COMPUTER VISION ASSIGNMENT 2

REAL-TIME FACE EMOTION RECOGNITION

May 23, 2022

Amit Jadhav i6306739

Tom Scholer i6223586

1 Introduction

In this assignment the application of Convolutional Neural Networks (CNN) is explored by executing the task of facial emotion recognition in real-time from the video. The main goal is to experiment with different architectures and come up with a final model that could best classify the emotions displayed in the FER2013 data set. Additionally the model's accuracy was tested on frames that were extracted from short videos or from the webcam feed of the authors. In the following sections the results of different models are presented together with an analysis of those results.

2 Implementation

In order to realise this project the programming language Python was used. The implementation including training, validation and testing of the models on the FER2013 Dataset was done on a Notebook using Google Colab because of the great accessibility for both students to contribute to the code. Google Colab was also chosen because of the ability to train the models within reasonable time by using their GPU at runtime.

For the deep learning framework Tensorflow and Keras were chosen because of their easy to use and straight forward way of building model architectures. Other python libraries that were used include Numpy, Matplotlib and Pandas. The trained models were then saved using pickle package and these models were tested on webcam video feed using Pycharm tool. This part of the assignment was done on Pycharm due to the restrictions placed on Google Colab on loading the webcam feed via

```
cv2.videoCapture(0)
```

for security reasons. This part of the code is therefore shared as a python file by the name "VideoCaptureOpenCV.py"

3 Data acquisition & preprocessing

1. The dataset FER2013 was used to train, validate and test the models. This dataset was acquired from Kaggle in form of a CSV file. The Data had to be restructured from a flat array of 2384 pixel values to a 48 x 48 two dimensional array that comprised the pixels of the image. Before feeding the data into the models. The pixel values were regularized by dividing each value by 255. Regularization was used as a method to prevent over-fitting.

2. The data was split into a training set, validation set and test set. The training set which comprised 28709 examples was used to train the weights of the model over several epochs.

The validation set which comprised 3589 examples was used to evaluate the model after each training epoch in order to determine the loss metric. The test data which comprised 3589 examples as well was used to evaluate the model after training was completed.

3. Data augmentation was performed to create additional training data using the 'ImageDataGenerator' class from keras which allowed to randomly change the images. The changes to the images include rotations, zooms, flips and shifts. For every training epoch the images will be randomly manipulated which gives the effect that the model is trained on a multiple of the original images thus preventing over-fitting and better accuracy on the test data.

4 Models

4.1 Model Architecture

Each model is build using the Keras framework by sequentially adding different layers one after another. The main part of the models are the convolution layers whose weights are trained in order to detect features. Convolution layers that are placed at the front of the model will learn to detect low level features like differently oriented lines or curved lines while convolution layers towards the end of the model will learn to detect high level features that combine low level features like for example eyes, a nose or a mouth.

Convolution layers were in most cases followed by max pooling layers which reduce the data for the following layer and guarantee the inclusion of peak values by taking the maximum across a 2x2 pool of pixel values that correspond to feature locations.

Dense layers which were used at the end of the models are fully connected neural networks that receive a flattened row of values and are trained to predict the correct emotion of the received data. The dense layer uses an activation function like Relu or Softmax and assigns a percentage to the 7 available categories where the Softmax activation is given by, $\max\left(\frac{e^{ActivationValue}}{\sum_{n=1}^{No.ofPerceptrons} e^{Activationvalue}}\right)$ on the visible output layer assigns the label with the highest value determines the prediction.

In the following sections the four best performing models can be inspected.

Model 1 (see figure 11)

Model 2 (see figure 12)

Model 3 (see figure 13)

Model 4 (see figure 14)

Model 5 (see figure 15)

4.2 Training the models

The models were trained for a maximum of 200 epochs. During each epoch the model goes through the full training set which is fed to the model in batches of 32. While training, the model keeps track of four metrics which are: the prediction accuracy of the training data, the prediction accuracy of the validation data, the loss for the training data and the loss for the validation data.

The learning rate was reduced by a factor of 4 (i.e. LR divided by 4) when validation loss

stagnated for at least 5 epochs using keras function **ReduceLROnPlateau** and whenever no improvements on the validation loss were observed for 15 epochs, the Keras function **EarlyStopping** served to stop training before 200 epochs were reached. It was observed that most models converged to about 99% of the optimal loss in around 70 epochs.

The optimization algorithm used was Adam (Adaptive Moment Estimation). Categorical Cross-entropy was used as the loss function.

5 Experiments

Experiments were conducted in order to observe the impact on the metrics that were tracked during training and testing by changing several hyper parameters. Figure 1 shows the results of these experiments.

1. Different learning rates were tested where a learning rate of 0.0005 resulted in the best performance, higher learning rates resulted in slower learning and needed many more epochs in order to achieve the same outcome.

2. The amount of convolution layers on a basic model can be seen in figure 1b where for this specific problem we could not observe a big difference between the different settings. It seems that a higher number of convolution layers achieves better results on the validation loss therefore the decision was made to stick with four convolution layers for the final model.

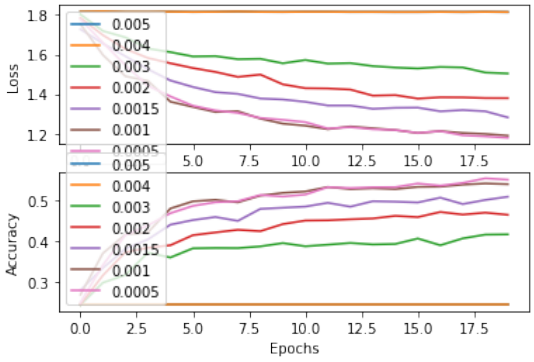
3. The kernel size that worked best was a $3x3$ kernel as shown in figure 1c. The kernel size of $5x5$ allowed for lesser number of convolution layers to be added since the layer dimensions were lost by a factor of 4 pixels after the first convolution from $48 * 48$ to $44 * 44$. In Model 5 Batch normalization was tried after the convolution layer but this approach did not increase the accuracy much while increasing the size of the model to almost 10 times in terms of total parameters.

4. Other optimizer functions such as Adamax and SGD were tried but Adam gave the best results for this use case.

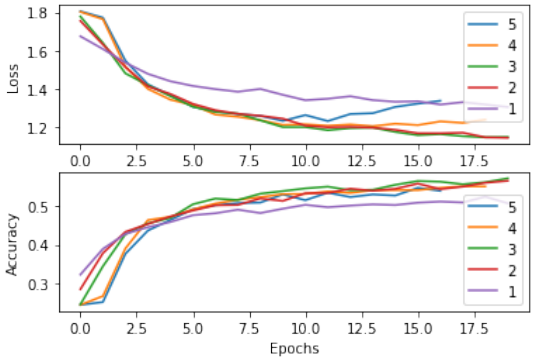
5. Different dropout layers were also tested (with dropout proportions ranging from 0.25 to as high as 0.8) and they were found to help well to prevent over-fitting by dropping the weights of a certain amount of neurons while training. However, they slowed down training the models and therefore they were used to a limited extent.

6 Results

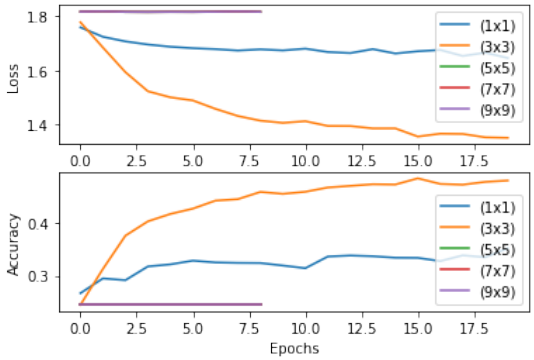
The results of the different models that were tested can be seen in table 1. Model 5 outperformed the other models on most metrics. However it worked with 278 Million parameters and was quite heavy with only a marginal improvement in the results. We therefore consider model 4 to be our best model. Models 1 through 4 share a similar architecture, the filter sizes in the convolution layers of these models gradually increase for each convolution layer which results in a much better performance on all the metrics. Model 3 and 4 achieved a higher accuracy while training and testing both. This might be due to the fact that model 1 and model 2 filter size starts off relatively



(a) Learning rate effects



(b) Number of convolution layers effect



(c) Kernel size effects

Figure 1: Effects of parameters on prediction performance

small on the first convolution layer and gradually increases whereas model 3 and 4 start off higher then gradually increase. We also observed that convolution filter size of 3x3 works better than a 5x5 filter. Since the input images are relatively small a lower filter size seems the way to go since we observed that higher filter size needs more time to train and the performance outperformed by lower filter size. Some of the classifications on the test data of model 5 can be seen in figure 2.

	Accuracy	Precision	Recall
Model 1	0.549178	0.497028	0.408095
Model 2	0.633881	0.601376	0.588281
Model 3	0.624407	0.624292	0.565599
Model 4	0.643912	0.602411	0.594083
Model 5	0.653664	0.642169	0.602060

Table 1: Performances on test data

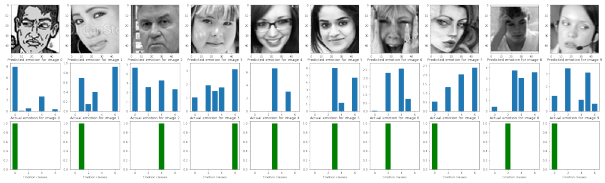


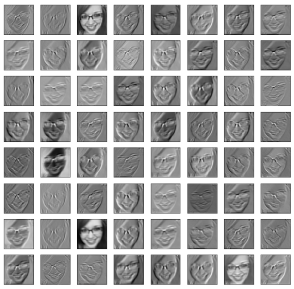
Figure 2: Prediction results best Model - Model 5

7 Visualisation of the learnt filters on different layers

In figure 3 the features that were learned in the different convolution layers can be inspected. It can be observed how the human face is broken down into feature maps of different scale in some of the layers of the network.

8 WebCam emotion recognition

The trained models were run on the webcam feed of one of the authors and his spouse by using the videoCapture function from cv2 and the predicted emotion was overlayed on to the video feed in real time using the putText function in cv2. The results were best for detecting happyness and neutral emotions. The model detected almost all emotions when given enough time except for confusion which alternated most of the times between



(a) 1st convolutional layer output

(b) 2nd convolutional layer output

(c) 3rd convolutional layer output

(d) 4th convolutional layer output

Figure 3: Feature maps of the outputs

anger and disgust. The various emotions have been given as snapshots in 7 figures: fig 4, fig 5, fig 6, fig 7, fig 8, fig 9 and fig 10.

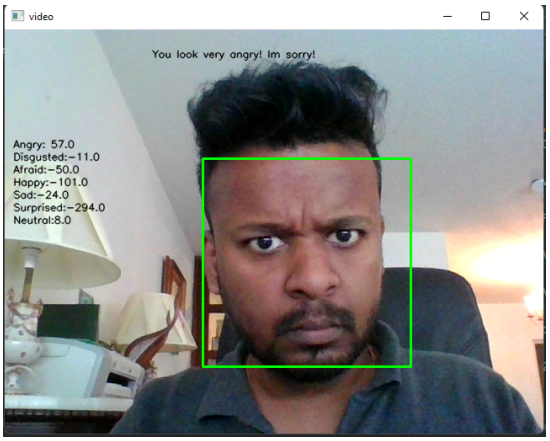


Figure 4: Angry (Notice Disgust values here)

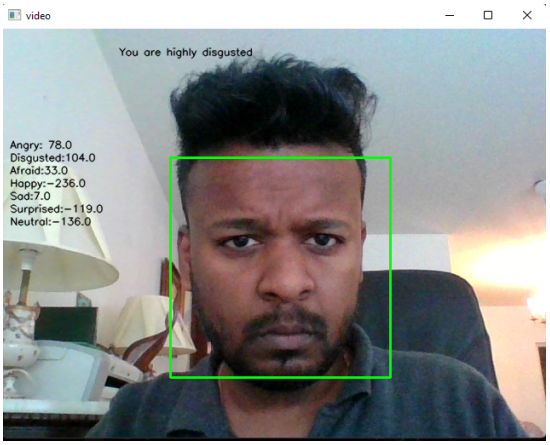


Figure 5: Disgusted (Notice Anger values here)

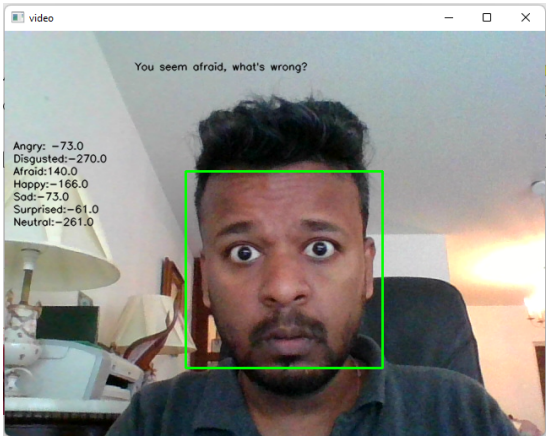


Figure 6: Afraid

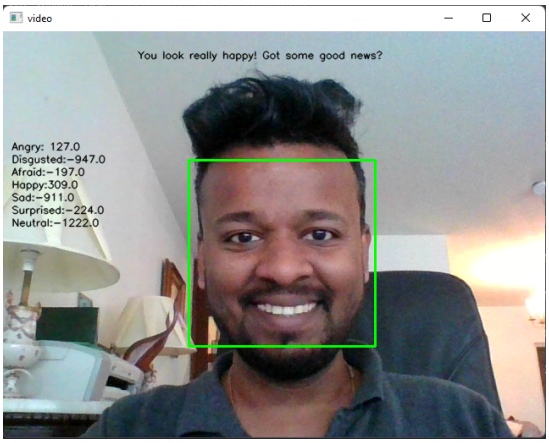


Figure 7: Happy

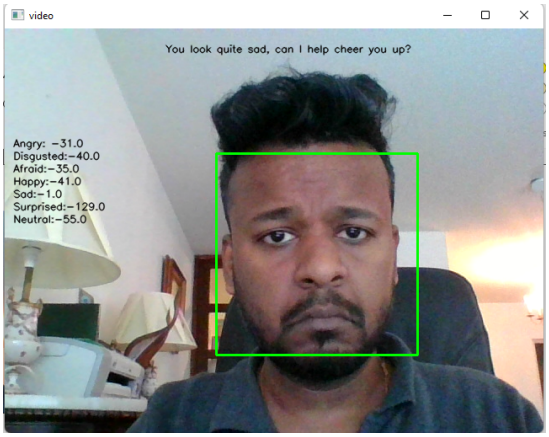


Figure 8: Sad

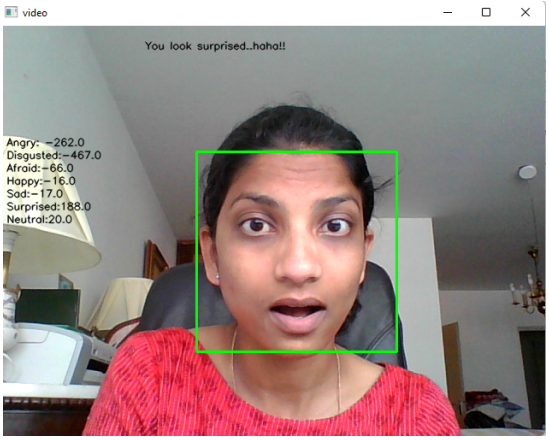


Figure 9: Surprised

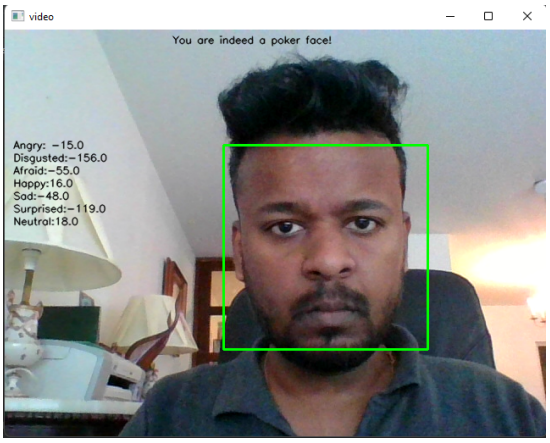


Figure 10: Neutral

9 Conclusion

The combination of Opencv-Python package along with Google's tensorflow empowered with Keras can serve as a quickly implementable option for real-time object recognition in videos. The one shortcoming is the availability of labelled and well balanced training datasets for these tasks. The neural network frameworks are well equipped to give outstanding accuracies with minimal validation loss given a properly balanced and diverse training data. We learnt that unlike facial emotion recognition, many such other tasks can be accomplished with very little code since the object (Webcam human face detection part) in this assignment was developed in a matter of 10 minutes using the haar-cascade classification and multi-detect combination. This same tool can be replicated with minor modifications to detect human beings first in the frames and then detect gestures/ actions since all human beings have roughly the same body shape with minor height / weight variations. One interesting application would be to have animal faces fed into the haar cascade xml files and some training datasets of happy / sad / angry etc. animal emotions.

Appendix

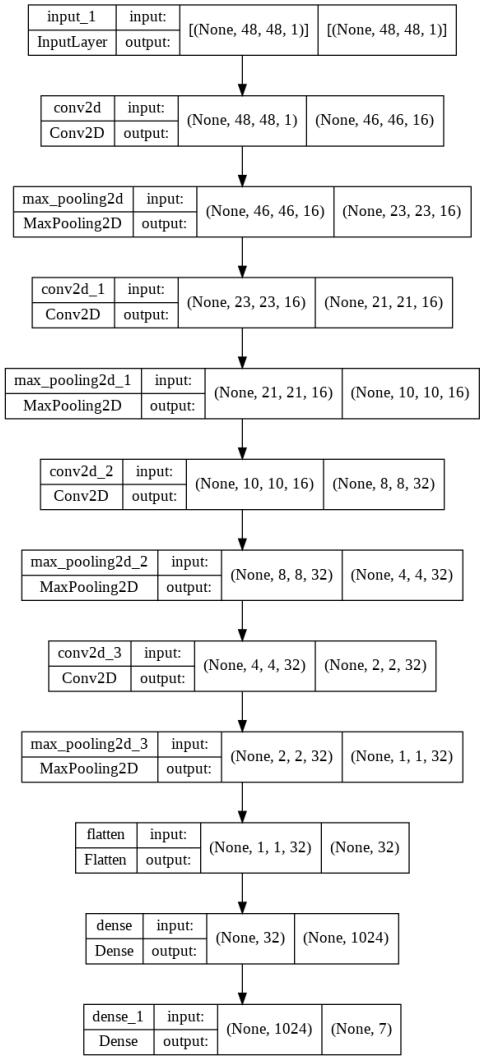


Figure 11: Model1 architecture

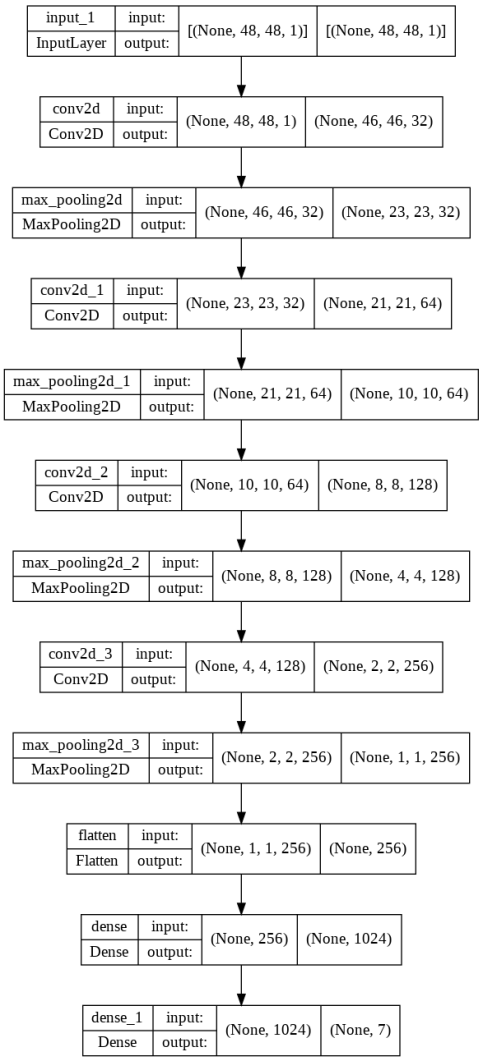


Figure 12: Model2 architecture

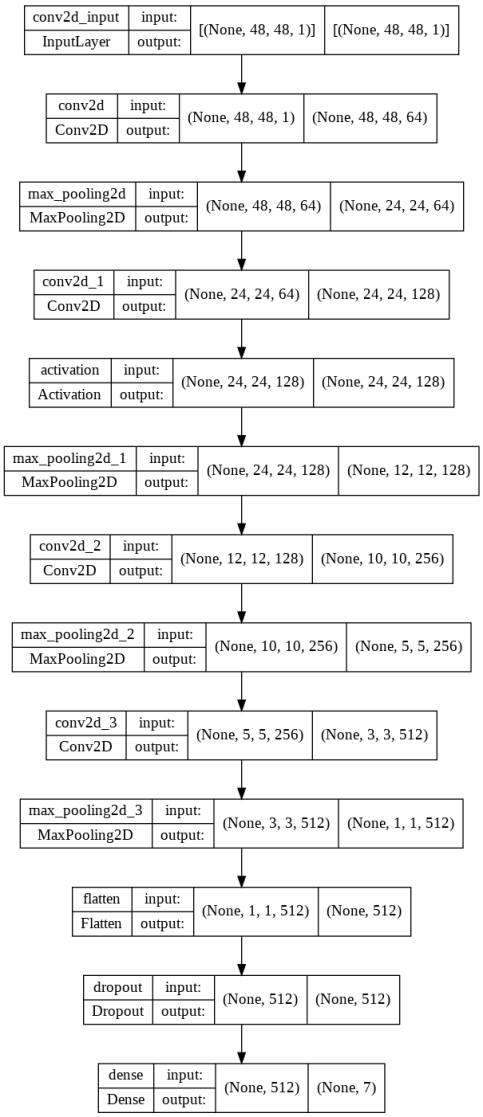


Figure 13: Model3 architecture

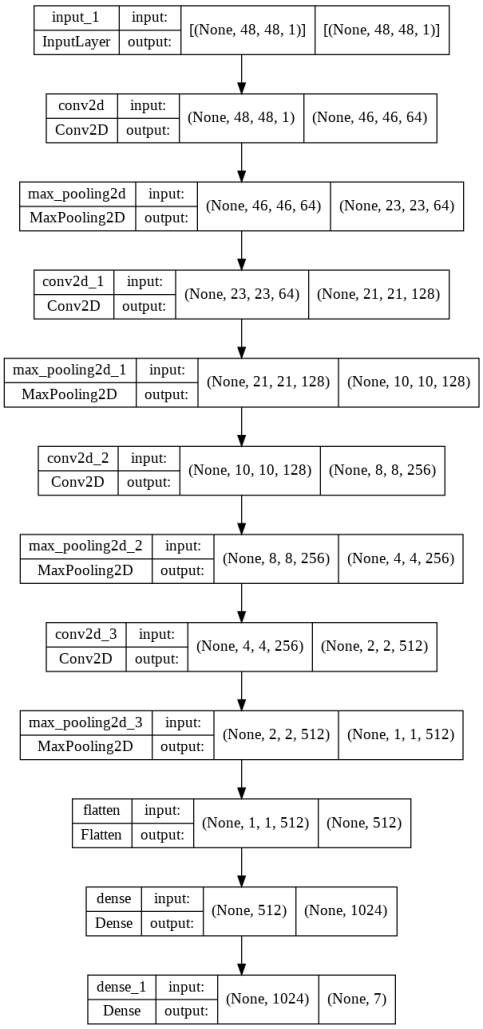


Figure 14: Model4 architecture

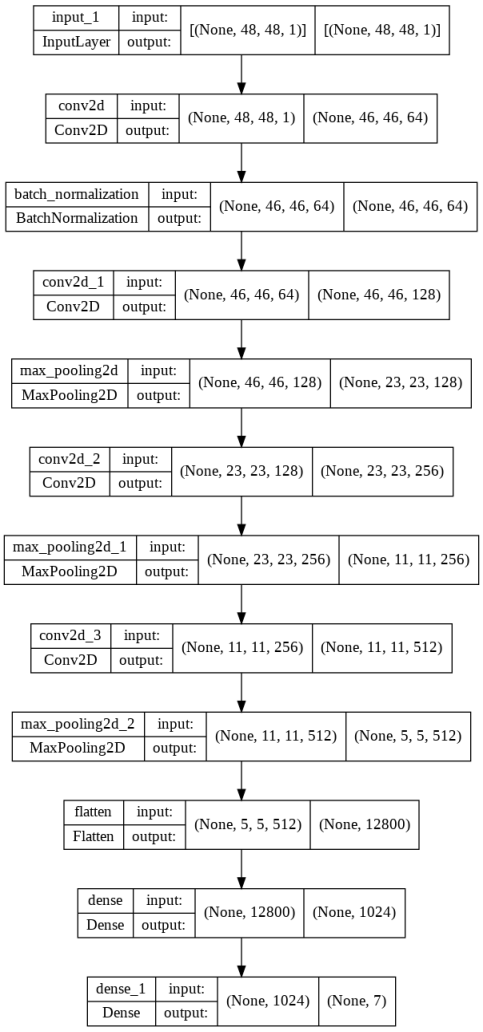
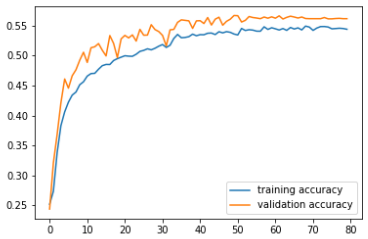
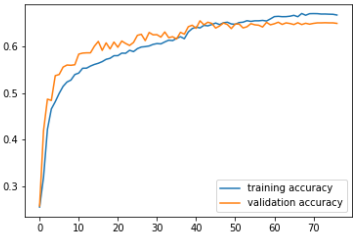


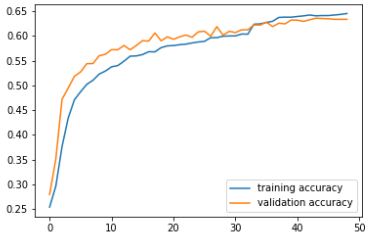
Figure 15: Model5 architecture



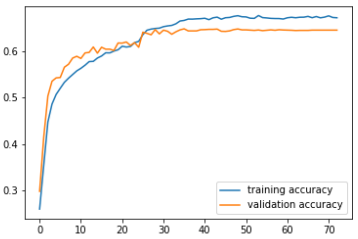
(a) Model 1 accuracy



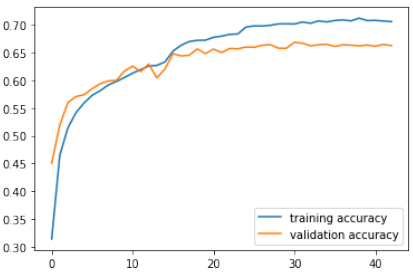
(b) Model 2 accuracy



(c) Model 3 accuracy

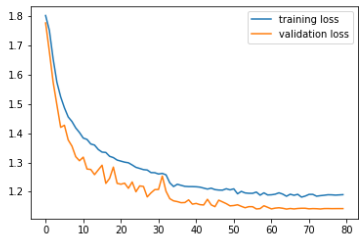


(d) Model 4 accuracy

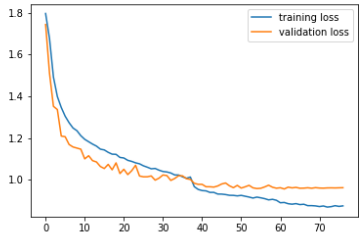


(e) Model 5 accuracy

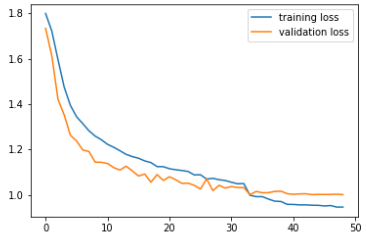
Figure 16: Training Vs. Validation Accuracy of the models



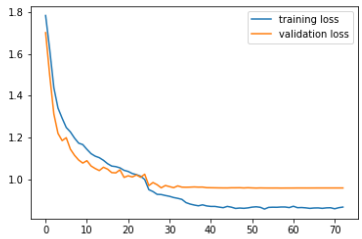
(a) Model 1 loss



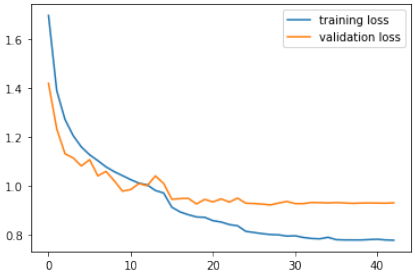
(b) Model 2 loss



(c) Model 3 loss



(d) Model 4 loss



(e) Model 5 loss

Figure 17: Training Vs. Validation Loss of the models