Faculty of Engineering

Ain Shams University

2021-2022

Neural Networks and Their Applications

CSE 616

PG2015

Project Report

Title: ***Wakeup Word Detection Implementation***

Made by:

2100966 حمدي أسامه محمد السيد فرج

Mechatronics PG Student

Wakeup Word Detection Project Report

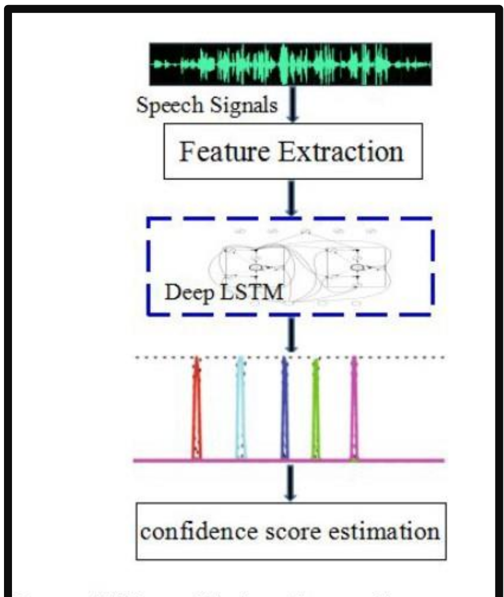
Made by: Hamdy Osama (2100966)

Git-Hub Project Link: https://github.com/hamdy-cryptic/CSE-616-WUW-Detector-Project

Video showing Project Operation: Wakeup Word Detector Demo, By Hamdy Osama

Overview of the Problem

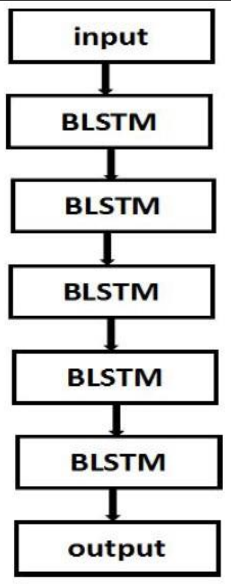
Wake Word Detection (also known as Hot word detection) is a technique mainly used in Chatbots to wake them. 'Okay Google', 'Siri' and 'Alexa' are the wake words used by Google assistant, Apple, and Amazon's Alexa respectively. The main function of any chatbot is to take an audio input by user and answer what it is designed for! Let's take an example of Google Assistant, as soon as it detects the wake words, it starts to take the audio input by user (and sends it to cloud for speech synthesis) and then gives the answer. That's why the Wake word prevents us to send everything on cloud for speech synthesis. As speech synthesis takes too much computational power, wakeup word saves this by only sending the audio fragment which user wants it to synthesize. In short, Wakeup word is a point in time from when we must start doing speech synthesis. In this project, we explore how can we detect that a wakeup word is being invoked. This work is inspired by the previous work of Pritish Mishra and Shilei Zhang.



Proposed Wake-up-Word spotting paradigm.

Detecting the Wakeup Word: Paper Solution

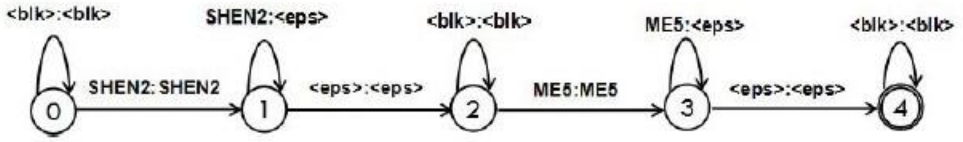
The authors use Bidirectional Deep Long Short-Term Memory (BDLSTM) to transform input features of the audio into a probability distribution of output characters. This is done by using a SoftMax Layer at the output with each node representing a letter, character, or syllable. By studying the sequence of characters being classified, the authors can recognize the speech being fed into the neural network, and they can compare the sequence of characters to a known sequence representing the wakeup word. The input features to the neural network include filter bank normalized data with its 1st and 2nd derivatives. In signal processing, a filter bank is an array of bandpass filters that separates the input signal into multiple components, each one carrying a single frequency sub-band of the original signal. Filter banks are extracted by applying fast Fourier transform over overlapping segments of input data stream. The author used 40 channels, with a sampling time of 10ms to extract the filter banks. After that, 1st and 2nd derivatives are calculated for the filter banks and



Neural network architecture using a DBLSTM.

normalized data is used as input to the neural network. Each bidirectional LSTM had 320 cells. Note that based on the author's experiments, unidirectional LSTMs performed like bidirectional LSTMs and advised to use unidirectional LSTM in online applications for its faster computation.

Connectionist Temporal Classification (CTC) training loss function is used by the authors to train the model as CTC is the core concept make it possible to transcribe unsegmented sequence data. It uses a SoftMax output layer to define a separate output distribution $P(k \mid t)$ at every step t along the input sequence for all the transcription labels plus an extra 'blank' symbol which represents a non-output. Intuitively the network decides whether to emit any label, or no label, at every time-step. Given a length T, input sequence x, and the output vectors y , the probability of emitting the label or blank with index k at time t is calculated and used to determine the sequence of letters and recognize if the sequence of the wakeup word is said.



An example of the WFST which depicts the Mandarin word 什么, corresponding syllable sequences "SHEN2 ME5". The "<eps>" symbol means no inputs are consumed or no outputs are emitted. While <blk> means blank label.

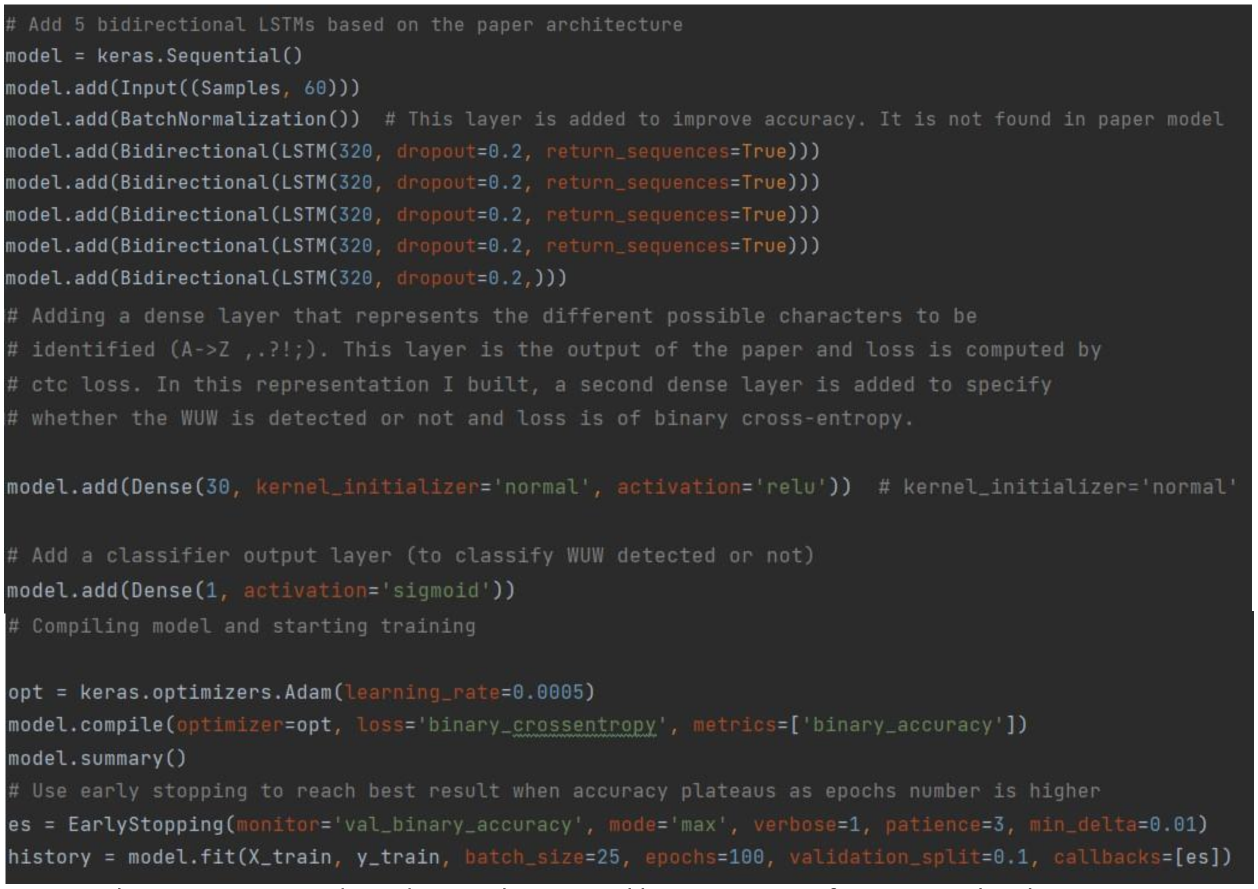
The weighted finite-state transducers (WFSTs) component maps a sequence of frame-level CTC labels (such as syllable or character) to a single wake-up word by first removing successive identical labels, and then blanks. The possible events in WUW spotting are hit, false alarm and false rejection. The performance is evaluated by presenting the false rejection rate (FRR) as function of the false alarm rate (FAR). This yields the Detect Error Tradeoff (DET) curve. To finetune the model, the author used data augmentation and compared the performance of the model based on different outputs like character, syllable, syllable with tone, and implemented a smaller version of the model with fewer parameters (160 cells per LSTM).

Detecting the Wakeup Word: My Implementation

Instead of training on a Chinese dataset, I used the mini-speech-commands dataset made by Google that has different English words labelled by the folder name. Also, I added some codes inspired by Pritish Mishra to complement the Google dataset by recordings made through my laptop's microphone. The mini-speech-commands had 8000 .wav files with 8 different spoken words. Each audio file was 1 second long with a sampling frequency of 256KHz. Data was split into 80% training, 20% testing. 10% of the training data was used for validation. Extracted features were like those of the author but with smaller number of channels (20 instead of 40) and larger sampling time (0.1s instead of 0.01s). This is done to reduce the training time and complexity of the input dimensions to the first layer. Data dimensions at input layer is ---> (training examples, samples = 10, features = 60).

The neural network proposed by the author in his paper was like the one I implemented, but I added a batch normalization layer before the five BLSTM layers. Also, instead of getting the output directly from the dense layer representing the letters of the input audio, I added a second layer with one node that transforms the problem into a binary classification problem (Wakeup word detected or not). This approach simplifies the model as it doesn't use a WFST to map the letters into words because the problem from the start was detecting the wakeup word only not spelling it too. Also, because of this modification in the output layer characteristics, binary cross entropy loss is superior to CTC loss and is used instead of CTC. The same performance metrics are used as a confusion matrix is used to visualize the performance of the model on testing data. Binary accuracy is used as the metric while training and validating the data. Real-time testing was done after training the model to verify its performance and notice its drawbacks.

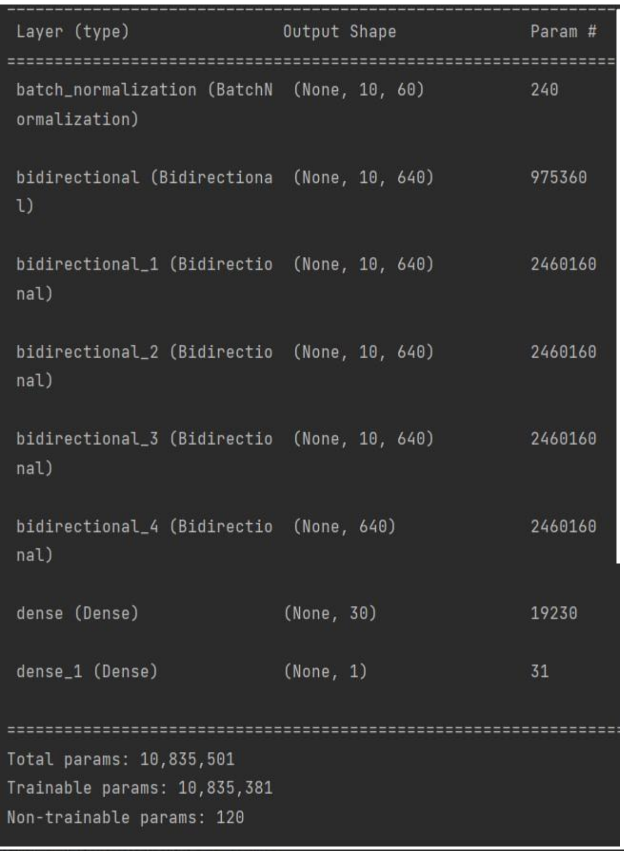
Building and Training the Model



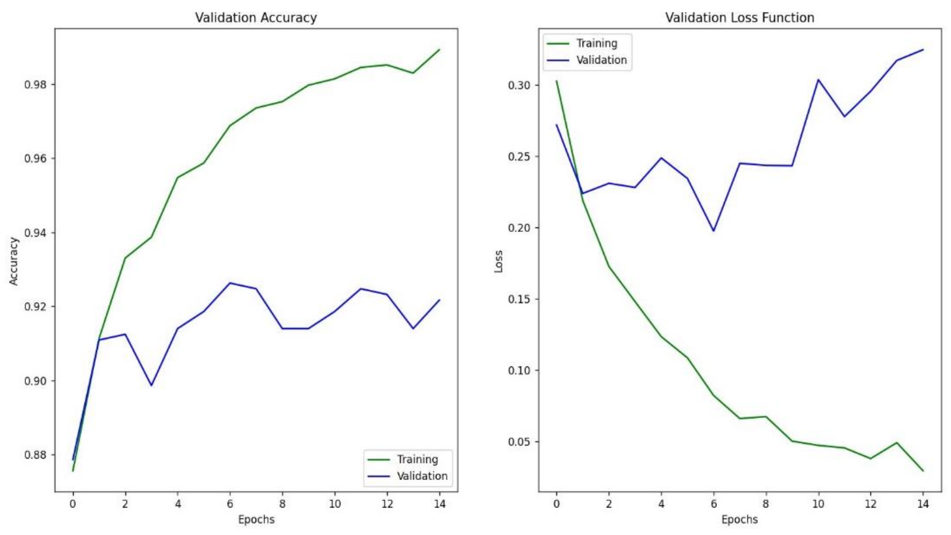
```
# Add 5 bidirectional LSTMs based on the paper architecture
model = keras.Sequential()
model.add(Input((Samples, 60)))
model.add(BatchNormalization())  # This layer is added to improve accuracy. It is not found in paper model
model.add(Bidirectional(LSTM(320, dropout=0.2, return_sequences=True)))
model.add(Bidirectional(LSTM(320, dropout=0.2, return_sequences=True)))
model.add(Bidirectional(LSTM(320, dropout=0.2, return_sequences=True)))
model.add(Bidirectional(LSTM(320, dropout=0.2, return_sequences=True)))
model.add(Bidirectional(LSTM(320, dropout=0.2,)))
# Adding a dense layer that represents the different possible characters to be
# identified (A->Z ,.?!;). This layer is the output of the paper and loss is computed by
# ctc loss. In this representation I built, a second dense layer is added to specify
# whether the WUW is detected or not and loss is of binary cross-entropy.

model.add(Dense(30, kernel_initializer='normal', activation='relu'))  # kernel_initializer='normal'

# Add a classifier output layer (to classify WUW detected or not)
model.add(Dense(1, activation='sigmoid'))
# Compiling model and starting training

opt = keras.optimizers.Adam(learning_rate=0.0005)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['binary_accuracy'])
model.summary()
# Use early stopping to reach best result when accuracy plateaus as epochs number is higher
es = EarlyStopping(monitor='val_binary_accuracy', mode='max', verbose=1, patience=3, min_delta=0.01)
history = model.fit(X_train, y_train, batch_size=25, epochs=100, validation_split=0.1, callbacks=[es])
```

The optimizer used is Adam with an initial learning rate of 0.0005. Other hyperparameters of the optimizer are kept as default in KERAS API. Note that we are using a sigmoid activation function instead of SoftMax in the output layer because of the architectural change (We have 1 node at output only while SoftMax is used with multiple node classifiers). Also, we implement early stopping during training to prevent overfitting as the training will stop if validation accuracy plateaus within three epochs. At first, a noticeable overfitting was seen during training and validation, so I added dropout at each LSTM (20% dropout ratio) to minimize the overfitting.

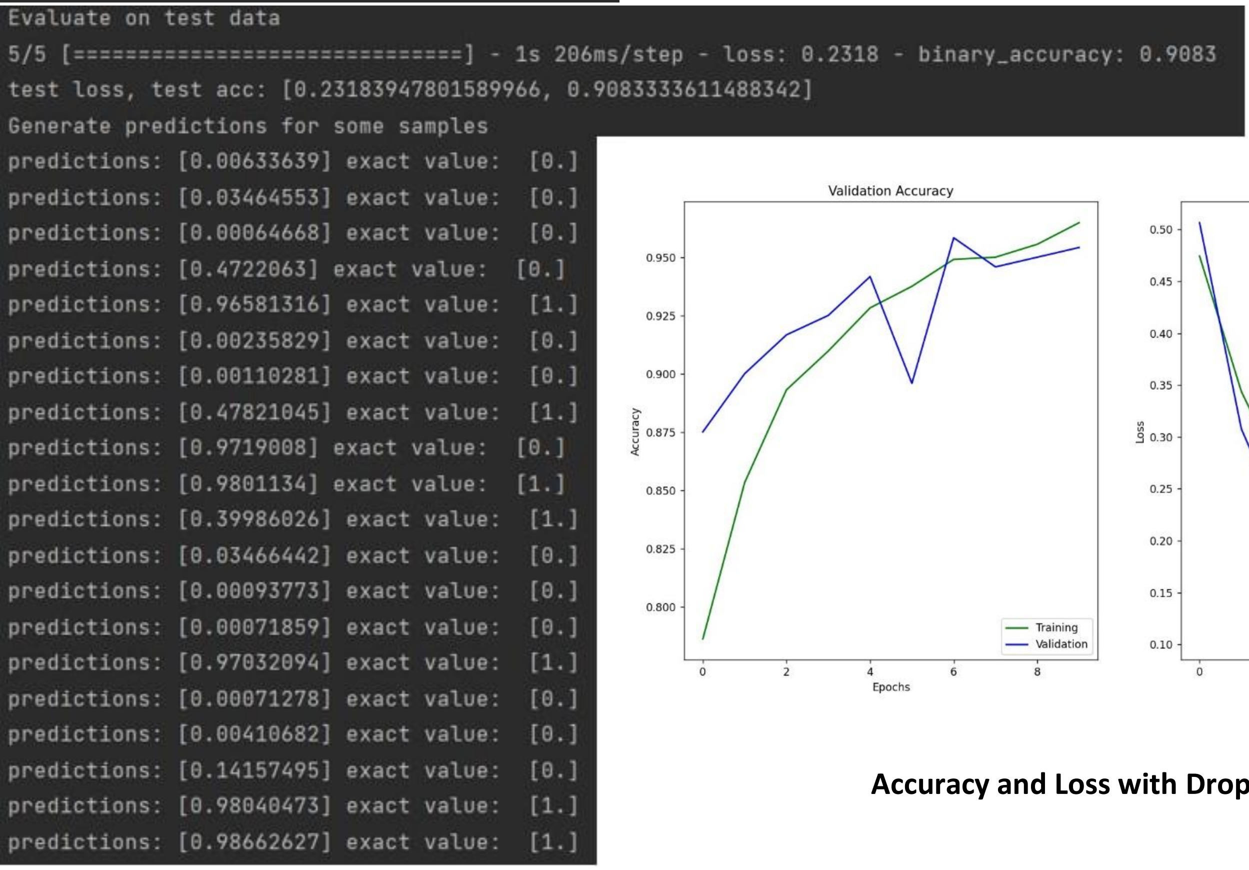
Results of Training



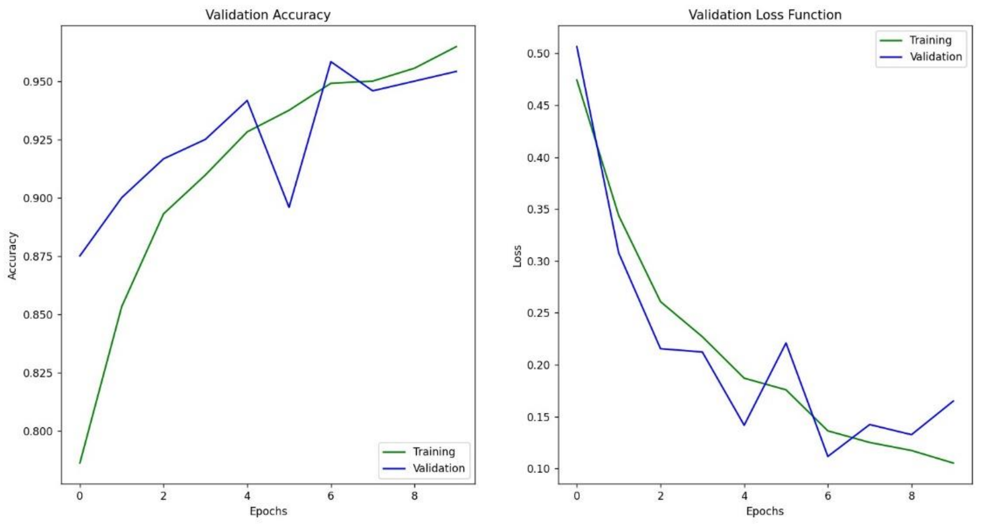
```
=================================================================
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, 10, 60)           240
 ormalization)

 bidirectional (Bidirectiona  (None, 10, 640)          975360
 l)

 bidirectional_1 (Bidirectio  (None, 10, 640)          2460160
 nal)

 bidirectional_2 (Bidirectio  (None, 10, 640)          2460160
 nal)

 bidirectional_3 (Bidirectio  (None, 10, 640)          2460160
 nal)

 bidirectional_4 (Bidirectio  (None, 640)              2460160
 nal)

 dense (Dense)               (None, 30)                19230

 dense_1 (Dense)             (None, 1)                 31

=================================================================
Total params: 10,835,501
Trainable params: 10,835,381
Non-trainable params: 120
```



Accuracy and Loss without Dropout

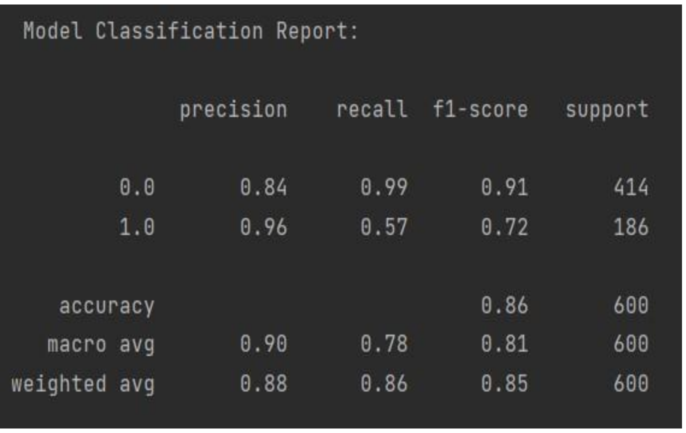


```
Evaluate on test data
5/5 [==============================] - 1s 206ms/step - loss: 0.2318 - binary_accuracy: 0.9083
test loss, test acc: [0.23183947801589966, 0.9083333611488342]
Generate predictions for some samples
predictions: [0.00633639] exact value:  [0.]
predictions: [0.03464553] exact value:  [0.]
predictions: [0.00064668] exact value:  [0.]
predictions: [0.4722063] exact value:  [0.]
predictions: [0.96581316] exact value:  [1.]
predictions: [0.00235829] exact value:  [0.]
predictions: [0.00110281] exact value:  [0.]
predictions: [0.47821045] exact value:  [1.]
predictions: [0.9719008] exact value:  [0.]
predictions: [0.9801134] exact value:  [1.]
predictions: [0.39986026] exact value:  [1.]
predictions: [0.03466442] exact value:  [0.]
predictions: [0.00093773] exact value:  [0.]
predictions: [0.00071859] exact value:  [0.]
predictions: [0.97032094] exact value:  [1.]
predictions: [0.00071278] exact value:  [0.]
predictions: [0.00410682] exact value:  [0.]
predictions: [0.14157495] exact value:  [0.]
predictions: [0.98040473] exact value:  [1.]
predictions: [0.98662627] exact value:  [1.]
```

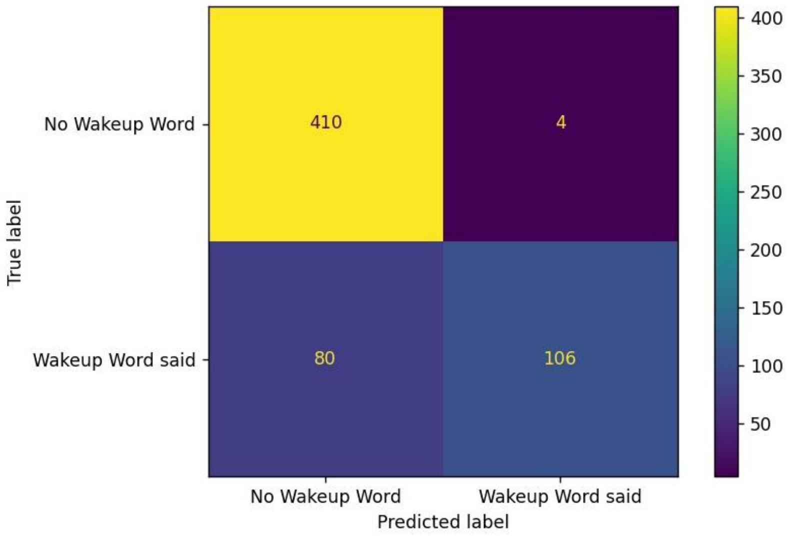


Accuracy and Loss with Dropout

Results of Testing



```
Model Classification Report:

              precision    recall  f1-score   support

         0.0       0.84      0.99      0.91       414
         1.0       0.96      0.57      0.72       186

    accuracy                           0.86       600
   macro avg       0.90      0.78      0.81       600
weighted avg       0.88      0.86      0.85       600
```



Discussion

The first problem seen is the relatively low ratio of training data labelled 1 and those labelled 0. This was fixed by purposely reducing the dataset till it reached the point at which data labeled 1 were consisting 1/3 of the data. This caused another problem of the overall small training set of 2400 audio examples used to train a relatively large model of more than 10 million parameters. Also, due to the lack of preprocessing made to recorded audio, most of the audio files recorded locally on my laptop had an initial pause in which I didn't speak, so that confused the neural network with background noise in which silence was present. The relatively small number of Wakeup word audio and its lack of preprocessing to remove idle sound made some True Wakeup word audio files to be predicted as No Wakeup Word said. This reduced the model accuracy during testing. This problem can be fixed by preparing a larger training dataset with better clipped audio.

While testing the model in real-time, I noticed that it can easily be fooled by words that sound like the wakeup word. The wakeup word chosen was the word 'go' and the model can label a word like 'no' as a wakeup word. This can be solved by better training to the model in future work. Also, Future work should include the speech recognition to be made after the wake-up word is detected. Note that the paper studied here made a general speech recognizer by building the model with SoftMax output layer and used it to detect whether the output sequence coincides with the wakeup word spelling or not. The approach implemented here used the same neural network as the paper but with a different output layer that simplifies the task of detecting the wakeup word; However, it is less general than the paper's implementation as the neural network would need to train every time we would like to change the wakeup word being detected.