Neural Networks approach for Audio clip recognition

Nitin Gummidela

Abstract

This project is aimed at modelling a neural network that classifies structured and unstructured audio data. For unstructured data urban sounds such as air conditioner and car horn were used and for structured data short speech commands were used. Models were trained using features called MFCCs (Mel Frequency Cepstral Coefficient) extracted from the audio samples. MFCCs contain the important content in the audio and discards the changes in the samples due to background noise. Experiments were performed using various CNN architectures. Further experiments were performed by conduting an abalation study with various filter sizes, feature bin sizes, number of layers, number of epochs, different optimizers and various other hyper parameters. A final test accuracy of 88 percent was achieved with a validation accuracy of 89 percent and training accuracy of 99 precent using 2D CNN model trained with data with augmentation. To achieve this two different databases 'URBANSOUND8K [1]' and 'Tensorflow speech recogniton challange [2]' dataset were mereged and used for training the model.

1 Introduction

Automatic audio classification is one of the growing research fields with a lot of day-to-day applications with the increase in the large-scale content base multimedia indexing and retrieval [13, 16, 10]. Audio samples can be structured data on unstructured data. Structured audio data consists of data with information such as human speech and instructions. Unstructured audio data includes sounds with no structure such as car horn, air conditioner, siren noise. Most of the research in this field restricts themselves to either of the two probelms. The work using the structured audio data exploits the sequential nature of the data where as the unstructured data exploits the randomness in the data. But little effort was made in building a consolidated model that works both for structrued and unstructed data. This project is a step towards that. In this model were developed that perform decently well both on structured and unstructure data. For this purpose 2 datasets UrbanSound8K[1] and Tensorflow Speech recognition challenge[2] dataset were consolidated and used to train the models. We experiment with various Convolutional Neural network models and try to boost the accuracy. The organization of the report is as follows. Section 2 gives a detailed description of the datasetes used and the way data was split into train,validation and testing. section 3 talks about the initial proposal and how changes were made in it to improve upon the models. section 4 gives an idea about the features. Section5 talks about the data preprocessing steps. Section 6 gives an idea of the model architectures, hyperparameters, training and validation accuracy and loss plots for different models. section 7 talks about the drawbacks of the models. section 8 shows the instructions for training the model using the GITHUB repository and how to use the GUI. It also contains a short recording showing how the GUI works. The the last section gives a short conclusion to the project.

2 Dataset Description

2 datasets UrbanSound8K[1] and Kaggle 'Tensorflow Speech Recognition challenge[2] datasets were used. Both the datasets contain audio clips with their corresponding labels. A detailed description of both the datasets is given in this section.
UrbanSound8K[1] dataset contains 8732 labeled sound excerpts under 4 seconds length from 10 different classes ('air conditioner', 'car horn', 'children playing', 'dog bark', 'drilling', 'engine idling', 'gun shot', 'jackhammer', 'siren', and 'street music'). The sampling frequency is different for each sample and is provided in a adjoining csv file with the data. Most of the data has a sampling frequency close to or less than 8000hz. All excerpts are taken from field recordings uploaded to 'www.freesound.org'. The files are pre-sorted into ten folds (folders named fold1-fold10). But for our purpose of the project split each fold into train and test data using a 90-10 split and the train data is further split into train and validataion data using a 90-10 split. Then all the train, test and validation data is consolidated.
Tensorflow speech recognition[2] dataset contains 105,872 labeled audio clips of 1 second length with 35 classes('down','learn', 'right', 'nine', 'eight', 'dog', 'bird', 'house', 'marvin', 'zero', 'sheila', 'bed', 'follow', 'off', 'happy', 'backward', 'on', 'cat', 'left', 'five', 'visual', 'one', 'no', 'two', 'yes', 'forward', 'tree', 'three', 'go', 'seven', 'six', 'wow', 'stop', 'four', 'up'). Each file consisted an audio recording of one of the speech command mentioned above. The sampling frequency of each clip in this dataset is 16000hz. The data set also contains two files which has the file names of the test and validation samples. So those files were filtered from the data and used correspondingly for validation and testing of the models.

3 Changes from the initial plan

Initial plan: The initial plan was to implement a 1D CNN directly on the audio files as a base model. As an improvement implementing a 2D CNN on the features extracted from the audio data (mel coefficients) just on the "Urban sound 8k dataset".
Further to extend it to speech recognition, use the same model and train it on another database available on kaggle for the "speech recognition challenge".
The plan was to initially train the model on the Urban sound dataset which has 10 classes. Then replace the last dense layer of the model with a different dense layer which has 35 classes (corresponding to the classes in speech recognition challenge). The last step would be to use a 45 class dense layer that is trained on all the data in the end to estimate the combined accuracy of the network recognizing both the urban noises and speech commands

Revised plan: Since I am trying to build a network that classifies both structured and unstructured audio clips I realized a better approach to the problem would be to use both the datasets train the same network and work on improving the accuracy of the network one a baseline working CNN was built. So I started by building a 2D network on the extracted features as a base model. The first step was to merge both the datasets, since the datasets have different audio clip properties such as sampling rate and clip duration. Then extract the features from them. The next step is to optimize a 2D CNN using these features and improve the accuracy by varying different features and the architectures of the CNN.

4 Feature description

All the models are built on features extracted from the audio clips. Features extracted from the signals can be classified as spectral ,rhythm features. For this project I concentrated on using just the spectral features. These are the spectral features that can be extracted from the audio data are power spectogram, Mel fequency cepstral coefficients (MFCCs), melspectrogram, chromagram, constant-Q chromogram, chroma energy Normalized statistics(CENS). A brief description of the features is provided below. A audio processing python library librosa[3] was used to extract these features from the clips.

4.1 Signal plot:

figures 1 and 2 are the plots of one example from both the datasets. The first sample is from the urban noise dataset and the second one is a sample from the speech recognition dataset. In the following subsections, different spectral feature representation of these samples are shown. Spectral features have a parameter known as the number of bins which is a representation of the resolution of the audio signal used to extract features. For example, using 15 bins divides the clip into 15 consecutive chunks and extracts the features from them. This spectral resolution was also considered a hyper parameter in the project. Another parameter is the bin size which is the resolution of the bin. This was also considered a hyperparameter and experimented with.

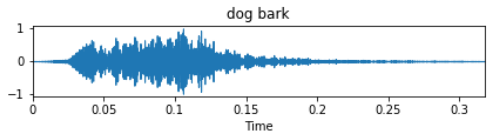

Figure 1: Dog bark

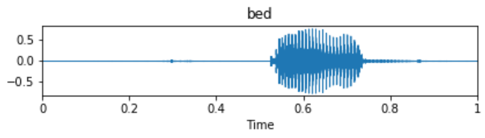

Figure 2: Bed

4.2 Spectrogram:

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. Spectrograms are generally used to identify words phonetically and various noises and animal calls. Spectrograms are made using band-pass filters, Fourier transform or a wavelet transform. A spectrogram is usually depicted as a heat map, i.e., as an image with the intensity shown by varying the colour or brightness. Figures 3 and 4 show the spectrogram of the above samples.

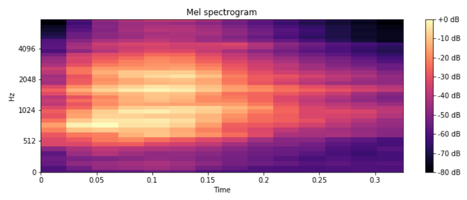

Figure 3: Dog bark mel spectrogram

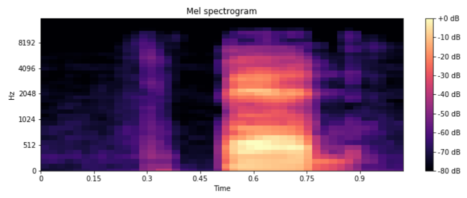

Figure 4: Bed mel spectrogram

4.3 Mel frequency cepstral coefficients:

Mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. Figures 5 and 6 show the MFCCs of the samples.

4.4 Chromogram:

Chromagram closely relates to the twelve different pitch classes. Chroma-based features, are a powerful tool for analyzing pitches which can be meaningfully categorized (often into twelve categories) and whose tuning approximates to the equal-tempered scale. One main property of chroma features is that they capture harmonic and melodic characteristics. Figures 7 and 8 show the chromogram]

4.5 Constant-Q Chromogram:

Constant Q chromogram is a chromogram with a constant Q transform. Constant Q transform can transform the data into frequency domain. Constant Q chromogram uses a logarithamically spaced frequency scale. Figure 9 and 10 show a constatnt Q cipe

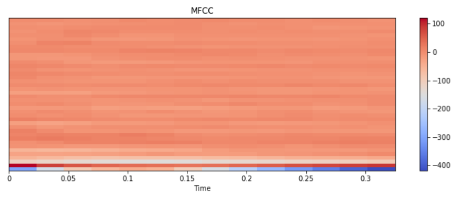

Figure 5: Dog bark MFCC

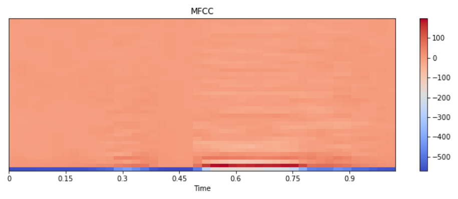

Figure 6: Bed MFCC

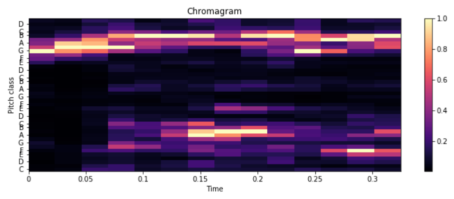

Figure 7: Dog bark chromogram

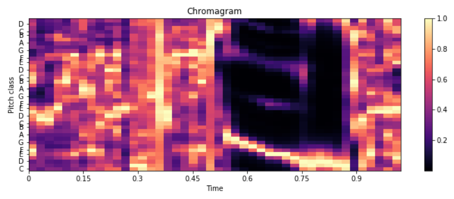

Figure 8: Bed chromogram

4.6 Chroma Energy Normalized statistics(CENS):

The main idea of CENS features is that taking statistics over large windows smooths local deviations in tempo, articulation. CENS are best used for tasks such as audio matching and similarity.

5 Data preprocessing:

The first step of data preprocessing performed was to equalize the sampling rate. Each sample in the urban noise dataset had a variable sampling rate but it was noticed that most of the clips gave a good quality audio representation at 8000hz. So all the samples in the dataset were downsampled to 8000hz. Similarly all the samples in the speech recognition dataset had a sampling frequency of 16000hz. They were also downsampled to 8000hz. There are two reasons behind this step. one was to merge the two datasets to that the

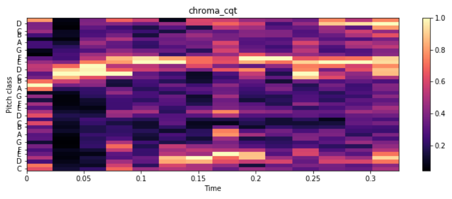

Figure 9: Dog bark constant Q chromogram

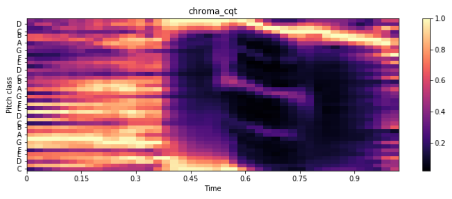

Figure 10: Bed Constant Q chromogram

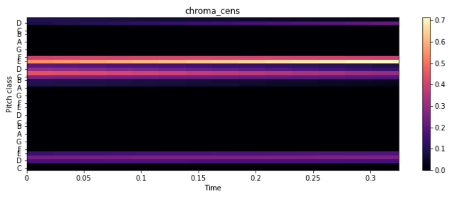

Figure 11: Dog bark CENS

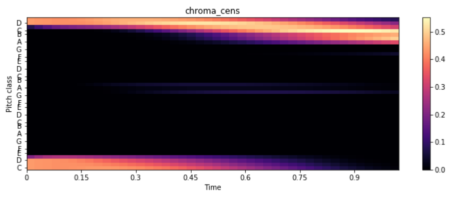

Figure 12: Bed CENS

feature extraction process can become easier. The second being the feature extraction process increases exponentially with time as the sampling rate increases. Therefore having a lower sampling helps in improving the data processing time. Although the accuracy of the model could be higher by using a higer frequency, a trade off was made due to time and hardware restrictions.

The next step was to extract the features from the merged dataset. For this python audio processing library "librosa" was used. All the above mentioed features were extracted and various models were build using either one of the features or multiple of them. These extracted features were correspondingly classified into train, test and validation sets for each label class and saved as .npy file. For training the models these npy files were loaded and then models were fitted.

After merging the datasets, the combined dataset has 45 classes. The following are the labels in order correspondingly according to their class number

['down', 'learn', 'right', 'nine', 'eight', 'dog', 'bird', 'house','marvin', 'zero', 'sheila', 'bed',

’follow’, ’off’, ’happy’, ’backward’, ’on’, ’cat’, ’left’, ’five’,’visual’, ’one’, ’no’, ’two’, ’yes’, ’forward’, ’tree’,’three’, ’go’, ’seven’, ’six’, ’wow’, ’stop’, ’four’, ’up’,’air_conditioner’,dog_bark’, ’street_music’, ’car_horn’, ’drilling’, ’children_playing’, ’siren’, ’engine_idling’, ’jackhammer’, ’gun_shot’]

6 Model Descriptions, Training curves and results:

Multiple models were built using different architectures. In the following section model descriptions, their corresponding training curves, accuracies and losses and confusion matrices

6.1 Model 1 : Baseline model

The first model was built using only one feature MFCCs. Initial model was established with 15 bins and a bin size of 20. So with these as the data related hyper parameters, the input size would be (None,20,15,1). The model that was built has the following architecture.
These are the training parameters for the model:

epochs	200
loss	categorical crossentropy
metrics	accuracy
optimizer	AdaDelta

The following are the training loss and accuracy curves.

The following are the train, validation and test losses and accuracy at the point where validation accuracy reached it’s maximum.

	Accuracy	loss
Train	0.9709	0.1019
Validation	0.8036	0.9778
Test	0.7819	1.1269

6.2 Model 2:

6.2.1 Model 2a:

After establishing a baseline model we experimented with various hyper parameters, adjusted the number of layers, kernel sizes of the layers, number of kernels in each layers, dropout percentages, number of dense layers, number of connections in the dense layers. The second model was made which has the following architecture.
The optimal hyper parameters used are as follows:

number of bins	15
Bin size	40
kernel size	2x2
Dropout	0.4

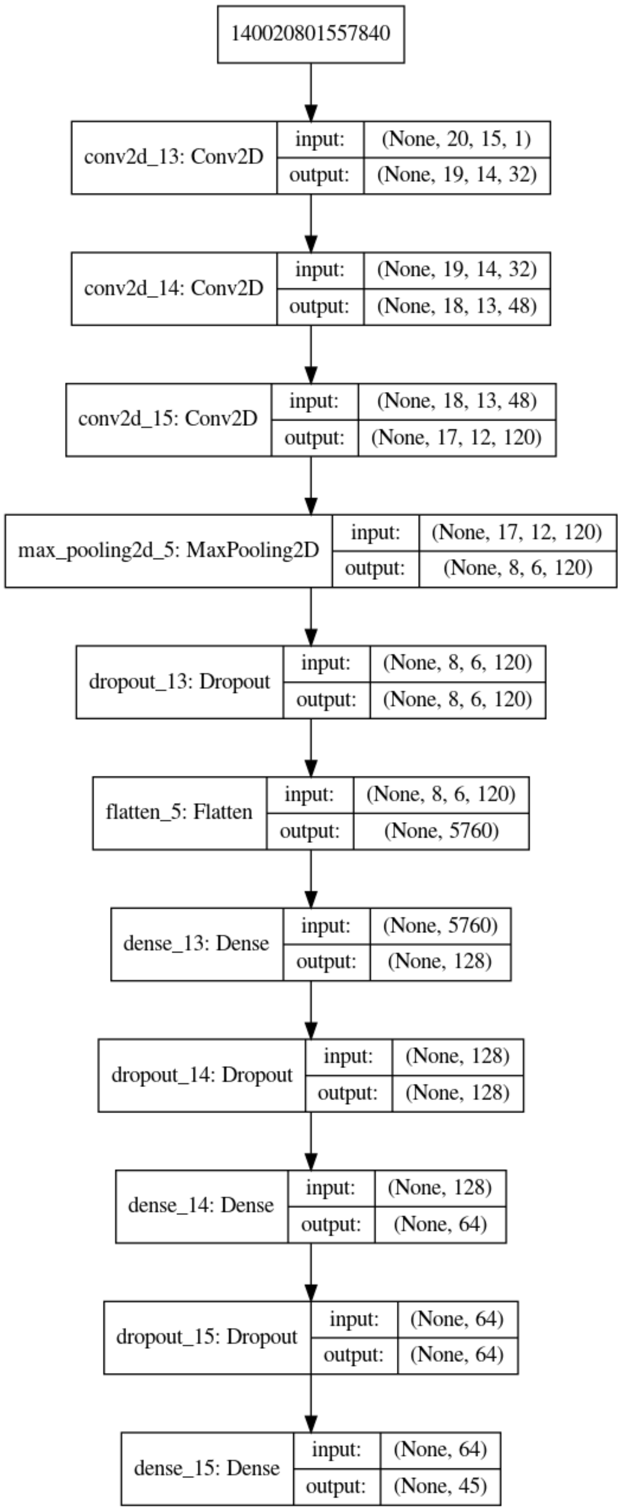

Figure 13: Baseline model

These are the training parameters for the model:

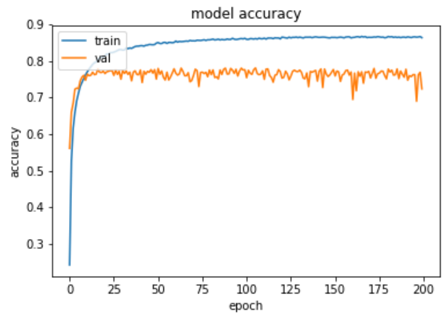

Figure 14: Baseline model accuracy

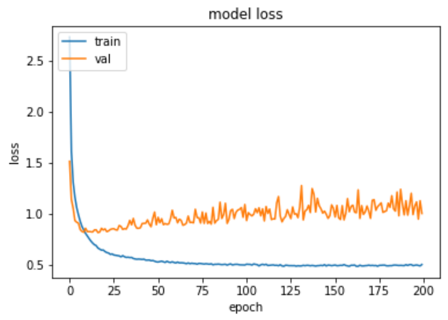

Figure 15: Baseline model loss

epochs	200
loss	categorical crossentropy
metrics	accuracy
optimizer	AdaDelta

The following are the training loss and accuracy curves.
The following are the train, validation and test losses and accuracy at the point where validation accuracy reached it's maximum.

	Accuracy	loss
Train	0.97328	0.2410
Validation	0.8492	0.6015
Test	0.8334	0.6722

6.2.2 Model 2b:

In the same model architecture the number of features used were also increased in experimented. Increasing the features didn't seem to improve the results significantly. So I reverted back to using just MFCCs as the main feature. In this case all the features used 15 bins and bin size as 40. and The input size in this case would be (None,40,15,5) -since the features are sent to the network as 5 different channels.

	Accuracy	loss
Train	0.9431	0.2116
Validation	0.8533	0.5672
Test	0.839	0.603

6.3 Model 3:

In this model batch normalization was added to model to see if it increases the testing accuracy. There was approximately 1 percent increase in the accuracy from the above model. Along with batch normalization, training learning rate was reduced by a factor of 0.1 if the validation accuracy doesn't increase for 10 epochs. Although this doesn't increase the final validation accuracy by a significant amount, it helped in converging faster to that accuracy. The architecture is as follows
The model accuracy and loss after 45 epochs are as follows.

	Accuracy	loss
Train	0.9291	0.2386
Validation	0.8588	0.5344
Test	0.839	0.603

6.4 Additional Data augmentation for model 2:

Additional data augmentation was used to try and improve accuracy. This was done by adding a random background noise to the data. One of six background noises (cleaning dishes, dude miaowing, exercise bike,pink noise, running tap, white noise) is chosen at random and is added as background noise to the original data using a random volume level between 0-10 percent of the full volume level. Another strategy for data augmentation was using a time offset. A random time offset was chosen and the data was offset by that amount. Also a silence class was added which has a size of 10 percent the size of the speech data set. Once the data augmentation is performed then the feature extraction was performed. Train, test and validation accuracies in this case are as follows. The solution for this could be to try and balance the dataset by finding more datasets that contains urban sounds data, or use some data augmentation on the urban noise dataset but not speech recognition dataset. It was also noticed that the models well when trained seperately on either the speech commands data or the urban sounds data. Thus, further models can be built using an ensemble strategy using seperately trained models.

	Accuracy	loss
Train	0.9917	0.0121
Validation	0.8942	0.7420
Test	0.8798	0.8956

7 Setbacks and solutions

Since in this project we are merging two datasets, there was a lot of information loss while equalizing the sampling rate. If the sampling rate was increased there is a chance that the accuracy of the networks can increase. Also the models are more biased towards the speech commands since the data containing speech commands is a lot more compared to the urban noise data. Approximately 80000 samples were used for 35 speech command classes and 6000 samples were used for 10 urban sounds classes.

8 Instructions for the code and testing using GUI

8.1 Installing dependencies:

Install the following dependencies into a python anaconda environment or your local client before you run the GUI.

- numpy (pip install numpy)
- Librosa (pip install librosa)
- keras (conda install -c conda-forge keras)
- tkinter (sudo apt-get install python3-tk)
- ttkthemes (pip install ttkthemes)

8.2 Execution

To run the GUI for using the neural network, after installing the above dependencies navigate to the GUI folder and run the main.py file. (python main.py). A link to a demo is given below.

8.3 Annotated code

The code for all the models along with the data processing is uploaded into the github repository which is provided below. The code is provided as annotated jupyter notebooks. The github repository also consists of raw data and extracted features with and without the data augmentation, saved models and the code for the GUI and the images used in the report.

8.4 Github repository

CSCE636-project

8.5 GUI Demo link

GUI Demo

9 Conclusion

In this project a neural network was built that classifies both speech commands and urban noises. Neural networks were trained by merging two datasets and by extracting features from that contain essential information. Further by using data augmentation such as background noise and random offsets, accuracy was boosted by approximately 5 percent. A final test accuracy of 88 percent was achieved.

References

[1] J. Salamon, C. Jacoby and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research", 22nd ACM International Conference on Multimedia, Orlando USA, Nov. 2014.

[2] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

[3] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

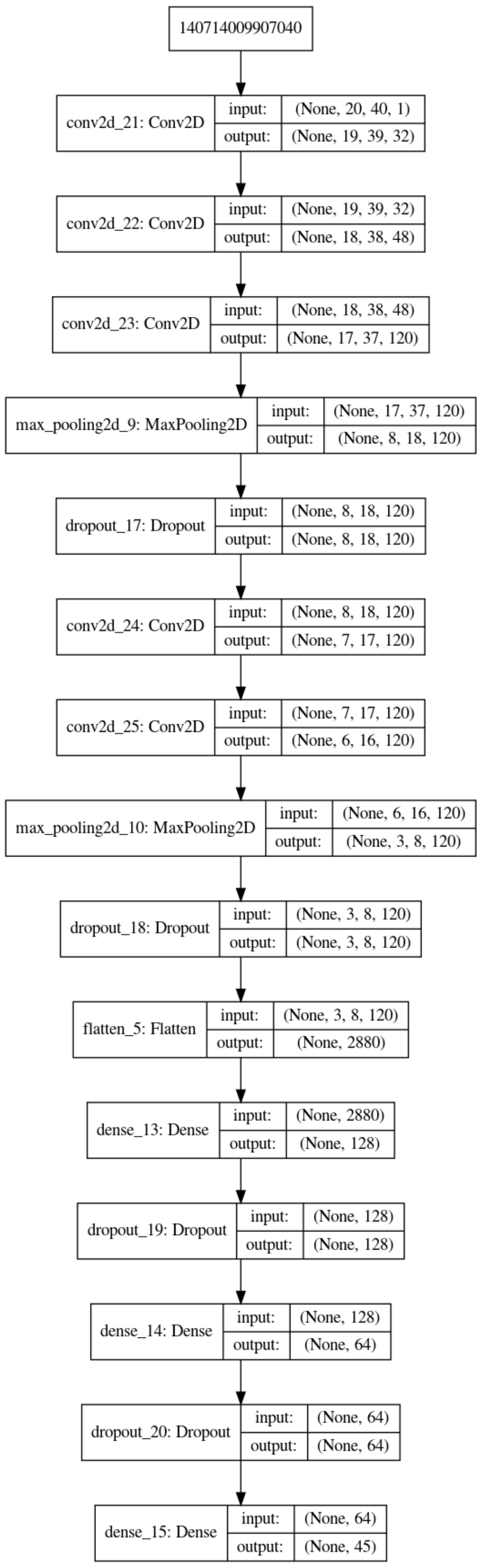

Figure 16: Model 2

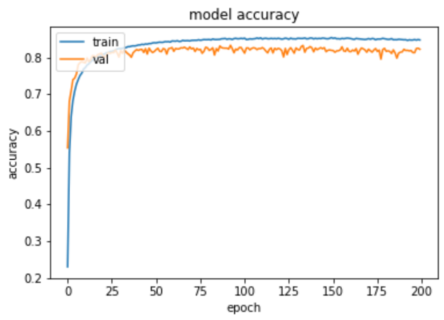

Figure 17: Model 2 accuracy

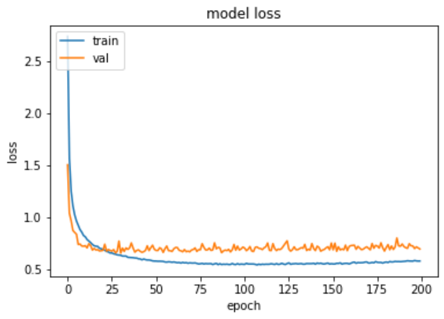

Figure 18: Model 2 loss

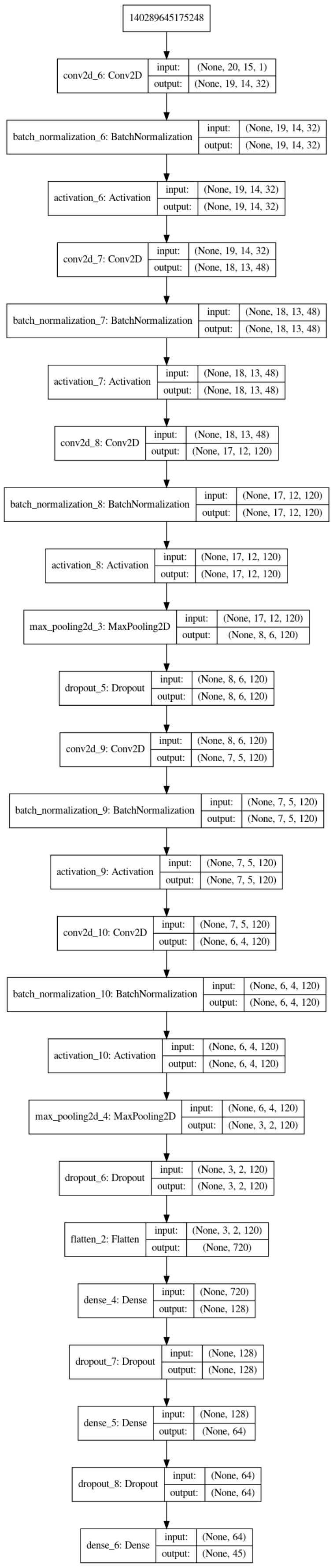
140289645175248
conv2d_6: Conv2D
input: (None, 20, 15, 1)
output: (None, 19, 14, 32)
batch_normalization_6: BatchNormalization
input: (None, 19, 14, 32)
output: (None, 19, 14, 32)
activation_6: Activation
input: (None, 19, 14, 32)
output: (None, 19, 14, 32)
conv2d_7: Conv2D
input: (None, 19, 14, 32)
output: (None, 18, 13, 48)
batch_normalization_7: BatchNormalization
input: (None, 18, 13, 48)
output: (None, 18, 13, 48)
activation_7: Activation
input: (None, 18, 13, 48)
output: (None, 18, 13, 48)
conv2d_8: Conv2D
input: (None, 18, 13, 48)
output: (None, 17, 12, 120)
batch_normalization_8: BatchNormalization
input: (None, 17, 12, 120)
output: (None, 17, 12, 120)
activation_8: Activation
input: (None, 17, 12, 120)
output: (None, 17, 12, 120)
max_pooling2d_3: MaxPooling2D
input: (None, 17, 12, 120)
output: (None, 8, 6, 120)
dropout_5: Dropout
input: (None, 8, 6, 120)
output: (None, 8, 6, 120)
conv2d_9: Conv2D
input: (None, 8, 6, 120)
output: (None, 7, 5, 120)
batch_normalization_9: BatchNormalization
input: (None, 7, 5, 120)
output: (None, 7, 5, 120)
activation_9: Activation
input: (None, 7, 5, 120)
output: (None, 7, 5, 120)
conv2d_10: Conv2D
input: (None, 7, 5, 120)
output: (None, 6, 4, 120)
batch_normalization_10: BatchNormalization
input: (None, 6, 4, 120)
output: (None, 6, 4, 120)
activation_10: Activation
input: (None, 6, 4, 120)
output: (None, 6, 4, 120)
max_pooling2d_4: MaxPooling2D
input: (None, 6, 4, 120)
output: (None, 3, 2, 120)
dropout_6: Dropout
input: (None, 3, 2, 120)
output: (None, 3, 2, 120)
flatten_2: Flatten
input: (None, 3, 2, 120)
output: (None, 720)
dense_4: Dense
input: (None, 720)
output: (None, 128)
dropout_7: Dropout
input: (None, 128)
output: (None, 128)
dense_5: Dense
input: (None, 128)
output: (None, 64)
dropout_8: Dropout
input: (None, 64)
output: (None, 64)
dense_6: Dense
input: (None, 64)
output: (None, 45)

Figure 19: Model 3