University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Mem2Speech: An Intelligent Character Recognition-to-Speech Application using Long Short-Term Memory Networks

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Jerome Patrick V. Gonzalvo

June 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Mem2Speech: An Intelligent Character Recognition-to-Speech Application using Long Short-Term Memory Networks" prepared and submitted by Jerome Patrick V. Gonzalvo in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Marvin John C. Ignacio, M.Sc. (*cand.*)
Adviser

EXAMINERS:

	Approved	Disapproved
1. Richard Bryann L. Chua, M.Sc. (*cand.*)		
2. Avegail D. Carpio, M.Sc.		
3. Marbert John C. Marasigan (*cand.*)		
4. Ma. Sheila A. Magboo, M.Sc.		
5. Vincent Peter C. Magboo, M.D., M.Sc.		
6. Berwin Jarret T. Yu (*cand.*)		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Smartphones are no longer used solely for communication but also for photography, video recording, internet surfing, etc. As technology continues to advance, it is possible to apply some techniques to perform tasks such as Optical Character Recognition (OCR) and Intelligent Character Recognition (ICR). Tess2Speech is one of the applications that perform these tasks and it uses the Tesseract OCR Engine. Each trained Tesseract model of Tess2Speech is able to recognize a handwrting style of a user but not that of other users. It is possible for each user to train the engine themselves to fit their handwriting but this may be an issue if they don't have the technical background needed to train the engine. This special problem uses Long Short-Term Memory (LSTM) networks for handwritten text recognition. It provides a user-friendly trainer that will help AI experts in training handwriting recognition models. It also provides a mobile application for recognizing handwritten texts using the trained models.

Keywords: Artificial Intelligence, Machine Learning, Deep Learning, Recurrent Neural Networks, Long Short-Term Memory Networks, Optical Character Recognition, Intelligent Character Recognition

Contents

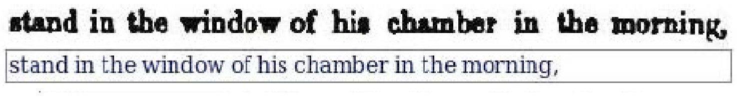
List of Figures

I. Introduction

A. Background of the Study

Smartphones have been known as most commonly used electronic devices in daily life today. They are no longer used solely for communication but also for photography, video recording, internet surfing, etc. [1] As technology continues to advance, it is possible to apply some techniques to perform tasks such as Optical Character Recognition (OCR) and Intelligent Character Recognition (ICR).

OCR is a technology that allows users to convert text or documents in images captured by an input device into an editable, searchable and reusable data type for further image processing [2]. It has been widely used in many areas, mainly in document preservation [3].


stand in the window of his chamber in the morning,

stand in the window of his chamber in the morning,

Figure 1: Optical Character Recognition [4]

ICR is an advanced OCR which allows different fonts to be learned by a computer. Among these fonts are different styles of handwriting. It is a challenging task because there is a high variablity of handwriting styles. High recognition rates are achieved in character recognition and isolated word recognition, but there is still room for development in achieving high recognition rates for unconstrained offline handwritten texts [5].

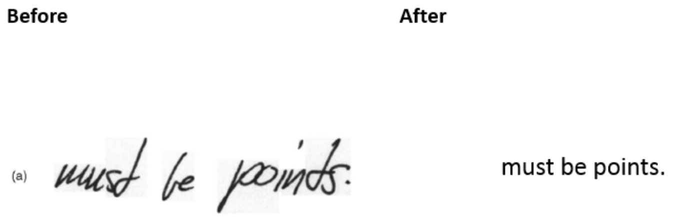


Figure 2: Intelligent Character Recognition

Machine learning (ML), an approach to Artifical Intelligence (AI), uses information from the data represented as features. Unlike knowledge-based systems which rely on hard-coded knowledge, ML uses these features to learn patterns.

Neural networks, one of the approaches used in ML, take in training data and produces an output based on the weights of each neuron in the network. Neural networks aim to find a set of weights that will produce the correct output for the majority of inputs [6] by adjusting the weights to fit the training data, and adjusting the hyperparameters. Hyperparameters influence the training. A few of these hyperparameters are the learning rate, which determines the rate of change on the weights; batch size, which determines the number of input data the network has to take before the adjustment of weights take place; solver type, the optimization function on adjusting the weights, and; the number of epochs, the number of passes to be made on the entire input data. The training data can't be used as the test data or validation data as it will eventually learn the training data too well but when introduced to unseen data, the network won't be able to produce the correct output. Thus, it is necessary to evaluate the network on a testing dataset.

The features used in ML are hand-crafted in a way that one has to carefully select them in order represent the data. This becomes an issue when the data has a lot of features to extracted.

Representation Learning, another approach to AI, uses ML to learn the features themselves [6]. This eliminates the problem of choosing the best features to extract from the data. But this is not enough for tasks that involve high-level, abstract features such as handwriting recognition.

Deep learning expresses these representations in terms of simpler ones [6]. It enables computers to build complex concepts out of simple ones. Moreover, the task of extracting the relevant features from data is left to the computer. One of the approaches used in Deep Learning is Long Short-Term Memory Networks.

Long Short-Term Memory Networks (LSTMs) have already been applied in several handwriting recognition tasks. The motivation for using LSTMs is its capbility to learn sequences that occur in time-series. Moreover, LSTMs can maintain long-term dependencies. According to Espana et al. [5], handwriting can be interpreted as a sequence of ink signals. With this in mind, LSTMs can be applied to the handwriting recognition task.

Custodios Tess2Speech application [7] has both OCR and ICR capabilities. This means that the application can recognize not only texts written on the smart phones screen but also images containing computer-typed or handwritten texts, and convert it to both editable text, and speech. His application used the Tesseract OCR engine, an open source OCR library currently owned by Google, to perform such tasks [8].

B. Statement of the Problem

Each trained Tesseract model of Tess2Speech is able to recognize a handwrting style of a user but not that of other users. This lead to the creation of the desktop trainer application to be trained by the user before they can use the mobile application in order to recognize their handwritings (refer to figure 3). A user won't be able to use the mobile application if they don't have the technical background needed to use the desktop trainer.

C. Objectives of the Study

1. This special problem creates a mobile based image-to-speech application that allows the owner of the application to:

    (a) Either upload an image that contains the handwritten or printed text, or;

    (b) Write the handwritten text itself on the phone's screen.

    (c) Output the voice equivalent of the converted texts.

    (d) Update the application's model through software updates.

2. This reasearch also creates a system that allows an AI Expert to:

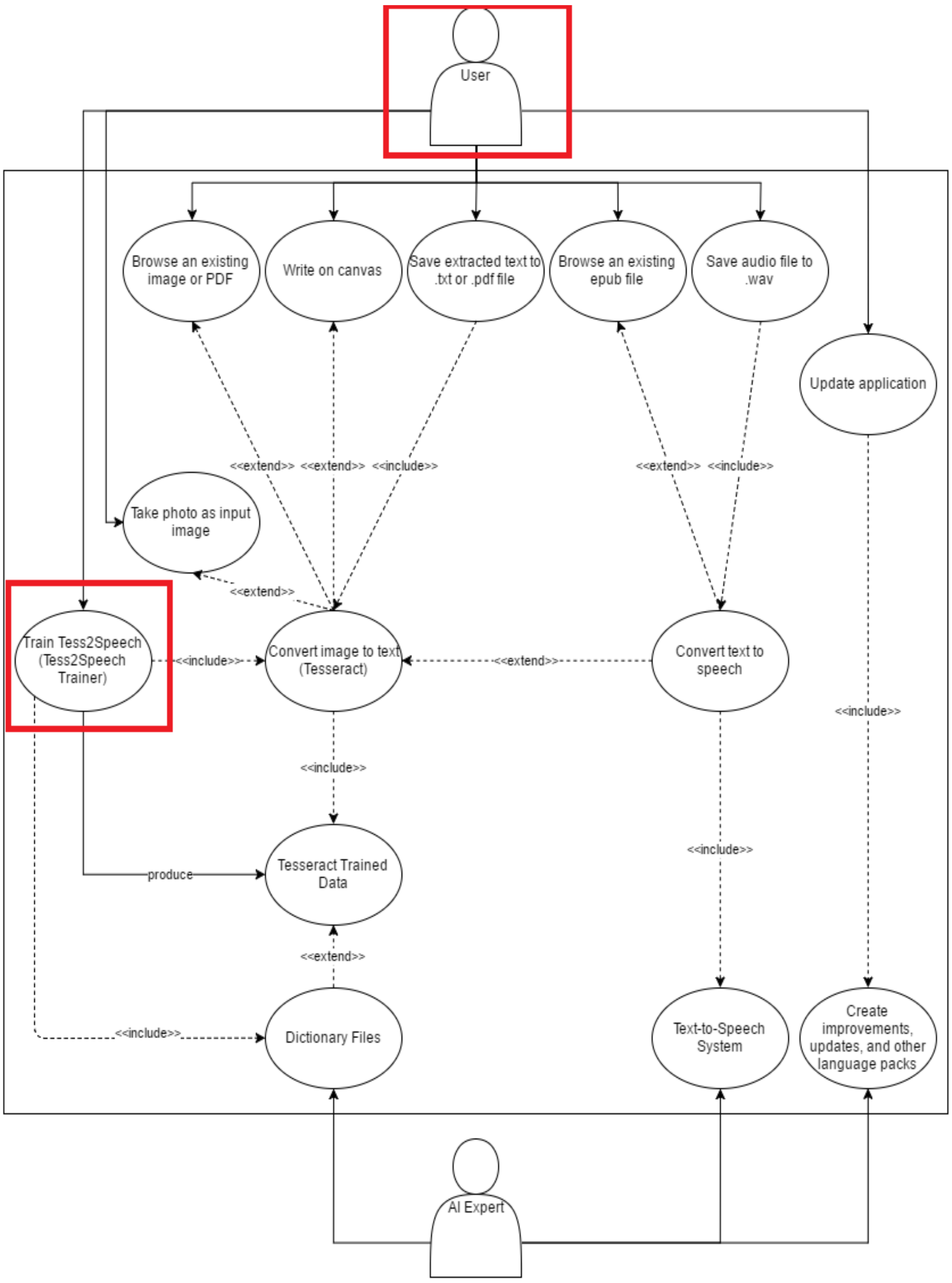


Figure 3: Use-case Diagram of Tess2Speech

(a) Prepare the dataset

i. Select the training dataset of handwritten text images

ii. Select the test dataset of handwritten text images

iii. Display the number of training and test data

(b) Create the recognition model

   i. Select a prepared dataset

   ii. Select LSTM Architecture

   iii. Select the following hyperparameters:

      A. Training epochs

      B. Solver type

      C. Learning rate

      D. Batch size

   iv. Train the classifier to create an recognition model

   v. Display training and validation error for each epoch using tensorboard

(c) Evaluate the recognition model

   i. Select recognition model to evaluate

   ii. Perform predictions using a test dataset

      A. Select the test dataset

      B. Extract the machine readable text from all the test data through the model and output to a log file

      C. Display the Character Error Rate (CER) for the whole data

D. Significance of the Project

1. By replacing the engine of Custodio's application with an engine that is flexible to the handwriting of any person, the user does not need to train the desktop application to recognize their handwriting; the training is now delegated to the AI Expert.

2. The user of the application is not restricted to a handwriting style as the engine will also be flexible to all handwriting styles (printed or cursive).

E. Scope and Limitations

1. Mem2Speech accepts JPEG, GIF, PNG, and BMP image files as input.
2. Words that Mem2Speech can convert to speech are limited to the English alphabet.
3. The training is not a functionality of the Mem2Speech mobile application. Training is separately done on the desktop.
4. The dataset to be used for training and testing is the IAM handwriting dataset. [9]
5. Handwriting must only contain characters that are alphanumeric. They can be cursive, or printed.
6. The user has a Text-to-Speech Engine installed on their mobile phone.

F. Assumptions

1. There are words or characters on the images that Mem2Speech is converting.

II. Review of Related Literature

Espana et. al. [5] presented techniques in preprocessing offline handwritten text by using Artificial Neural Networks (ANNs) for cleaning the handwritten text images, correcting the slope and slant of the cleaned images, and normalizing their sizes. As for their recognizer, they used Hidden Markov Models (HMMs) motivated by the idea that handwriting can be interpreted as a left-to-right sequence of ink signals.

Frinken et. al. [10] used LSTM Networks Language Modeling for Handwriting Recognition to model a target language. Since LSTMs are capable of dealing with long-term dependencies, the context the network can consider is not limited to a fixed size which is a limitation of using statistical n-grams. They tested their system on the IAM handwriting database [9] containing handwritten English text and it showed that their system outperforms statiscal n-gram models.

Breuel and Shafait [11] presented a simple application of Long Short-Term Memory (LSTM) Networks for recognizing printed English and Fraktur, without language modeling. They compared the performance of their architecture against other OCR systems [4, 12, 13]. It was shown that despite the improvements being small in terms of accuracy, their architecture outperformed the mentioned OCR systems in terms of differences between OCR systems and in terms of practical applications, greatly reducing the need for manual post-processing procedures such as language modeling. They focused on 1D LSTM networks as it was shown in their preliminary experiments that they are faster and better than its 2D variant which is used for handwriting recognition.

Ul-Hasan et. al. [14] used Bidirectional LSTM Networks for Offline Printed Urdu Nastaleeq Script Recognition. Their architecture used a Connectionist Temporal Classification (CTC) output layer to avoid the requirement of pre-segmenting input data as it was found to be a limiting factor to the utility of traditional Recurrent Neural Networks (RNNs) for handwriting recognition. [15]

Doetsch et. al. [16] demonstrated a modified topology for long LSTM RNNs that manipulate gating units. They also proposed an efficient training framework based on a mini-batch training on sequence level combined with a sequence chunking approach. This framework was evaluated on the IAM handwriting database [9], and the RIMES handwriting database [17] containing handwritten English, and handwritten French text, respectively.

Zamora et. al. [18] integrated neural network language models in the decoding process of three systems: one based on bidirectional RNNs, another based on HMMs, and a combination of both. They evaluated each of these modified systems on the IAM handwriting database [9] and it was shown that the reduction of word error rates is better with neural network language models than statistical N-gram language models.

Bluche et. al. [19] described the Arabic handwriting recognition systems proposed by A2iA to the NIST OpenHaRT2013 evaluation. The systems A2iA proposed were based on an optical model using LSTM networks, trained to recognize different forms of Arabic characters directly from the image, without explicit feature extraction nor segmentation. They also used large vocabulary selection techniques and n-gram language modeling to provide a full paragraph recognition, without explicit word segmentation.

An improvement of this architecture is presented in [20] by using a technique called "dropout" which was origiginally proposed in [21]. Dropout involves randomly removing some hidden units in a neural network during training but keeping all of them during testing. This is applied to only feed-forward connections and not to recurrent connections. By doing so, this process can be seen as a way to combine high-level features learned by recurrent layers. They evaluated their system on three handwriting datasets: RIMES [17], IAM [9], and OpenHaRT [22] containing handwritten French, English, and Arabic text, respectively. Their system achieved the best results among that of other systems mentioned in their work on RIMES and OpenHaRT databases.

Voigtlaender et. al. [23] used their own GPU-based implementation of deep

Multidimensional LSTMs (MdLSTMs) for handwriting recognition that processes input images in a diagonal-wise fashion to reduce training time. They also used the IAM handwriting dataset [9] and RIMES [17] dataset. They achieved good performance by pulling the non-recurrent parts of the computations out of the loops for the recurrences, using custom CUDA kernels for the MdLSTM gating mechanism. On raw data, their recognitions system achieved a word error rate of 8.5% on the validation set, 11% on the test set; the recognition system achieved a character error rate of 3% and 4.3% on the respective mentioned sets.

Ray et. al. [24] proposed a segmentation-free and script-independent approach for Oriya text recognition. They used Deep Bidirectional LSTM network architecture because of its ability to access long-term dependencies, learn sequence alignment and work from raw input. Their architecture used a Connectionist Temporal Classification (CTC) for training to learn labels of an unsegmented sequence with unknown alignment. Due to the use of CTC and forward backward algorithms for alignment of the output labels, the need for a language model or postprocessing schemes was eliminated.

III. Theoretical Framework

A. Optical Character Recognition

A..1 Input Image

These are digitalized images like scanned or captured text images. It could be of different formats, i.e. TIFF, JPEG, GIF, PNG, and BMP image.

A..2 Classification/Recognition

It is the step wherein optical patterns are classified into alphanumeric and other characters. The information should be readable to both human and machine.

A..3 Output Text

The text that is extracted from the input image is displayed in the output text.

B. Intelligent Character Recognition

ICR is an advanced OCR which allows different fonts to be learned by a computer. Among these fonts are different styles of handwriting.

C. Recurrent Neural Networks, Long Short-Memory Networks, Dropout

A recurrent neural network (RNN) is a neural network model proposed for time series analysis. They are considered good at context-aware processing and to recognize patterns occurring in timeseries. [25] An illustration of a rolled version of this network can be found in figure 4 and its unrolled version in 5.

But RNNs are not good at maintaining long-term dependencies due to the vanishing gradient problem (also known as the exploding gradient problem). [26]

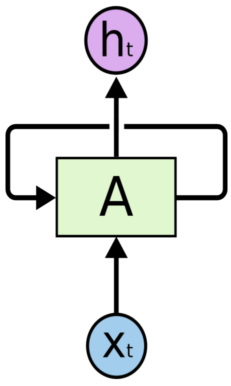


Figure 4: Recurrent Neural Networks. They are networks with loops in them allowing information to persist. A is the chunk of the neural network (hidden layer) that looks at some input x_t and outputs a value h_t

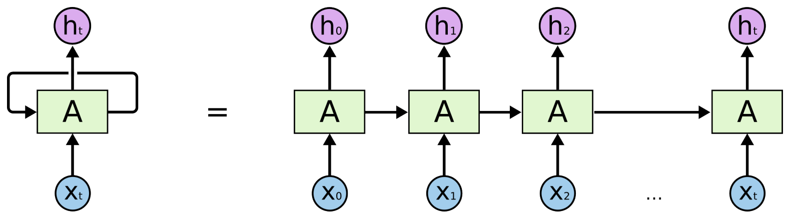


Figure 5: Recurrent neural networks can be thought of as multiple copies of the same network connected next to each other.

C..1 Long Short-Term Memory Networks (LSTMs)

Traditional RNNs have not shown competitive performance in large scale tasks like OCR and speech recognition due to the vanishing gradient problem [27, 28]. LSTM Networks [29], a type of recurrent neural network, address this problem.

LSTMs follow the same chain-like structure as RNNs but each repeating module has a different structure [25] (see figure 6; figure 7 for the notation).

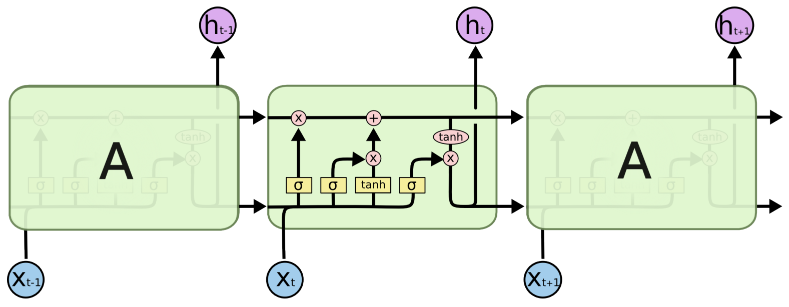


Figure 6: A Long Short-Term Memory Network

The key to LSTMs is the cell state (see figure 8). [25]

LSTMs have the ability to remove or add information to the cell state by using

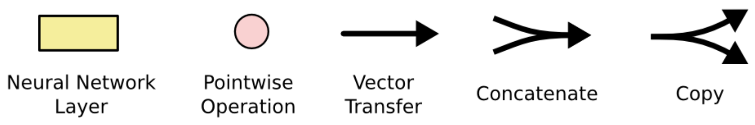


Figure 7: Notation for LSTM.

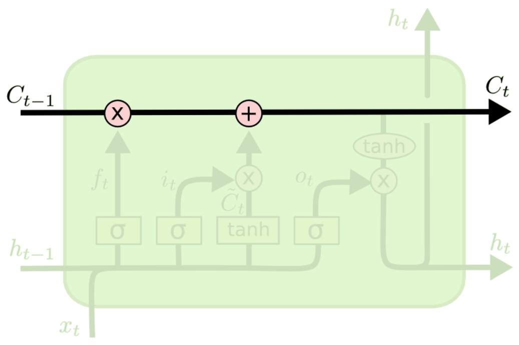


Figure 8: The cell-state. This is where the information flows.

gates and it has three of them.

The first gate, "forget gate" (see figure 10), decides which information gets thrown away from the cell state. [25]

The second gate, "input gate" (see figure 11), decides which information gets stored into the cell state. [25]

The old cell state C_{t1} is then updated into the new cell state C_t (see figure 12). [25]

The last gate, output gate (see figure 13), will then decide which information is going to be in the output based on the cell state. [25]

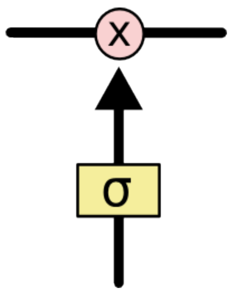
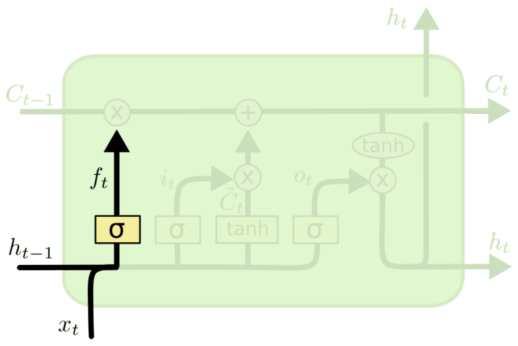
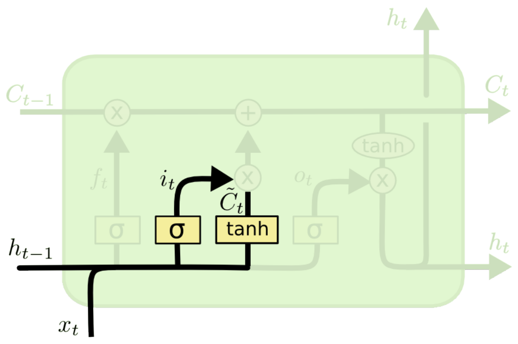


Figure 9: An LSTM gate. It consists of a sigmoid layer and a pointwise multiplication operator. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means let nothing through, while a value of one means let everything through!



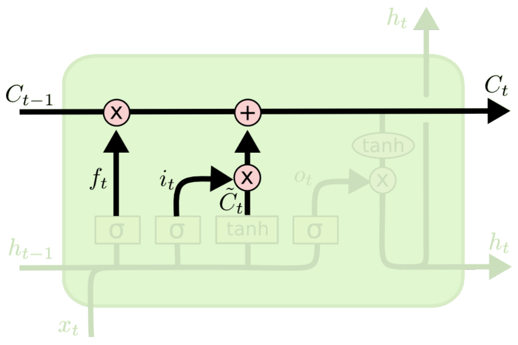
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 10: The Forget Gate. It looks at ht1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct1. A 1 represents completely keep this while a 0 represents completely get rid of this.



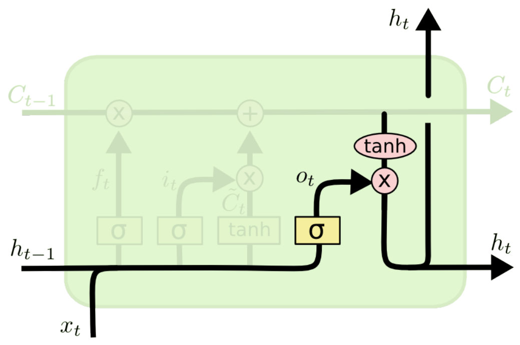
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 11: The Input Gate. The input gate layer decides the values to be updated. The tanh layer creates a vector of new candidate values C̆t that could be added to the cell state. The outputs of these two are then concatenated to create an update for the cell state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 12: Updating the Cell State. The old state is multiplied by ft to throw away information from the old cell state. Then new candidate values it * C̆t are then added to update the cell state value.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 13: The Output Gate. A sigmoid layer decides what parts of the cell state are going to be in the output. Then the cell state goes through a $\tanh$ layer (to push the values to be between 1 and 1) and multiplied to the output of the sigmoid layer.

C..2 Multidimensional Long Short-Term Memory Networks (MdLSTMs)

While LSTMs operate on one dimension of data (the x-axis of an image or the time-axis for audio), MdLSTMs operate over both axes capturing more writing variations than that of plain LSTMs, and directly work on raw input images.

For this study, the architecture proposed by Voigtlaender et al. [23] will be used. The two key components of the architecture are the use of MdLSTM layers, and the use Connectionist Temporal Classification (CTC) which handles the alignment between the input and the output sequence. The architecture also uses dropout which will be discussed next.

C..3 Dropout

Dropout involves randomly removing some hidden units in a neural network during training but keeping all of them during testing. This is applied to only feed-forward connections and not to recurrent connections. By doing so, this process can be seen as a way to combine high-level features learned by recurrent layers. [20]

D. Text-to-Speech

The focus of this project is on Character Recognition. The installed Text-to-Speech (TTS) system will be used and no further improvement of the TTS system will be done. Users are allowed to download any TTS Engines that they prefer in Google Play Store.

E. Android OS

Android is a mobile operating system (OS) which was developed and currently being developed by Google [30]. Its operating system is based on the Linux kernel and is primarily designed for smart phones and tablets. Android applications are developed using Java Programming Language and Android Software Development Kit (SDK).

F. Java Programming Language

Java Programming Language is a general-purpose, concurrent, class-based, object-oriented language. Its syntax is somewhat related to C and C++ but is organized differently. Java language is strongly typed which means that it has a strict set of rules when it comes to programming syntax. Java programming language also distinguishes run-time errors and compile-time errors. It is also a relatively high-level programming language since it includes automatic storage management and garbage collectors which automatically performs deallocation unlike in C and C++ which uses explicit deallocation [31]. Java needs Java Virtual Machine (JVM) installed on the computer to work. Java Programming Language is also used in Android application development alongside other Android SDKs.

G. Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android Application Development.

H. Text-to-Speech System

Any Text-to-Speech system installed can be used on Mem2Speech. This means that the performance of Mem2Speech' TTS is based on the TTS being used on the phone. Text-to-Speech systems can be downloaded on Google Play Store. TTS settings can be changed on Android's phone settings.

I. Mem2Speech Canvas

Mem2Speech will use the canvas implemented in Tess2Speech [7].

J. Mem2Speech Camera

Almost all smart phones have a camera, and it is accessible by applications, utilizing a camera, installed on the phone. Mem2Speech will access the phone's camera.

K. Settings

The preference screen of Mem2Speech contains different settings such as help, and Licenses.

L. Mem2Speech Trainer

Mem2Speech Trainer is a user-friendly desktop application which allows an AI expert to upload datasets, build their own architectures, train handriting recognition models and test them, and so on. It produces models from which an AI expert can choose for Mem2Speech to use.

IV. Design and Implementation

A. Use-case Diagram

Mem2Speech caters to two types of users: the general users and the AI experts. The general users are the application users that uses the main functionalities of Mem2Speech, while the AI experts are those that perform the training of the models and create updates for the application. The use-case diagram of Mem2Speech is shown in 14.

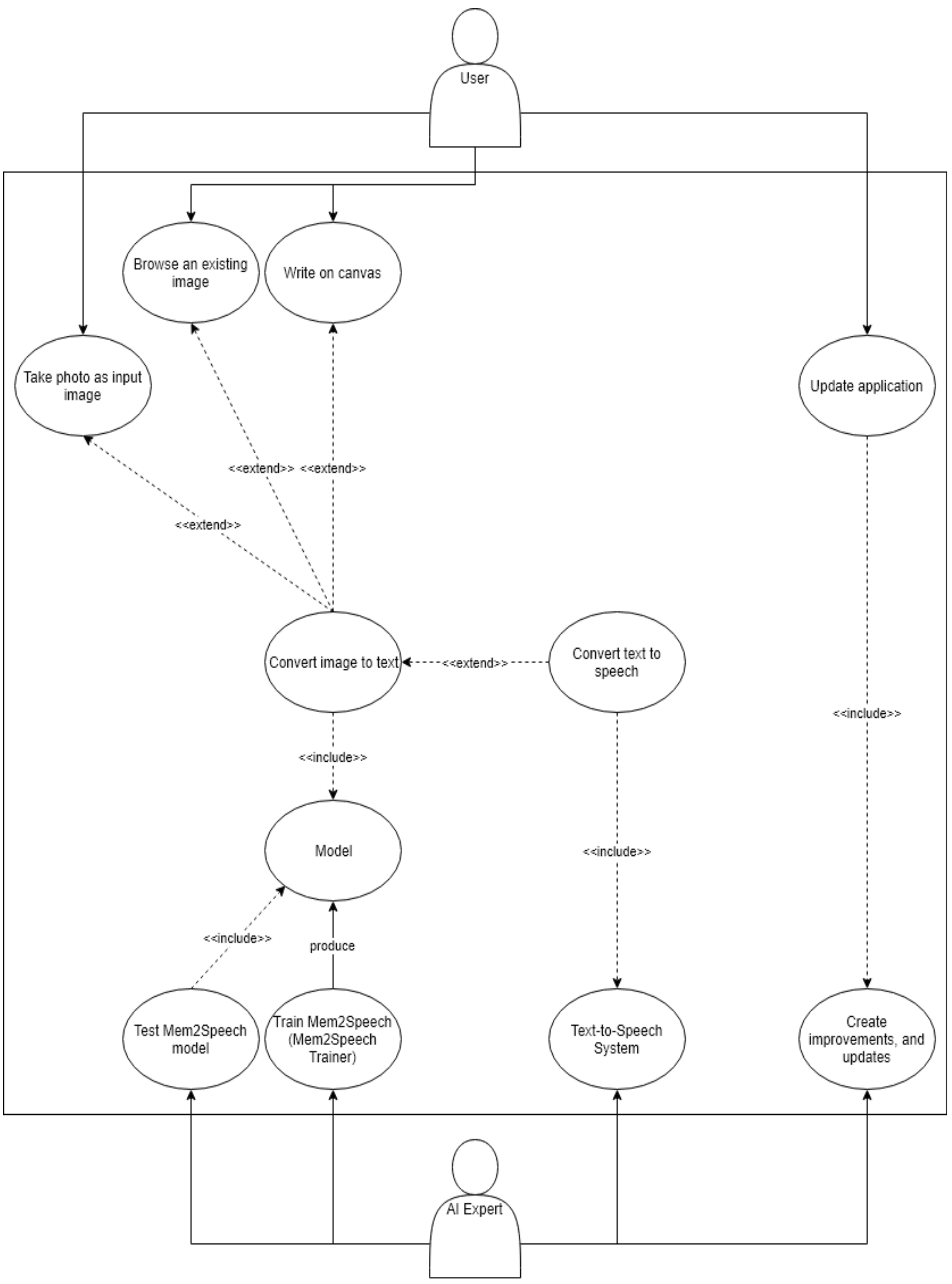


Figure 14: Use-case Diagram of Mem2Speech

B. Block Diagram of Mem2Speech Mobile Application

Mem2Speech combines the LSTM-based model, which converts the input images into text, and a Text-to-speech engine to convert it to speech. The block diagram is shown in 15.

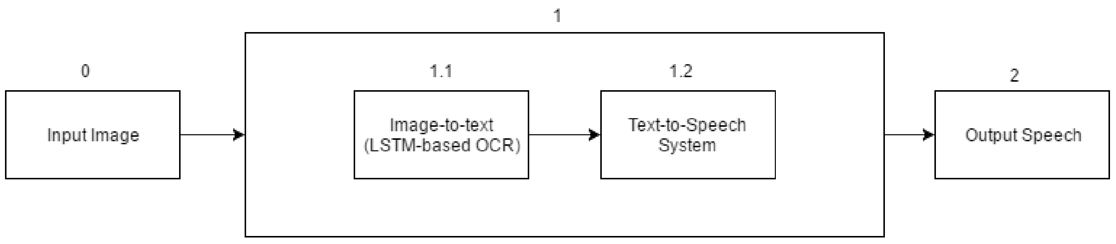


Figure 15: Block Diagram of Mem2Speech

C. Flowcharts

C..1 General User

Input file refers to the image that contains the text the user wants to convert into an text and/or speech. This can be PNG, JPEG, BMP, and GIF for images. Mem2Speech gives the users choices on what input they want to process. The users can choose to browse their phone for an existing input file, or use the canvas to write on the screen. The input file is fed to the OCR engine and converted to text that is displayed in a textbox. The generated texts can now be converted into speech by pressing play button. The flow chart of this application is shown in 16.

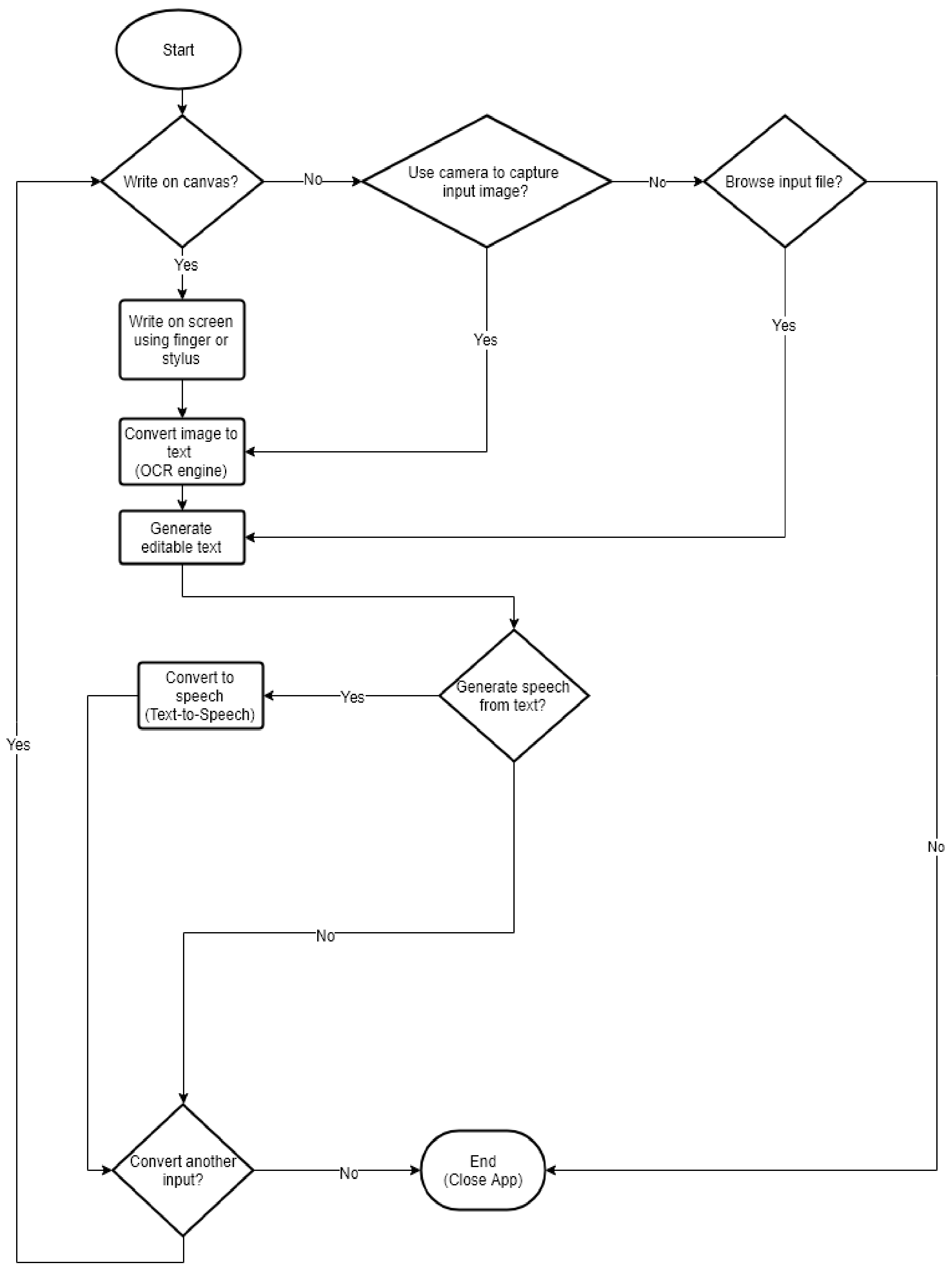

Start
Write on canvas?
No
Use camera to capture input image?
No
Browse input file?
Yes
Write on screen using finger or stylus
Yes
Yes
Convert image to text (OCR engine)
Generate editable text
Convert to speech (Text-to-Speech)
Yes
Generate speech from text?
Yes
No
No
Convert another input?
No
End (Close App)

Figure 16: Flowchart of Mem2Speech Mobile Application

C..2 AI Expert

If the AI expert encounters a new dataset, the AI expert may use it to produce a better model. The AI expert may also create a new model that uses a different architecture and see if it performs better than the working one.

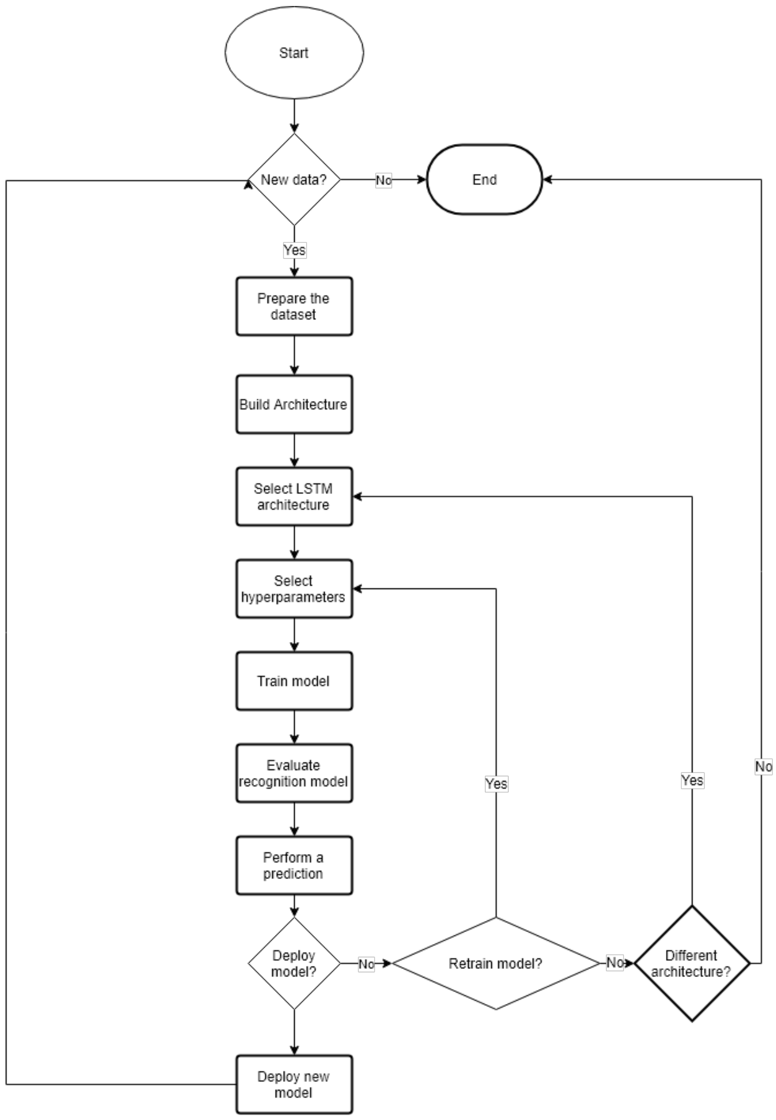


Figure 17: Flowchart of AI Expert

D. Technical Architecture

The underlying desktop application implementation makes use of Tensorflow [32], a deep learning library in Python.

1. These are the minimum system requirements for Mem2Speech mobile appli-

cation:

   (a) Android Lollipop (5.0) or higher

   (b) At least 2 GB of free disk space, for the dependencies.

2. These are the desktop application requirements:

   (a) At least 2GHz CPU

   (b) At least 4 GB RAM or higher

   (c) At least 2 GB free disk space

   (d) CUDA-capable GPUs with at least Compute Capability 5.0

3. These are the dependencies:

   (a) Anaconda with Python 3.6

   (b) Tensorflow 3.4

   (c) CUDA

V. Results

A. Mem2Speech Trainer

The trainer allows an AI expert to manage the datasets, build and view architectures, and facilitate training and testing.

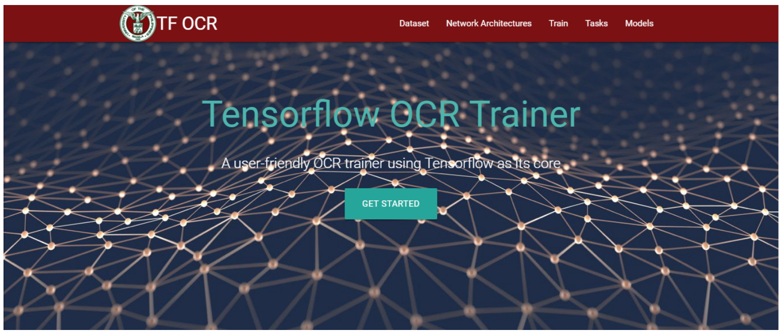
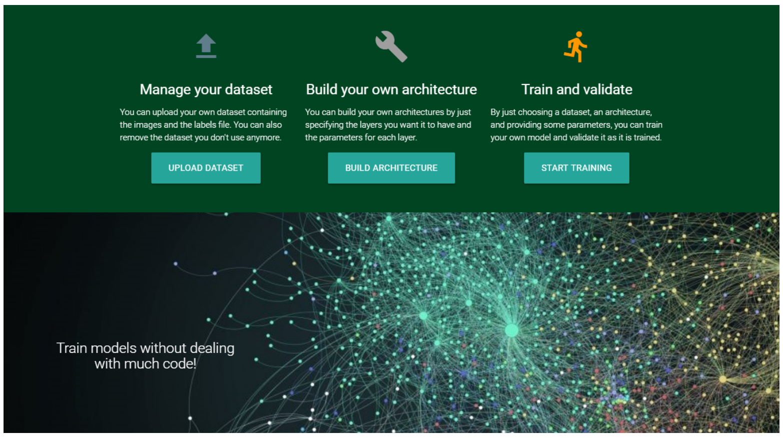


Figure 18: First part of the homepage



Manage your dataset

You can upload your own dataset containing the images and the labels file. You can also remove the dataset you don't use anymore.

Build your own architecture

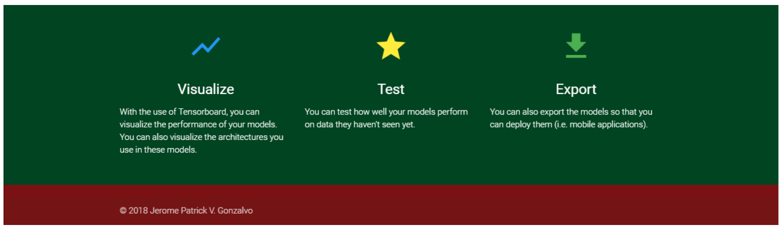
You can build your own architectures by just specifying the layers you want it to have and the parameters for each layer.

Train and validate

By just choosing a dataset, an architecture, and providing some parameters, you can train your own model and validate it as it is trained.

Train models without dealing with much code!

Figure 19: Second part of the homepage



Visualize

With the use of Tensorboard, you can visualize the performance of your models. You can also visualize the architectures you use in these models.

Test

You can test how well your models perform on data they haven't seen yet.

Export

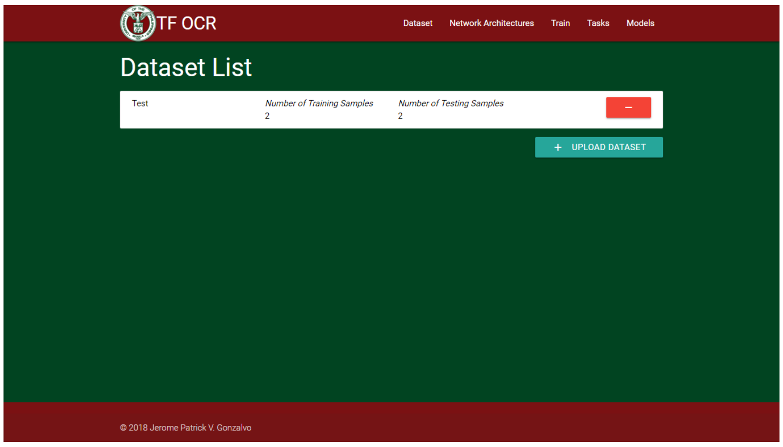
You can also export the models so that you can deploy them (i.e. mobile applications).

© 2018 Jerome Patrick V. Gonzalvo

Figure 20: Third part of the homepage

A..1 Dataset Management

The dataset management page allows the AI expert to upload datasets, and remove them.



	Number of Training Samples	*Number of Testing Samples*
Test	2	2

Figure 21: Dataset Viewing

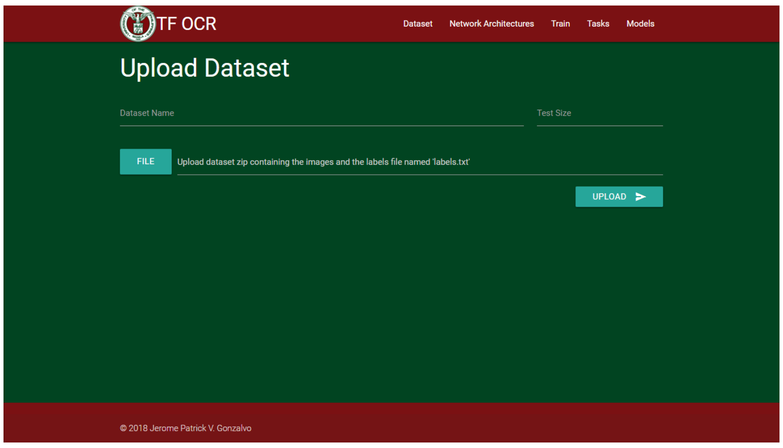

TF OCR
Dataset
Network Architectures
Train
Tasks
Models
Upload Dataset
Dataset Name
Test Size
FILE
Upload dataset zip containing the images and the labels file named 'labels.txt'
UPLOAD
© 2018 Jerome Patrick V. Gonzalvo

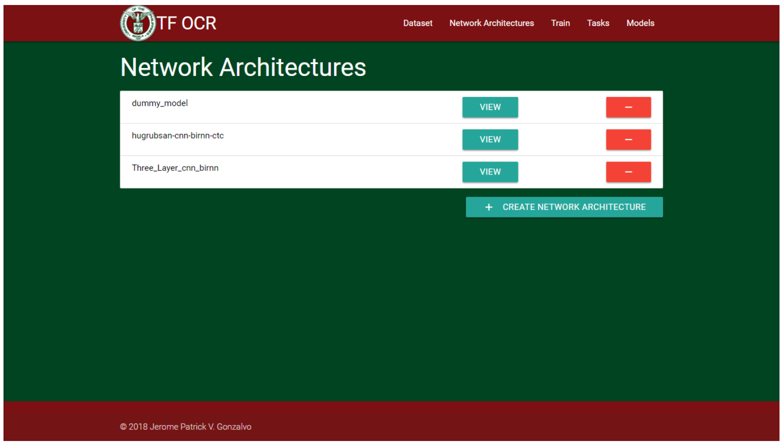
Figure 22: Dataset Uploading

A..2 Architecture Building and Viewing

The architecure building and viewing page allows the AI expert to build their own architectures and view them.

A..3 Training

The training page contains the form that the AI expert is required to fill out to start training.



Network Architectures

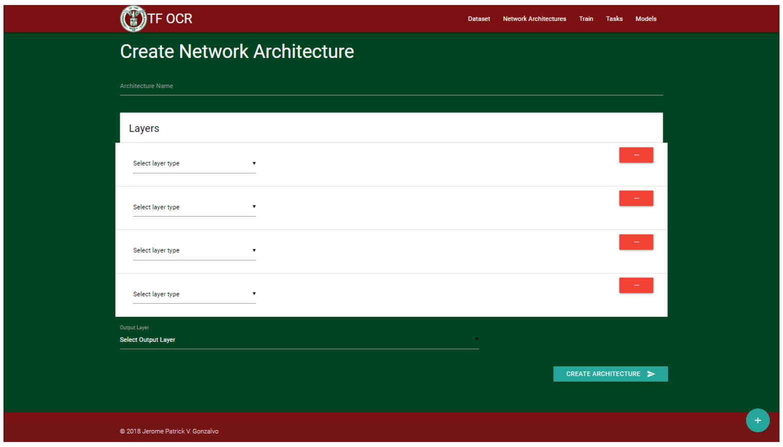
dummy_model

hugrubsan-cnn-birnn-ctc

Three_Layer_cnn_birnn

© 2018 Jerome Patrick V. Gonzalvo

Figure 23: Viewing the list of network architectures created



Create Network Architecture

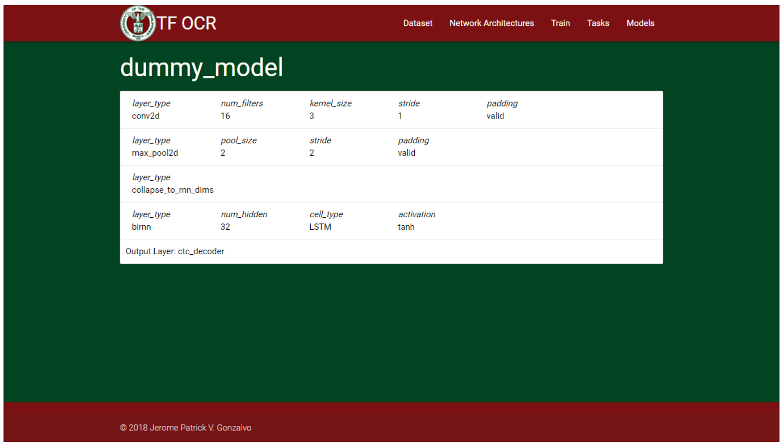
Architecture Name

Layers

Output Layer
Select Output Layer

© 2018 Jerome Patrick V. Gonzalvo

Figure 24: Creating a network architecture



dummy_model

layer_type conv2d	*num_filters* 16	*kernel_size* 3	*stride* 1	*padding* valid
layer_type max_pool2d	*pool_size* 2	*stride* 2	*padding* valid	
layer_type collapse_to_rnn_dims				
layer_type birnn	*num_hidden* 32	*cell_type* LSTM	*activation* tanh	

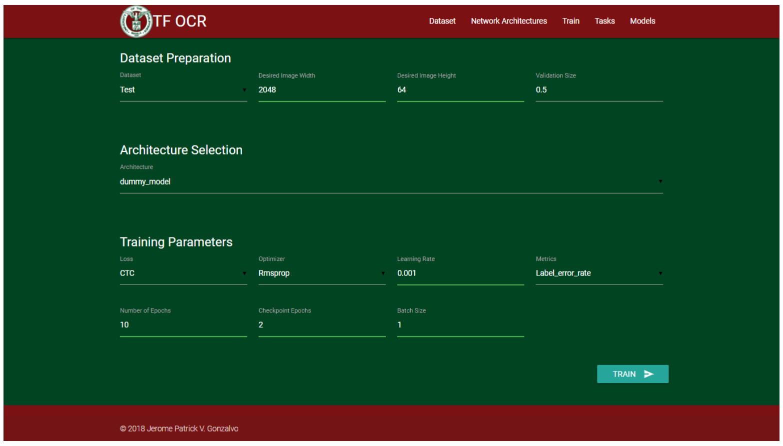
Output Layer: ctc_decoder

© 2018 Jerome Patrick V. Gonzalvo

Figure 25: Viewing a network architecture

A..4 Task Management

The task management page allows the AI expert to view the running tasks. These tasks may be training, testing, or model visualization.



Dataset Preparation

Dataset	Desired Image Width	Desired Image Height	Validation Size
Test	2048	64	0.5

Architecture Selection

Architecture: dummy_model

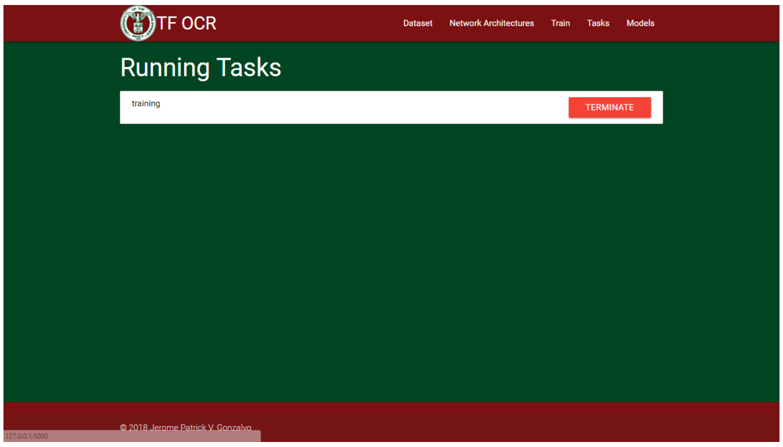
Training Parameters

Loss	Optimizer	Learning Rate	Metrics
CTC	Rmsprop	0.001	Label_error_rate

Number of Epochs	Checkpoint Epochs	Batch Size
10	2	1

© 2018 Jerome Patrick V. Gonzalvo

Figure 26: Training form to be filled out by the AI expert



Running Tasks

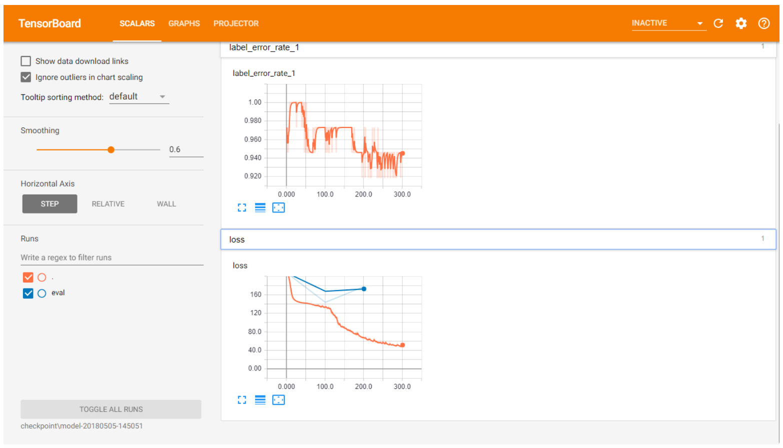
training

© 2018 Jerome Patrick V. Gonzalvo

Figure 27: Viewing list of running tasks

A..5 Visualization

The visualization page allows the AI expert to view the progress of training, testing.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing: 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
☑ .
☑ eval

checkpoint\model-20180505-145051

Figure 28: Training and testing loss, and label error rate plot

B. Mem2Speech Mobile Application

The mobile application takes in a handwriting input on the canvas and performs the recognition with the press of a button.

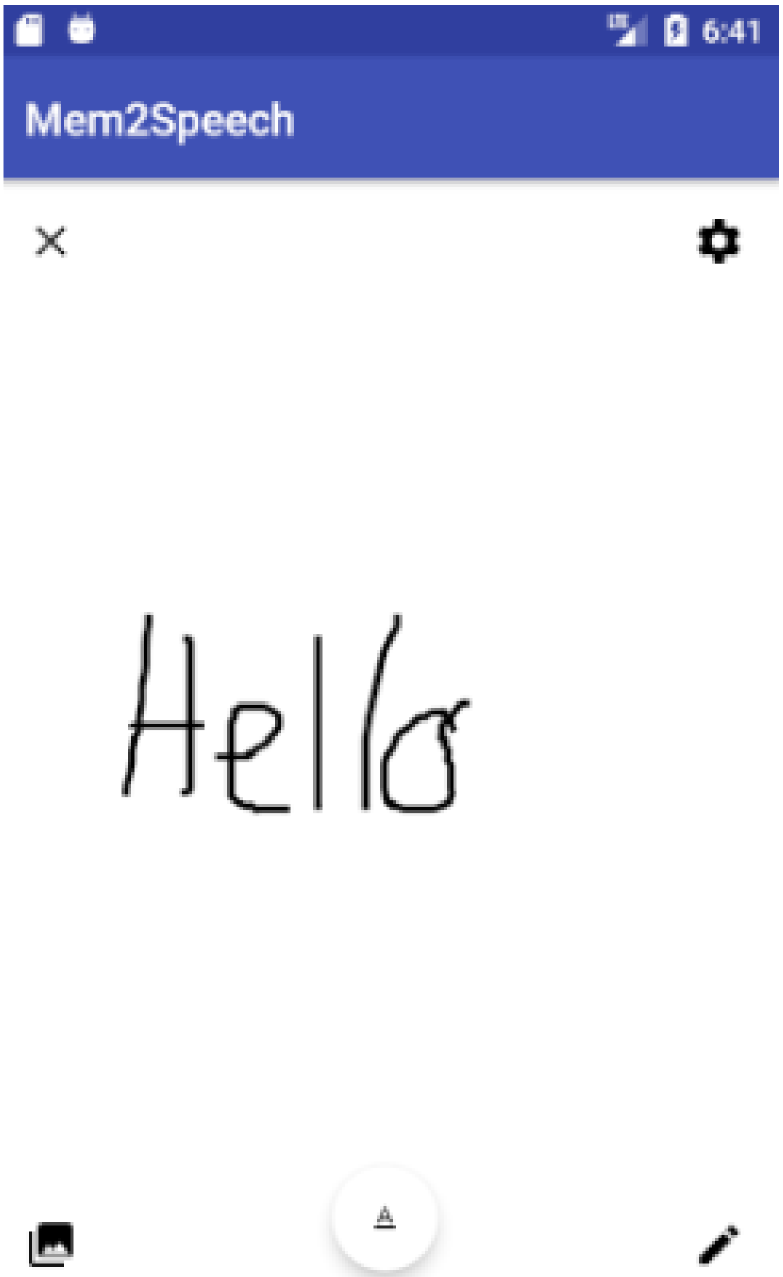
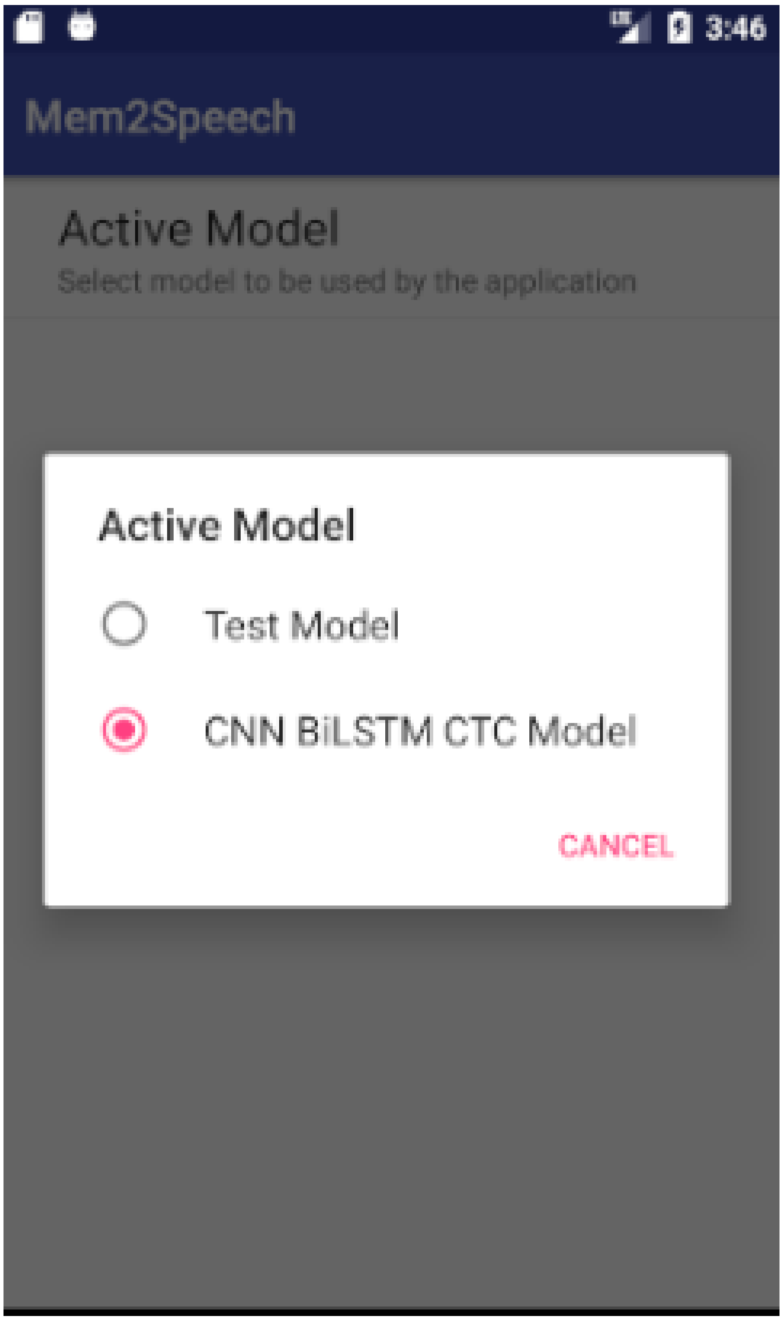


Figure 29: Drawing canvas for handwriting

It is also possible to switch between models in the application.



Active Model
Select model to be used by the application

Active Model

( ) Test Model
[✓ CNN BiLSTM CTC Model]

Figure 30: Selecting model to be used by the application

VI. Discussions

A. Mem2Speech Trainer

A..1 Training, Validation, and Testing

The researcher used an architecture based on the MdLSTM architecture proposed by Voigtlaender et. al. [23].

The input, having a shape of [Batch size, Height, Width, Number of channels], goes through five parts of repeating 3x3 convolution, max pooling, and MdLSTM layers. A dropout of 0.25 is applied to all forward connections of all convolutional layers except for the first one so as to not drop the single color channel of the input image. The outputs of this network are passed onto the fully connected layer having N hidden units where N is the number character classes in the charset (there are 79 classes in the charset used for this special problem), and a softmax activation function. The outputs of the fully connected layer are passed onto the CTC loss function to calculate the loss. Said outputs are also passed onto the CTC beam search decoder function to get the decoded outputs. The outputs from the decoder are then passed with the labels to calculate the label error rate. These outputs are transformed to get the actual output to be decoded using the charset.

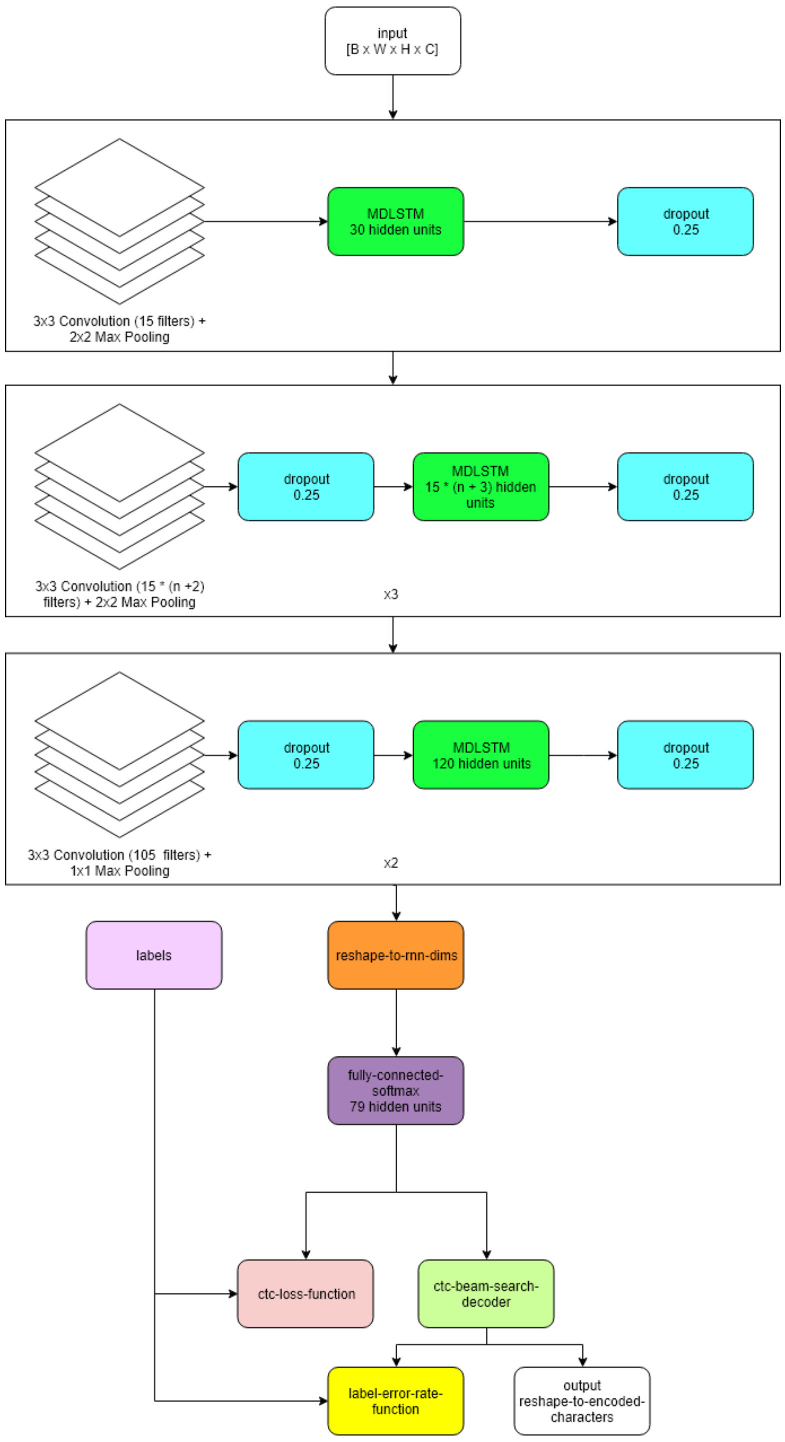


Figure 31: The architecture of the model

The IAM handwriting dataset [9] specifically the dataset containing 13,353 isolated lines. Not all examples from this dataset are used since some transcriptions do not match the true word written on the image. The dataset is split into training and testing sets first – 20% of the dataset is used as the test set. The training set is split into training and validation sets – 20% of the training set is used as the validation set. All of the images are resized and padded to 2564x64 pixels, binarized, and inverted before feeding them into the model.

The solver type used is Adam [33] with Nesterov momentum [34] with a learning rate of 0.0005.

The model is trained for about 80 epochs. Checkpoints are saved, and validation is run every 20 epochs. The batch size of images is only 18 since the machine couldn't handle bigger batch sizes, and it is suggested in the work of Voigtlaender et. al. [23]. The training and validation progress is shown in 32

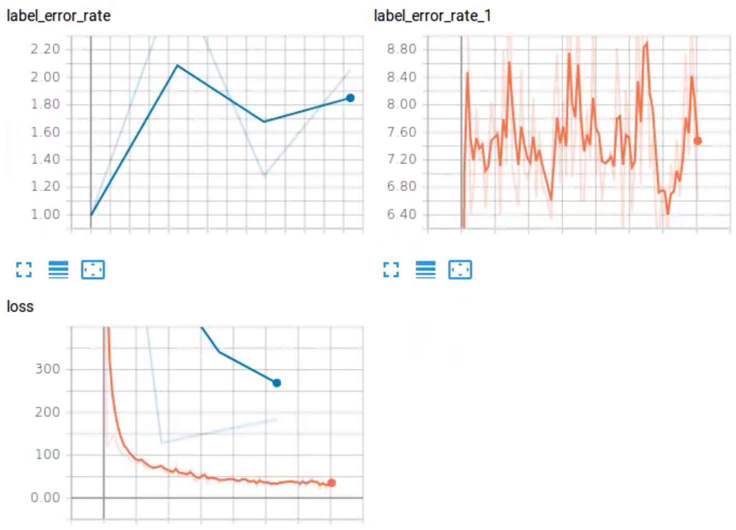


Figure 32: Training (orange) and validation (blue) progress

The label error rate is not going lower since the predictions is still far from the ground truths. The loss does get low but the model is not able to perform well even though the architecture is based on an existing on the paper [23] using the same dataset and learning parameters such as the learning rates, optimizer, the number of layers. One possible reason for this is that the architecture used on the paper used their own GPU-based implementation of MdLSTM through custom

CUDA kernels and cuBLAS, and the researcher is not able to reproduce such an implementation given how difficult it is to implement.

The researcher is not able to train a model that will cater to a different number of handwriting styles due to the lack of time and resources. The reason why the trainer is created is for the AI experts to build their own architectures, try out different handwriting datasets, and try out different training parameters to be able to create a model that will cater to almost, if not all, handwriting styles.

B. Mem2Speech Mobile Application

Mem2Speech uses trained and tested models to recognize handwriting. The user can upload an image that contains the handwriten text, or write the text itself on the screen through the canvas and convert it to text then to speech. The user can also change the model being used by the application.

Some features (such as the conversion of PDF images to text) of Tess2Speech are not added to this application since the researcher focused on the creation of the trainer and training handwriting recognition models.

C. Significance of the Mem2Speech Project

This special problem lead to the creation of a user-friendly trainer for AI experts to easily train handwriting recognition models making it possible to create better models using different datasets, different architectures, and different training configurations.

The users won't have to train a handwriting recognition model themselves since that task is left to the AI experts. All they have to do is either upload an image containing the handwritten text, or write the text itself on the screen through the canvas – the application will convert it to text then the user may also convert it to speech.

VII. Conclusions

Mem2Speech is a handwriting recognition application which should be able to recognize handwritten texts. Unfortunately, due to the lack of time and resources, the researcher was not able to train a handwriting recognition that will cater to a number of handwriting styles. With the help of the desktop trainer, AI experts can try out different datasets, different architectures, and different training configurations to train better handwriting recognition models.

The development of the trainer helped in the generalization of deep learning tasks by simplifying the common processes occuring in said tasks. With that, it may be possible to modify the trainer such that it would cater to a wide range of deep learning tasks.

VIII. Recommendations

In order to create a better handwriting recognition model, a more powerful machine may have to be used to accomodate larger batch sizes for faster training. If finding a more powerful machine is not possible, different datasets, different architectures, and different training parameters may be used to create a better handwriting recognition model. Applying more preprocessing techniques such as slant-correction may also help. Instead of predicting the labels per character, predicting the labels per word by using language models may yield a better performance.

IX. Bibliography

[1] S. Ramiah, T. Y. Liong, and M. Jayabalan, "Detecting text based image with optical character recognition for english translation and speech using android," in *2015 IEEE Student Conference on Research and Development (SCOReD)*, pp. 272–277, IEEE, 2015.

[2] S. Barve, "Optical character recognition using artificial neural network," *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 1, no. 4, pp. 2278–1323, 2012.

[3] S. B. Ahmed, S. Naz, M. I. Razzak, S. F. Rashid, M. Z. Afzal, and T. M. Breuel, "Evaluation of cursive and non-cursive scripts using recurrent neural networks," *Neural Computing and Applications*, vol. 27, no. 3, pp. 603–613, 2016.

[4] T. M. Breuel, "The ocropus open source ocr system," in *Electronic Imaging 2008*, pp. 68150F–68150F, International Society for Optics and Photonics, 2008.

[5] S. Espana-Boquera, M. J. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez, "Improving offline handwritten text recognition with hybrid hmm/ann models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 4, pp. 767–779, 2011.

[6] Y. B. Ian Goodfellow and A. Courville, "Deep learning.".

[7] A. A. Custodio, *Tess2Speech: An Intelligent Character Recognition-To-Speech Application for Android Using Google's Tesseract Optical Character Recognition Engine.* PhD thesis, University of the Philippines Manila, 2016.

[8] Google, "Tesseract-ocr," 2006.

[9] U.-V. Marti and H. Bunke, "The iam-database: an english sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.

[10] V. Frinken, F. Zamora-Martínez, S. Espana-Boquera, M. J. Castro-Bleda, A. Fischer, and H. Bunke, "Long-short term memory neural networks language modeling for handwriting recognition," in *Pattern Recognition (ICPR), 2012 21st International Conference on*, pp. 701–704, IEEE, 2012.

[11] T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, and F. Shafait, "High-performance ocr for printed english and fraktur using lstm networks," in *2013 12th International Conference on Document Analysis and Recognition*, pp. 683–687, IEEE, 2013.

[12] R. W. Smith, "History of the tesseract ocr engine: what worked and what didn't," in *IS&T/SPIE Electronic Imaging*, pp. 865802–865802, International Society for Optics and Photonics, 2013.

[13] E. Mendelson, "Abbyy finereader professional 9.0," *PC Magazine*, 2008.

[14] A. Ul-Hasan, S. B. Ahmed, F. Rashid, F. Shafait, and T. M. Breuel, "Offline printed urdu nastaleeq script recognition with bidirectional lstm networks," in *2013 12th International Conference on Document Analysis and Recognition*, pp. 1061–1065, IEEE, 2013.

[15] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, ACM, 2006.

[16] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pp. 279–284, IEEE, 2014.

[17] E. Grosicki and H. El Abed, "Icdar 2009 handwriting recognition competition," in *2009 10th International Conference on Document Analysis and Recognition*, pp. 1398–1402, IEEE, 2009.

[18] F. Zamora-Martínez, V. Frinken, S. España-Boquera, M. J. Castro-Bleda, A. Fischer, and H. Bunke, "Neural network language models for off-line handwriting recognition," *Pattern Recognition*, vol. 47, no. 4, pp. 1642–1652, 2014.

[19] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant, "The a2ia arabic handwritten text recognition system at the open hart2013 evaluation," in *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pp. 161–165, IEEE, 2014.

[20] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pp. 285–290, IEEE, 2014.

[21] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[22] "Nist 2013 open handwriting recognition and translation evaluation plan." http://www.nist.gov/itl/iad/mig/upload/OpenHaRT2013EvalPlanv1-7.pdf. Accessed: 2016-11-5.

[23] P. Voigtlaender, P. Doetsch, and H. Ney, "Handwriting recognition with large multidimensional long short-term memory recurrent neural networks," in *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pp. 228–233, IEEE, 2016.

[24] A. Ray, S. Rajeswar, and S. Chaudhury, "Text recognition using deep blstm networks," in *Advances in Pattern Recognition (ICAPR), 2015 Eighth International Conference on*, pp. 1–6, IEEE, 2015.

[25] O. Christopher, "Understanding lstm networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[26] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks.," *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[27] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[28] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] Google, "Android os." https://www.android.com/. Accessed: 2016-11-6.

[31] J. Gosling, *The Java language specification.* Addison-Wesley Professional, 2000.

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

X. Appendix

A. Source Code

A..1 OCR Trainer

File: Tensorflow-OCR-Trainer/requirements.txt

```
flask
werkzeug
six
numpy
scipy
opencv-python
scikit-learn
tensorflow
requests
```

File: Tensorflow-OCR-Trainer/config.py

```
class BaseConfig(object):
    DEBUG = False
    TESTING = False
    ARCHITECTURES_DIRECTORY = "architectures"
    DATASET_DIRECTORY = "dataset"
    CHARSET_DIRECTORY = "charset"
    ALLOWED_IMAGE_EXTENSIONS = {'png', 'jpg', 'jpeg'}
    ALLOWED_LABELS_FILE_EXTENSIONS = {'txt', 'csv'}
    ALLOWED_ZIP_EXTENSIONS = {'gz', 'rar', 'zip'}
    MODELS_DIRECTORY = "checkpoint"
    VISUALIZATION_HOST = "localhost"
    OUTPUT_GRAPH_FILENAME = "graph.pb"
    MODEL_ZIP_FILENAME = "model_files.gz"
    SERVING_MODEL_CONFIG_FILENAME = "serving_model_config.json"
    IMAGE_CONFIG_FILENAME = "image_config.json"


class DevelopmentConfig(BaseConfig):
    DEBUG = True
    TESTING = True


class TestingConfig(BaseConfig):
    DEBUG = False
    TESTING = True
```

File: Tensorflow-OCR-Trainer/run.py

```
from trainer import app

if __name__ == '__main__':
    app.run()
```

File: Tensorflow-OCR-Trainer/setup.py

```
from setuptools import setup

setup(
    name='Optimized_OCR',
    packages=['trainer'],
    include_package_data=True,
    install_requires=[
        'flask',
        'tensorflow',
        'cv2',
        'numpy',
        'werkzeug',
        'six',
        'scikit-learn'
    ],
)
```

File: Tensorflow-OCR-Trainer/trainer/controllers.py

```
import os
import json
import csv
import shutil
import multiprocessing
import time
import zipfile
import requests

from werkzeug.utils import secure_filename
from flask import request
from collections import OrderedDict
from sklearn.model_selection import train_test_split

from trainer import app
from trainer.backend import GraphKeys, dataset_utils
from trainer.backend.dataset_utils import read_dataset_list
from trainer.backend.train_ocr import train_model
from trainer.backend.train_ocr import evaluate_model
from trainer.backend.train_ocr import continue_training_model
from trainer.backend import create_serving_model
from trainer.backend import visualize
from trainer.backend import create_optimized_graph


def _allowed_labels_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() \
           in app.config['ALLOWED_LABELS_FILE_EXTENSIONS']


def _allowed_image_files(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() \
           in app.config['ALLOWED_IMAGE_EXTENSIONS']


def _allowed_zip_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() \
           in app.config['ALLOWED_ZIP_EXTENSIONS']


def upload_dataset(dataset_zip):
    dataset_name = get('dataset_name')
    if _allowed_zip_file(dataset_zip.filename):
        dataset_zip_path = _create_path(app.config['DATASET_DIRECTORY'], secure_filename(dataset_zip.
             filename))
        dataset_zip.save(dataset_zip_path)
        dataset_path = _create_path(app.config['DATASET_DIRECTORY'], dataset_name)
        os.makedirs(dataset_path)
        _extract_zip_files(dataset_zip_path, dataset_path)
        delete_file(dataset_zip_path)
        split_dataset("labels.txt")
        return dataset_name + " has been uploaded."
    return "An error occurred in uploading the dataset."


def _extract_zip_files(src, dest_dir):
    zip_ref = zipfile.ZipFile(src, 'r')
    zip_ref.extractall(dest_dir)
    zip_ref.close()


def get_dataset(dataset_name):
    return _create_path(app.config['DATASET_DIRECTORY'], dataset_name)


def _get_number_of_lines(filename):
    counter = 0
    with open(filename) as f:
        for i, l in enumerate(f):
            counter = i
    return counter + 1


def get_dataset_list_with_amount_of_training_and_testing_data():
    dataset_list = []
    dataset_names = get_directory_list_from_config('DATASET_DIRECTORY')
    for dataset_name in dataset_names:
        dataset_dict = OrderedDict()
        dataset_path = get_dataset(dataset_name)
        number_training_samples = _get_number_of_lines(_create_path(dataset_path, 'train.csv'))
        number_testing_samples = _get_number_of_lines(_create_path(dataset_path, 'test.csv'))
        dataset_dict['name'] = dataset_name
        dataset_dict['num_training_examples'] = number_training_samples
        dataset_dict['num_testing_examples'] = number_testing_samples
        dataset_list.append(dataset_dict)
    return dataset_list


def get_directory_list_from_config(directory_key):
    directory_names = os.listdir(app.config[directory_key])
    return directory_names


def _create_labels_file(filename, features, labels):
    with open(filename, 'w') as f:
        writer = csv.writer(f, lineterminator='\n', delimiter=' ')
        writer.writerows(zip(features, labels))


def split_dataset(labels_file):
    dataset_path = _create_path(app.config['DATASET_DIRECTORY'], get('dataset_name'))
    features, labels = read_dataset_list(_create_path(dataset_path,
                                                     secure_filename(labels_file)))
    x_train, x_test, y_train, y_test = train_test_split(features,
                                                        labels,
                                                        test_size=float(get('test_size')))
    _create_labels_file(_create_path(dataset_path, 'train.csv'), x_train, y_train)
    _create_labels_file(_create_path(dataset_path, 'test.csv'), x_test, y_test)


def _create_path(*args):
    return os.path.join(*args)
```

```
def get_enum_values(enum_type):
    return [key.value.replace('"', '') for key in enum_type]


def _get_layer(layer_index):
    layer = OrderedDict()
    layer["layer_type"] = get(_create_network_key(layer_index, "layer_type"))
    layer_type = layer["layer_type"]
    if layer_type == GraphKeys.LayerTypes.CONV2D.value:
        layer["num_filters"] = int(get(_create_network_key(layer_index, "num_filters")))
        layer["kernel_size"] = int(get(_create_network_key(layer_index, "kernel_size1")))
        if get(_create_network_key(layer_index, "kernel_size2")):
            layer["kernel_size"] = [layer["kernel_size"], int(get(_create_network_key(layer_index, "
                kernel_size2")))]
        layer["stride"] = int(get(_create_network_key(layer_index, "stride")))
        layer["padding"] = get(_create_network_key(layer_index, "padding"))
        layer["activation"] = get(_create_network_key(layer_index, "activation"))
    elif layer_type == GraphKeys.LayerTypes.MAX_POOL2D.value:
        layer["pool_size"] = int(get(_create_network_key(layer_index, "pool_size")))
        layer["stride"] = int(get(_create_network_key(layer_index, "stride")))
        layer["padding"] = get(_create_network_key(layer_index, "padding"))
    elif layer_type == GraphKeys.LayerTypes.BIRNN.value:
        layer["num_hidden"] = int(get(_create_network_key(layer_index, "num_hidden")))
        layer["cell_type"] = get(_create_network_key(layer_index, "cell_type"))
        layer["activation"] = get(_create_network_key(layer_index, "activation"))
    elif layer_type == GraphKeys.LayerTypes.MDRNN.value:
        layer["num_hidden"] = int(get(_create_network_key(layer_index, "num_hidden")))
        layer["cell_type"] = get(_create_network_key(layer_index, "cell_type"))
        layer["activation"] = get(_create_network_key(layer_index, "activation"))
    elif layer_type == GraphKeys.LayerTypes.DROPOUT.value:
        layer["keep_prob"] = float(get(_create_network_key(layer_index, "keep_prob")))
    return layer


def get_architecture_file_contents(architecture_name):
    architecture_path = get_architecture_path(architecture_name)
    return json.load(open(architecture_path), object_pairs_hook=OrderedDict)


def delete_file(file_path):
    os.remove(file_path)


def delete_folder(folder_name):
    shutil.rmtree(folder_name)


def _create_network_key(layer_index, param):
    network_key = "network[" + str(layer_index) + "][" + param + "]"
    return network_key


def _generate_architecture_dict():
    resulting_dict = OrderedDict()
    network = []
    number_of_layers = len([value for key, value in request.form.items() if 'layer_type' in key.lower()])
    for layer_index in range(number_of_layers):
        layer = _get_layer(layer_index)
        network.append(layer)
    resulting_dict['network'] = network
    resulting_dict['output_layer'] = get("output_layer")
    return resulting_dict


def get_model_path(model):
    return _create_path(app.config['MODELS_DIRECTORY'], model)


def visualize_model(model_name, host):
    model_path = get_model_path(model_name)
    visualization_task = multiprocessing.Process(target=visualize, args=(model_path, host))
    visualization_task.name = "visualize-{}".format(model_name)
    visualization_task.start()


def get_log(model_name, log_name):
    model_path = get_model_path(model_name)
    log_path = _create_path(model_path, log_name+".log")
    content = []
    if os.path.exists(log_path):
        with open(log_path) as f:
            content = f.readlines()
        content = [x.strip() for x in content]
    return content


def request_connection(url):
    status_code = 404
    while status_code == 404:
        try:
            status_code = requests.get(url).status_code
        except requests.exceptions.ConnectionError:
            time.sleep(5)
            continue

def save_model_as_json():
    architecture_dict = _generate_architecture_dict()
    architecture_name = get('architecture_name')
    architecture_path = get_architecture_path(architecture_name)
    _write_json(architecture_path, architecture_dict)
    return architecture_name + " has been created."


def _write_json(path, content):
    with open(path, 'w') as fp:
        json.dump(content, fp, indent=0)


def get_architecture_path(architecture_name):
    return _create_path(app.config['ARCHITECTURES_DIRECTORY'], architecture_name) + ".json"


def _get_abs_path(filename):
    return _create_path(os.getcwd(), filename, '')


def get(param):
    return request.form[param]


def getlist(param):
    return request.form.getlist(param)


def stop_running(task):
    for running_task in multiprocessing.active_children():
        if running_task.name == task:
            running_task.terminate()
            running_task.join()
            return task.capitalize() + " is terminated."
    return task.capitalize() + " was already terminated."


def get_running_tasks():
    return [running_task.name for running_task in multiprocessing.active_children()]


def _export_serving_model(model_name, input_name="features", output_names="output"):
    model_path = get_model_path(model_name)
    run_params = json.load(open(_create_path(model_path, "run_config.json")), object_pairs_hook=OrderedDict)
    serving_model_path, input_shape = create_serving_model(model_path,
                                                           run_params,
                                                           input_name)
    optimized_model_path = _create_path(model_path, app.config["OUTPUT_GRAPH_FILENAME"])
    create_optimized_graph(serving_model_path, output_names, output_graph_filename=optimized_model_path)
    delete_folder(serving_model_path)
    serving_model_config = OrderedDict()
    input_node = OrderedDict()
    image_config = OrderedDict()
    image_config['image_width'] = run_params['desired_image_width']
    image_config['image_height'] = run_params['desired_image_height']
    input_node['input_name'] = input_name
    input_node['input_shape'] = input_shape
    serving_model_config['input_nodes'] = [input_node]
    serving_model_config['output_names'] = output_names.split(',')
    serving_model_config_path = _create_path(model_path, app.config['SERVING_MODEL_CONFIG_FILENAME'])
    image_config_path = _create_path(model_path, app.config['IMAGE_CONFIG_FILENAME'])
    _write_json(serving_model_config_path, serving_model_config)
    _write_json(image_config_path, image_config)


def package_model_files(model_name):
    _export_serving_model(model_name)
    model_path = get_model_path(model_name)
    with zipfile.ZipFile(_create_path(model_path, app.config['MODEL_ZIP_FILENAME']), mode='w') as f_out:
        for model_file in [app.config['SERVING_MODEL_CONFIG_FILENAME'],
                           app.config["OUTPUT_GRAPH_FILENAME"],
                           app.config['IMAGE_CONFIG_FILENAME']]:
            model_file_path = _create_path(model_path, model_file)
            f_out.write(model_file_path, arcname=model_file)
            delete_file(model_file_path)
    return _get_abs_path(model_path)


def _test_task(model_name):
    checkpoint_dir = get_model_path(model_name)
    run_params = json.load(open(_create_path(checkpoint_dir, "run_config.json")), object_pairs_hook=
        OrderedDict)
    dataset_dir = get_dataset(run_params['dataset_name'])
    charset_file = run_params['charset_file']
    testing_task = multiprocessing.Process(target=evaluate_model,
                                           args=(run_params,
                                                 dataset_dir,
                                                 charset_file,
                                                 checkpoint_dir,
                                                 ' '))
    testing_task.start()
    return testing_task


def _retrain_task(model_name):
    checkpoint_dir = get_model_path(model_name)
    run_params = json.load(open(_create_path(checkpoint_dir, "run_config.json")), object_pairs_hook=
        OrderedDict)
    run_params['learning_rate'] = float(get('learning_rate'))
    run_params['checkpoint_epochs'] = int(get('checkpoint_epochs'))
    run_params['num_epochs'] = int(get('num_epochs'))
    dataset_dir = get_dataset(run_params['dataset_name'])
    run_config_path = _create_path(checkpoint_dir, 'run_config.json')
    _write_json(run_config_path, run_params)
    continue_training_task = multiprocessing.Process(target=continue_training_model,
                                                     args=(run_params,
                                                           checkpoint_dir,
                                                           dataset_dir))
    continue_training_task.start()
    return continue_training_task


def run_learning_task(task):
    if task == 'training':
        dataset_name = get('dataset_name')
        model_name = "model-" + time.strftime("%Y%m%d-%H%M%S")
        checkpoint_dir = get_model_path(model_name)
        running_task = _train_task(get('architecture_name'),
                                   dataset_name,
                                   checkpoint_dir,
                                   int(get('desired_image_width')),
                                   int(get('desired_image_height')),
                                   int(get('num_epochs')),
                                   int(get('checkpoint_epochs')),
                                   int(get('batch_size')),
                                   'charsets/chars.txt',
                                   float(get('learning_rate')),
                                   get('optimizer'),
                                   getlist('metrics'),
                                   get('loss'),
                                   get('validation_size'))
        _set_running_task_name(running_task, task, model_name)
    elif task == 'testing':
        running_task = _test_task(get('model_name'))
        _set_running_task_name(running_task, task, get('model_name'))
    elif task == 'retrain':
        running_task = _retrain_task(get('model_name'))
        _set_running_task_name(running_task, task, get('model_name'))


def _set_running_task_name(running_task, task, checkpoint_dir):
    running_task.name = "{}-{}".format(task, checkpoint_dir)


def _train_task(architecture_name,
                dataset_name,
                checkpoint_dir,
                desired_image_width,
                desired_image_height,
                num_epochs,
                checkpoint_epochs,
                batch_size,
                charset_file,
                learning_rate,
                optimizer,
                metrics,
                loss,
                validation_size):
    if validation_size:
        validation_size = float(validation_size)
    dataset_dir = get_dataset(dataset_name)
    os.mkdir(checkpoint_dir)
    run_params = get_architecture_file_contents(architecture_name)
    classes = dataset_utils.get_characters_from(charset_file)
    run_params['loss'] = loss
    run_params['metrics'] = metrics
    run_params['desired_image_width'] = desired_image_width
    run_params['desired_image_height'] = desired_image_height
    run_params['batch_size'] = batch_size
    run_params['dataset_name'] = dataset_name
    run_params['charset_file'] = charset_file
    run_params['num_classes'] = len(classes) + 1
    run_params['checkpoint_epochs'] = checkpoint_epochs
    run_params['validation_size'] = validation_size
    run_params['num_epochs'] = num_epochs
    run_params['learning_rate'] = learning_rate
    run_params['optimizer'] = optimizer
    run_config_path = _create_path(checkpoint_dir, 'run_config.json')
    _write_json(run_config_path, run_params)
    task = multiprocessing.Process(target=train_model,
                                   args=(
                                       run_params,
                                       dataset_dir,
                                       checkpoint_dir,
                                       learning_rate,
                                       metrics,
                                       loss,
                                       optimizer,
                                       charset_file,
                                       validation_size,
                                       ' ',
                                       num_epochs,
                                       batch_size,
                                       checkpoint_epochs
                                   ))
    task.start()
    return task
```

File: Tensorflow-OCR-Trainer/trainer/views.py

```
from flask import request, render_template, flash, redirect, url_for, send_from_directory
from trainer import app
from trainer.backend import GraphKeys
from trainer.controllers import package_model_files
from trainer.controllers import delete_file
from trainer.controllers import delete_folder
from trainer.controllers import get_architecture_path
from trainer.controllers import get_architecture_file_contents
from trainer.controllers import get_dataset
from trainer.controllers import get_dataset_list_with_amount_of_training_and_testing_data
from trainer.controllers import get_directory_list_from_config
from trainer.controllers import get_enum_values
from trainer.controllers import get_model_path
from trainer.controllers import get_running_tasks
from trainer.controllers import get_log
from trainer.controllers import request_connection
from trainer.controllers import run_learning_task
from trainer.controllers import save_model_as_json
from trainer.controllers import stop_running
from trainer.controllers import upload_dataset
from trainer.controllers import visualize_model


@app.route('/')
def index():
    return render_template("index.html")


@app.route('/architectures', methods=['GET', 'POST'])
def architectures():
    if request.method == 'POST':
        saving_message = save_model_as_json()
        flash(saving_message)
    network_architectures = _get_network_architectures()
    return render_template("architectures.html",
                           network_architectures=network_architectures)


@app.route('/view_architecture/<architecture_name>')
def view_architecture(architecture_name):
    architecture = get_architecture_file_contents(architecture_name)
    return render_template("view_architecture.html",
                           architecture=architecture,
                           architecture_name=architecture_name)


def _get_network_architectures():
    network_architectures = get_directory_list_from_config('ARCHITECTURES_DIRECTORY')
    network_architectures = _get_names(network_architectures)
    return network_architectures


def _get_names(network_architectures):
    return [network_architecture.split('.')[0]
            for network_architecture in network_architectures]


@app.route('/create_network_architecture')
def create_network_architecture():
    return render_template("create_network_architecture.html",
                           layer_types=get_enum_values(GraphKeys.LayerTypes),
                           padding_types=get_enum_values(GraphKeys.PaddingTypes),
                           cell_types=get_enum_values(GraphKeys.CellTypes),
                           activation_functions=get_enum_values(GraphKeys.ActivationFunctions),
                           output_layers=get_enum_values(GraphKeys.OutputLayers))


@app.route('/delete/<architecture_name>', methods=['POST'])
def delete_architecture(architecture_name):
    delete_file(get_architecture_path(architecture_name))
    flash(architecture_name + " has been deleted.")
    return redirect(url_for('architectures'))


@app.route('/dataset_form')
def dataset_form():
    return render_template("dataset_form.html")


@app.route('/dataset', methods=['GET', 'POST'])
def dataset():
    if request.method == 'POST':
        if 'dataset_zip' not in request.files:
            flash('No labels file part.')
            return redirect(request.url)
        dataset_zip = request.files['dataset_zip']
        if not dataset_zip:
            flash('No dataset zip file selected')
            return redirect(request.url)
        upload_message = upload_dataset(dataset_zip)
        flash(upload_message)
    return render_template("dataset.html", dataset_list=_get_dataset_list_and_details())


def _get_dataset_list_and_details():
    return get_dataset_list_with_amount_of_training_and_testing_data()

@app.route('/delete_dataset/<dataset_name>', methods=['POST'])
def delete_dataset(dataset_name):
    delete_folder(get_dataset(dataset_name))
    flash(dataset_name + " has been deleted.")
    return redirect(url_for('dataset'))


def _get_dataset_list():
    dataset_list = get_directory_list_from_config('DATASET_DIRECTORY')
    return dataset_list


@app.route('/train')
def train():
    return render_template("train.html",
                           dataset_list=_get_dataset_list(),
                           network_architectures=_get_network_architectures(),
                           losses=get_enum_values(GraphKeys.Losses),
                           optimizers=get_enum_values(GraphKeys.Optimizers),
                           metrics=get_enum_values(GraphKeys.Metrics))


@app.route('/retrain/<model_name>')
def retrain(model_name):
    return render_template("retrain.html", model_name=model_name,
                           optimizers=get_enum_values(GraphKeys.Optimizers))


@app.route('/tasks/<task>', methods=['GET', 'POST'])
def tasks(task):
    if request.method == 'POST':
        run_learning_task(task)
        flash(task + " has started.")
    running_tasks = get_running_tasks()
    return render_template('tasks.html', running_tasks=running_tasks)


@app.route('/terminate/<task>', methods=['POST'])
def terminate(task):
    message = stop_running(task)
    flash(message)
    return redirect(url_for('tasks', task='view'))


@app.route('/models')
def models():
    return render_template("models.html",
                           models=get_directory_list_from_config('MODELS_DIRECTORY'),
                           dataset_list=_get_dataset_list())


@app.route('/visualize/<model_name>')
def visualize(model_name):
    visualize_model(model_name, app.config['VISUALIZATION_HOST'])
    request_connection("http://localhost:6006")
    return redirect("http://localhost:6006")

@app.route('/view_logs/<model_name>')
def view_logs(model_name):
    train_log = get_log(model_name, "train")
    test_log = get_log(model_name, "test")
    return render_template("view_logs.html",
                           train_log=train_log,
                           test_log=test_log)


@app.route('/delete_model/<model_name>', methods=['POST'])
def delete_model(model_name):
    delete_folder(get_model_path(model_name))
    flash(model_name + " has been deleted.")
    return redirect(url_for('models'))


@app.route('/{}/<model_name>/{}'.format(app.config['MODELS_DIRECTORY'], app.config['MODEL_ZIP_FILENAME']))
def export_model(model_name):
    model_abs_path = package_model_files(model_name)
    return send_from_directory(model_abs_path, app.config['MODEL_ZIP_FILENAME'])


@app.errorhandler(404)
def url_error(e):
    print(e)
    return render_template("404.html"), 404


@app.errorhandler(500)
def server_error(e):
    print(e)
    return render_template("500.html"), 500
```

File: Tensorflow-OCR-Trainer/trainer/¨init¨.py

```
from flask import Flask
app = Flask(__name__)
app.secret_key = 'pancake'
app.config.from_object('config.DevelopmentConfig')

import trainer.views
```

File: Tensorflow-OCR-Trainer/trainer/backend/dataset˙utils.py

```
import cv2
import numpy as np
```

```python
import os

from trainer.backend.EncoderDecoder import EncoderDecoder


def read_dataset_list(dataset_list_file, delimiter=' '):
    features = []
    labels = []
    with open(dataset_list_file) as f:
        data = f.readlines()
    data = [x.strip() for x in data]
    for example in data:
        example = example.split(delimiter)
        features.append(example[0])
        labels.append(example[-1])
    return features, labels


def read_images(data_dir, image_paths, image_extension='png'):
    print('Reading images...')
    images = []
    for image_name in image_paths:
        images.append(cv2.imread(os.path.join(data_dir, image_name) + '.' + image_extension))
    print('Done reading images. Number of images read:', len(image_paths))
    return images


def binarize(images):
    print('Binarizing images...')
    binarized_images = []
    for image in images:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        _, binarized_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binarized_image = binarized_image[:, :, np.newaxis]
        binarized_images.append(binarized_image)
    print('Done binarizing images.')
    return binarized_images


def invert(images):
    print('Inverting color of images...')
    inverted_images = []
    for image in images:
        inverted_image = cv2.bitwise_not(image)
        inverted_image = inverted_image[:, :, np.newaxis]
        inverted_images.append(inverted_image)
    print('Done inverting color of images.')
    return inverted_images


def images_as_float32(images):
    float32_images = []
    for image in images:
        float32_images.append(image.astype(np.float32))
    return float32_images


def resize(images, desired_height=None, desired_width=None):
    print("Resizing images...")
    resized_images = []
    for image in images:
        resized_image = _resize(image, desired_height, desired_width)
        resized_images.append(resized_image)
    print("Done resizing images.")
    return resized_images


def _resize(image, desired_height=None, desired_width=None):
    dim = (desired_width, desired_height)
    if dim is (None, None):
        return image
    raw_height, raw_width, num_channels = image.shape
    scaled_width = int(raw_width * (desired_height / raw_height))
    scaled_width_image = cv2.resize(image, (scaled_width, desired_height))
    scaled_width_image_array = np.array(scaled_width_image).astype(np.uint8)
    scaled_image = scaled_width_image_array.reshape(desired_height, scaled_width)
    padding = np.full((desired_height, desired_width - scaled_width + 1), 255)
    padded_image = np.concatenate((scaled_image, padding), axis=1)
    padded_image = padded_image[:, 0:desired_width]
    return np.array(padded_image).astype(np.uint8)


def _resize_to_square(desired_height, desired_width, image, raw_height, raw_width):
    ratio = float(desired_width) / max(raw_height, raw_width)
    scaled_width = int(raw_width * ratio)
    scaled_height = int(raw_height * ratio)
    resized_image = cv2.resize(image, (scaled_width, scaled_height))
    square_image = _add_padding(desired_height, desired_width, resized_image, scaled_height, scaled_width)
    return square_image


def _add_padding(desired_height, desired_width, image, scaled_height, scaled_width):
    delta_width = desired_width - scaled_width
    delta_height = desired_height - scaled_height
    top, bottom = delta_height // 2, delta_height - (delta_height // 2)
    left, right = delta_width // 2, delta_width - (delta_width // 2)
    color = [255, 255, 255]
    square_image = cv2.copyMakeBorder(image,
                                      top,
                                        bottom,
                                        left,
                                        right,
                                        cv2.BORDER_CONSTANT,
                                        value=color)
    return square_image


def get_characters_from(charset_file):
    return ''.join([line.rstrip('\n') for line in open(charset_file)])


def encode(labels, classes):
    encoder_decoder = EncoderDecoder()
    encoder_decoder.initialize_encode_and_decode_maps_from(
        classes
    )
    encoded_labels = []
    for label in labels:
        encoded_labels.append(encoder_decoder.encode(label))
    return encoded_labels


def pad(labels, max_label_length=120):
    padded_labels = []
    for label in labels:
        while len(label) < max_label_length:
            label = np.append(label, [-1])
        padded_labels.append(label)
    return padded_labels
```

File: Tensorflow-OCR-Trainer/trainer/backend/EncoderDecoder.py

```
class EncoderDecoder:
    def __init__(self):
        self.charset = ""
        self.eos_token = '`'
        self.encode_map = {}
        self.decode_map = {}

    def initialize_encode_and_decode_maps_from(self, charset_string):
        for index, char in enumerate(list(charset_string)):
            self._add_to_encode_decode_maps(char, index)

    def _add_to_encode_decode_maps(self, char, index):
        self.encode_map[char] = index
        self.decode_map[index] = char

    def encode(self, string_to_encode):
        return [self.encode_map[c] for c in list(string_to_encode)]

    def decode(self, encoded_string_to_decode):
        return ''.join([self.decode_map[i] for i in encoded_string_to_decode])
```

File: Tensorflow-OCR-Trainer/trainer/backend/GraphKeys.py

```
from enum import Enum

class LayerTypes(Enum):
    CONV2D = "conv2d"
    MAX_POOL2D = "max_pool2d"
    COLLAPSE_TO_RNN_DIMS = "collapse_to_rnn_dims"
    MDRNN = "mdrnn"
    BIRNN = "birnn"
    L2_NORMALIZE = "l2_normalize"
    BATCH_NORM = "batch_norm"
    DROPOUT = "dropout"

class PaddingTypes(Enum):
    SAME = "same"
    VALID = "valid"

class CellTypes(Enum):
    LSTM = "LSTM"
    GRU = "GRU"
    GLSTM = "GLSTM"

class ActivationFunctions(Enum):
    TANH = "tanh"
    RELU = "relu"
    RELU6 = "relu6"

class Optimizers(Enum):
    ADAM = "adam"
    MOMENTUM = "momentum"
    ADADELTA = "adadelta"
    RMSPROP = "rmsprop"
    NADAM = "nadam"

class Metrics(Enum):
    LABEL_ERROR_RATE = "label_error_rate"

class Losses(Enum):
    CTC = "ctc"

class OutputLayers(Enum):
    CTC_DECODER = "ctc_decoder"
```

File: Tensorflow-OCR-Trainer/trainer/backend/train˙ocr.py

```python
import os

from sklearn.model_selection import train_test_split

from trainer.backend import dataset_utils
from trainer.backend.tf import train
from trainer.backend.tf import test


def train_model(run_params, dataset_dir, checkpoint_dir,
                learning_rate, metrics, loss, optimizer,
                charset_file, validation_size=None,
                labels_delimiter=' ', num_epochs=1, batch_size=1,
                checkpoint_epochs=1):
    labels_file = os.path.join(dataset_dir, "train.csv")
    images, labels, num_classes = _prepare_dataset(charset_file,
                                                   dataset_dir,
                                                   run_params['desired_image_width'],
                                                   run_params['desired_image_height'],
                                                   labels_delimiter,
                                                   labels_file)
    features = {'train': images}
    labels_dict = {'train': labels}
    if validation_size:
        features, labels_dict = _train_validation_split(images, labels, validation_size)

    run_params["learning_rate"] = learning_rate
    run_params["optimizer"] = optimizer
    run_params["metrics"] = metrics
    run_params["loss"] = loss

    train(params=run_params,
          features=features,
          labels=labels_dict,
          num_classes=num_classes,
          checkpoint_dir=checkpoint_dir,
          batch_size=batch_size,
          num_epochs=num_epochs,
          save_checkpoint_every_n_epochs=checkpoint_epochs)


def continue_training_model(run_params, checkpoint_dir, dataset_dir):
    labels_file = os.path.join(dataset_dir, "train.csv")
    images, labels, num_classes = _prepare_dataset(run_params['charset_file'],
                                                   dataset_dir,
                                                   run_params['desired_image_width'],
                                                   run_params['desired_image_height'],
                                                   ' ',
                                                   labels_file)
    features = {'train': images}
    labels_dict = {'train': labels}
    validation_size = run_params['validation_size']
    if validation_size:
        features, labels_dict = _train_validation_split(images, labels, validation_size)
    train(params=run_params,
          features=features,
          labels=labels_dict,
          num_classes=num_classes,
          checkpoint_dir=checkpoint_dir,
          batch_size=run_params['batch_size'],
          num_epochs=run_params['num_epochs'],
          save_checkpoint_every_n_epochs=run_params['checkpoint_epochs'])


def _train_validation_split(images, labels, validation_size):
    features = {}
    labels_dict = {}
    x_train, x_validation, y_train, y_validation = train_test_split(
        images,
        labels,
        test_size=validation_size
    )
    features['train'] = x_train
    features['validation'] = x_validation
    labels_dict['train'] = y_train
    labels_dict['validation'] = y_validation
    return features, labels_dict


def evaluate_model(architecture_params, dataset_dir, charset_file,
                   checkpoint_dir, labels_delimiter=' '):
    labels_file = os.path.join(dataset_dir, "test.csv")
    images, labels, num_classes = _prepare_dataset(charset_file,
                                                   dataset_dir,
                                                   architecture_params['desired_image_width'],
                                                   architecture_params['desired_image_height'],
                                                   labels_delimiter,
                                                   labels_file)
    test(architecture_params, images, labels, checkpoint_dir)


def _prepare_dataset(charset_file, dataset_dir, desired_image_width, desired_image_height, labels_delimiter,
      labels_file):
    image_paths, labels = dataset_utils.read_dataset_list(
        labels_file, delimiter=labels_delimiter)
    max_label_length = len(max(labels, key=len))
    images = dataset_utils.read_images(data_dir=dataset_dir,
                                       image_paths=image_paths,
                                       image_extension='png')
    images = dataset_utils.binarize(images)
    images = dataset_utils.resize(images,
                                  desired_height=desired_image_height,
                                  desired_width=desired_image_width)
    images = dataset_utils.invert(images)
    classes = dataset_utils.get_characters_from(charset_file)
    images = dataset_utils.images_as_float32(images)
    labels = dataset_utils.encode(labels, classes)
    num_classes = len(classes) + 1
    labels = dataset_utils.pad(labels, max_label_length)
    return images, labels, num_classes
```

File: Tensorflow-OCR-Trainer/trainer/backend/¨init¨.py

```
from trainer.backend.train_ocr import train_model
from trainer.backend.GraphKeys import LayerTypes
from trainer.backend.GraphKeys import PaddingTypes
from trainer.backend.tf.util_ops import visualize
from trainer.backend.tf.experiment_ops import create_serving_model
from trainer.backend.tf.experiment_ops import create_optimized_graph
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/ctc˙ops.py

```
import tensorflow as tf
from tensorflow.contrib import slim


def ctc_beam_search_decoder(inputs, sequence_length):
    decoded, log_probabilities = tf.nn.ctc_beam_search_decoder(inputs, sequence_length)
    return decoded[0], log_probabilities


def convert_to_ctc_dims(inputs, num_classes, num_steps, num_outputs):
    outputs = tf.reshape(inputs, [-1, num_outputs])
    logits = slim.fully_connected(outputs, num_classes)
    logits = tf.reshape(logits, [-1, num_steps, num_classes])
    logits = tf.transpose(logits, (1, 0, 2))
    return logits
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/DenseToSparseTest.py

```
import numpy as np
import tensorflow as tf

from trainer.backend.tf.util_ops import dense_to_sparse


class DenseToSparseTest(tf.test.TestCase):
    def test_convert_dense_tensor_to_sparse_tensor(self):
        dense_array = np.array([1, 2, 3]).astype(np.int32)
        dense = tf.placeholder(tf.int32, shape=[None])
        sparse = dense_to_sparse(dense)
        sparse_converted_to_dense = tf.sparse_tensor_to_dense(sparse)
        with self.test_session() as sess:
            dense_from_sparse = sess.run(sparse_converted_to_dense, feed_dict={dense: dense_array})
        self.assertAllEqual(dense_from_sparse, dense_array)


if __name__ == '__main__':
    tf.test.main()
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/experiment˙ops.py

```
import numpy as np
import tensorflow as tf
import logging

from tensorflow.python.estimator.export.export import ServingInputReceiver
from tensorflow.python.tools import freeze_graph

from trainer.backend.GraphKeys import Optimizers
from trainer.backend.GraphKeys import OutputLayers
from trainer.backend.GraphKeys import Metrics
from trainer.backend.GraphKeys import Losses

from trainer.backend.tf import ctc_ops, losses, metric_functions
from trainer.backend.tf.replicate_model_fn import TowerOptimizer
from trainer.backend.tf.util_ops import feed, dense_to_sparse, get_sequence_lengths
from trainer.backend.tf.ValidationHook import ValidationHook

from tensorflow.contrib import slim
from tensorflow.contrib.opt import NadamOptimizer

tf.logging.set_verbosity(tf.logging.INFO)


def _get_loss(loss, labels, inputs, num_classes):
    if loss == Losses.CTC.value:
        ctc_inputs = ctc_ops.convert_to_ctc_dims(inputs,
                                                 num_classes=num_classes,
                                                 num_steps=inputs.shape[1],
                                                 num_outputs=inputs.shape[-1])
        labels = dense_to_sparse(labels, token_to_ignore=-1)
        return losses.ctc_loss(labels=labels,
                               inputs=ctc_inputs,
                               sequence_length=get_sequence_lengths(inputs))
    raise NotImplementedError(loss + " loss not implemented")


def _sparse_to_dense(sparse_tensor, name="sparse_to_dense"):
    return tf.sparse_to_dense(tf.to_int32(sparse_tensor.indices),
                              tf.to_int32(sparse_tensor.dense_shape),
                              tf.to_int32(sparse_tensor.values),
                              name=name)


def _get_optimizer(learning_rate, optimizer_name):
    if optimizer_name == Optimizers.MOMENTUM.value:
        return tf.train.MomentumOptimizer(learning_rate,
                                          momentum=0.9,
                                          use_nesterov=True)
    if optimizer_name == Optimizers.ADAM.value:
        return tf.train.AdamOptimizer(learning_rate)
    if optimizer_name == Optimizers.ADADELTA.value:
        return tf.train.AdadeltaOptimizer(learning_rate)
    if optimizer_name == Optimizers.RMSPROP.value:
        return tf.train.RMSPropOptimizer(learning_rate)
    if optimizer_name == Optimizers.NADAM.value:
        return NadamOptimizer(learning_rate)
    raise NotImplementedError(optimizer_name + " optimizer not supported")


def train(params, features, labels, num_classes, checkpoint_dir,
          batch_size=1, num_epochs=1,
          save_checkpoint_every_n_epochs=1):
    _set_logger_to_file(checkpoint_dir, 'train')
    num_steps_per_epoch = len(features['train']) // batch_size
    save_checkpoint_steps = save_checkpoint_every_n_epochs * num_steps_per_epoch
    params['num_classes'] = num_classes
    params['log_step_count_steps'] = num_steps_per_epoch
    training_hooks = []
    estimator = tf.estimator.Estimator(model_fn=_train_model_fn,
                                       params=params,
                                       model_dir=checkpoint_dir,
                                       config=tf.estimator.RunConfig(
                                           save_checkpoints_steps=save_checkpoint_steps,
                                           log_step_count_steps=num_steps_per_epoch,
                                           save_summary_steps=num_steps_per_epoch
                                       ))
    if features.get('validation'):
        training_hooks.append(ValidationHook(
            model_fn=_eval_model_fn,
            params=params,
            input_fn=_input_fn(features['validation'],
                               labels['validation'],
                               batch_size,
                               num_epochs=1,
                               shuffle=False),
            checkpoint_dir=checkpoint_dir,
            every_n_steps=save_checkpoint_steps
        ))
    estimator.train(input_fn=_input_fn(features['train'], labels['train'], batch_size),
                    steps=num_epochs * num_steps_per_epoch,
                    hooks=training_hooks)


def test(params, features, labels, checkpoint_dir):
    _set_logger_to_file(checkpoint_dir, 'test')
    params['summary_dir'] = checkpoint_dir + '/test'
    estimator = tf.estimator.Estimator(model_fn=_test_model_fn,
                                       params=params,
                                       model_dir=checkpoint_dir)
    estimator.evaluate(input_fn=_input_fn(features,
                                          labels,
                                          batch_size=params['batch_size'],
                                          num_epochs=1,
                                          shuffle=False))


# see https://stackoverflow.com/a/44296581
def _set_logger_to_file(checkpoint_dir, task):
    log = logging.getLogger('tensorflow')
    log.setLevel(logging.DEBUG)
    # create formatter and add it to the handlers
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    # create file handler which logs even debug messages
    fh = logging.FileHandler(checkpoint_dir + '/' + task + '.log')
    fh.setLevel(logging.DEBUG)
    fh.setFormatter(formatter)
    log.addHandler(fh)


def predict(params, features, checkpoint_dir):
    estimator = tf.estimator.Estimator(model_fn=_predict_model_fn,
                                       params=params,
                                       model_dir=checkpoint_dir)
    predictions = estimator.predict(input_fn=_input_fn(features, num_epochs=1))
    for i, p in enumerate(predictions):
        print(i, p)


def _input_fn(features, labels=None, batch_size=1, num_epochs=None, shuffle=True):
    if labels:
        labels = np.array(labels, dtype=np.int32)
    return tf.estimator.inputs.numpy_input_fn(
        x={'features': np.array(features)},
        y=labels,
        batch_size=batch_size,
        num_epochs=num_epochs,
        shuffle=shuffle
    )


def _add_to_summary(name, value):
    tf.summary.scalar(name, value)


def _create_train_op(loss, learning_rate, optimizer):
    optimizer = _get_optimizer(learning_rate, optimizer)
    optimizer = TowerOptimizer(optimizer)
    return slim.learning.create_train_op(loss, optimizer, global_step=tf.train.get_or_create_global_step())


def _create_model_fn(mode, predictions, loss=None, train_op=None,
                     eval_metric_ops=None, training_hooks=None,
                     evaluation_hooks=None, export_outputs=None):
    return tf.estimator.EstimatorSpec(mode=mode,
                                      predictions=predictions,
                                      loss=loss,
                                      train_op=train_op,
                                      eval_metric_ops=eval_metric_ops,
                                      training_hooks=training_hooks,
                                      evaluation_hooks=evaluation_hooks,
                                      export_outputs=export_outputs)


def _get_output(rnn_outputs, output_layer, num_classes):
    if output_layer == OutputLayers.CTC_DECODER.value:
        ctc_inputs = ctc_ops.convert_to_ctc_dims(rnn_outputs,
                                                 num_classes=num_classes,
                                                 num_steps=rnn_outputs.shape[1],
                                                 num_outputs=rnn_outputs.shape[-1])
        decoded, _ = ctc_ops.ctc_beam_search_decoder(ctc_inputs, get_sequence_lengths(rnn_outputs))
        return _sparse_to_dense(decoded, name="output")
    raise NotImplementedError(output_layer + " not implemented")


def _get_metrics(metrics, y_pred, y_true, num_classes):
    metrics_dict = {}
    for metric in metrics:
        if metric == Metrics.LABEL_ERROR_RATE.value:
            ctc_inputs = ctc_ops.convert_to_ctc_dims(y_pred,
                                                     num_classes=num_classes,
                                                     num_steps=y_pred.shape[1],
                                                     num_outputs=y_pred.shape[-1])
            y_pred, _ = ctc_ops.ctc_beam_search_decoder(ctc_inputs, get_sequence_lengths(y_pred))
            y_true = dense_to_sparse(y_true, token_to_ignore=-1)
            value = metric_functions.label_error_rate(y_pred,
                                                      y_true,
                                                      metric)
        else:
            raise NotImplementedError(metric + " metric not implemented")
        metrics_dict[metric] = value
    return metrics_dict


def create_serving_model(checkpoint_dir, run_params, input_name="features"):
    serving_model_path, input_shape = _export_serving_model(checkpoint_dir, run_params, input_name)
    return serving_model_path, input_shape


def _export_serving_model(checkpoint_dir, model_params, input_name="features"):
    model_params["input_name"] = input_name
    checkpoint = tf.train.get_checkpoint_state(checkpoint_dir)
    input_checkpoint = checkpoint.model_checkpoint_path
    with tf.Session(graph=tf.Graph()) as sess:
        saver = tf.train.import_meta_graph(input_checkpoint + '.meta', clear_devices=True)
        saver.restore(sess, input_checkpoint)
        input_layer = tf.get_default_graph().get_operation_by_name('input_layer').outputs[0]
        input_shape = input_layer.get_shape().as_list()
        input_shape[0] = 1
        estimator = tf.estimator.Estimator(model_fn=_serving_model_fn,
                                           params=model_params,
                                           model_dir=checkpoint_dir)

    def _serving_input_receiver_fn():
        serialized_tf_example = tf.placeholder(dtype=input_layer.dtype,
                                               shape=input_shape,
                                               name=input_name)
        receiver_tensors = {input_name: serialized_tf_example}
        return ServingInputReceiver(receiver_tensors, receiver_tensors)

    serving_model_path = estimator.export_savedmodel(checkpoint_dir, _serving_input_receiver_fn,
                                                     as_text=True)
    return serving_model_path, input_shape


def create_optimized_graph(model_filename,
                           output_nodes="output",
                           output_graph_filename="optimized_graph.pb"):
    freeze_graph.freeze_graph(
        input_graph=None,
        input_saver=None,
        input_binary=False,
        input_checkpoint=None,
        output_node_names=output_nodes,
        restore_op_name=None,
        filename_tensor_name=None,
        output_graph=output_graph_filename,
```

```python
        clear_devices=True,
        initializer_nodes=None,
        input_saved_model_dir=model_filename
    )


def _write_graph(output_graph_def, output_graph_filename):
    with tf.gfile.GFile(output_graph_filename, "wb") as f:
        f.write(output_graph_def.SerializeToString())


def _serving_model_fn(features, mode, params):
    features = features[params["input_name"]]
    return _predict_model_fn(features, mode, params)


def _predict_model_fn(features, mode, params):
    features = _network_fn(features, mode, params)
    outputs = _get_output(features, params["output_layer"], params["num_classes"])
    predictions = {
        "outputs": outputs
    }

    return _create_model_fn(mode, predictions=predictions,
                            export_outputs={
                                "outputs": tf.estimator.export.PredictOutput(predictions)
                            })


def _test_model_fn(features, labels, mode, params):
    loss, metrics, predictions = _get_evaluation_parameters(features, labels, mode, params)
    _add_to_summary("test_loss", loss)
    for metric_key in metrics:
        _add_to_summary("test_" + metric_key, metrics[metric_key])
    evaluation_hooks = [tf.train.LoggingTensorHook(predictions, every_n_iter=1),
                        tf.train.SummarySaverHook(save_steps=1,
                                                  output_dir=params['summary_dir'],
                                                  summary_op=tf.summary.merge_all())
                        ]
    return _create_model_fn(mode, predictions=predictions, loss=loss,
                            evaluation_hooks=evaluation_hooks)


def _eval_model_fn(features, labels, mode, params):
    loss, metrics, predictions = _get_evaluation_parameters(features, labels, mode, params)

    evaluation_hooks = [tf.train.LoggingTensorHook(predictions, every_n_iter=1)]
    for metric_key in metrics:
        metrics[metric_key] = metric_functions.create_eval_metric(metrics[metric_key])
    return _create_model_fn(mode, predictions=predictions, loss=loss,
                            eval_metric_ops=metrics, evaluation_hooks=evaluation_hooks)


def _train_model_fn(features, labels, mode, params):
    loss, metrics, predictions = _get_evaluation_parameters(features, labels, mode, params)

    train_op = _create_train_op(loss,
                                learning_rate=params["learning_rate"],
                                optimizer=params["optimizer"])

    training_hooks = []
    for metric_key in metrics:
        _add_to_summary(metric_key, metrics[metric_key])
        training_hooks.append(tf.train.LoggingTensorHook(
            {metric_key: metric_key},
            every_n_iter=params["log_step_count_steps"])
        )
    return _create_model_fn(mode,
                            predictions=predictions,
                            loss=loss,
                            train_op=train_op,
                            training_hooks=training_hooks)


def _get_evaluation_parameters(features, labels, mode, params):
    features, predictions = _get_fed_features_and_resulting_predictions(features, mode, params)
    loss = _get_loss(params["loss"], labels=labels,
                     inputs=features, num_classes=params["num_classes"])
    metrics = _get_metrics(params["metrics"],
                           y_pred=features,
                           y_true=labels,
                           num_classes=params["num_classes"])
    return loss, metrics, predictions


def _get_fed_features_and_resulting_predictions(features, mode, params):
    features = features['features']
    features = _network_fn(features, mode, params)
    outputs = _get_output(features, params["output_layer"], params["num_classes"])
    predictions = {
        "outputs": outputs
    }
    return features, predictions


def _network_fn(features, mode, params):
    features = _set_dynamic_batch_size(features)
    for layer in params["network"]:
        features = feed(features, layer, is_training=mode == tf.estimator.ModeKeys.TRAIN)
    return features


def _set_dynamic_batch_size(inputs):
    new_shape = inputs.get_shape().as_list()
    new_shape[0] = -1
    inputs = tf.reshape(inputs, new_shape, name="input_layer")
    return inputs
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/layers.py

```
import tensorflow as tf

from tensorflow.contrib import rnn, slim


def get_sequence_lengths(inputs):
    used = tf.sign(tf.reduce_max(tf.abs(inputs), 2))
    length = tf.reduce_sum(used, 1)
    length = tf.cast(length, tf.int32)
    return length


def reshape(tensor: tf.Tensor, new_shape: list, name="reshape"):
    return tf.reshape(tensor, new_shape, name=name)


def bidirectional_rnn(inputs, num_hidden, cell_type='LSTM',
                      activation='tanh', concat_output=True,
                      scope=None):
    with tf.variable_scope(scope, "bidirectional_rnn", [inputs]):
        cell_fw = _get_cell(num_hidden, cell_type, activation)
        cell_bw = _get_cell(num_hidden, cell_type, activation)
        outputs, _ = tf.nn.bidirectional_dynamic_rnn(cell_fw,
                                                     cell_bw,
                                                     inputs,
                                                     sequence_length=get_sequence_lengths(inputs),
                                                     dtype=tf.float32)
        if concat_output:
            return tf.concat(outputs, 2)
        return outputs


def _get_activation(name):
    if name == 'tanh':
        return tf.nn.tanh
    if name == 'relu':
        return tf.nn.relu
    if name == 'relu6':
        return tf.nn.relu6
    raise NotImplementedError(name, "activation function not implemented")


def _get_cell(num_filters_out, cell_type='LSTM', activation='tanh'):
    cell_type = cell_type or 'LSTM'
    activation = activation or 'tanh'
    activation_function = _get_activation(activation)
    if cell_type == 'LSTM':
        return rnn.LSTMCell(num_filters_out,
                            initializer=slim.xavier_initializer(),
                            activation=activation_function)
    if cell_type == 'GRU':
        return rnn.GRUCell(num_filters_out,
                           kernel_initializer=slim.xavier_initializer(),
                           activation=activation_function)
    if cell_type == 'GLSTM':
        return rnn.GLSTMCell(num_filters_out,
                             initializer=slim.xavier_initializer(),
                             activation=activation_function)
    raise NotImplementedError(cell_type, "is not supported.")


def mdrnn(inputs, num_hidden, cell_type='LSTM', activation='tanh', scope=None):
    with tf.variable_scope(scope, "multidimensional_rnn", [inputs]):
        hidden_sequence_horizontal = _bidirectional_rnn_scan(inputs,
                                                             num_hidden // 2,
                                                             cell_type=cell_type,
                                                             activation=activation)
        with tf.variable_scope("vertical"):
            transposed = tf.transpose(hidden_sequence_horizontal, [0, 2, 1, 3])
            output_transposed = _bidirectional_rnn_scan(transposed, num_hidden // 2, cell_type=cell_type)
        output = tf.transpose(output_transposed, [0, 2, 1, 3])
        return output


def images_to_sequence(inputs):
    _, _, width, num_channels = _get_shape_as_list(inputs)
    s = tf.shape(inputs)
    batch_size, height = s[0], s[1]
    return reshape(inputs, [batch_size * height, width, num_channels])


def _get_shape_as_list(tensor):
    return tensor.get_shape().as_list()


def sequence_to_images(tensor, height):
    num_batches, width, depth = tensor.get_shape().as_list()
    if num_batches is None:
        num_batches = -1
    else:
        num_batches = num_batches // height
    reshaped = tf.reshape(tensor,
                          [num_batches, width, height, depth])
    return tf.transpose(reshaped, [0, 2, 1, 3])


def _bidirectional_rnn_scan(inputs, num_hidden, cell_type='LSTM', activation='tanh'):
    with tf.variable_scope("BidirectionalRNN", [inputs]):
        height = inputs.get_shape().as_list()[1]
        inputs = images_to_sequence(inputs)
        output_sequence = bidirectional_rnn(inputs, num_hidden, cell_type, activation)
        output = sequence_to_images(output_sequence, height)
        return output


def conv2d(inputs, num_filters, kernel_size, activation="relu", stride=1, padding='SAME', scope=None):
    if isinstance(kernel_size, int):
        kernel_size = [kernel_size, kernel_size]
    padding = padding or 'SAME'
    activation = activation or "relu"
    return slim.conv2d(inputs, num_filters, kernel_size,
                       activation_fn=_get_activation(activation),
                       padding=padding,
                       stride=stride,
                       scope=scope)


def max_pool2d(inputs, pool_size, padding='VALID', stride=2, scope=None):
    if isinstance(pool_size, int):
        pool_size = [pool_size, pool_size]
    padding = padding or 'VALID'
    stride = stride or 2
    return slim.max_pool2d(inputs, pool_size, padding=padding, stride=stride, scope=scope)


def dropout(inputs, keep_prob, is_training, scope=None):
    return slim.dropout(inputs, keep_prob, scope=scope, is_training=is_training)


def collapse_to_rnn_dims(inputs):
    batch_size, height, width, num_channels = inputs.get_shape().as_list()
    if batch_size is None:
        batch_size = -1
    nwhc_cnn_outputs = tf.transpose(inputs, (0, 2, 1, 3))
    batch_major_rnn_inputs = tf.reshape(nwhc_cnn_outputs,
                                        [batch_size, width, height * num_channels]
                                        )
    return batch_major_rnn_inputs


def batch_norm(inputs, is_training):
    return slim.batch_norm(inputs, is_training=is_training)


def l2_normalize(inputs, axis):
    return tf.nn.l2_normalize(inputs, axis)
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/layers˙test.py

```
import tensorflow as tf

from trainer.backend.tf.layers import images_to_sequence
from trainer.backend.tf.layers import mdrnn
from trainer.backend.tf.layers import sequence_to_images


def _create_input(shape):
    return tf.placeholder(tf.float32, shape)


class MDRNNTest(tf.test.TestCase):
    def testImagesToSequenceDims(self):
        inputs = _create_input([2, 7, 11, 5])
        outputs = images_to_sequence(inputs)
        expected_shape = (14, 11, 5)
        self._testAssertShapesAreEqual(outputs, expected_shape)

    def _testAssertShapesAreEqual(self, result, expected_shape):
        self.assertEqual(tuple(result.get_shape().as_list()), expected_shape)

    def testImagesToSequenceDimsDynamicBatchSize(self):
        inputs = tf.placeholder(tf.float32, [None, 7, 11, 5])
        outputs = images_to_sequence(inputs)
        expected_shape = (None, 11, 5)
        self._testAssertShapesAreEqual(outputs, expected_shape)

    def testSequenceToImagesDims(self):
        inputs = _create_input([14, 11, 5])
        outputs = sequence_to_images(inputs, 7)
        expected_shape = (2, 7, 11, 5)
        self._testAssertShapesAreEqual(outputs, expected_shape)

    def testSequenceToImagesDimsDynamicBatchSize(self):
        inputs = _create_input([None, 11, 5])
        outputs = sequence_to_images(inputs, 7)
        expected_shape = (None, 7, 11, 5)
        self._testAssertShapesAreEqual(outputs, expected_shape)

    def testMDRNN(self):
        inputs = _create_input([2, 7, 11, 5])
        outputs = mdrnn(inputs, num_hidden=8)
        expected_shape = (2, 7, 11, 8)
        self._testAssertShapesAreEqual(outputs, expected_shape)

    def testMRNNDynamicBatchSize(self):
        inputs = _create_input([None, 7, 11, 5])
        outputs = mdrnn(inputs, num_hidden=8)
        expected_shape = (None, 7, 11, 8)
        self._testAssertShapesAreEqual(outputs, expected_shape)


if __name__ == "__main__":
    tf.test.main()
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/losses.py

```
import tensorflow as tf

def ctc_loss(labels, inputs, sequence_length, preprocess_collapse_repeated_labels=False,
             ctc_merge_repeated=True, inputs_are_time_major=True):
    return tf.reduce_mean(tf.nn.ctc_loss(labels, inputs, sequence_length,
                          preprocess_collapse_repeated=preprocess_collapse_repeated_labels,
                          ctc_merge_repeated=ctc_merge_repeated,
                          time_major=inputs_are_time_major))
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/metric˙functions.py

```
import tensorflow as tf


def create_eval_metric(values):
    return tf.metrics.mean(values)


def label_error_rate(y_pred, y_true, name="label_error_rate"):
    return tf.reduce_mean(tf.edit_distance(tf.cast(y_pred, tf.int32), y_true), name=name)
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/replicate˙model˙fn.py

```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
"""Utilities to replicate model_fn's over local GPUs.

This file contains util that allow to replicate 'Estimator.model_fn' over
GPUs.  Replicated version of a 'model_fn' is returned that can subsequently
be used with 'Estimator'.
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import copy
from collections import defaultdict
from contextlib import contextmanager

import six
from tensorflow.core.framework import node_def_pb2
from tensorflow.python.client import device_lib
from tensorflow.python.estimator import model_fn as model_fn_lib
from tensorflow.python.estimator import util
from tensorflow.python.estimator.export import export_output as export_output_lib
from tensorflow.python.framework import device as framework_device
from tensorflow.python.framework import ops as ops_lib
from tensorflow.python.framework import sparse_tensor
from tensorflow.python.ops import array_ops
from tensorflow.python.ops import control_flow_ops
from tensorflow.python.ops import math_ops
from tensorflow.python.ops import sparse_ops
from tensorflow.python.ops import state_ops
from tensorflow.python.ops import variable_scope
from tensorflow.python.ops.losses import losses
from tensorflow.python.platform import tf_logging
from tensorflow.python.training import device_setter as device_setter_lib
from tensorflow.python.training import optimizer as optimizer_lib


def replicate_model_fn(model_fn,
                       loss_reduction=losses.Reduction.SUM_BY_NONZERO_WEIGHTS,
                       devices=None):
    """Replicate 'Estimator.model_fn' over GPUs.

  The given 'model_fn' specifies a single forward pass of a model.  To replicate
  such a model over GPUs, each GPU gets its own instance of the forward pass
  (a.k.a. a tower).  The input features and labels get sharded into the chunks
  that correspond to the number of GPUs.  Each tower computes a loss based
  on its input.  For each such loss, gradients are computed.  After that, the
```

~~~
available losses are aggregated to form aggregated loss.  Available
gradients are summed.  Then, they update weights using the specified
optimizer.

If `devices` are `None`, then all available GPUs are going to be used for
replication.  If no GPUs are available, then the model is going to be
placed on the CPU.

Two modes of local replication over available GPUs are supported:
  1)  If exactly 1 GPU is detected, then variables and operations are placed
      onto the GPU.
  2)  If more than 1 GPU is detected, then variables are going to be placed on
      the CPU.  Replicas of operations are placed on each individual GPU.

Here is an example of how one might use their `model_fn` to run over GPUs:
  ```python
     ...
     def model_fn(...):  # See `model_fn` in `Estimator`.
       loss = ...
       optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
       optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)
       if mode == tf.estimator.ModeKeys.TRAIN:
         #  See the section below on `EstimatorSpec.train_op`.
         return EstimatorSpec(mode=mode, loss=loss,
                              train_op=optimizer.minimize(loss))

       #  No change for `ModeKeys.EVAL` or `ModeKeys.PREDICT`.
       return EstimatorSpec(...)
     ...
     classifier = tf.estimator.Estimator(
       model_fn=tf.contrib.estimator.replicate_model_fn(model_fn))
  ~~~

Please see `DNNClassifierIntegrationTest` for an example with a canned
Estimator.

On `EstimatorSpec.train_op`:
`model_fn` returns `EstimatorSpec.train_op` for
`tf.estimator.GraphKeys.TRAIN`. It is typically derived using an optimizer.
Towers are expected to populate it in the same way. Gradients from all towers
are reduced and applied in the last tower. To achieve that in the case of
multiple towers, `TowerOptimizer` needs to be used. See `TowerOptimizer`.

On sharding input features and labels:
Input features and labels are split for consumption by each tower. They are
split across the dimension 0. Features and labels need to be batch major.

On reduction algorithms:
Certain algorithms were chosen for aggregating results of computations on
multiple towers:
  - Losses from all towers are reduced according to `loss_reduction`.
  - Gradients from all towers are reduced according to `loss_reduction`
    for each trainable variable.
  - `eval_metrics_ops` are reduced per metric using `reduce_mean`.
  - `EstimatorSpec.predictions` and `EstimatorSpec.export_outputs` are
    reduced using concatenation.
  - For all other fields of `EstimatorSpec` the values of the first tower
    are taken.

On distribution of variables:
Variables are not duplicated between towers. Instead, they are placed on a
single device as defined above and shared across towers.

On overhead:
If only one device is specified, then aggregation of loss and gradients
doesn't happen. Replication consists of placing `model_fn` onto the
specified device.

On current limitations:
  - `predictions` are not supported for `ModeKeys.EVAL`. They are required
     for `tf.contrib.estimator.add_metrics`.

Args:
  model_fn: `model_fn` as defined in `Estimator`. See the section above about
    the train_op argument of `EstimatorSpec`.
  loss_reduction: controls whether losses are summed or averaged.
  devices: Optional list of devices to replicate the model across. This
    argument can be used to replice only on the subset of available GPUs.
    If `None`, then all available GPUs are going to be used for replication.
    If no GPUs are available, then the model is going to be placed on the CPU.

Raises:
  ValueError: if there is no `loss_reduction` or if TowerOptimizer is
    mis-used.

Returns:
  A replicated version of the supplied `model_fn`. Returned function that
    conforms to the requirements of `Estimator`'s `model_fn` and can be used
    instead of the supplied `model_fn`.
"""
  return _replicate_model_fn_with_mode(
      model_fn,
      loss_reduction,
      devices,
      # TODO(isaprykin): Query the system configuration to choose modes other
      # than `SHARED_LOCAL_PARAMETER_SERVER`, even though it is often
      # appropriate.
      mode=_VariableDistributionMode.SHARED_LOCAL_PARAMETER_SERVER)
```
```

```
class _VariableDistributionMode(object):
    """Modes for variable distribution used for forcing a particular one.

  Forcing a mode is meant for performance experimentation purposes rather than
  for general use cases.
  """

    SHARED_LOCAL_PARAMETER_SERVER = 1
    """Variables are placed on a single device and shared across all devices.

  Two ways to achieve this distribution over available GPUs are supported:
    1)  If exactly 1 GPU is detected, then variables and operations are placed
        onto GPU.
    2)  If more than 1 GPU is detected, then variables are going to be placed on
        the CPU.  Replicas of operations are placed on each individual GPU.
  """

    SHARED_ROUND_ROBIN = 2
    """Variables are placed on all devices in a round-robin fashion.

  Every subsequent variable is placed on the next device.  There is only one
  copy of each variable that is shared across all devices.
  """


def _replicate_model_fn_with_mode(
        model_fn,
        loss_reduction,
        devices=None,
        mode=_VariableDistributionMode.SHARED_LOCAL_PARAMETER_SERVER):
    """A version of `replicate_model_fn` that allows to specify a `mode`."""
    if loss_reduction == losses.Reduction.NONE:
        raise ValueError('Tower losses need to be reduced in some way, yet {} '
                         'reduction is specified.'.format(loss_reduction))
    if not devices:
        devices = _get_local_devices('GPU') or _get_local_devices('CPU')

    is_a_single_gpu_case = len(devices) == 1 and 'GPU' in devices[0].upper()
    consolidation_device = devices[0] if is_a_single_gpu_case else '/CPU:0'

    ps_devices = [consolidation_device]
    if mode == _VariableDistributionMode.SHARED_ROUND_ROBIN:
        ps_devices = devices

    tf_logging.info('Replicating the `model_fn` across {}.  Variables are going '
                    'to be placed on {}.  Consolidation device is going to be {}.'
                    .format(devices, ps_devices, consolidation_device))

    def single_device_model_fn(features, labels, mode, params=None, config=None):
        """`model_fn` on a single device without reduction overhead."""
        return _get_loss_towers(
            model_fn=model_fn,
            mode=mode,
            features=[features],
            labels=[labels],
            params=params,
            loss_reduction=loss_reduction,
            config=config,
            devices=devices,
            local_ps_devices=ps_devices)[0]  # One device, so one spec is out.

    def replicated_model_fn(features, labels, mode, params=None, config=None):
        """Replicated version of `model_fn` to be used instead."""
        feature_shards, label_shards = _split_batch(
            features, labels, len(devices), device=consolidation_device)
        tower_specs = _get_loss_towers(
            model_fn=model_fn,
            mode=mode,
            features=feature_shards,
            labels=label_shards,
            params=params,
            loss_reduction=loss_reduction,
            config=config,
            devices=devices,
            local_ps_devices=ps_devices)

        if mode == model_fn_lib.ModeKeys.TRAIN:
            train_op = _minimize_towers(tower_specs)
            return _train_spec(
                tower_specs, train_op, aggregation_device=consolidation_device)
        elif mode == model_fn_lib.ModeKeys.EVAL:
            return _eval_spec(tower_specs, aggregation_device=consolidation_device)
        elif mode == model_fn_lib.ModeKeys.PREDICT:
            return _predict_spec(tower_specs, aggregation_device=consolidation_device)

    if len(devices) == 1:
        return single_device_model_fn
    else:
        return replicated_model_fn


class TowerOptimizer(optimizer_lib.Optimizer):
    """Gathers gradients from all towers and reduces them in the last one."""

    COLLECTION_FOR_GRAPH_STATES = 'replicate_model_fn_graph_states'

    def __init__(self, optimizer_or_optimizer_fn):
        """Wrap an existing optimizer for gathering gradients across towers.
```

```
Each invocation of model_fn has to call the same optimizers in the same
order.

Multiple optimizers that use the same or different losses are supported.

If TowerOptimizer is used but `replicate_model_fn` isn't, then no
aggregation will happen.  All calls will simply be forwarded to the
underlying optimizer. The behavior is similar if there is only one tower.

If TowerOptimizer is used together with SyncReplicasOptimizer that wraps
the user's optimizer, then it's the SyncReplicasOptimizer that needs to be
wrapped with TowerOptimizer.

Args:
  optimizer_or_optimizer_fn: an instance of optimizer to wrap.  That
    instance is going to be used for optimizer-specific logic.  This can
    also be a no-argument function that returns such an optimizer instance.
"""
    self._optimizer_or_optimizer_fn = optimizer_or_optimizer_fn

@staticmethod
def has_been_used():
    return TowerOptimizer._graph_state().has_tower_optimizer_been_used

def get_slot(self, *args, **kwargs):
    return self._get_optimizer().get_slot(*args, **kwargs)

def get_slot_names(self, *args, **kwargs):
    return self._get_optimizer().get_slot_names(*args, **kwargs)

def get_name(self, *args, **kwargs):
    return self._get_optimizer().get_name(*args, **kwargs)

def variables(self, *args, **kwargs):
    return self._get_optimizer().variables(*args, **kwargs)

def compute_gradients(self, loss, *args, **kwargs):
    """Compute gradients, but first, if needed, scale the loss."""
    loss = _scale_loss(loss,
                       self._graph_state().loss_reduction,
                       self._graph_state().number_of_towers)
    return self._get_optimizer().compute_gradients(loss, *args, **kwargs)

def apply_gradients(self, grads_and_vars, global_step=None, **kwargs):
    """Collect gradients updates to apply them with the last tower."""
    if self._graph_state().number_of_towers == 1:
        # Avoid the overhead of reduction if there's only one tower.
        #
        # There assumed to be only one tower if aggregation-related methods were
        # not called by `_get_loss_towers`, for example if the model_fn uses
        # TowerEstimator, but `replicate_model_fn` isn't used.
        return self._get_optimizer().apply_gradients(grads_and_vars, global_step,
                                                     **kwargs)

    self._graph_state().collect_gradients(grads_and_vars)

    if not self._graph_state().is_the_last_tower:
        with ops_lib.control_dependencies(_extract_tensors(grads_and_vars)):
            return self._construct_no_op_train_op()
    else:
        # Gradients need to be gathered and applied in the scope of the first
        # tower, so that the tensors are accessible via names without prefixes.
        var_scope, name_scope = self._graph_state().scopes_of_the_first_tower
        with variable_scope.variable_scope(var_scope):
            with ops_lib.name_scope(name_scope):
                return self._apply_gathered_gradients(global_step, **kwargs)

def _apply_gathered_gradients(self, global_step, **kwargs):
    graph_state = self._graph_state()
    optimizer = self._get_optimizer()

    grad_lists = {}
    for grad, var in graph_state.get_latest_gradients_from_all_towers():
        if grad is not None:
            grad_lists.setdefault(var, []).append(grad)

    aggregated_grads = []
    with ops_lib.name_scope('gradient_aggregating'):
        for var, grads in six.iteritems(grad_lists):
            grad = _compute_sum_on_device(grads, var.device)
            aggregated_grads.append((grad, var))
    return optimizer.apply_gradients(
        aggregated_grads, global_step=global_step, **kwargs)

def _get_optimizer(self):
    if callable(self._optimizer_or_optimizer_fn):
        # If optimizer is given as a function then we need to wait till we are
        # under the right graph context before constructing it.  That's why the
        # optimizer is constructed in _get_optimizer() rather than __init__().
        self._optimizer_or_optimizer_fn = self._optimizer_or_optimizer_fn()
    self._graph_state().has_tower_optimizer_been_used = True
    return self._optimizer_or_optimizer_fn

def _construct_no_op_train_op(self):
    return control_flow_ops.no_op(name='train_op_placeholder')

@staticmethod
def _graph_state():
        graph_states = ops_lib.get_default_graph().get_collection_ref(
            TowerOptimizer.COLLECTION_FOR_GRAPH_STATES)
        if not graph_states:
            graph_states.append(TowerOptimizer._PerGraphState())
        return graph_states[-1]

    @staticmethod
    def _did_towers_have_same_optimizer_calls():
        graph_state = TowerOptimizer._graph_state()
        return graph_state.did_towers_have_same_optimizer_calls()

    @staticmethod
    def _clear_graph_state():
        # Clearing the Graph collection will prevent _PerGraphState from being
        # serialized.
        ops_lib.get_default_graph().clear_collection(
            TowerOptimizer.COLLECTION_FOR_GRAPH_STATES)

    class _PerGraphState(object):
        """Gradient reduction related state of a Tensorflow graph."""

        def __init__(self):
            self._collected_grads_and_vars = defaultdict(list)
            self._current_tower_index = 0
            self._number_of_towers = 1
            self._loss_reduction = None
            # Scopes of the first tower that don't have a prefix:
            self._variable_scope = None
            self._name_scope = None
            # If needed, alert that TowerOptimizer needs to be used with model_fn.
            self._has_tower_optimizer_been_used = False
            # TODO: figure out a way to include the name without causing errors in importing the meta_graph.
            # self.name = "graph_reduction"

        def collect_gradients(self, grads_and_vars):
            self._collected_grads_and_vars[self._current_tower_index].append(
                grads_and_vars)

        def get_latest_gradients_from_all_towers(self):
            """Get gradients across towers for the last called optimizer."""
            grads_and_vars = []
            index_of_last_gradients = len(
                self._collected_grads_and_vars[self._current_tower_index]) - 1
            for tower_id in range(self._current_tower_index + 1):
                grads_and_vars.extend(
                    self._collected_grads_and_vars[tower_id][index_of_last_gradients])
            return grads_and_vars

        def set_reduction_across_towers(self, loss_reduction, number_of_towers):
            self._loss_reduction = loss_reduction
            self._number_of_towers = number_of_towers

        @contextmanager
        def tower(self, tower_id, var_scope, name_scope):
            if tower_id == 0:
                self._variable_scope = var_scope
                self._name_scope = name_scope
            self._current_tower_index = tower_id
            yield

        @property
        def scopes_of_the_first_tower(self):
            return self._variable_scope, self._name_scope

        @property
        def is_the_last_tower(self):
            return self._current_tower_index == (self._number_of_towers - 1)

        @property
        def number_of_towers(self):
            return self._number_of_towers

        @property
        def loss_reduction(self):
            return self._loss_reduction

        @property
        def has_tower_optimizer_been_used(self):
            return self._has_tower_optimizer_been_used

        @has_tower_optimizer_been_used.setter
        def has_tower_optimizer_been_used(self, value):
            self._has_tower_optimizer_been_used = value

        def did_towers_have_same_optimizer_calls(self):
            total_number_of_grads = sum([
                len(grads)
                for _, grads in six.iteritems(self._collected_grads_and_vars)
            ])
            return total_number_of_grads % self._number_of_towers == 0


def _get_local_devices(device_type):
    local_device_protos = device_lib.list_local_devices()
    return [
        device.name
        for device in local_device_protos
        if device.device_type == device_type
    ]
```

```
def _split_batch(features, labels, number_of_shards, device):
    """Split input features and labes into batches."""

    def ensure_divisible_by_shards(sequence):
        batch_size = ops_lib.convert_to_tensor(sequence).get_shape()[0]
        if batch_size % number_of_shards != 0:
            raise ValueError(
                'Batch size {} needs to be divisible by the number of GPUs, which '
                'is {}.'.format(batch_size, number_of_shards))

    def split_dictionary(dictionary):
        """Split a dictionary into shards."""
        shards = [{} for _ in range(number_of_shards)]
        for name, tensor in six.iteritems(dictionary):
            if isinstance(tensor, sparse_tensor.SparseTensor):
                for i, shard in enumerate(
                        sparse_ops.sparse_split(
                            sp_input=tensor, num_split=number_of_shards, axis=0)):
                    shards[i][name] = shard
            else:
                ensure_divisible_by_shards(tensor)
                for i, shard in enumerate(array_ops.split(tensor, number_of_shards)):
                    shards[i][name] = shard
        return shards

    with ops_lib.name_scope('split_inputs'):
        with ops_lib.device(device):
            if isinstance(features, dict):
                feature_shards = split_dictionary(features)
            else:
                ensure_divisible_by_shards(features)
                feature_shards = array_ops.split(features, number_of_shards)

            if labels is None:
                label_shards = None
            elif isinstance(labels, dict):
                label_shards = split_dictionary(labels)
            else:
                ensure_divisible_by_shards(labels)
                label_shards = array_ops.split(labels, number_of_shards)
    return feature_shards, label_shards


_DEFAULT_NAME_SCOPE_PATTERN = 'tower_{}'


def _get_loss_towers(model_fn,
                     mode,
                     features,
                     labels,
                     params,
                     config,
                     devices,
                     local_ps_devices,
                     loss_reduction,
                     name_scope_pattern=_DEFAULT_NAME_SCOPE_PATTERN):
    """Replicate the loss computation across devices."""
    tower_specs = []

    model_fn_args = util.fn_args(model_fn)
    optional_params = {}
    if 'params' in model_fn_args:
        optional_params['params'] = copy.deepcopy(params)
    if 'config' in model_fn_args:
        optional_params['config'] = copy.deepcopy(config)

    # pylint: disable=protected-access
    round_robin_strategy = device_setter_lib._RoundRobinStrategy(
        num_tasks=len(local_ps_devices))
    TowerOptimizer._graph_state().set_reduction_across_towers(
        loss_reduction, len(devices))

    for i, device in enumerate(devices):
        is_the_first_tower = (i == 0)

        device_setter = _local_device_setter(
            worker_device=device,
            ps_devices=local_ps_devices,
            ps_strategy=round_robin_strategy)

        # We would like to preserve the names of the variables and ops that the user
        # might be relying on. Names without a prefix are going to resolve to
        # variables and ops of the first tower.
        name_scope = name_scope_pattern
        if is_the_first_tower:
            name_scope = ''

        with variable_scope.variable_scope(
                '', reuse=not is_the_first_tower) as var_scope:
            with ops_lib.name_scope(name_scope.format(i)) as name_scope:
                with TowerOptimizer._graph_state().tower(
                        tower_id=i, var_scope=var_scope, name_scope=name_scope):
                    with ops_lib.device(device_setter):
                        labels_shard = None
                        if labels:
                            labels_shard = labels[i]
                        tower_spec = model_fn(
                            mode=mode,
                            features=features[i],
                            labels=labels_shard,
                            **optional_params)

                        if (tower_spec.train_op is not None and len(devices) > 1 and
                                not TowerOptimizer.has_been_used()):
                            raise ValueError('Please wrap optimizers with TowerOptimizer'
                                             ' in order to use replicate_model_fn with'
                                             ' multiple `devices`.')

                        # Scaling the loss here doesn't actually affect gradients.  Another
                        # instance of scaling happens inside the TowerOptimizer.
                        tower_spec = _scale_tower_loss(
                            tower_spec, loss_reduction, number_of_towers=len(devices))
                        tower_specs.append(tower_spec)

    if not TowerOptimizer._did_towers_have_same_optimizer_calls():
        raise ValueError('Each invocation of model_fn was supposed to make the same'
                         ' optimizer calls.')
    TowerOptimizer._clear_graph_state()
    # pylint: enable=protected-access
    return tower_specs


def _local_device_setter(worker_device, ps_devices, ps_strategy):
    """A device setter that puts distributes Var/Ops to PS/workers."""
    ps_ops = ['Variable', 'VariableV2', 'VarHandleOp']

    def local_device_chooser(op):
        current_device = framework_device.DeviceSpec.from_string(op.device or '')

        node_def = op if isinstance(op, node_def_pb2.NodeDef) else op.node_def
        if node_def.op in ps_ops:
            ps_device_spec = framework_device.DeviceSpec.from_string(
                '{}'.format(ps_devices[ps_strategy(op)]))

            ps_device_spec.merge_from(current_device)
            return ps_device_spec.to_string()
        else:
            worker_device_spec = framework_device.DeviceSpec.from_string(
                worker_device or '')
            worker_device_spec.merge_from(current_device)
            return worker_device_spec.to_string()

    return local_device_chooser


def _scale_tower_loss(tower_spec, loss_reduction, number_of_towers):
    """Produce an EstimatorSpec with approproriately scaled loss."""
    if tower_spec.loss is None:
        return tower_spec

    estimator_spec = _asdict(tower_spec)
    estimator_spec['loss'] = _scale_loss(tower_spec.loss, loss_reduction,
                                             number_of_towers)
    return model_fn_lib.EstimatorSpec(**estimator_spec)


def _scale_loss(loss, loss_reduction, number_of_towers):
    """If needed, scale down the loss for averaging loss by summing."""
    if loss is None:
        return None
    if number_of_towers == 1:
        return loss

    if loss_reduction != losses.Reduction.SUM:
        return math_ops.div(loss, 1.0 * number_of_towers, name='averaged_loss')
    else:
        return loss


def _minimize_towers(tower_specs):
    """`train_op` of the last tower applies aggregated gradients."""
    return tower_specs[-1].train_op


def _compute_sum_on_device(values, device, name=None):
    with ops_lib.device(device):
        if isinstance(values[0], ops_lib.IndexedSlices):
            if name:
                raise ValueError('The name {} is not expected to be given to '
                                 'IndexedSlices {}'.format(name, values))

            values_concat = array_ops.concat([v.values for v in values], axis=0)
            indices_concat = array_ops.concat([v.indices for v in values], axis=0)
            return ops_lib.IndexedSlices(values_concat, indices_concat,
                                             values[0].dense_shape)
        else:
            return math_ops.add_n(values, name=name)


def _train_spec(tower_specs,
                train_op,
                aggregation_device,
                aggregated_loss_name='loss'):
    """Populate replicated EstimatorSpec for `GraphKeys.TRAIN`."""
    # Spec of the last tower is used as the template for the final spec, because
    # some `EstimatorSpec.training_hooks` rely on calls made in model_fn.  For
    # example, `SyncReplicasOptimizerHook` validates the
    # `SyncReplicasOptimizer.apply_gradients` call. `TowerEstimator` makes that
    # call only in the last tower.
    estimator_spec = _asdict(tower_specs[-1])
    estimator_spec['mode'] = model_fn_lib.ModeKeys.TRAIN
    estimator_spec['train_op'] = train_op
    estimator_spec['loss'] = _compute_sum_on_device(
        [spec.loss for spec in tower_specs], aggregation_device,
        aggregated_loss_name)
    return model_fn_lib.EstimatorSpec(**estimator_spec)


def _eval_spec(tower_specs, aggregation_device, aggregated_loss_name='loss'):
    """Populate replicated EstimatorSpec for `GraphKeys.EVAL`."""
    estimator_spec = _asdict(tower_specs[0])
    estimator_spec['mode'] = model_fn_lib.ModeKeys.EVAL
    estimator_spec['loss'] = _compute_sum_on_device(
        [spec.loss for spec in tower_specs], aggregation_device,
        aggregated_loss_name)

    update_ops = []
    for tower_spec in tower_specs:
        for name, (_, update_op) in six.iteritems(tower_spec.eval_metric_ops):
            update_ops.append(update_op)

    with ops_lib.control_dependencies(update_ops):
        reduced_update_op = _reduce_metric_variables(len(tower_specs))

    eval_metric_ops = {}
    for name, (metric_tensor, _) in six.iteritems(tower_specs[0].eval_metric_ops):
        eval_metric_ops[name] = (metric_tensor, reduced_update_op)
    estimator_spec['eval_metric_ops'] = eval_metric_ops
    return model_fn_lib.EstimatorSpec(**estimator_spec)


def _reduce_metric_variables(number_of_towers):
    """Aggregate local variables used in metrics into the first tower."""
    if number_of_towers == 1:
        return control_flow_ops.no_op(name='no_eval_metric_reduction')

    metric_variables = ops_lib.get_collection("metric_variables")
    variables_per_tower = len(metric_variables) // number_of_towers

    if len(metric_variables) % number_of_towers != 0:
        raise ValueError(
            'Different `EstimatorSpec.eval_metric_ops` across `model_fn()` calls.'
            ' Expected {} local variables, but got {} instead.'.format(
                variables_per_tower * number_of_towers, len(metric_variables)))

    # `metric_variables` has the size of `variables_per_tower` x
    #  number_of_towers.  Each tower is produced by calling the same model_fn.
    #  First `variables_per_tower` correspond to the first tower.  Each such
    #  variable has an replica at the `(variables_per_tower * i)` position, where
    #  `i` is `[1.. number_of_towers]`.  We are going to add values from replicas
    #  to each variable of the first tower.  We then zero out replica values, so
    #  that `_reduce_metric_variables` operation is idempotent.  If a metric
    #  is then computed based on local variables from the first tower, then the
    #  resulting metric is an estimate for all `number_of_towers` towers.
    ops = []
    for i in range(0, variables_per_tower):
        next_replica_id = i + variables_per_tower
        replicas = [
            metric_variables[replica_id]
            for replica_id in range(next_replica_id, len(metric_variables),
                                    variables_per_tower)
        ]  # `replicas` doesn't contain the first-tower variable.

        reduce_op = state_ops.assign_add(metric_variables[i],
                                         math_ops.add_n(replicas))

        with ops_lib.control_dependencies([reduce_op]):
            for replica in replicas:
                zeros_for_replica = array_ops.zeros(
                    array_ops.shape(replica), dtype=replica.dtype)
                zero_out_replica_op = state_ops.assign(replica, zeros_for_replica)
                ops.append(zero_out_replica_op)

    return control_flow_ops.group(*ops)


def _predict_spec(tower_specs, aggregation_device):
    """Populate replicated EstimatorSpec for `GraphKeys.PREDICT`."""
    estimator_spec = _asdict(tower_specs[0])
    estimator_spec['mode'] = model_fn_lib.ModeKeys.PREDICT

    with ops_lib.device(aggregation_device):
        estimator_spec['predictions'] = _concat_tensor_dicts(
            *[tower_spec.predictions for tower_spec in tower_specs])

        export_outputs_dict = _dict_concat(
            *[tower_spec.export_outputs for tower_spec in tower_specs])

        export_outputs = {}
        for name, export_output_list in six.iteritems(export_outputs_dict):
            if isinstance(export_output_list[0], export_output_lib.PredictOutput):
                export_outputs[name] = export_output_lib.PredictOutput(
                    outputs=_concat_tensor_dicts(*[
                        export_output.outputs for export_output in export_output_list
                    ]))
            elif isinstance(export_output_list[0],
                            export_output_lib.RegressionOutput):
                export_outputs[name] = export_output_lib.RegressionOutput(
                    value=array_ops.concat(
                        [export_output.value for export_output in export_output_list],
                        axis=0))
            elif isinstance(export_output_list[0],
                            export_output_lib.ClassificationOutput):
                scores = None
                if export_output_list[0].scores is not None:
                    scores = array_ops.concat(
                        [export_output.scores for export_output in export_output_list],
                        axis=0)

                classes = None
                if export_output_list[0].classes is not None:
                    classes = array_ops.stack(
                        [export_output.classes for export_output in export_output_list],
                        axis=0)

                export_outputs[name] = export_output_lib.ClassificationOutput(
                    scores=scores, classes=classes)

    estimator_spec['export_outputs'] = export_outputs
    return model_fn_lib.EstimatorSpec(**estimator_spec)


def _concat_tensor_dicts(*tensor_dicts):
    return {
        name: array_ops.concat(tensors, axis=0, name=name)
        for name, tensors in six.iteritems(_dict_concat(*tensor_dicts))
    }


def _extract_tensors(tensors_and_vars):
    tensors = []
    for tensor_and_var in tensors_and_vars:
        tensor, _ = tensor_and_var
        if isinstance(tensor, ops_lib.IndexedSlices):
            tensors.append(tensor.values)
        elif tensor is not None:
            tensors.append(tensor)
    return tensors


def _dict_concat(*dicts):
    list_dict = {}
    for d in dicts:
        if d is None:
            continue

        for k, v in six.iteritems(d):
            list_dict.setdefault(k, []).append(v)
    return list_dict


def _asdict(namedtuple):
    """Returns a namedtuple as a dictionary.

  This is required because `_asdict()` in Python 3.x.x is broken in classes
  that inherit from `collections.namedtuple`. See
  https://bugs.python.org/issue24931 for more details.

  Args:
    namedtuple: An object that inherits from `collections.namedtuple`.

  Returns:
    A dictionary version of the tuple.
  """
    return {k: getattr(namedtuple, k) for k in namedtuple._fields}
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/util˙ops.py

```
import tensorflow as tf

from tensorboard import main as tb

from trainer.backend.tf import layers
from trainer.backend.GraphKeys import LayerTypes


def get_sequence_lengths(inputs):
    used = tf.sign(tf.reduce_max(tf.abs(inputs), 2))
    length = tf.reduce_sum(used, 1)
    length = tf.cast(length, tf.int32)
    return length


def feed(inputs, layer, is_training):
    layer_type = layer["layer_type"]
    if layer_type == LayerTypes.CONV2D.value:
        return layers.conv2d(inputs, num_filters=layer["num_filters"],
                             kernel_size=layer["kernel_size"],
                             activation=layer.get("activation"),
                             padding=layer.get("padding"),
                             scope=layer.get("name"))
    if layer_type == LayerTypes.MAX_POOL2D.value:
        return layers.max_pool2d(inputs, pool_size=layer["pool_size"],
                                 padding=layer.get("padding"),
                                stride=layer.get("stride"),
                                scope=layer.get("name"))
    if layer_type == LayerTypes.BIRNN.value:
        return layers.bidirectional_rnn(inputs, num_hidden=layer["num_hidden"],
                                        cell_type=layer.get("cell_type"),
                                        activation=layer.get("activation"),
                                        scope=layer.get("name"))
    if layer_type == LayerTypes.MDRNN.value:
        return layers.mdrnn(inputs, num_hidden=layer["num_hidden"],
                            cell_type=layer.get("cell_type"),
                            activation=layer.get("activation"),
                            scope=layer.get("name"))
    if layer_type == LayerTypes.DROPOUT.value:
        return layers.dropout(inputs, keep_prob=layer["keep_prob"],
                              is_training=is_training,
                              scope=layer.get("name"))
    if layer_type == LayerTypes.COLLAPSE_TO_RNN_DIMS.value:
        return layers.collapse_to_rnn_dims(inputs)
    if layer_type == LayerTypes.L2_NORMALIZE.value:
        return layers.l2_normalize(inputs, [1, 2])
    if layer_type == LayerTypes.BATCH_NORM.value:
        return layers.batch_norm(inputs, is_training=is_training)
    raise NotImplementedError(layer_type + " layer not implemented.")


def dense_to_sparse(tensor, token_to_ignore=0):
    indices = tf.where(tf.not_equal(tensor, tf.constant(token_to_ignore, dtype=tensor.dtype)))
    values = tf.gather_nd(tensor, indices)
    shape = tf.shape(tensor, out_type=tf.int64)
    return tf.SparseTensor(indices, values, shape)


def visualize(model, host):
    tf.flags.FLAGS.logdir = model
    tf.flags.FLAGS.host = host
    tb.main()
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/util˙ops˙test.py

```
import tensorflow as tf

from trainer.backend.tf.layers import bidirectional_rnn
from trainer.backend.tf.ctc_ops import convert_to_ctc_dims, ctc_beam_search_decoder
from trainer.backend.tf.util_ops import get_sequence_lengths
from trainer.backend.tf.losses import ctc_loss


def _create_input(shape):
    return tf.placeholder(tf.float32, shape)


class GetSequenceLengthTest(tf.test.TestCase):
    def setUp(self):
        self.inputs = _create_input([1, 128, 64])
        self.inputs = bidirectional_rnn(self.inputs, 8)
        self.labels = tf.sparse_placeholder(dtype=tf.int32)
        self.sequence_lengths = get_sequence_lengths(self.inputs)

    def testCTCLOSS(self):
        ctc_inputs = convert_to_ctc_dims(self.inputs,
                                         num_classes=79,
                                         num_steps=self.inputs.shape[1],
                                         num_outputs=self.inputs.shape[-1])
        ctc_loss(self.labels, ctc_inputs, self.sequence_lengths)

    def testCTCDecoder(self):
        ctc_inputs = convert_to_ctc_dims(self.inputs,
                                         num_classes=79,
                                         num_steps=self.inputs.shape[1],
                                         num_outputs=self.inputs.shape[-1])
        ctc_beam_search_decoder(ctc_inputs, self.sequence_lengths)


if __name__ == "__main__":
    tf.test.main()
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/ValidationHook.py

```
import tensorflow as tf


class ValidationHook(tf.train.SessionRunHook):
    def __init__(self, model_fn, params, input_fn, checkpoint_dir,
                 every_n_secs=None, every_n_steps=None):
        self._iter_count = 0
        self._estimator = tf.estimator.Estimator(
            model_fn=model_fn,
            params=params,
            model_dir=checkpoint_dir
        )
        self._input_fn = input_fn
        self._timer = tf.train.SecondOrStepTimer(every_n_secs, every_n_steps)
        self._should_trigger = False

    def begin(self):
        self._timer.reset()
        self._iter_count = 0

    def before_run(self, run_context):
        self._should_trigger = self._timer.should_trigger_for_step(self._iter_count)
    def after_run(self, run_context, run_values):
        if self._should_trigger:
            self._estimator.evaluate(
                self._input_fn
            )
            self._timer.update_last_triggered_step(self._iter_count)
        self._iter_count += 1
```

File: Tensorflow-OCR-Trainer/trainer/backend/tf/¨init¨.py

```
from trainer.backend.tf.experiment_ops import train
from trainer.backend.tf.experiment_ops import test
```

File: Tensorflow-OCR-Trainer/trainer/templates/404.html

```
{% extends "base.html" %}
{% block title %}Page not found!{% endblock %}
{% block body %}
    <div class="container white-text">
        <div class="row">
            <h5>Page not found.</h5>
            <p>What you were looking for is not here</p>
            <p><a href="{{ url_for('index') }}">Go back to the home page.</a></p>
        </div>
    </div>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/500.html

```
{% extends "base.html" %}
{% block title %}Page not found!{% endblock %}
{% block body %}
    <div class="container white-text">
        <div class="row">
            <h5>Internal server error.</h5>
            <p>Something went wrong. </p>
            <p><a href="{{ url_for('index') }}">Go back to the home page.</a></p>
        </div>
    </div>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/architectures.html

```
{% extends "base.html" %}
{% block title %}Architectures{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Network Architectures</h3>
        <ul class="collection black-text">
        {% for network_architecture in network_architectures %}
            <li class="collection-item row">
                <ul>
                    <li class="col s6">{{ network_architecture }}</li>
                    <li class="col s3">
                        <a class="right waves-effect waves-light btn right"
                           href="{{ url_for('view_architecture', architecture_name=network_architecture) }}"
                           target="_blank">
                            View
                        </a>
                    </li>
                    <li class="col s3">
                        <form name="deletion" method="post" action="{{ url_for('delete_architecture',
                              architecture_name=network_architecture) }}">
                            <button type="submit" class="waves-effect waves-light btn red right" title="
                                 delete">
                                <i class="material-icons">remove</i>
                            </button>
                        </form>
                    </li>
                </ul>
            </li>
        {% endfor %}
        </ul>
        <a href="{{ url_for('create_network_architecture') }}" class="right waves-effect waves-light btn">
            <i class="material-icons left">add</i>
            Create Network Architecture
        </a>
    </div>
{% endblock %}
{% block extra_script%}
    <script type="text/javascript" src="{{ url_for('static', filename='js/delete-file.js') }}"></script>
    <script type="text/javascript" src="{{ url_for('static', filename='js/toast.js') }}"></script>
    <script>
        toast({{ get_flashed_messages()|tojson }})
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link rel=stylesheet type=text/css href="{{ url_for('static', filename='css/materialize.min.css') }}"
         media="screen,projection">
    <link rel=stylesheet type=text/css href="{{ url_for('static', filename='css/custom.css') }}" media="
         screen,projection">
    <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.png') }}">
</head>
```

```
<body>
<header class="page-header">
    <nav>
        <div class="nav-wrapper up-maroon">
            <div class="container">
                <img class="responsive-img" src="{{ url_for('static', filename="img/upm_logo.png") }}">
                <a class="brand-logo" href="{{ url_for('index') }}">TF OCR</a>
                <a href="#" data-activates="mobile-demo" class="button-collapse"><i class="material-icons">
                     menu</i></a>
                <ul id="nav-mobile" class="right hide-on-med-and-down">
                    <li><a href="{{ url_for('dataset') }}">Dataset</a></li>
                    <li><a href="{{ url_for('architectures') }}">Network Architectures</a></li>
                    <li><a href="{{ url_for('train') }}">Train</a></li>
                    <li><a href="{{ url_for('tasks', task='view') }}">Tasks</a></li>
                    <li><a href="{{ url_for('models') }}">Models</a></li>
                </ul>
                <ul id="mobile-demo" class="side-nav">
                    <li><a href="{{ url_for('dataset') }}">Dataset</a></li>
                    <li><a href="{{ url_for('architectures') }}">Network Architectures</a></li>
                    <li><a href="{{ url_for('train') }}">Train</a></li>
                    <li><a href="{{ url_for('tasks', task='view') }}">Tasks</a></li>
                    <li><a href="{{ url_for('models') }}">Models</a></li>
                </ul>
            </div>
        </div>
    </nav>
</header>
<main>
    {% block body %}{% endblock %}
</main>
<footer class="page-footer up-maroon">
    <div class="footer-copyright">
        <div class="container">
            &copy; 2018 Jerome Patrick V. Gonzalvo
        </div>
    </div>
</footer>
<script type="text/javascript" src="{{ url_for('static', filename='js/jquery-3.2.1.min.js') }}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/materialize.min.js') }}"></script>
<script type="text/javascript" src="{{ url_for('static', filename='js/script.js') }}"></script>
{% block extra_script %}
{% endblock %}
</body>
</html>
```

File: Tensorflow-OCR-Trainer/trainer/templates/create˙network˙architecture.html

```
{% extends "base.html" %}
{% block title %}Create Network Architecture{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Create Network Architecture</h3>
        <form id="my_form" method="post" action="{{ url_for('architectures') }}">
            <div class="row valign-wrapper">
                <div class="input-field col s12">
                    <input id="architecture_name" class="validate"
                           type="text" name="architecture_name"
                           required>
                    <label for="architecture_name">Architecture Name</label>
                </div>
                <div class="fixed-action-btn">
                    <a id="add-layer" class="btn-floating btn-large waves-effect waves-light" title="add
                         layer">
                        <i class="material-icons left">add</i>
                        Add Layer
                    </a>
                </div>
            </div>
            <ul id="layers" class="collection with-header black-text overflowing">
                <li class="collection-header"><h5>Layers</h5></li>
            </ul>
            <div class="row">
                <div class="input-field col s8">
                    <select id="output-layer-select" name="output_layer" required>
                        <option value="" disabled selected>Select Output Layer</option>
                        {% for output_layer in output_layers %}
                            <option value="{{ output_layer }}">{{ output_layer.replace('_', ' ')|upper }}</
                                 option>
                        {% endfor %}
                    </select>
                    <label for="output-layer-select">Output Layer</label>
                </div>
            </div>
            <div class="row">
                <button class="btn waves-effect waves-light right" type="submit" name="action">Create
                     Architecture
                    <i class="material-icons right">send</i>
                </button>
            </div>
        </form>
    </div>
{% endblock %}
{% block extra_script %}
    <script type="text/javascript" src="{{ url_for('static', filename='js/network_architecture_creation.js')
          }}"></script>
    <script>
        create_dynamic_layer_builder({{ layer_types|tojson }}, {{ padding_types|tojson }}, {{ cell_types|
             tojson }}, {{ activation_functions|tojson }});
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/dataset.html

```
{% extends "base.html" %}
{% block title %}Dataset{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Dataset List</h3>
        <ul class="collection black-text">
        {% if not dataset_list %}
            <li class="collection-item row">No dataset has been uploaded yet.
                Click <a href="{{ url_for('dataset_form') }}">here</a> to upload a dataset.
            </li>
        {% endif %}
        {% for dataset in dataset_list %}
            <li class="collection-item row">
                <ul>
                   <li class="col s3">{{ dataset['name'] }}</li>
                   <li class="col s3">
                       <i>Number of Training Samples</i>
                       <ul>
                           <li>{{ dataset['num_training_examples'] }}</li>
                       </ul>
                   </li>
                   <li class="col s3">
                       <i>Number of Testing Samples</i>
                       <ul>
                           <li>{{ dataset['num_testing_examples'] }}</li>
                       </ul>
                   </li>
                   <li class="col s3 right">
                        <form name="deletion" method="post" action="{{ url_for('delete_dataset',
                             dataset_name=dataset['name']) }}">
                            <button type="submit" class="waves-effect waves-light btn red right" title="
                                 delete">
                                <i class="material-icons">remove</i>
                            </button>
                        </form>
                    </li>
                </ul>
            </li>
        {% endfor %}
        </ul>
        <a href="{{ url_for('dataset_form') }}" class="right waves-effect waves-light btn">
            <i class="material-icons left">add</i>
            Upload Dataset
        </a>
    </div>
{% endblock %}
{% block extra_script%}
    <script type="text/javascript" src="{{ url_for('static', filename='js/delete-file.js') }}"></script>
    <script type="text/javascript" src="{{ url_for('static', filename='js/toast.js') }}"></script>
    <script>
        toast({{ get_flashed_messages()|tojson }})
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/dataset˙form.html

```
{% extends "base.html" %}
{% block title %}Upload Dataset{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Upload Dataset</h3>
        <form method="post" enctype="multipart/form-data" action="/dataset">
            <div class="row">
                <div class="input-field col s9">
                    <input id="dataset_name" class="validate" type="text" name="dataset_name" required>
                    <label for="dataset_name">Dataset Name</label>
                </div>
                <div class="input-field col s3">
                    <input id="test_size" name="test_size" type="number" step="0.01" min="0.01" max="0.5"
                         required>
                    <label for="test_size">Test Size</label>
                </div>
            </div>
            <div class="file-field input-field col s12">
                <div class="btn">
                    <span>File</span>
                    <input class="validate" type="file" name="dataset_zip" required>
                </div>
                <div class="file-path-wrapper">
                    <input class="file-path validate" type="text" placeholder="Upload dataset zip containing
                          the images and the labels file named 'labels.txt'">
                </div>
            </div>
            <button class="btn waves-effect waves-light right" type="submit" name="action">Upload
                <i class="material-icons right">send</i>
            </button>
        </form>
    </div>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/index.html

```
{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block body %}
    <div id="index-banner" class="parallax-container">
      <div class="section no-pad-bot">
        <div class="container">
      <br><br>
      <h1 class="header center teal-text text-lighten-2">Tensorflow OCR Trainer</h1>
      <div class="row center">
        <h5 class="header col white-text s12 light">A user-friendly OCR trainer using Tensorflow as its
             core</h5>
      </div>
      <div class="row center">
        <a href="{{ url_for('dataset') }}" id="download-button" class="btn-large waves-effect waves-
             light teal lighten-1">Get Started</a>
      </div>
      <br><br>
    </div>
  </div>
  <div class="parallax"><img src="{{ url_for('static', filename="img/neural-network.jpeg") }}" alt="
       Unsplashed background img 1"></div>
</div>

<div class="container white-text">
  <div class="section">
    <div class="row">
      <div class="col s12 m4">
        <div class="icon-block">
          <h2 class="center blue-grey-text"><i class="medium material-icons">file_upload</i></h2>
          <h5 class="center">Manage your dataset</h5>

          <p class="light">
              You can upload your own dataset containing the images and the labels file.
              You can also remove the dataset you don't use anymore.
          </p>
          <p class="center">
              <a href="{{ url_for('dataset_form') }}" id="download-button" class="btn-large waves-effect
                     waves-light teal lighten-1">Upload dataset</a>
          </p>
        </div>
      </div>

      <div class="col s12 m4">
        <div class="icon-block">
          <h2 class="center grey-text"><i class="medium material-icons">build</i></h2>
          <h5 class="center">Build your own architecture</h5>

          <p class="light">
              You can build your own architectures by just specifying the layers
              you want it to have and the parameters for each layer.
          </p>

          <p class="center">
              <a href="{{ url_for('create_network_architecture') }}" id="download-button" class="btn-
                   large waves-effect waves-light teal lighten-1">Build architecture</a>
          </p>
        </div>
      </div>

      <div class="col s12 m4">
        <div class="icon-block">
          <h2 class="center orange-text"><i class="medium material-icons">directions_run</i></h2>
          <h5 class="center">Train and validate</h5>

          <p class="light">
              By just choosing a dataset, an architecture, and providing some parameters,
              you can train your own model and validate it as it is trained.
          </p>

          <p class="center">
              <a href="{{ url_for('train') }}" id="download-button" class="btn-large waves-effect waves-
                   light teal lighten-1">Start Training</a>
          </p>
        </div>
      </div>
    </div>

  </div>
</div>

<div class="parallax-container valign-wrapper">
  <div class="section no-pad-bot">
    <div class="container white-text">
      <div class="row center">
        <h5 class="header col s12 light">Train models without dealing with much code!</h5>
      </div>
    </div>
  </div>
  <div class="parallax"><img src="{{ url_for('static', filename="img/upload-bg.jpeg") }}" alt="
       Unsplashed background img 2"></div>
</div>

<div class="container white-text">
  <div class="section">
      <div class="row">
      <div class="col s12 m4">
        <div class="icon-block">
          <h2 class="center blue-text"><i class="medium material-icons">show_chart</i></h2>
          <h5 class="center">Visualize</h5>
          <p class="light">
              With the use of Tensorboard, you can visualize the performance of your models.
              You can also visualize the architectures you use in these models.
          </p>
        </div>
      </div>
```

```
          <div class="col s12 m4">
            <div class="icon-block">
              <h2 class="center yellow-text"><i class="medium material-icons">star</i></h2>
              <h5 class="center">Test</h5>

              <p class="light">
                  You can test how well your models perform on data they haven't seen yet.
              </p>
            </div>
          </div>

          <div class="col s12 m4">
            <div class="icon-block">
              <h2 class="center green-text"><i class="medium material-icons">file_download</i></h2>
              <h5 class="center">Export</h5>

              <p class="light">
                  You can also export the models so that you can
                  deploy them (i.e. mobile applications).
              </p>
            </div>
          </div>
        </div>
      </div>
    </div>
{% endblock %}
{% block extra_script %}
    <script>
    $(document).ready(function () {
        $('.parallax').parallax();
    })
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/models.html

```
{% extends "base.html" %}
{% block title %}Models{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Trained Models</h3>
        <ul class="collection black-text">
        {% if not models %}
            <li class="collection-item row">No trained models yet.
                Click <a href="{{ url_for('train') }}">here</a> to train a model.
            </li>
        {% endif %}
        {% for model in models %}
            <li class="collection-item row">
                <ul>
                   <li class="col m3">
                       {{ model }}
                   </li>
                   <li class="col m8">
                       <div class="row">
                           <div class="col m2">
                               <a class="right waves-effect waves-light btn" href="{{ url_for('retrain',
                                    model_name=model) }}">
                                    Retrain
                               </a>
                           </div>
                           <div class="col m2">
                               <form id="testing" action="{{ url_for('tasks', task='testing') }}" method="
                                    post">
                                       <ul>
                                           <li>
                                               <button type="submit" class="waves-effect waves-light btn
                                                    right">
                                                   Test
                                               </button>
                                           </li>
                                       </ul>
                                   <input type="hidden" name="model_name" value="{{ model }}">
                               </form>
                           </div>
                           <div class="col m3">
                               <a class="right waves-effect waves-light btn" href="{{ url_for('export_model
                                    ', model_name=model) }}" target="_blank">
                                   Export
                               </a>
                           </div>
                           <div class="col m3">
                               <a class="right waves-effect waves-light btn" href="{{ url_for('visualize',
                                    model_name=model) }}" target="_blank">
                                   Visualize
                               </a>
                           </div>
                           <div class="col m2">
                               <a class="right waves-effect waves-light btn" href="{{ url_for('view_logs',
                                    model_name=model) }}" target="_blank">
                                   Logs
                               </a>
                           </div>
                       </div>
                   </li>
                   <li class="col m1">
                       <form name="deletion" method="post" action="{{ url_for('delete_model', model_name=
                            model) }}">
                           <button type="submit" class="waves-effect waves-light btn red" title="delete">
```

```
                                <i class="material-icons">remove</i>
                            </button>
                        </form>
                    </li>
                </ul>
            </li>
        {% endfor %}
        </ul>
    </div>
{% endblock %}
{% block extra_script%}
    <script type="text/javascript" src="{{ url_for('static', filename='js/delete-file.js') }}"></script>
    <script type="text/javascript" src="{{ url_for('static', filename='js/toast.js') }}"></script>
    <script>
        toast({{ get_flashed_messages()|tojson }})
    </script>
    <script>
        $(document).ready(function () {
           $('select').material_select();
        });
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/retrain.html

```
{% extends "base.html" %}
{% block title %}Retrain{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Retrain</h3>
        <form action="{{ url_for('tasks', task='retrain') }}" method="post">
        <input name="model_name" type="hidden" value="{{ model_name }}">
        <div class="section">
            <h5>Training Parameters</h5>
            <div class="row">
                <div class="input-field col s4">
                    <input id="learning_rate" class="validate"
                       type="number" name="learning_rate"
                       min="0.0000001"
                       max="1"
                       step="any"
                       required>
                    <label for="learning_rate">Learning Rate</label>
                </div>
                <div class="input-field col s4">
                    <input id="num_epochs"
                           name="num_epochs"
                           class="validate"
                           type="number"
                           min="1"
                           step="1">
                    <label for="num_epochs">Number of Epochs</label>
                </div>
                <div class="input-field col s4">
                    <input id="checkpoint_epochs"
                           name="checkpoint_epochs"
                           class="validate"
                           type="number"
                           min="1"
                           step="1">
                    <label for="checkpoint_epochs">Checkpoint Epochs</label>
                </div>
            </div>
        </div>
        <div class="row">
            <button class="btn waves-effect waves-light right" type="submit" name="action">Retrain
                <i class="material-icons right">send</i>
            </button>
        </div>
        </form>
    </div>
{% endblock %}
{% block extra_script %}
    <script>
        $(document).ready(function () {
           $('select').material_select();
        });
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/tasks.html

```
{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Running Tasks</h3>
        <ul class="collection black-text">
        {% if not running_tasks %}
            <li class="collection-item">No task is running.</li>
        {% else %}
            {% for running_task in running_tasks %}
                <li class="collection-item row">
                    <ul>
                        <li class="col s3">{{ running_task }}</li>
                        <li class="col s3 right">
                            <form method="post" action="{{ url_for('terminate', task=running_task) }}">
                                <button type="submit" class="waves-effect waves-light btn red right">
                                     Terminate</button>
                            </form>
                        </li>
                    </ul>
                </li>
            {% endfor %}
        {% endif %}
        </ul>
    </div>
{% endblock %}
{% block extra_script %}
    <script type="text/javascript" src="{{ url_for('static', filename='js/toast.js') }}"></script>
    <script>
        toast({{ get_flashed_messages()|tojson }})
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/train.html

```
{% extends "base.html" %}
{% block title %}Home{% endblock %}
{% block body %}
    <div class="container white-text">
        <form action="{{ url_for('tasks', task='training') }}" method="post">
        <div class="section">
            <h5>Dataset Preparation</h5>
            <div class="row">
                <div class="input-field col s3">
                    <select id="dataset-select" name="dataset_name" required>
                        <option value="" disabled selected>Select Dataset</option>
                        {% for dataset_name in dataset_list %}
                            <option value="{{ dataset_name }}">{{ dataset_name|capitalize }}</option>
                        {% endfor %}
                    </select>
                    <label for="dataset-select">Dataset</label>
                </div>
                <div class="input-field col s3">
                    <input id="desired_image_width"
                           name="desired_image_width"
                           class="validate"
                           type="number"
                           min="16"
                           step="1"
                           required>
                    <label for="desired_image_width">Desired Image Width</label>
                </div>
                <div class="input-field col s3">
                    <input id="desired_image_height"
                           name="desired_image_height"
                           class="validate"
                           type="number"
                           min="16"
                           step="1"
                           required>
                    <label for="desired_image_height">Desired Image Height</label>
                </div>
                <div class="input-field col s3">
                    <input id="validation_size" name="validation_size" type="number" step="0.01" min="0.01"
                         max="0.5">
                    <label for="validation_size">Validation Size</label>
                </div>
            </div>
        </div>
        <div class="section">
            <h5>Architecture Selection</h5>
            <div class="row">
                <div class="input-field col s12">
                    <select id="architecture-select" name="architecture_name" required>
                        <option value="" disabled selected>Select Architecture</option>
                        {% for network_architecture in network_architectures %}
                            <option value="{{ network_architecture }}">{{ network_architecture }}</option>
                        {% endfor %}
                    </select>
                    <label for="architecture-select">Architecture</label>
                </div>
            </div>
        </div>
        <div class="section">
            <h5>Training Parameters</h5>
            <div class="row">
                <div class="input-field col s3">
                    <select id="loss-select" name="loss" required>
                        <option value="" disabled selected>Select Loss</option>
                        {% for loss in losses %}
                            <option value="{{ loss }}">{{ loss|upper }}</option>
                        {% endfor %}
                    </select>
                    <label for="loss-select">Loss</label>
                </div>
                <div class="input-field col s3">
                    <select id="optimizer-select" name="optimizer" required>
                        <option value="" disabled selected>Select Optimizer</option>
                        {% for optimizer in optimizers %}
                            <option value="{{ optimizer }}">{{ optimizer|capitalize }}</option>
                        {% endfor %}
                    </select>
                    <label for="optimizer-select">Optimizer</label>
                </div>
                <div class="input-field col s3">
                    <input id="learning_rate" class="validate"
                       type="number" name="learning_rate"
                       min="0.0000001"
                       max="1"
                       step="any"
                       required>
                    <label for="learning_rate">Learning Rate</label>
                </div>
                <div class="input-field col s3">
                    <select id="metrics-select" name="metrics" required multiple>
                        <option value="" disabled selected>Select Metrics</option>
                        {% for metric in metrics %}
                            <option value="{{ metric }}">{{ metric|capitalize }}</option>
                        {% endfor %}
                    </select>
                    <label for="metrics-select">Metrics</label>
                </div>
            </div>
            <div class="row">
                <div class="input-field col s3">
                    <input id="num_epochs"
                           name="num_epochs"
                           class="validate"
                           type="number"
                           min="1"
                           step="1">
                    <label for="num_epochs">Number of Epochs</label>
                </div>
                <div class="input-field col s3">
                    <input id="checkpoint_epochs"
                           name="checkpoint_epochs"
                           class="validate"
                           type="number"
                           min="1"
                           step="1">
                    <label for="checkpoint_epochs">Checkpoint Epochs</label>
                </div>
                <div class="input-field col s3">
                    <input id="batch_size"
                           name="batch_size"
                           class="validate"
                           type="number"
                           min="1"
                           step="1">
                    <label for="batch_size">Batch Size</label>
                </div>
            </div>
        </div>
        <div class="row">
            <button class="btn waves-effect waves-light right" type="submit" name="action">Train
                <i class="material-icons right">send</i>
            </button>
        </div>
        </form>
    </div>
{% endblock %}
{% block extra_script %}
    <script>
        $(document).ready(function () {
           $('select').material_select();
        });
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/view˙architecture.html

```
{% extends "base.html" %}
{% block title %}{{ architecture_name }}{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>{{ architecture_name }}</h3>
        <ul class="collection black-text">
        {% for layer in architecture['network'] %}
            <li class="collection-item row">
                <ul>
                {% for layer_parameter in layer %}
                    <li class="col s2">
                        <i>{{ layer_parameter }}</i>
                        <ul>
                            <li>{{ layer[layer_parameter] }}</li>
                        </ul>
                    </li>
                {% endfor %}
                </ul>
            </li>
        {% endfor %}
            <li class="collection-item row">Output Layer: {{ architecture['output_layer'] }}</li>
        </ul>
    </div>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/templates/view˙logs.html

```
{% extends "base.html" %}
{% block title %}Training Log{% endblock %}
{% block body %}
    <div class="container white-text">
        <h3>Logs</h3>
        {% if train_log %}
            <ul class="collapsible">
                <li>
                  <div class="collapsible-header black-text"><i class="material-icons">arrow_drop_down</i>
                       Training</div>
                  <div class="collapsible-body">
                      {% for line in train_log %}
                          <p>{{ line }}</p>
                      {% endfor %}
                  </div>
                </li>
            </ul>
        {% endif %}
        {% if test_log %}
            <ul class="collapsible">
                <li>
                  <div class="collapsible-header black-text"><i class="material-icons">arrow_drop_down</i>
                       Testing</div>
                  <div class="collapsible-body">
                      {% for line in test_log %}
                          <p>{{ line }}</p>
                      {% endfor %}
                  </div>
                </li>
            </ul>
        {% endif %}
    </div>
{% endblock %}
{% block extra_script %}
    <script>
        $(document).ready(function(){
            $('.collapsible').collapsible();
        });
    </script>
{% endblock %}
```

File: Tensorflow-OCR-Trainer/trainer/static/js/delete-file.js

```
$(document).ready(function () {
   $('form[name="deletion"]').submit(function () {
       return confirm("Are you sure you want to delete this?");
   });
});
```

File: Tensorflow-OCR-Trainer/trainer/static/js/network˙architecture˙creation.js

```
var current_layer_index = -1;

$(document).ready(function () {
    $('select').material_select();
});

function create_dynamic_layer_builder(layer_types, padding_types, cell_types, activation_functions){
    function populate(selector, values) {
        $.each(values, function(index, value){
            $(selector).append($('<option>', {"value": value}).text(value))
        });
    }

    function create_selector(key, placeholder_text, values) {
        var placeholder = $('<option>', {"value": ""})
            .prop('disabled', true)
            .prop('selected', true).text(placeholder_text);
        var selector = $('<select>', {"name": key})
            .prop('required', true)
            .append(placeholder);
        populate(selector, values);
        return selector
    }

    function create_int_input_field(input_name) {
        return $('<input>').attr({'type': 'number', 'step': "1", 'min': "1", "name": input_name, "id":
             input_name}).prop('required', true)
    }


    function create_network_layer_param(key, layer_index){
        return "network[" + layer_index + "][" + key + "]"
    }

    var conv2d_params = function(layer_index) {
        return {"Num Filters": create_int_input_field(create_network_layer_param("num_filters", layer_index)
             ),
                "Kernel Size 1": create_int_input_field(create_network_layer_param("kernel_size1",
                     layer_index)),
                "Kernel Size 2": create_int_input_field(create_network_layer_param("kernel_size2",
                     layer_index)).prop('required', false),
                "Stride": create_int_input_field(create_network_layer_param("stride", layer_index)),
                "Padding": create_selector(create_network_layer_param("padding", layer_index), "Select
                     padding", padding_types),
                "Activation": create_selector(create_network_layer_param("activation", layer_index), "Select
                      activation", activation_functions)}
    };

    var maxpool2d_params = function(layer_index) {
        return {"Pool size": create_int_input_field(create_network_layer_param("pool_size", layer_index)),
                "Stride": create_int_input_field(create_network_layer_param("stride", layer_index)),
                "Padding": create_selector(create_network_layer_param("padding", layer_index), "Select
                     padding", padding_types)}
    };

    var mdrnn_params = function(layer_index) {
        return {"Num Hidden": create_int_input_field(create_network_layer_param("num_hidden", layer_index)),
                "Cell Type": create_selector(create_network_layer_param("cell_type", layer_index), "Select
                     cell type", cell_types),
                "Activation": create_selector(create_network_layer_param("activation", layer_index), "Select
                      activation", activation_functions)}
    };

    var birnn_params = function(layer_index) {
        return {"Num Hidden": create_int_input_field(create_network_layer_param("num_hidden", layer_index)),
                "Cell Type": create_selector(create_network_layer_param("cell_type", layer_index), "Select
                     cell type", cell_types),
                "Activation": create_selector(create_network_layer_param("activation", layer_index), "Select
                      activation", activation_functions)}
    };

    var dropout_params = function(layer_index) {
        return {"Keep Prob": $('<input>').attr({'type': 'number',
            'step': "any", "max": "1", "min": "0.000001",  "name": create_network_layer_param("keep_prob",
                 layer_index)}).prop('required', true)
            .attr('placeholder', 'Keep Prob')}
    };

    var layer_params = {
        "conv2d": conv2d_params,
        "max_pool2d": maxpool2d_params,
        "mdrnn": mdrnn_params,
        "birnn": birnn_params,
        "l2_normalize": function(layer_index){},
        "dropout": dropout_params,
        "collapse_to_rnn_dims": function(layer_index){},
        "batch_norm": function(layer_index){}
    };

    $('#add-layer').click(function () {
        current_layer_index++;

        var layer_type_selector_behavior = function () {
            var layer_type = $(this).val();
            var layer_index = $(this).attr('name').match(/(\d+)/g);
            var params_container = $(this).parent().parent().parent().children('.layer_parameters');
            params_container.empty();

            function append_layer_params(input_fields){
                $.each(input_fields, function(key, input_field){
                    var input_field_label = $('<label>', {'for': input_field.attr('id')});
                    input_field_label.text(input_field_label.text() + key);
                    params_container.append(
                        $('<li>', {"class": "input-field col s2"})
                            .append(input_field)
                            .append($(input_field_label))
                    );
                    if (input_field.is('select')){
                        input_field.material_select();
                    }
                });
            }

            append_layer_params(layer_params[layer_type](layer_index));
        };

        var layer_type_selector = create_selector(
            create_network_layer_param("layer_type", current_layer_index),
            "Select layer type",
            layer_types).change(layer_type_selector_behavior);

        var remove_button = $(
            $('<a>', {'class': 'waves-effect waves-light btn red circle'})
                .append($('<i>', {'class': 'material-icons'}).text("remove"))
        ).click(function () {
            current_layer_index--;
            $(this).parent().parent().parent().remove();
        });

        $('#layers').append(
            $('<li>', {"class": "layer collection-item row overflowing"}).append(
                $($('<ul>'))
                    .append($('<ul>', {"class": "layer_parameters"}))
                    .prepend($('<li>', {"class": "input-field col s3"}).append(layer_type_selector))
                    .append($('<li>', {"class": "col s1 right"}).append(remove_button))
                )
        );
        layer_type_selector.material_select();
    });
}

$(document).ready(function () {
   $('#my_form').submit(function(){
       if (current_layer_index <= -1){
           alert("Please add some layers.");
           return false;
       }
   })
});
```

File: Tensorflow-OCR-Trainer/trainer/static/js/script.js

```
$(document).ready(function(){
    $(".button-collapse").sideNav();
});
```

File: Tensorflow-OCR-Trainer/trainer/static/js/toast.js

```
function toast(messages){
    $.each(messages, function(index, message){
        Materialize.toast(message, 3000);
    });
}
```

File: Tensorflow-OCR-Trainer/trainer/static/css/custom.css

```
body {
    display: flex;
    min-height: 100vh;
    flex-direction: column;
    background-color: #014421 !important;
}

main {
    flex: 1 0 auto;
}

.parallax-container {
    flex: 1 0 auto;
}

.up-maroon {
    background-color: #7b1113 !important;
}

.up-green {
    background-color: #014421 !important;
}

.brand-logo{
    font-size: 0.1em;
    padding: 0 auto;
}

.overflowing{
    overflow: visible;
}
```

A..2 Mobile Application

File: Mem2Speech/build.gradle

```
// Top-level build file where you can add
     configuration options common to all sub-
     projects/modules.

buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle
             :3.0.1'


        // NOTE: Do not place your application
             dependencies here; they belong
        // in the individual module build.gradle
             files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

File: Mem2Speech/settings.gradle

```
include ':app'
```

File: Mem2Speech/app/build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "gonzalvo.dpsm.cas.upm.edu.ph.
             mem2speech"
        minSdkVersion 23
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.
             test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('
                 proguard-android.txt'), 'proguard-
                 rules.pro'
        }
    }
}

dependencies {
    compile 'com.android.support:preference-v7:26.1.0
         '
    implementation fileTree(dir: 'libs', include: ['
         *.jar'])
    implementation 'com.android.support:appcompat-v7
         :26.1.0'
    implementation 'com.android.support:support-v4
         :26.1.0'
    implementation 'com.android.support.constraint:
         constraint-layout:1.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.
         test:runner:1.0.1'
    androidTestImplementation 'com.android.support.
         test.espresso:espresso-core:3.0.1'
    implementation 'com.android.support:design:26.1.0
         '
    //noinspection GradleDynamicVersion
    compile 'org.tensorflow:tensorflow-android:+'
}
```

File: Mem2Speech/app/proguard-rules.pro

```
# Add project specific ProGuard rules here.
# You can control the set of applied configuration
     files using the
# proguardFiles setting in build.gradle.
#
# For more details, see
#   http://developer.android.com/guide/developing/
     tools/proguard.html

# If your project uses WebView with JS, uncomment the
      following
# and specify the fully qualified class name to the
     JavaScript interface
# class:
#-keepclassmembers class fqcn.of.javascript.interface
     .for.webview {
#   public *;
#}

# Uncomment this to preserve the line number
     information for
# debugging stack traces.
```

```
#-keepattributes SourceFile,LineNumberTable

# If you keep the line number information, uncomment
     this to
# hide the original source file name.
#-renamesourcefileattribute SourceFile
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/ConvertedTextActivity.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;

import android.content.Intent;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class ConvertedTextActivity extends AppCompatActivity {

    private final int MAX_CHARS_TTS_CAN_READ = 3900;

    private TextView textView;
    private TextToSpeech tts;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_converted_text);

        Intent intent = getIntent();
        final String recognizedText = intent.getStringExtra(ToText.EXTRA_RECOGNIZED_TEXT);

        textView = findViewById(R.id.converted_text);
        textView.setText(recognizedText);

        tts = new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {
                if (status != TextToSpeech.ERROR){
                    tts.setLanguage(Locale.US);
                }
            }
        });
    }

    public void toSpeech(View view){
        final String uId = this.hashCode() + "";
        String text = textView.getText().toString();
        if(text.length() >= MAX_CHARS_TTS_CAN_READ) {
            final List<String> splittedText = splitText(text);
            for(final String queueElement : splittedText) {
                tts.speak(queueElement, TextToSpeech.QUEUE_ADD, null, uId);
            }
        } else {
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, uId);
        }
    }

    private List<String> splitText(String text) {
        final int textLength = text.length();
        final List<String> splittedText = new ArrayList<>();

        int index = 0;
        while (index < textLength) {
            splittedText.add(text.substring(index, Math.min(index + MAX_CHARS_TTS_CAN_READ, textLength)));
            index += MAX_CHARS_TTS_CAN_READ;
        }
        return splittedText;
    }

    @Override
    protected void onPause() {
        if(tts != null){
            tts.stop();
            tts.shutdown();
        }
        super.onPause();
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/MainActivity.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;

import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.provider.MediaStore;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ImageView;

import java.io.FileNotFoundException;
import java.lang.ref.WeakReference;

import gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.canvas.DrawingView;
```

```
public class MainActivity extends AppCompatActivity {

    private DrawingView drawingView;
    private boolean isPen = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        drawingView = findViewById(R.id.drawing);
        PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
    }

    public void switchTool(View view){
        ImageView penButton = view.findViewById(R.id.write);
        drawingView.switchTool();
        if(isPen){
            penButton.setRotation(180);
            isPen = false;
        } else {
            penButton.setRotation(360);
            isPen = true;
        }
    }

    public void clear(View view){
        AlertDialog.Builder builder = new AlertDialog.Builder(view.getContext());
        builder.setMessage("Erase canvas?");
        DialogInterface.OnClickListener dialogClickListener = new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int which) {
                switch(which){
                    case DialogInterface.BUTTON_POSITIVE:
                        drawingView.clearCanvas();
                        break;
                    case DialogInterface.BUTTON_NEGATIVE:
                        break;
                }
            }
        };
        builder.setPositiveButton("Erase", dialogClickListener);
        builder.setNegativeButton("Cancel", dialogClickListener);
        builder.show();
    }

    public void recognize(View view){
        Bitmap bitmap = drawingView.getDrawingCache();
        startToTextTask(bitmap);
    }

    public void selectImage(View view){
        Intent selectImageIntent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.
             EXTERNAL_CONTENT_URI);
        startActivityForResult(selectImageIntent, 0);
    }

    public void viewSettings(View view){
        Intent intent = new Intent(this, SettingsActivity.class);
        startActivity(intent);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT_OK){
            Uri targetUri = data.getData();
            Bitmap selectedImageBitmap;
            try {
                selectedImageBitmap = BitmapFactory.decodeStream(getContentResolver().openInputStream(
                     targetUri));
                startToTextTask(selectedImageBitmap);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }
    }

    private void startToTextTask(Bitmap bitmap) {
        new ToText(new WeakReferenceContextWrapper(new WeakReference<Context>(MainActivity.this))).execute(
             bitmap);
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/SettingsActivity.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class SettingsActivity extends AppCompatActivity {

    public static final String KEY_PREF_MODEL_LIST = "model_preference";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction().replace(
                android.R.id.content,
                new SettingsFragment())
                .commit();
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/SettingsFragment.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;


import android.os.Bundle;
import android.preference.PreferenceFragment;

public class SettingsFragment extends PreferenceFragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }

}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/ToText.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;

import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.os.AsyncTask;
import android.preference.PreferenceManager;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.lang.ref.WeakReference;
import java.util.ArrayList;
import java.util.List;

import gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer.OfflineRecognizer;
import gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer.Recognizer;
import gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer.RecognizerConfig;

import static gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.SettingsActivity.KEY_PREF_MODEL_LIST;

class ToText extends AsyncTask<Object, Void, Void> {

    static final String EXTRA_RECOGNIZED_TEXT = "ph.edu.upm.cas.dpsm.gonzalvo.mem2speech.recognizedtext";

    private final WeakReference<Context> contextReference;
    private ProgressDialog progressDialog;
    private String convertedText;

    ToText(WeakReferenceContextWrapper weakReferenceContextWrapper) {
        this.contextReference = weakReferenceContextWrapper.getContextReference();
        this.convertedText = "";
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        createDialog();
    }

    private void createDialog() {
        this.progressDialog = new ProgressDialog(getContext());
        this.progressDialog.setCancelable(false);
        this.progressDialog.setCanceledOnTouchOutside(false);
        this.progressDialog.setMessage("Converting to text...");
        this.progressDialog.show();
    }

    @Override
    protected Void doInBackground(Object... params) {
        Bitmap bitmap = (Bitmap) params[0];
        String[] charset = readCharsetFromFile();
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(getContext());
        String modelPref = sharedPref.getString(KEY_PREF_MODEL_LIST, "");
        String modelPath = "models/" + modelPref;
        RecognizerConfig config = new RecognizerConfig(
                charset,
                modelPath + "/image_config.json",
                modelPath + "/serving_model_config.json"
        );
        Recognizer recognizer = new OfflineRecognizer(
                config,
                getAssetManager(),
                modelPath + "/graph.pb"
        );
        convertedText = recognizer.recognizeHandwritingFrom(bitmap);
        return null;
    }

    private AssetManager getAssetManager() {
        return getContext().getAssets();
    }
    private String[] readCharsetFromFile() {
        List<String> charset = new ArrayList<>();
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(getAssetManager().open("charset.txt
                  ")));
            String line;

            while((line = br.readLine()) != null){
                charset.add(line);
            }
            charset.add("");
            br.close();
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
        String[] charsetArr = new String[charset.size()];
        charsetArr = charset.toArray(charsetArr);
        return charsetArr;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        this.progressDialog.cancel();
        Intent intent = new Intent(getContext(), ConvertedTextActivity.class);
        intent.putExtra(EXTRA_RECOGNIZED_TEXT, convertedText);
        getContext().startActivity(intent);
    }

    private Context getContext() {
        return contextReference.get();
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/WeakReferenceContextWrapper.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech;

import android.content.Context;

import java.lang.ref.WeakReference;

class WeakReferenceContextWrapper {
    private final WeakReference<Context> contextReference;

    public WeakReferenceContextWrapper(WeakReference<Context> contextReference) {
        this.contextReference = contextReference;
    }

    public WeakReference<Context> getContextReference() {
        return contextReference;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/canvas/Color˙WidthDTO.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.canvas;

import android.graphics.Path;

import java.util.HashMap;
import java.util.Map;

class Color_WidthDTO {
    private final Map<Path, Integer> colorsMap;
    private final Map<Path, Float> widthMap;

    Color_WidthDTO() {
        this.colorsMap = new HashMap<>();
        this.widthMap = new HashMap<>();
    }

    Integer getColor(Path path) {
        return this.colorsMap.get(path);
    }

    Float getStrokeWidth(Path path) {
        return this.widthMap.get(path);
    }

    void putPenProperty(Path path, int selectedColor, float strokeWidth) {
        this.colorsMap.put(path, selectedColor);
        this.widthMap.put(path, strokeWidth);
    }

    void clear() {
        this.colorsMap.clear();
        this.widthMap.clear();
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/canvas/DrawingView.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.canvas;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
```

```java
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

import java.util.ArrayList;

public class DrawingView extends View {
    private Path path;
    private final Paint paint;
    private float x, y;
    private final float STROKE_WIDTH = 10f;
    private float strokeWidth;
    private int selectedColor = Color.BLACK;

    private final Color_WidthDTO color_widthDTO;
    private final ArrayList<Path> paths;

    public DrawingView(final Context context, final AttributeSet attributes) {
        super(context, attributes);

        this.strokeWidth = STROKE_WIDTH;

        this.color_widthDTO = new Color_WidthDTO();
        this.paths = new ArrayList<>();

        this.path = new Path();
        this.paint = new Paint();
        this.paint.setAntiAlias(true);
        this.paint.setColor(Color.BLACK);
        this.paint.setStyle(Paint.Style.STROKE);
        this.paint.setStrokeJoin(Paint.Join.ROUND);
        this.paint.setStrokeWidth(5f);
        this.setDrawingCacheEnabled(true);
        this.setDrawingCacheQuality(View.DRAWING_CACHE_QUALITY_HIGH);
    }

    @Override
    protected void onSizeChanged(int width, int height, int oldWidth, int oldHeight) {
        super.onSizeChanged(width, height, oldWidth, oldHeight);

        final Bitmap bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
        new Canvas(bitmap);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        for(final Path path : paths){
            this.paint.setColor(color_widthDTO.getColor(path));
            this.paint.setStrokeWidth(color_widthDTO.getStrokeWidth(path));
            canvas.drawPath(path, this.paint);
        }
        this.paint.setColor(this.selectedColor);
        this.paint.setStrokeWidth(this.strokeWidth);
        canvas.drawPath(this.path, this.paint);
    }

    private void startTouch(final float x, final float y) {
        this.path.moveTo(x, y);
        this.x = x;
        this.y = y;
    }

    private void moveTouch(final float x, final float y) {
        final float dx = Math.abs(x - this.x);
        final float dy = Math.abs(y - this.y);

        float TOLERANCE = 2;
        if(dx >= TOLERANCE || dy >= TOLERANCE) {
            this.path.quadTo(this.x, this.y, (x + this.x)/2, (y + this.y)/2);
            this.x = x;
            this.y = y;
        }
    }

    private void upTouch() {
        savePath();

        this.path.setLastPoint(this.x, this.y);
        this.x+=5;
        savePath();
        this.x-=5;

        this.path.setLastPoint(this.x, this.y);
        this.y--;
        savePath();
        this.path = new Path();
        this.y++;

        this.x-=5;
        this.path.setLastPoint(this.x, this.y);
        this.x+=5;
        savePath();

        this.path.reset();
    }
    private void savePath() {
        this.path.lineTo(this.x, this.y);
        this.paths.add(this.path);
        this.color_widthDTO.putPenProperty(this.path, this.selectedColor, this.strokeWidth);
        this.path = new Path();
    }

    public void clearCanvas() {
        this.color_widthDTO.clear();
        this.paths.clear();
        this.path.reset();
        this.destroyDrawingCache();
        invalidate();
    }

    public void switchTool() {
        if(penIsActive()) {
            switchToEraser();
        } else {
            switchToPen();
        }
        this.destroyDrawingCache();
    }

    private boolean penIsActive() {
        return this.selectedColor == Color.BLACK;
    }

    private void switchToPen() {
        this.selectedColor = Color.BLACK;
        this.strokeWidth = STROKE_WIDTH;
    }

    private void switchToEraser() {
        this.selectedColor = Color.WHITE;
        this.strokeWidth = 25f;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        final float x = event.getX();
        final float y = event.getY();
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                startTouch(x, y);
                break;
            case MotionEvent.ACTION_MOVE:
                moveTouch(x, y);
                break;
            case MotionEvent.ACTION_UP:
                upTouch();
                break;
        }
        invalidate();
        return true;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/GraphComponents.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

class GraphComponents {
    private final OutputNodes outputNodes;
    private final InputNodes inputNodes;

    GraphComponents(String jsonString) {
        inputNodes = new InputNodes();
        outputNodes = new OutputNodes();
        try {
            JSONObject obj = new JSONObject(jsonString);
            JSONArray jsonInputNodes = obj.getJSONArray("input_nodes");
            for (int i = 0; i < jsonInputNodes.length(); i++){
                JSONObject jsonInputNode = jsonInputNodes.getJSONObject(i);
                JSONArray jsonInputShape = jsonInputNode.getJSONArray("input_shape");
                long[] inputShape = new long[jsonInputShape.length()];
                for (int j = 0; j < jsonInputShape.length(); j++)
                    inputShape[j] = jsonInputShape.getLong(j);
                inputNodes.addInputNode(jsonInputNode.getString("input_name"), inputShape);
            }
            JSONArray jsonOutputNames = obj.getJSONArray("output_names");
            for (int i = 0; i < jsonOutputNames.length(); i++)
                outputNodes.addOutputNode(jsonOutputNames.getString(i), 120);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    InputNodes getInputNodes() {
        return inputNodes;
    }

    OutputNodes getOutputNodes() {
        return outputNodes;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/ImageConfig.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

import org.json.JSONException;
import org.json.JSONObject;

public class ImageConfig {
    private int imageWidth;
    private int imageHeight;

    ImageConfig(String imageConfigJsonString) {
        try {
            JSONObject obj = new JSONObject(imageConfigJsonString);
            this.imageHeight = obj.getInt("image_height");
            this.imageWidth = obj.getInt("image_width");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    public int getImageWidth() {
        return imageWidth;
    }

    public int getImageHeight() {
        return imageHeight;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/InferenceInterface.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

import android.content.res.AssetManager;

import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

import java.util.ArrayList;
import java.util.List;

class InferenceInterface {

    private final TensorFlowInferenceInterface inferenceInterface;

    InferenceInterface(AssetManager assetManager, String modelFilename){
        this.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);
    }

    List<int[]> runInference(InputNode[] inputNodes, OutputNodes outputNodes){
        List<int[]> outputs = new ArrayList<>();
        for (InputNode inputNode : inputNodes){
            inferenceInterface.feed(inputNode.getInputName(), inputNode.getInputValue(), inputNode.
                 getInputShape());
        }
        inferenceInterface.run(outputNodes.getOutputNodeNames());
        for (String outputNodeName : outputNodes.getOutputNodeNames()){
            int[] output = new int[outputNodes.getShape(outputNodeName)];
            inferenceInterface.fetch(outputNodeName, output);
            outputs.add(output);
        }
        return outputs;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/InputNode.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

class InputNode {
    private final String inputName;
    private final float[] inputValue;
    private final long[] inputShape;

    InputNode(String inputName, float[] inputValue, long[] inputShape) {
        this.inputName = inputName;
        this.inputValue = inputValue;
        this.inputShape = inputShape;
    }

    String getInputName() {
        return inputName;
    }

    float[] getInputValue() {
        return inputValue;
    }

    long[] getInputShape() {
        return inputShape;
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/InputNodes.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;


import java.util.HashMap;
import java.util.Map;
```

```java
class InputNodes {
    private final Map<String, long[]> inputNameShapeMap;

    InputNodes() {
        inputNameShapeMap = new HashMap<>();
    }

    public long[] getInputShape(String inputName){
        return inputNameShapeMap.get(inputName);
    }

    void addInputNode(String inputName, long[] inputShape){
        inputNameShapeMap.put(inputName, inputShape);
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/OfflineRecognizer.java

```java
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.graphics.Canvas;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class OfflineRecognizer implements Recognizer {
    private final RecognizerConfig config;

    private final InferenceInterface inferenceInterface;
    private final AssetManager assetManager;

    public OfflineRecognizer(RecognizerConfig config,
                             AssetManager assetManager,
                             String modelFilename) {
        this.config = config;
        this.assetManager = assetManager;
        inferenceInterface = new InferenceInterface(this.assetManager, modelFilename);
    }

    @Override
    public String recognizeHandwritingFrom(Bitmap bitmap) {
        float[] features = getFeaturesFrom(bitmap);
        String servingModelConfig = loadJSONFrom(config.getServingModelConfigFilename());
        GraphComponents graphComponents = new GraphComponents(servingModelConfig);
        InputNodes inputNodes = graphComponents.getInputNodes();
        InputNode inputNode = new InputNode(
                "features",
                features,
                inputNodes.getInputShape("features")
        );
        OutputNodes outputNodes = graphComponents.getOutputNodes();
        List<int[]> outputs = inferenceInterface.runInference(new InputNode[]{inputNode}, outputNodes);
        System.out.println(decode(outputs.get(0)));
        return decode(outputs.get(0));
    }

    private String loadJSONFrom(String jsonFilename) {
        String json;
        try {
            InputStream is = assetManager.open(jsonFilename);
            int size = is.available();
            byte[] buffer = new byte[size];
            is.read(buffer);
            is.close();
            json = new String(buffer, "UTF-8");
        } catch (IOException ex) {
            ex.printStackTrace();
            return null;
        }
        return json;
    }

    private Bitmap resize(Bitmap image, int maxHeight, int maxWidth) {
        if (maxHeight > 0 && maxWidth > 0) {
            int width = image.getWidth();
            int height = image.getHeight();
            float ratioBitmap = (float) width / (float) height;
            float ratioMax = (float) maxWidth / (float) maxHeight;

            int finalWidth = maxWidth;
            int finalHeight = maxHeight;
            if (ratioMax > ratioBitmap) {
                finalWidth = (int) ((float)maxHeight * ratioBitmap);
            } else {
                finalHeight = (int) ((float)maxWidth / ratioBitmap);
            }
            image = Bitmap.createScaledBitmap(image, finalWidth, finalHeight, true);
            if (finalWidth == maxWidth) {
                image = padBottom(image, maxHeight);
            } else {
                image = padRight(image, maxWidth);
            }
            return image;
        } else {
            return image;
        }
    }

    private Bitmap padRight(Bitmap bitmap, int maxWidth) {
        Bitmap outputImage = Bitmap.createBitmap(maxWidth, bitmap.getHeight(), Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(outputImage);
        canvas.drawARGB(255, 255, 255, 255);
        canvas.drawBitmap(bitmap, 0, 0, null);
        return outputImage;
    }

    private Bitmap padBottom(Bitmap bitmap, int maxHeight) {
        Bitmap outputImage = Bitmap.createBitmap(bitmap.getWidth(), maxHeight, Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(outputImage);
        canvas.drawARGB(255, 255, 255, 255);
        canvas.drawBitmap(bitmap, 0, 0, null);
        return outputImage;
    }

    private float[] getFeaturesFrom(Bitmap bitmap) {
        String imageConfigJsonString = loadJSONFrom(config.getImageConfigFile());
        ImageConfig imageConfig = new ImageConfig(imageConfigJsonString);
        bitmap = resize(bitmap, imageConfig.getImageHeight(), imageConfig.getImageWidth());
        int[] pixels = getPixelValuesFrom(bitmap);
        float[] floatValues = new float[pixels.length];
        for (int i = 0; i < pixels.length - 1; ++i) {
            final int pixel = pixels[i];
            int b = pixel & 0xff;
            floatValues[i] = (float)((0xff - b)/255.0);
        }
        return floatValues;
    }

    private int[] getPixelValuesFrom(Bitmap bitmap) {
        int[] intValues = new int[bitmap.getWidth() * bitmap.getHeight()];
        bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(), bitmap.getHeight());
        return intValues;
    }

    private String decode(int[] result) {
        String[] charset = config.getCharset();
        StringBuilder sb = new StringBuilder();
        for(int encodedCharacter : result){
            sb.append(charset[encodedCharacter]);
        }
        return sb.toString().replaceAll("\\|", " ");
    }

}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/OutputNodes.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

import java.util.HashMap;
import java.util.Map;

class OutputNodes {
    private final Map<String, Integer> outputNodes;

    OutputNodes() {
        this.outputNodes = new HashMap<>();
    }

    void addOutputNode(String outputNodeName, int shape) {
        outputNodes.put(outputNodeName, shape);
    }

    String[] getOutputNodeNames(){
        return outputNodes.keySet().toArray(new String[outputNodes.size()]);
    }

    int getShape(String outputNodeName) {
        return outputNodes.get(outputNodeName);
    }
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/Recognizer.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;


import android.graphics.Bitmap;

public interface Recognizer {
    String recognizeHandwritingFrom(Bitmap bitmap);
}
```

File: Mem2Speech/app/src/main/java/gonzalvo/dpsm/cas/upm/edu/ph/mem2speech/recognizer/RecognizerConfig.java

```
package gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.recognizer;

public class RecognizerConfig {
    private final String[] charset;
    private final String servingModelConfigFile;
    private final String imageConfigFile;

    public RecognizerConfig(String[] charset, String imageConfigFile, String servingModelConfigFile) {
        this.imageConfigFile = imageConfigFile;
        this.charset = charset;
        this.servingModelConfigFile = servingModelConfigFile;
    }

    String[] getCharset() {
        return charset;
    }

    String getImageConfigFile() {
        return imageConfigFile;
    }

    String getServingModelConfigFilename() {
        return servingModelConfigFile;
    }
}
```

File: Mem2Speech/app/src/main/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/
     apk/res/android"
    package="gonzalvo.dpsm.cas.upm.edu.ph.mem2speech"
         >

    <uses-permission android:name="android.permission
         .READ_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:extractNativeLibs="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.
                     action.MAIN" />

                <category android:name="android.
                     intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".
             ConvertedTextActivity" />
        <activity android:name=".SettingsActivity"></
             activity>
    </application>

</manifest>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙collections˙black˙24dp.xml

```
<vector android:height="24dp" android:viewportHeight=
     "24.0"
    android:viewportWidth="24.0" android:width="24dp"
         xmlns:android="http://schemas.android.com/
        apk/res/android">
    <path android:fillColor="#FF000000"
        android:pathData="M22,16L22,4c0,-1.1
        -0.9,-2 -2,-2L8,2c-1.1,0 -2,0.9 -2,2v12c0
        ,1.1 0.9,2 2,2h12c1.1,0 2,-0.9 2,-2zM11,12
        l2.03,2.71L16,11l4,5L8,16l3,-4zM2,6v14c0
        ,1.1 0.9,2 2,2h14v-2L4,20L4,6L2,6z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙edit˙black˙24dp.xml

```
<vector android:height="24dp" android:viewportHeight=
     "24.0"
    android:viewportWidth="24.0" android:width="24dp"
         xmlns:android="http://schemas.android.com/
        apk/res/android">
    <path android:fillColor="#FF000000"
        android:pathData="M3,17.25V21h3.75L17
        .81,9.94l-3.75,-3.75L3,17.25zM20.71,7.04c0
        .39,-0.39 0.39,-1.02 0,-1.41l-2.34,-2.34c
        -0.39,-0.39 -1.02,-0.39 -1.41,0l-1.83,1.83
        3.75,3.75 1.83,-1.83z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙erase˙24dp.xml

```
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M19,6.41L17.59,5 12,10.59
             6.41,5 5,6.41 10.59,12 5,17.59 6.41,19
             12,13.41 17.59,19 19,17.59 13.41,12z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙launcher˙background.xml

```
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportHeight="108"
    android:viewportWidth="108">
    <path
        android:fillColor="#26A69A"
        android:pathData="M0,0h108v108h-108z" />
    <path
        android:fillColor="#00000000"
        android:pathData="M9,0L9,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,0L19,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M29,0L29,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M39,0L39,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,0L49,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,0L59,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,0L69,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,0L79,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M89,0L89,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M99,0L99,108"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,9L108,9"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,19L108,19"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,29L108,29"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,39L108,39"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,49L108,49"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,59L108,59"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,69L108,69"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,79L108,79"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,89L108,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M0,99L108,99"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,29L89,29"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,39L89,39"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,49L89,49"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,59L89,59"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,69L89,69"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M19,79L89,79"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M29,19L29,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M39,19L39,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M49,19L49,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M59,19L59,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M69,19L69,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
    <path
        android:fillColor="#00000000"
        android:pathData="M79,19L79,89"
        android:strokeColor="#33FFFFFF"
        android:strokeWidth="0.8" />
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙recognize˙24dp.xml

```
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M5,17v2h14v-2L5,17zM9
             .5,12.8h5l0.9,2.2h2.1L12.75,4h-1.5L6
             .5,15h2.1l0.9,-2.2zM12,5.98L13.87,11h
             -3.74L12,5.98z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙settings˙black˙24dp.xml

```
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M19.43,12.98c0.04,-0.32
             0.07,-0.64 0.07,-0.98s-0.03,-0.66
             -0.07,-0.98l2.11,-1.65c0.19,-0.15
             0.24,-0.42 0.12,-0.64l-2,-3.46c
             -0.12,-0.22 -0.39,-0.3 -0.61,-0.22l
             -2.49,1c-0.52,-0.4 -1.08,-0.73
             -1.69,-0.98l-0.38,-2.65C14.46,2.18
             14.25,2 14,2h-4c-0.25,0 -0.46,0.18
             -0.49,0.42l-0.38,2.65c-0.61,0.25
             -1.17,0.59 -1.69,0.98l-2.49,-1c
             -0.23,-0.09 -0.49,0 -0.61,0.22l-2,3.46c
             -0.13,0.22 -0.07,0.49 0.12,0.64l2
             .11,1.65c-0.04,0.32 -0.07,0.65
             -0.07,0.98s0.03,0.66 0.07,0.98l
             -2.11,1.65c-0.19,0.15 -0.24,0.42
             -0.12,0.64l2,3.46c0.12,0.22 0.39,0.3
             0.61,0.22l2.49,-1c0.52,0.4 1.08,0.73
             1.69,0.98l0.38,2.65c0.03,0.24 0.24,0.42
              0.49,0.42h4c0.25,0 0.46,-0.18
             0.49,-0.42l0.38,-2.65c0.61,-0.25
             1.17,-0.59 1.69,-0.98l2.49,1c0.23,0.09
             0.49,0 0.61,-0.22l2,-3.46c0.12,-0.22
             0.07,-0.49 -0.12,-0.64l-2.11,-1.65zM12
             ,15.5c-1.93,0 -3.5,-1.57 -3.5,-3.5s1
             .57,-3.5 3.5,-3.5 3.5,1.57 3.5,3.5
             -1.57,3.5 -3.5,3.5z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable/ic˙speech˙24dp.xml

```
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M7,7.07L8.43,8.5c0.91,-0.91
              2.18,-1.48 3.57,-1.48s2.66,0.57
             3.57,1.48L17,7.07C15.72,5.79 13.95,5
             12,5s-3.72,0.79 -5,2.07zM12,1C8.98,1
             6.24,2.23 4.25,4.21l1.41,1.41C7.28,4
             9.53,3 12,3s4.72,1 6.34,2.62l1.41,-1.41
             C17.76,2.23 15.02,1 12,1zM14.86,10.01L9
             .14,10C8.51,10 8,10.51 8,11.14v9.71c0
             ,0.63 0.51,1.14 1.14,1.14h5.71c0.63,0
             1.14,-0.51 1.14,-1.14v-9.71c0.01,-0.63
             -0.5,-1.13 -1.13,-1.13zM15,20L9,20v-8
             h6v8z"/>
</vector>
```

File: Mem2Speech/app/src/main/res/drawable-v24/ic˙launcher˙foreground.xml

```
<vector xmlns:android="http://schemas.android.com/apk
     /res/android"
    xmlns:aapt="http://schemas.android.com/aapt"
    android:width="108dp"
    android:height="108dp"
    android:viewportHeight="108"
    android:viewportWidth="108">
    <path
        android:fillType="evenOdd"
        android:pathData="M32,64C32,64 38.39,52.99
             44.13,50.95C51.37,48.37 70.14,49.57
             70.14,49.57L108.26,87.69L108,109.01L75
             .97,107.97L32,64Z"
        android:strokeColor="#00000000"
        android:strokeWidth="1">
        <aapt:attr name="android:fillColor">
            <gradient
                android:endX="78.5885"
                android:endY="90.9159"
                android:startX="48.7653"
                android:startY="61.0927"
                android:type="linear">
                <item
                    android:color="#44000000"
                    android:offset="0.0" />
                <item
                    android:color="#00000000"
                    android:offset="1.0" />
            </gradient>
        </aapt:attr>
    </path>
    <path
        android:fillColor="#FFFFFF"
        android:fillType="nonZero"
        android:pathData="M66.94,46.02L66.94,46.02C72
             .44,50.07 76,56.61 76,64L32,64C32,56.61
              35.56,50.11 40.98,46.06L36.18,41.19C35
             .45,40.45 35.45,39.3 36.18,38.56C36
             .91,37.81 38.05,37.81 38.78,38.56L44
             .25,44.05C47.18,42.57 50.48,41.71
             54,41.71C57.48,41.71 60.78,42.57
             63.68,44.05L69.11,38.56C69.84,37.81
             70.98,37.81 71.71,38.56C72.44,39.3
             72.44,40.45 71.71,41.19L66.94,46.02ZM62
             .94,56.92C64.08,56.92 65,56.01 65,54.88
             C65,53.76 64.08,52.85 62.94,52.85C61
             .8,52.85 60.88,53.76 60.88,54.88C60
             .88,56.01 61.8,56.92 62.94,56.92ZM45
             .06,56.92C46.2,56.92 47.13,56.01
             47.13,54.88C47.13,53.76 46.2,52.85
             45.06,52.85C43.92,52.85 43,53.76
             43,54.88C43,56.01 43.92,56.92
             45.06,56.92Z"
        android:strokeColor="#00000000"
        android:strokeWidth="1" />
</vector>
```

File: Mem2Speech/app/src/main/res/layout/activity˙converted˙text.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
     xmlns:android="http://schemas.android.com/apk/
     res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-
         auto"
    tools:contextReference="gonzalvo.dpsm.cas.upm.edu
         .ph.mem2speech.ConvertedTextActivity">

    <TextView
        android:id="@+id/converted_text"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_margin="16dp" />

    <android.support.design.widget.
         FloatingActionButton
        android:id="@+id/to_speech"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|start"
        android:layout_margin="16dp"
        android:src="@drawable/ic_speech_24dp"
        android:onClick="toSpeech"
        app:backgroundTint="#FFFFFF"
        app:layout_anchorGravity="bottom|left|start"
             />

</android.support.design.widget.CoordinatorLayout>
```

File: Mem2Speech/app/src/main/res/layout/activity˙main.xml

```
<android.support.design.widget.CoordinatorLayout
     xmlns:android="http://schemas.android.com/apk/
     res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-
         auto"
    tools:contextReference="gonzalvo.dpsm.cas.upm.edu
         .ph.mem2speech.MainActivity">

    <gonzalvo.dpsm.cas.upm.edu.ph.mem2speech.canvas.
         DrawingView
        android:id="@+id/drawing"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#FFFFFFFF" />

    <ImageView
        android:id="@+id/clear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|start"
        android:layout_margin="16dp"
        android:src="@drawable/ic_erase_24dp"
        android:onClick="clear"
        app:backgroundTint="#FFFFFF"
        app:layout_anchorGravity="top|left|start"
        android:contentDescription="@string/
             clear_canvas_description" />

    <ImageView
        android:id="@+id/about"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top|end"
        android:layout_margin="16dp"
        android:src="@drawable/ic_settings_black_24dp
             "
        app:backgroundTint="#FFFFFF"
        app:layout_anchorGravity="top|right|end"
        android:contentDescription="@string/
             settings_description"
        android:onClick="viewSettings"/>

    <ImageView
        android:id="@+id/view_gallery"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|start"
        android:layout_margin="16dp"
        android:src="@drawable/
             ic_collections_black_24dp"
        app:backgroundTint="#FFFFFF"
        app:layout_anchorGravity="bottom|left|start"
        android:contentDescription="@string/
             image_from_gallery_description"
        android:onClick="selectImage"/>

    <ImageView
        android:id="@+id/write"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:src="@drawable/ic_edit_black_24dp"
        android:onClick="switchTool"
        app:backgroundTint="#FFFFFF"
        app:layout_anchorGravity="bottom|right|end"
        android:contentDescription="@string/
             pen_description" />

    <android.support.design.widget.
         FloatingActionButton
        android:id="@+id/recognize"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:layout_margin="16dp"
        android:src="@drawable/ic_recognize_24dp"
        android:onClick="recognize"
        app:backgroundTint="#FFFFFF"
        app:layout_anchor="@+id/drawing"
        app:layout_anchorGravity="bottom|center"
        android:contentDescription="@string/
             recognize_description"/>

</android.support.design.widget.CoordinatorLayout>
```

File: Mem2Speech/app/src/main/res/mipmap-anydpi-v26/ic˙launcher.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.
     com/apk/res/android">
    <background android:drawable="@drawable/
         ic_launcher_background" />
    <foreground android:drawable="@drawable/
         ic_launcher_foreground" />
</adaptive-icon>
```

File: Mem2Speech/app/src/main/res/mipmap-anydpi-v26/ic˙launcher˙round.xml

```
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.
     com/apk/res/android">
    <background android:drawable="@drawable/
         ic_launcher_background" />
    <foreground android:drawable="@drawable/
         ic_launcher_foreground" />
</adaptive-icon>
```

File: Mem2Speech/app/src/main/res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="Model">
        <item>test_model</item>
        <item>cnn_bilstm_ctc_model</item>
        <item>overfitting_model</item>
        <item>three_layer_cnn_birnn_ctc_model</item>
    </string-array>
    <string-array name="modelAlias">
        <item>Test Model</item>
        <item>CNN BiLSTM CTC Model</item>
        <item>Overfitting Model</item>
        <item>Three Layer CNN BiRNN CTC Model</item>
    </string-array>
</resources>
```

File: Mem2Speech/app/src/main/res/values/attrs.xml

```
<resources>

    <!-- Declare custom theme attributes that allow
         changing which styles are
         used for button bars depending on the API
              level.
         ?android:attr/buttonBarStyle is new as of
              API 11 so this is
         necessary to support previous API levels. --
              >
    <declare-styleable name="ButtonBarContainerTheme"
         >
        <attr name="metaButtonBarStyle" format="
             reference" />
        <attr name="metaButtonBarButtonStyle" format=
             "reference" />
    </declare-styleable>

</resources>
```

File: Mem2Speech/app/src/main/res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="black_overlay">#66000000</color>
</resources>
```

File: Mem2Speech/app/src/main/res/values/strings.xml

```
<resources>
    <string name="app_name">Mem2Speech</string>
    <string name="pen_description">Tap to toggle pen/
         eraser.</string>
    <string name="image_from_gallery_description">tap
          to recognize text from image from your
         gallery</string>
    <string name="settings_description">tap for
         settings</string>
    <string name="clear_canvas_description">tap to
         clear canvas</string>
    <string name="recognize_description">Tap to
         recognize text on canvas</string>
    <string name="model_selection_title">Active Model
         </string>

<!-- TODO: Remove or change this placeholder text -->
    <string name="hello_blank_fragment">Hello blank
         fragment</string>
</resources>
```

File: Mem2Speech/app/src/main/res/values/styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.
         Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary
             </item>
        <item name="colorPrimaryDark">@color/
             colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</
             item>
        <item name="preferenceTheme">@style/
             PreferenceThemeOverlay</item>
    </style>

   <style name="FullscreenTheme" parent="AppTheme">
        <item name="android:actionBarStyle">@style/
             FullscreenActionBarStyle</item>
        <item name="android:windowActionBarOverlay">
             true</item>
        <item name="android:windowBackground">@null</
             item>
        <item name="metaButtonBarStyle">?android:attr
             /buttonBarStyle</item>
        <item name="metaButtonBarButtonStyle">?
             android:attr/buttonBarButtonStyle</item
             >
    </style>

    <style name="FullscreenActionBarStyle" parent="
         Widget.AppCompat.ActionBar">
        <item name="android:background">@color/
             black_overlay</item>
    </style>

</resources>
```

File: Mem2Speech/app/src/main/res/xml/preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.
      android.com/apk/res/android">
    <ListPreference
        android:key="model_preference"
        android:title="@string/model_selection_title"
        android:entries="@array/modelAlias"
        android:entryValues="@array/Model"
        android:summary="Select model to be used by
             the application"
        android:defaultValue="test_model"/>
</PreferenceScreen>
```

XI. Acknowledgement

First and foremost, I want to thank my adviser, professor Marvin John Ignacio for guiding me throughout this Special Problem. He helped me break down the complex concepts of deep learning into simpler ones making it easier for me to understand and apply. I want to thank him for being considerate and patient with me despite the seemingly insurmountable roadblocks I have encountered, and helping me overcome.

I want to thank the Stack Overflow community, the Github community who answered my queries making it possible for me to fill the gaps in my code.

I want to thank my professors in the university for imparting a lot of knowledge that I'll be able to use not just within the four corners of each classroom but also in the real world.

Lastly, I want to thank my family for being really patient with me given that I got delayed for a year and for being supportive despite that, my friends for making my stay in college a fun and memorable one, and my significant other for being ever so sweet, patient, and kind towards me. I love you :'>