# ACS61011 Deep Learning Assignment: Individual Project

Saba Firdaus Ansaria
Reg No: 210110201

March 2022

## Contents

# List of Figures



# List of Tables

# 1 Introduction

The goal of this project is to build an automated speech recognition system (ASR) using deep learning methods. Convolutional neural network models have been explored to fulfill the task. In this report, each section is responsible for demonstrating a specific task from dataset preparation to model building and then exploring different machine learning techniques for the project.

# 2 Dataset

The dataset consists 20 different words with 1000 examples each word. For making the task simple, a subset is chosen with the given *dataPreprocess.m* and the wav files have been converted to spectograms. Finally, these spectograms are split into training and validation set using the default ratio in *dataPreprocess.m*. The distribution is given in Figure 1.

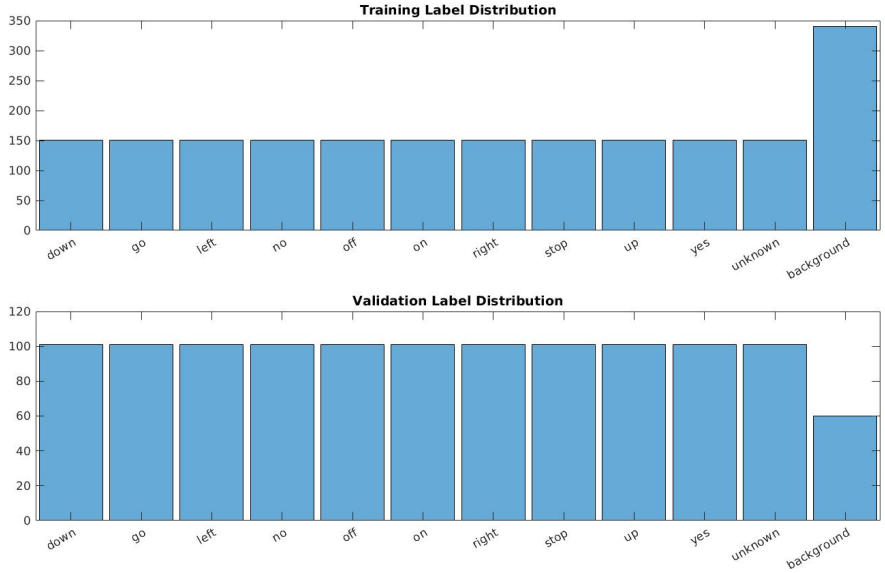



Figure 1: Training and Validation data distribution.

# 3 Model Architecture

Convolutional neural networks have been explored for building the ASR model as prescribed in the given report. The input layer is a 2D convolutional layer with [3x3] kernal size and 32 filters. For the next layers, several blocks consisting 2D convolution ([3x3] kernal,512 filters), batchnormalisation and 2D maxpooling ([2x2]) are used as shown in Figure 2. Two feed forward layers are there after the convolutional blocks. The blocks are build in a dynamic way as proposed in the given project guidelines. The model building and training are done in Keras.

Details of the model and the dynamic sequential blocks are shown below.

```
Model: "cnn_speech"
```

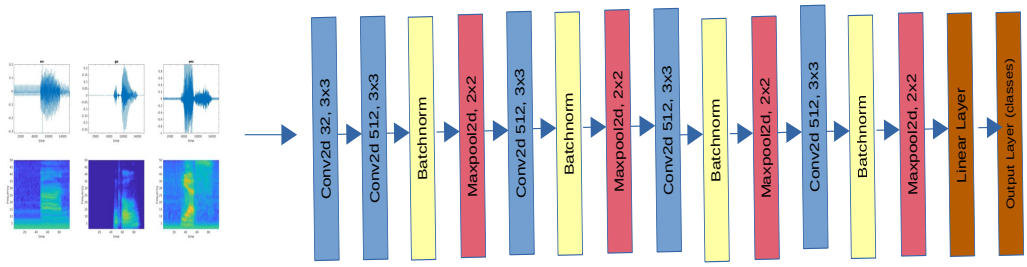



Figure 2: Model Architecture

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             multiple                  320

 sequential (Sequential)     (None, 5, 2, 512)         7235584

 flatten (Flatten)           multiple                  0

 dense (Dense)               multiple                  2621952

 dense_1 (Dense)             multiple                  6156

=================================================================
Total params: 9,864,012
Trainable params: 9,859,916
Non-trainable params: 4,096
_________________________________________________________________


    Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 94, 46, 512)       147968

 batch_normalization (BatchN  (None, 94, 46, 512)      2048
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 47, 23, 512)      0
 )

 conv2d_2 (Conv2D)           (None, 45, 21, 512)       2359808

 batch_normalization_1 (Batc  (None, 45, 21, 512)      2048
 hNormalization)
```

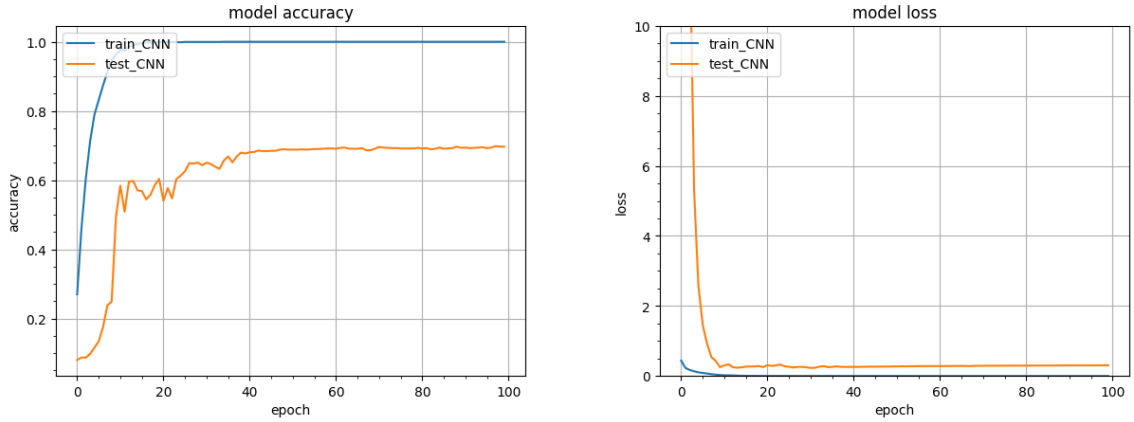
```
max_pooling2d_1 (MaxPooling  (None, 23, 11, 512)      0
2D)

conv2d_3 (Conv2D)           (None, 21, 9, 512)        2359808

batch_normalization_2 (Batc  (None, 21, 9, 512)       2048
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 11, 5, 512)       0
2D)

conv2d_4 (Conv2D)           (None, 9, 3, 512)         2359808

batch_normalization_3 (Batc  (None, 9, 3, 512)        2048
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 5, 2, 512)        0
2D)

=================================================================
Total params: 7,235,584
Trainable params: 7,231,488
Non-trainable params: 4,096
_________________________________________________________________
```

The dynamic block is executed using for loop as shown in the code snippet.

```
num_of_blocks = 4
self.input_cnn = Conv2D(32, (3, 3), strides=(1, 1), activation="relu")
self.cnn_block = Sequential()
for layers in range(num_of_blocks):
    self.cnn_block.add(Conv2D(512, (3, 3), strides=(1, 1), activation="relu"))
    self.cnn_block.add(BatchNormalization())
    self.cnn_block.add(MaxPool2D((2,2),padding='same'))
self.flatten = Flatten()
self.dense1 = Dense(512, activation="relu")
self.dense2 = Dense(12, activation="softmax")
```

# 4 Experiment and Results

The model is trained using Keras and it has achieved 69.68% accuracy on the validation set. This model has 4 convolutional blocks with 512 filters each. The confusion matrix is shown in Figure 4. The training vs validation accuracy and loss is shown in Figure 3. The model is trained 100 epoch with Adam optimiser and 128 batch size.




(a) CNN model accuracy. (b) CNN model loss.

Figure 3: Model accuracy and loss on training and validation data from the CNN model

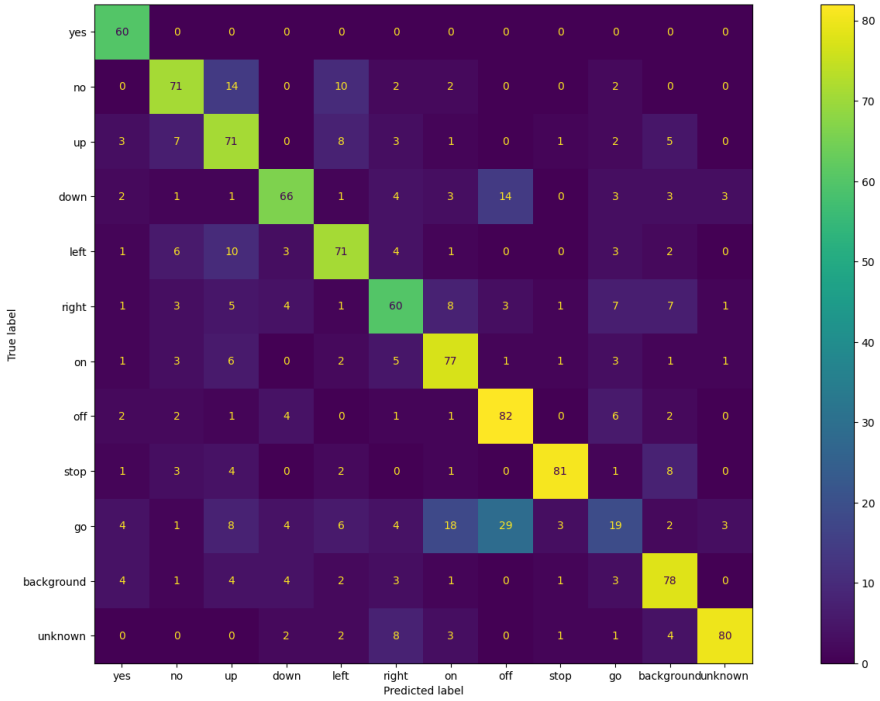
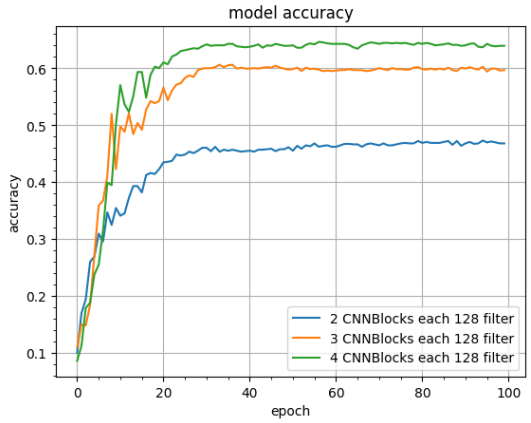



Figure 4: Confusion matrix of CNN model performance on validation data.

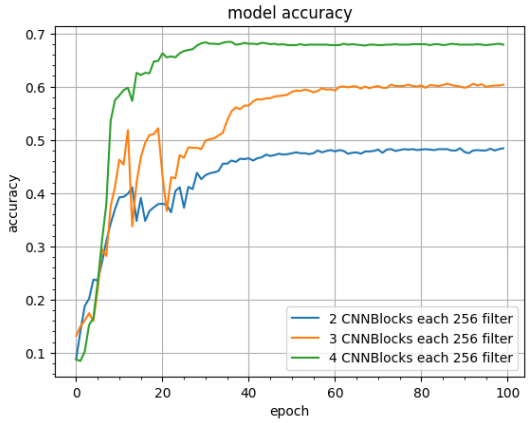
# 5 Model structure Investigation

The systematic investigation is necessary to understand the number of layers and number of filters in a model. In this scenario, I have changed the number of layers and filters by adding or reducing the convolutional blocks dynamically using **num_of_blocks** variable. In the first scenario, the number of layers have been changed while keeping the number of filters fixed. For example, a model with two (convolution, batchnormalisation, maxpool) blocks, a model with three (convolution, batchnormalisation, maxpool) blocks

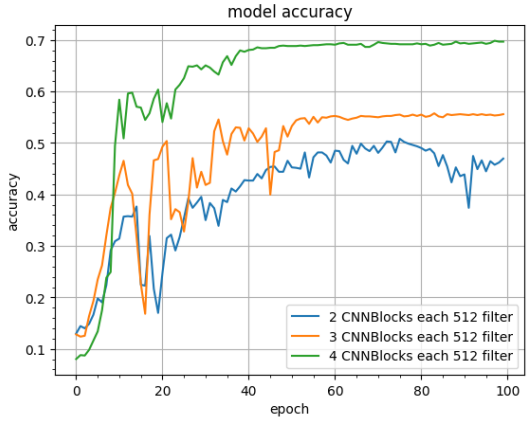
and, a model with four (convolution, batchnormalisation, maxpool) blocks are compared according to their performance on the ASR task while each of the convolution layers have 128 number of filters. These three models are called *model_2CNN_blocks_128_filter*, *model_3CNN_blocks_128_filter*, *model_4CNN_blocks_128_filter* (Figure 5a, 6a). Similarly, the models, with 256 filter blocks and 512 filter blocks are compared and the results are shown in Figure 5 and 6. The accuracy vs epoch plots shows the progression of the performance ( x-axis is plotted as accuracy/100 , i.e, 0.6 means 60% accuracy) on the validation set over the training period (100 epochs).




(a) Each CNN layer has 128 filters.



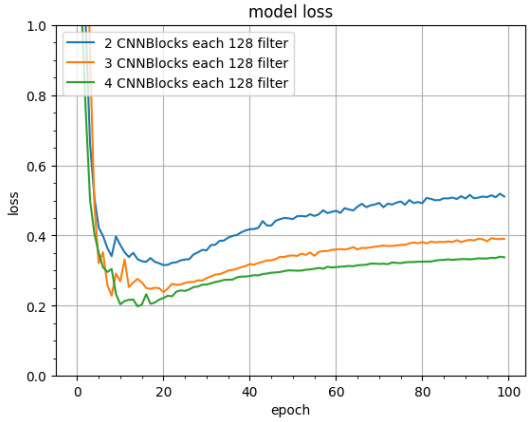

(b) Each CNN layer has 256 filters.



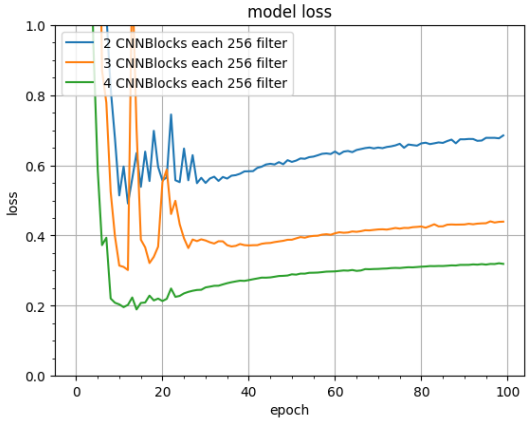

(c) Each CNN layer has 512 filters.

Figure 5: Model accuracy on validation data comparison by changing the number of layers while keeping the number of filters each layer same

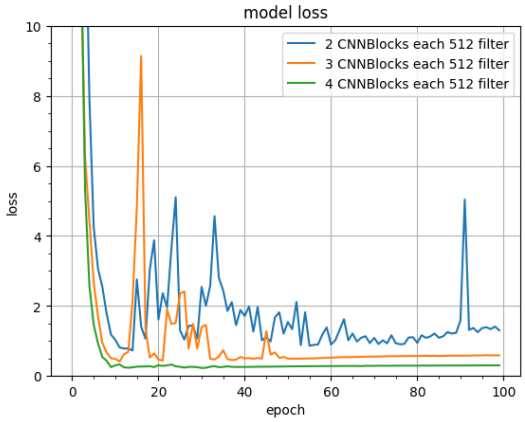
Furthermore, the models with different number of filters have been compared while keeping the number of (convolution, batchnormalisation, maxpool) blocks same. The results are shown in Figure 7 and 8. Clearly, it is evident that increasing the number of filters and the layers are beneficial to this ASR model among these 9 different models I have tested. However, this statement can not be generalised for every other model. Because it is dependant on the model and the amount of data it is trained on. The best accuracy achieved by these models on the validation data are shown in Table 1.




(a) Each CNN layer has 128 filters.




(b) Each CNN layer has 256 filters.

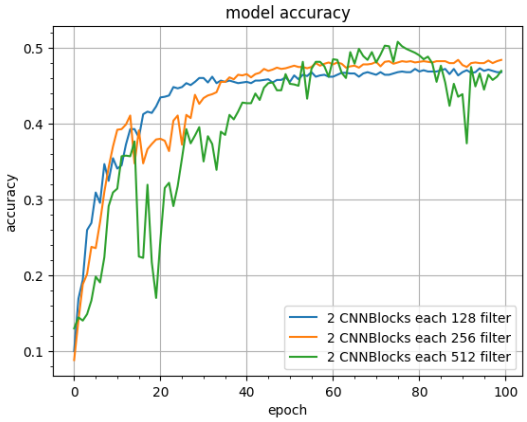



(c) Each CNN layer has 512 filters.

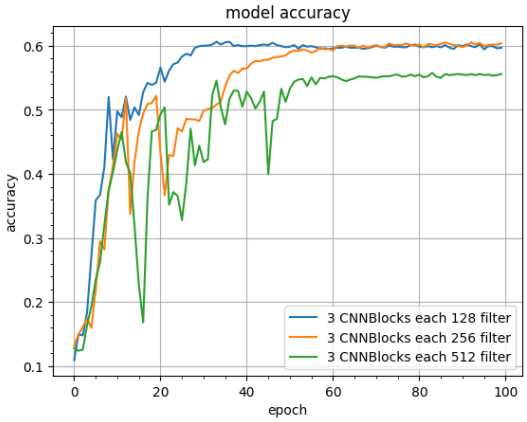
Figure 6: Model loss on validation data comparison by changing the number of layers while keeping the number of filters each layer same

| Model Name | Accuracy(%) | Total epochs |
|---|---|---|
| model_2CNN_blocks_128_filter | 47.22 | 100 |
| model_3CNN_blocks_128_filter | 60.63 | 100 |
| model_4CNN_blocks_128_filter | 64.56 | 100 |
| model_2CNN_blocks_256_filter | 48.42 | 100 |
| model_3CNN_blocks_256_filter | 60.30 | 100 |
| model_4CNN_blocks_256_filter | 68.40 | 100 |
| model_2CNN_blocks_512_filter | 50.81 | 100 |
| model_3CNN_blocks_512_filter | 55.59 | 100 |
| model_4CNN_blocks_512_filter | **69.68** | 100 |

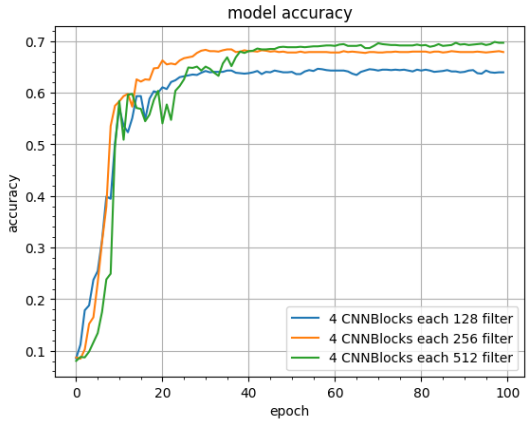
Table 1: Models with different number of layers and filters comparison on the validation data




(a) 2 CNN layer model.

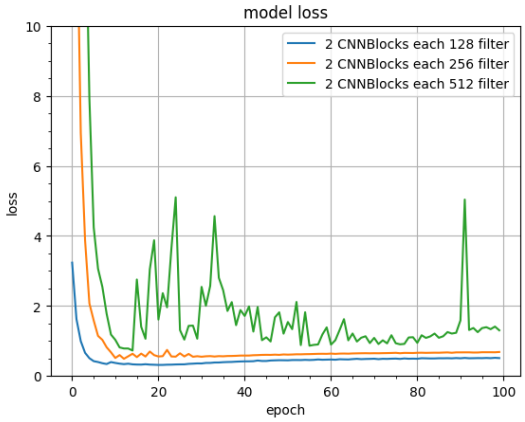



(b) 3 CNN layer model.

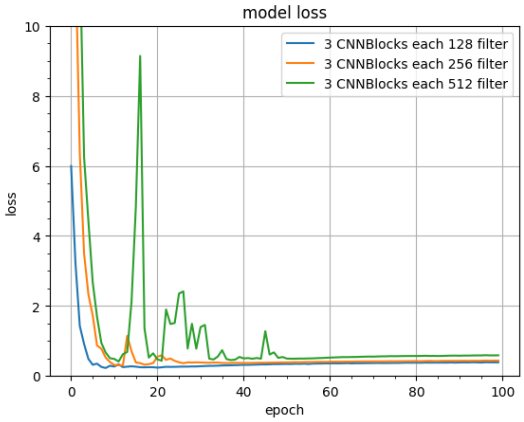



(c) 4 CNN layer model.

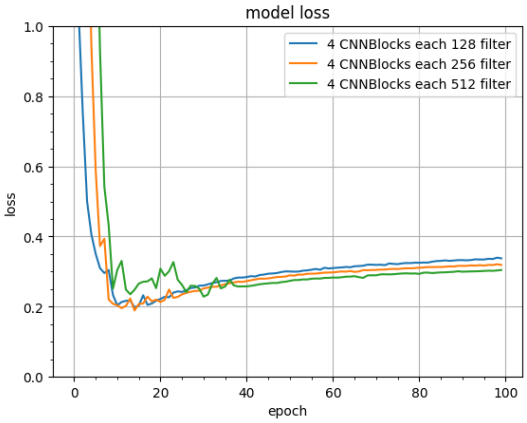
Figure 7: Model accuracy on validation data comparison by changing the number of filters per layer while keeping the number layer same




(a) 2 CNN layer model.




(b) 3 CNN layer model.




(c) 4 CNN layer model.

Figure 8: Model loss on validation data comparison by changing the number of filters per layer while keeping the number layer same

# 6 Extended Dataset and Results

After evaluating the model on the given subset, I have added five more classes *"wow","two","three","five"* to extend the task. Also the training and validation split ratio is now 75%, 25%. The data distribution is shown in Figure 9.

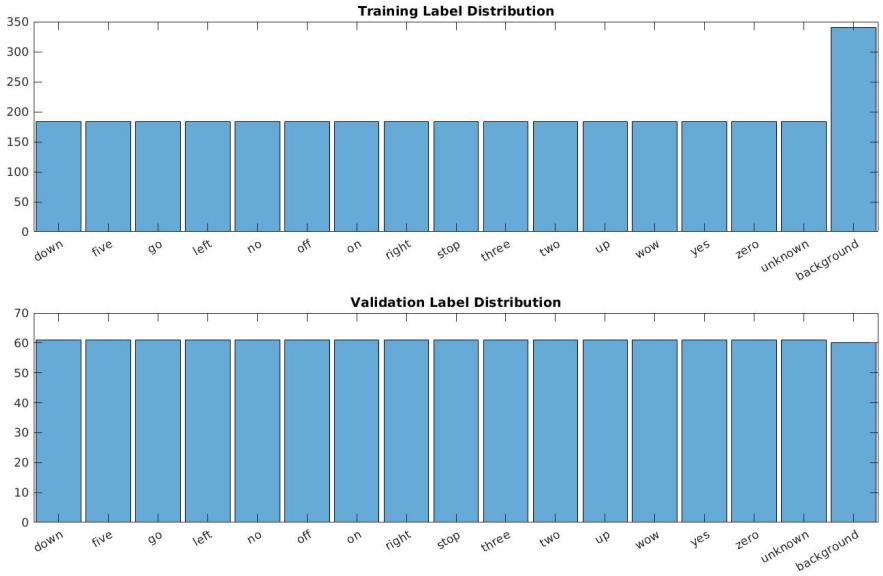
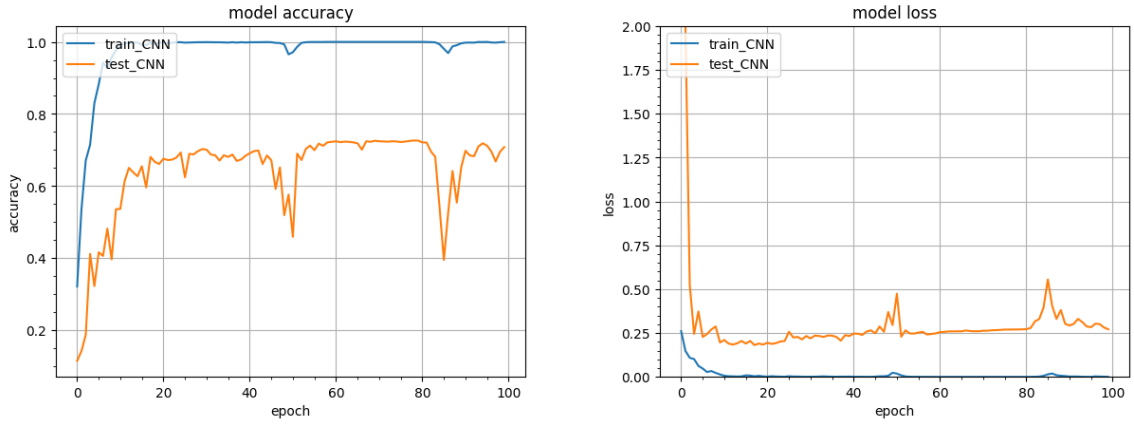



Figure 9: Training and Validation data distribution on the extended dataset with more five classes along with previous twelve classes.

This extended dataset is the trained on the model which achieved best result on the previous subset ,i.e, *model_4CNN_blocks_512_filter*. The training and validation results are shown in Figure 10 and the confusion matrix is shown in Figure 11.




(a) CNN model accuracy.

(b) CNN model loss.

Figure 10: Model accuracy and loss on validation data (extended dataset with five new classes) from the CNN model

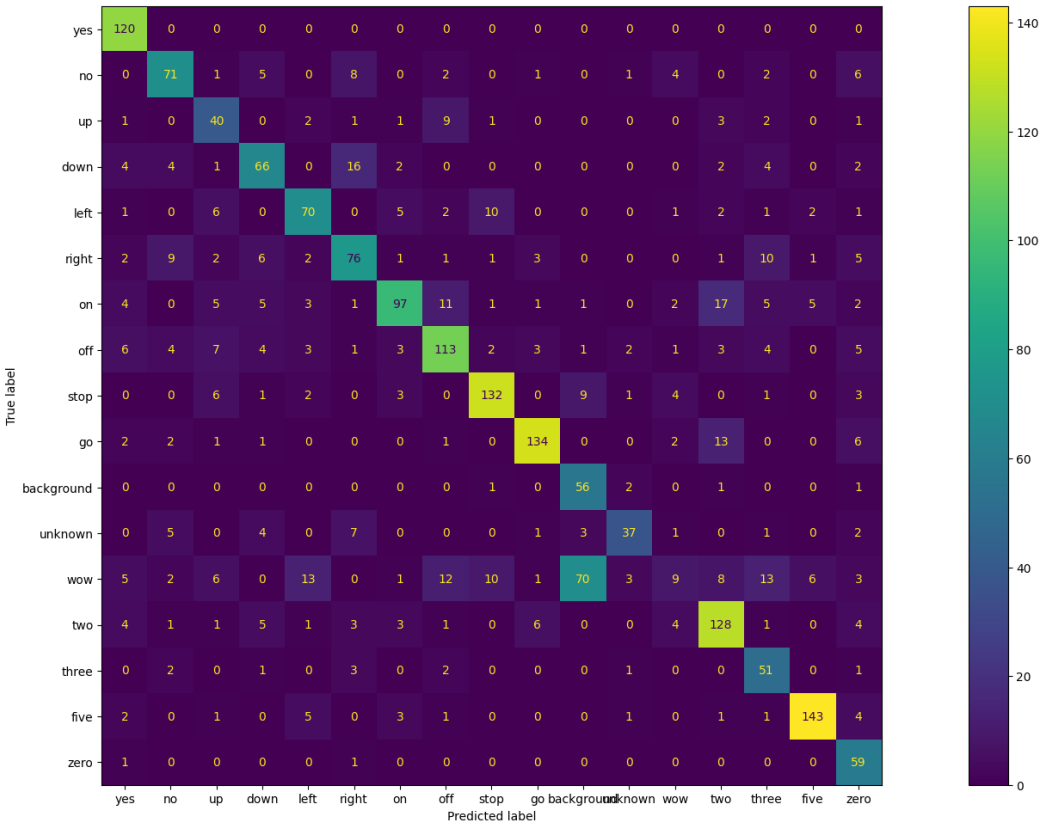
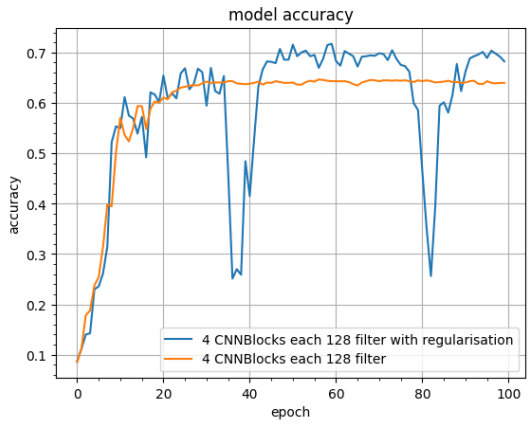


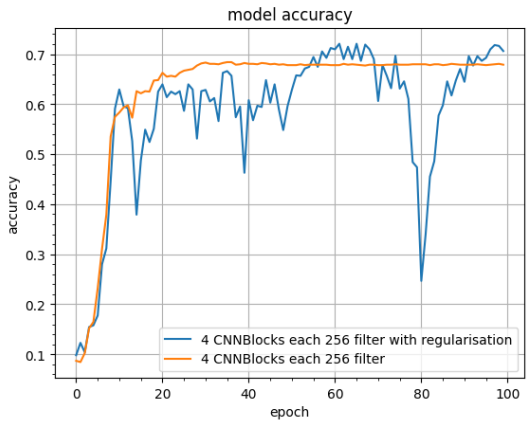

Figure 11: Confusion matrix of CNN model performance on validation data (extended dataset with five new classes).

# 7 Regularisation and Training Regime

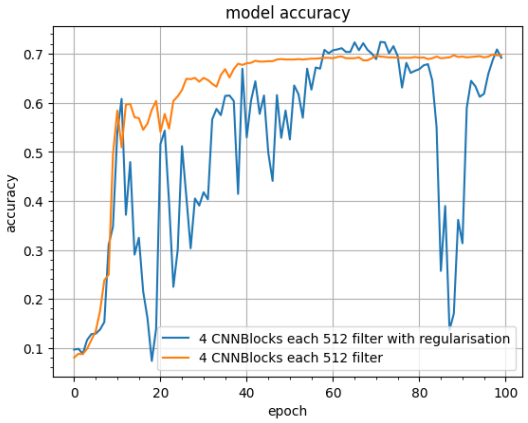
Regularisation is a training regime which helps to prevent the models from overfitting. Layer regularisation penalises the layers with large weights which helps the model achieve better optimisation. Here, Keras libraries are used to perform L2 (0.01) layer regularisation in the feed-forward neural layer. In Table 2, the models are compared with their regularised versions, where L2 layer regularisation is applied in the second to last feed-forward layer. All the models with regularisation achieved better top accuracy than their non-regularised counterparts. However, Figure 12 and 13 show a jittery training despite of having top accuracy. The reason may be related to the learning rate of the model while training. Here upto this point the learning rate is 0.001. While using regularisation, it is essential to decay the learning rate over time. This is demonstrated in the next section.




(a) 4 layer CNN with 128 filters model with and without regularisation.




(b) 4 layer CNN with 256 filters model with and without regularisation.

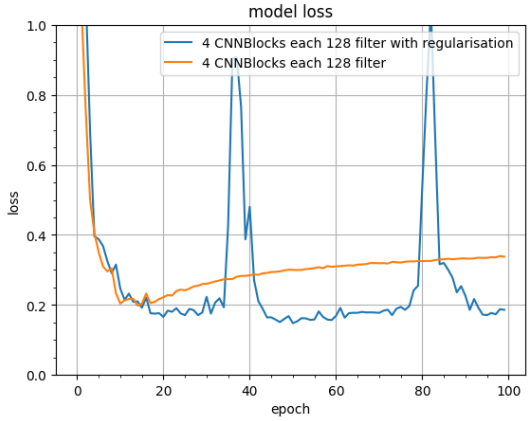



(c) 4 layer CNN with 512 filters model with and without regularisation.

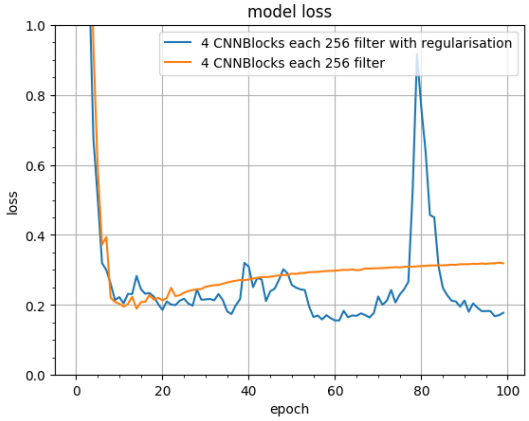
Figure 12: Model accuracy on validation data comparison between models with and without regularisation

| Model Name | Accuracy(%) | Total epochs |
|---|---|---|
| model_4CNN_blocks_128_filters | 64.56 | 100 |
| model_4CNN_blocks_256_filters | 68040 | 100 |
| model_4CNN_blocks_512_filters | 69.68 | 100 |
| model_4CNN_blocks_128_filters_regularisation | 71.73 | 100 |
| model_4CNN_blocks_256_filters_regularisation | 72.08 | 100 |
| model_4CNN_blocks_512_filters_regularisation | **72.42** | 100 |

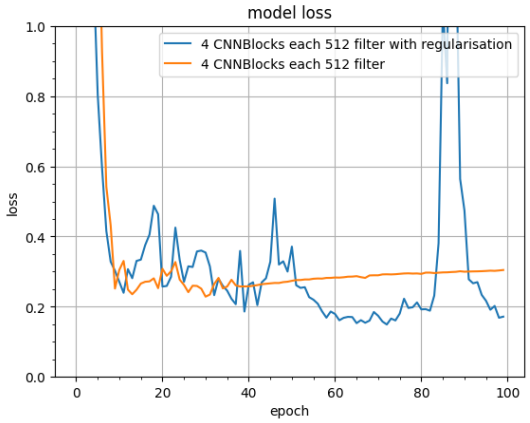
Table 2: Model without regularisation and with regularisation comparison on accuracy on the validation data




(a) 4 layer CNN with 128 filters model with and without regularisation.




(b) 4 layer CNN with 256 filters model with and without regularisation.




(c) 4 layer CNN with 512 filters model with and without regularisation.

Figure 13: Model loss on validation data comparison between models with and without regularisation

# 8 Convolutional Model with Residual Connection

As the final part of the miniproject, it was advised to implement the GoogleNet like architecture. However, due to computational complexity and the time to train such model, I tried to look at other approaches. Residual networks used skip connections which connect and adds the output from different layers in the model hierarchy. It has been seen that these residual models with skip connections performs better than traditional convolution models despite having smaller number of parameters. The residual CNN model is built using the previous convolutional model. Here, the (convolution, batchnormalisation, maxpool) blocks are connected with previous hierarchies using skip connections.

As discussed in the previous section, the initial learning rate 0.001 is decayed after 100 epoch to 0.0001 and the model is further trained for 100 epochs. The effects can be clearly seen. The decaying learning rate approach make the training more smoother with less jitters. The training vs validation accuracy is shown in Figure 14. The performance is compared with the best performing models in this report and shown in Table 3. Clearly the residual CNN model achieved higher accuracy than the other models. In The implemented code, the class name is **ResNetSpeech**. The confusion matrix is shown in Figure 15.

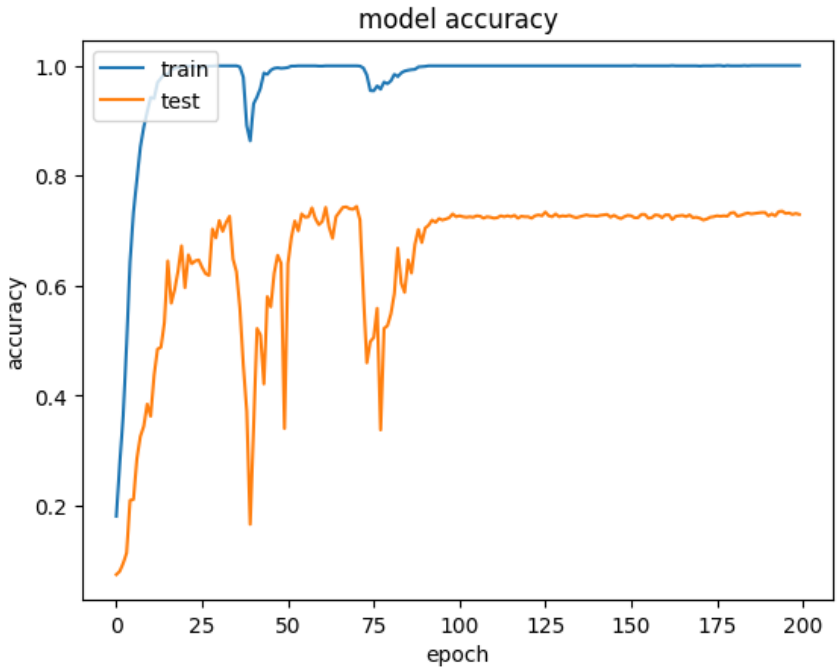



Figure 14: CNN model with residual connection performance on validation data.

| Model Name | Accuracy(%) |
|---|---|
| model_4CNN_blocks_512_filters | 69.68 |
| model_4CNN_blocks_512_filters_regularisation | 72.42 |
| CNN_model with residual connection | 74.64 |

Table 3: Model comparison on accuracy on the validation data

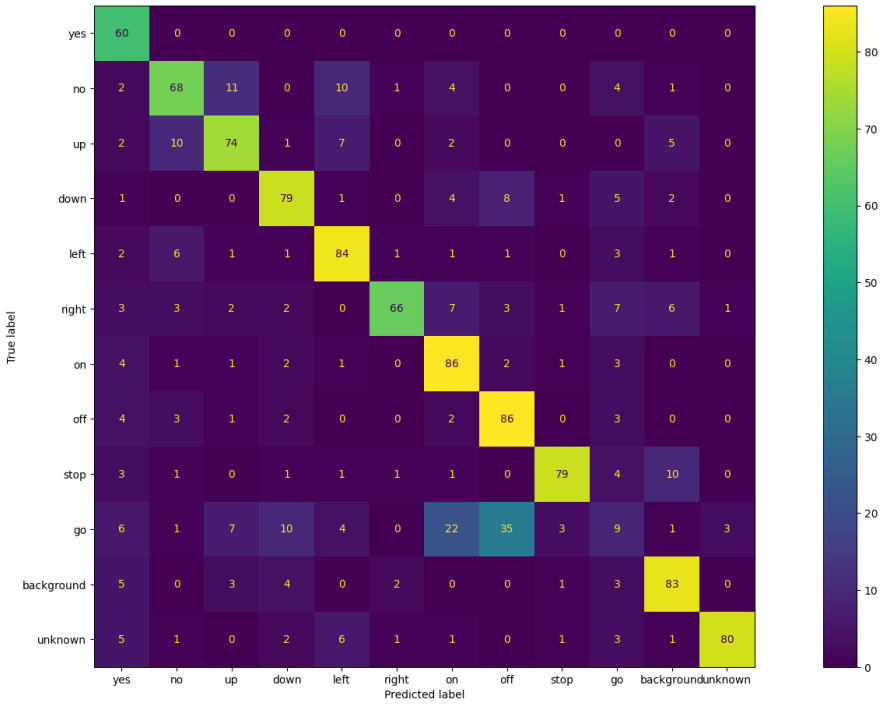



Figure 15: Confusion matrix of CNN model with residual connection on validation data.

# 9 Code

```
'''
main.py for running the main training code
CNN_model.py has all the model classes
plot_results.py plots the graphs by loading model history
model_load.py loads a given model and plots confusion matrix
'''


import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from CNN_model import CNNSpeech
from CNN_model import ResNetSpeech
from CNN_model import CNNSpeechRegularised
from keras.utils.vis_utils import plot_model
import pickle
from keras import backend as K



# building dataset from image directory
def dataset(train,val):
    train_set = keras.utils.image_dataset_from_directory(
```

```
        directory=train,
        labels="inferred",
        color_mode="grayscale",
        label_mode="categorical",
        batch_size=128,
        image_size=(98, 50))

    val_set = keras.utils.image_dataset_from_directory(
        directory=val,
        labels="inferred",
        color_mode="grayscale",
        label_mode="categorical",
        batch_size=128,
        image_size=(98, 50))
    return train_set, val_set


#main training function
def train(train_set,val_set, model, batch_size, epochs):

    # compile the keras model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # fit the keras model on the dataset
    history = model.fit(train_set,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=True, validation_data=val_set)

    return model, history

# training function for learning rate decay experiments in two stages
def train_learning_rate_exp(train_set,val_set, model, batch_size, epochs):

    # compile the keras model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # fit the keras model on the dataset
    history = model.fit(train_set,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=True, validation_data=val_set)
    train_acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    K.set_value(model.optimizer.learning_rate, 0.0001)
    history = model.fit(train_set,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=True, validation_data=val_set)
```

```
    train_acc = train_acc + history.history['accuracy']
    val_acc = val_acc + history.history['val_accuracy']

    plt.plot(train_acc)
    plt.plot(val_acc)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

    return model, history


# elementary plotting function
def plot(history):
    print(history.history.keys())
    # history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

#model evaluation
def evaluate_model(model, val_set):
    # Generate generalization metrics
    #score = model.evaluate(val_set, verbose=1)
    #print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
    _, accuracy = model.evaluate(val_set)
    print('Accuracy: %.2f' % (accuracy*100))


def save_model(model):
    model.save_weights("saved_models/model_weights", save_format='tf')


def show_model(model,filename):
```

```
    plot_model(model, to_file=filename, show_shapes=True, show_layer_names=True)

# save model training history
def save_history(history,filename):
    outbuffer = open(filename, 'wb')
    pickle.dump(history,outbuffer)
    outbuffer.close()

# load model history given a path
def load_history(filename):
    inbuffer = open(filename,'rb')
    history = pickle.load(inbuffer)
    return history


def save_model(model,filename):
    model.save(filename)

if __name__ == '__main__':
    train_data = "/home/saba/PycharmProjects/saba/speechImageData/TrainingData"
    val_data = "/home/saba/PycharmProjects/saba/speechImageData/ValidationData"
    train_set, val_set = dataset(train_data,val_data)
    model = CNNSpeech()
    # model = CNNSpeechRegularised()
    # model = ResNetSpeech()
    batch_size = 128
    epochs = 100
    model, history = train(train_set,val_set,model,batch_size,epochs)
    # model, history = train_learning_rate_exp(train_set, val_set, model, batch_size,
    print(model.optimizer.learning_rate)
    plot(history)
    evaluate_model(model, val_set)
    save_history(history, "history/test.hist")
    save_model(model, "model/test")
    print(model.summary())



'''
model class file
'''

import tensorflow as tf
from keras.models import Sequential
from keras.layers import BatchNormalization, MaxPool2D
from keras.layers import Conv2D
from keras.layers import Activation
from keras.layers import Flatten
```

```
from keras.layers import Dropout
from keras.layers import Dense
from keras.models import load_model
from tensorflow import keras


class CNNSpeech(tf.keras.Model):

  def __init__(self):
    super().__init__()
    num_of_blocks = 4
    self.input_cnn = Conv2D(32, (3, 3), strides=(1, 1), activation="relu")
    self.cnn_block = Sequential()
    # dynamic creation of convolution blocks
    for layers in range(num_of_blocks):
        self.cnn_block.add(Conv2D(512, (3, 3), strides=(1, 1), activation="relu"))
        self.cnn_block.add(BatchNormalization())
        self.cnn_block.add(MaxPool2D((2,2),padding='same'))
    self.flatten = Flatten()
    self.dense1 = Dense(512, activation="relu")
    self.dense2 = Dense(12, activation="softmax")

  def call(self, inputs):
    # print(inputs.shape)
    x = self.input_cnn(inputs)
    # print(x.shape)
    x = self.cnn_block(x)
    # print(x.shape)
    x = self.flatten(x)
    x = self.dense1(x)
    x = self.dense2(x)
    return x



class CNNSpeechRegularised(tf.keras.Model):

  def __init__(self):
    super().__init__()
    num_of_blocks = 4
    self.input_cnn = Conv2D(32, (3, 3), strides=(1, 1), activation="relu")
    self.cnn_block = Sequential()
    # dynamic creation of convolution blocks
    for layers in range(num_of_blocks):
        self.cnn_block.add(Conv2D(512, (3, 3), strides=(1, 1), activation="relu"))
        self.cnn_block.add(BatchNormalization())
        self.cnn_block.add(MaxPool2D((2,2),padding='same'))
    self.flatten = Flatten()
```

```
    self.dense1 = Dense(512, activation="relu",
    kernel_regularizer=keras.regularizers.l2(l=0.01))
    self.dense2 = Dense(128, activation="relu")
    self.dense3 = Dense(17, activation="softmax")

  def call(self, inputs):
    # print(inputs.shape)
    x = self.input_cnn(inputs)
    # print(x.shape)
    x = self.cnn_block(x)
    # print(x.shape)
    x = self.flatten(x)
    x = self.dense1(x)
    x = self.dense2(x)
    x = self.dense3(x)
    return x


class ResNetSpeech(tf.keras.Model):
  def __init__(self):
    super().__init__()
    num_of_blocks = 2
    self.input_cnn = Conv2D(32, (3, 3), strides=(1, 1), activation="relu")
    self.cnn_block1 = Sequential()
    # dynamic creation of convolution blocks
    for layers in range(num_of_blocks):
        self.cnn_block1.add(Conv2D(128, (3, 3), strides=(1, 1), activation="relu"))
        self.cnn_block1.add(BatchNormalization())
        self.cnn_block1.add(MaxPool2D((2,2),padding='same'))

    self.cnn_block2 = Sequential()
    for layers in range(num_of_blocks):
      self.cnn_block2.add(Conv2D(128, (1, 1), strides=(1, 1), activation="relu"))
      self.cnn_block2.add(BatchNormalization())
      # self.cnn_block2.add(MaxPool2D((2, 2), padding='same'))

    self.cnn_block3 = Sequential()
    for layers in range(num_of_blocks):
      self.cnn_block3.add(Conv2D(256, (3, 3), strides=(1, 1), activation="relu"))
      self.cnn_block3.add(BatchNormalization())
      self.cnn_block3.add(MaxPool2D((2, 2), padding='same'))

    self.cnn_block4 = Sequential()
    for layers in range(num_of_blocks):
      self.cnn_block4.add(Conv2D(256, (1, 1), strides=(1, 1), activation="relu"))
      self.cnn_block4.add(BatchNormalization())
      # self.cnn_block2.add(MaxPool2D((2, 2), padding='same'))
```

```
    self.flatten = Flatten()
    self.dense1 = Dense(256, activation="relu",
    kernel_regularizer=keras.regularizers.l2(l=0.01))
    self.dense2 = Dense(128, activation="sigmoid")
    self.dense3 = Dense(12, activation="sigmoid")

  def call(self, inputs):
    # print(inputs.shape)
    x = self.input_cnn(inputs)
    # print(x.shape)
    x = self.cnn_block1(x)
    # print(x.shape)
    x_ = x
    x = self.cnn_block2(x)
    # print(x.shape)
    x = tf.keras.layers.Add()([x, x_])
    # print(x.shape)
    x = self.cnn_block3(x)
    # print(x.shape)
    x_ = x
    x = self.cnn_block4(x)
    # print(x.shape)
    x = tf.keras.layers.Add()([x, x_])
    x = self.flatten(x)
    x = self.dense1(x)
    x = self.dense2(x)
    x = self.dense3(x)
    return x




'''
This program does different types of plotting based on model comparisons
'''

import pickle
import matplotlib.pyplot as plt


def plot_three_comparison(history1,history2,history3):
    print(history1.history.keys())
    # summarize history for test accuracy
    plt.plot(history1.history['val_accuracy'])
    plt.plot(history2.history['val_accuracy'])
    plt.plot(history3.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
```

```
    plt.xlabel('epoch')
    plt.legend(['4 CNNBlocks each 128 filter', '4 CNNBlocks each 256 filter',
    '4 CNNBlocks each 512 filter'], loc='lower right')
    plt.grid(True)
    plt.minorticks_on()
    plt.show()
    # summarize history for loss
    print(history1.history.keys())
    # summarize history for test accuracy
    plt.plot(history1.history['val_loss'])
    plt.plot(history2.history['val_loss'])
    plt.plot(history3.history['val_loss'])
    ax = plt.gca()
    ax.set_ylim([0, 1])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['4 CNNBlocks each 128 filter', '4 CNNBlocks each 256 filter',
    '4 CNNBlocks each 512 filter'], loc='upper right')
    plt.grid(True)
    plt.minorticks_on()
    plt.show()


def plot_two_comparison(history1,history2):
    print(history1.history.keys())
    # summarize history for test accuracy
    plt.plot(history1.history['val_accuracy'])
    plt.plot(history2.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['4 CNNBlocks each 512 filter with regularisation',
    '4 CNNBlocks each 512 filter'], loc='lower right')
    plt.grid(True)
    plt.minorticks_on()
    plt.show()
    # summarize history for loss
    print(history1.history.keys())
    # summarize history for test accuracy
    plt.plot(history1.history['val_loss'])
    plt.plot(history2.history['val_loss'])
    ax = plt.gca()
    ax.set_ylim([0, 1])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['4 CNNBlocks each 512 filter with regularisation',
```

```
    '4 CNNBlocks each 512 filter'], loc='upper right')
    plt.grid(True)
    plt.minorticks_on()
    plt.show()


def plot(history):
    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train_CNN', 'test_CNN'], loc='upper left')
    plt.minorticks_on()
    plt.grid(True)
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    ax = plt.gca()
    ax.set_ylim([0, 2])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_CNN', 'test_CNN'], loc='upper left')
    plt.minorticks_on()
    plt.grid(True)
    plt.show()


def load_history(filename):
    inbuffer = open(filename,'rb')
    history = pickle.load(inbuffer)
    return history


if __name__ == '__main__':
    history1 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_block_128.hist")
    history2 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_block_256.hist")
    history3 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_block_512.hist")
    plot_three_comparison(history1, history2, history3)
    history1 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_block_512_regularised.hist")
```

```
    history2 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_block_512.hist")
    plot_two_comparison(history1,history2)
    history1 = load_history("/home/saba/PycharmProjects/saba/
    history/model_4_512_extendeddata.hist")
    plot(history1)



'''
load model and print confusion matrix.
'''
from CNN_model import CNNSpeech
from keras.utils.vis_utils import plot_model
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras.metrics import Metric
from sklearn.metrics import plot_confusion_matrix
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt



def dataset(train,val):
    train_set = keras.utils.image_dataset_from_directory(
        directory=train,
        labels="inferred",
        color_mode="grayscale",
        label_mode="categorical",
        batch_size=128,
        image_size=(98, 50))

    val_set = keras.utils.image_dataset_from_directory(
        directory=val,
        labels="inferred",
        color_mode="grayscale",
        label_mode="categorical",
        batch_size=128,
        image_size=(98, 50))
    return train_set, val_set

if __name__ == '__main__':
    train_data = "/home/saba/PycharmProjects/saba/speechImageData/TrainingData"
    val_data = "/home/saba/PycharmProjects/saba/speechImageData/ValidationData"
    train_set, val_set = dataset(train_data,val_data)
    classes = ["yes","no","up","down","left","right","on","off","stop","go",
```

```
"background", "unknown"]
true_label = []
pred_label = []
model= keras.models.load_model("/home/saba/PycharmProjects/
saba/model/model_resnet")
model.compile()
for data, label in val_set:
    y_pred = model.predict(data)
    y_pred = np.argmax(y_pred,axis=1)
    label = label.numpy()
    label = np.where(label==1)[1]
    true_label.append(label)
    pred_label.append(y_pred)

y_pred = np.concatenate(pred_label,axis=0)
y_true = np.concatenate(true_label,axis=0)
print(y_pred.shape)
print(y_true.shape)
conf_obj = confusion_matrix(y_true, y_pred)
display = ConfusionMatrixDisplay(confusion_matrix=conf_obj,
display_labels=classes)
display.plot()
plt.show()
```