CV Assignment 2

Abhimanyu Anand, i6332095

May 2023

1 Introduction

This project aims to develop an emotion recognition system using Deep Learning methods for Computer Vision. The deep learning model must be capable of accurately classifying human emotions from facial images. The project consists of several stages, including data reading, data preprocessing, augmentation, designing and training several convolutional neural network (CNN) architectures; performance evaluation, and visualization of learned features.

Images have been taken from the FER Dataset available on Kaggle. Several augmentation techniques are applied, while testing, to make the model more robust. Model is then trained on several CNN architectures designed to predict emotions from the collected data. Model performance is evaluated using metrics such as accuracy, loss, precision, and recall. Additionally, visualization techniques are applied to gain insights into the learned features. These visualizations provide valuable understanding of the internal representation of each layer within the CNN model with respect to the emotions.

2 Dataset

The dataset used for this exercise is the FER dataset available on Kaggle. It has 35887 images with width and height of 48. The dataset is in a csv file, which was first converted into a numpy array and then subsequently into an image. The dataset consists of 7 classes - anger (0), disgust (1), fear (2), happy (3), sadness (4), surprise (5) and neutral (6). You can see the distribution of the classes in figure 2. A few sample images of each class from the dataset is shown in figure .

For loading the dataset, I converted the dataset into 2 parts - a train and test part. The data was randomly chosen to be the training data based on the input config parameters. We talk more about this in further sections. I used the pytorch data libraries to split the dataset.

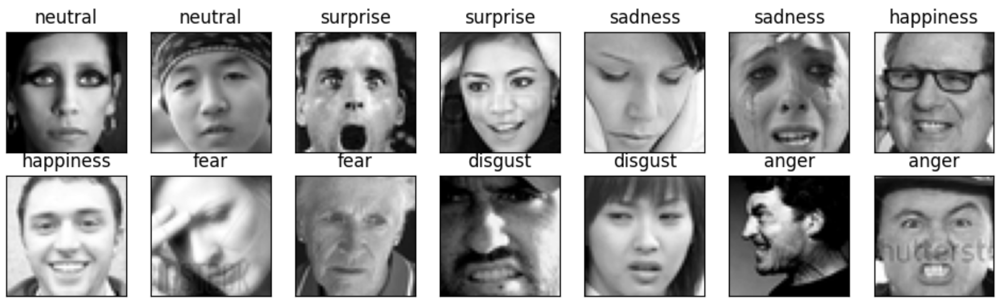


Figure 1: Dataset sample images

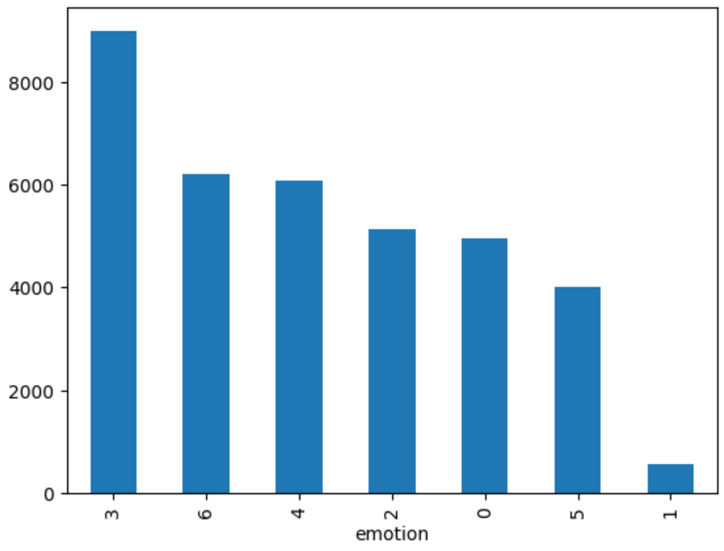


Figure 2: Dataset class split

3 Augmentation

I have implemented many CNN architectures and for each of them, I have done experimentation with a few augmentations techniques and without any augmentation techniques. Pytorch allows many augmentations for the data which include rotation, cropping, contrast and brightness adjustment, scaling and normalizing, etc.

I tried both approaches with a few architectures to see the importance of augmentation. With more augmentation and normalization I found that the train and test accuracy were closer to each other. I was avoiding data fitting with more augmentation. In this implementation, I have included brightness, contrast, and saturation changes for each image along with normalization and scaling by 0.5.

4 CNN

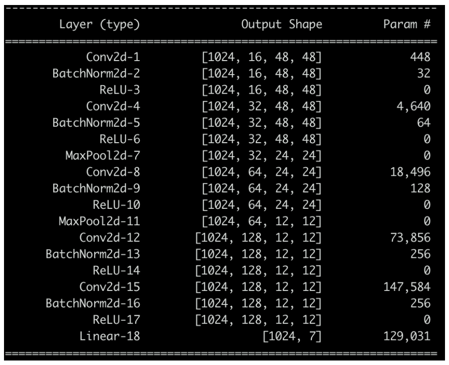
The proposed model for emotion recognition is implemented as a sequential convolutional neural network (CNN). The sequential nature of the model allows for a straightforward and intuitive organization of the different layers and operations. Each layer in the CNN performs a specific computation on the input data, gradually transforming it and extracting increasingly abstract features. I have tried 6 architectures experimenting with MaxPools and Average Pooling layers to reduce the dimensions of the input image and extract the most important features of each image for best classification.

During the training process of the models, various parameters can be adjusted to optimize performance. These include the choice of model architecture, learning rate, use adaptive learning rate or fixed, number of epochs, learning rate decay by epochs, batch size, and the use of augmentations. Additionally, the train-test ratio can be specified to allocate the data for training and testing purposes.

4.1 EmoCNN

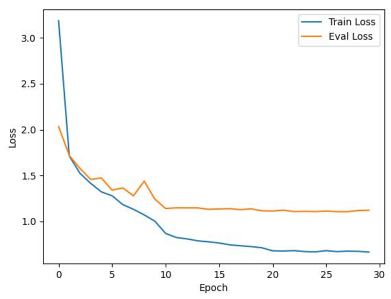
The EmoCNN consists of 5 convolutional layers followed by ReLU and batch normalizations. 3 maxpool layers are also implemented to reduce the input dimensions to focus on the most important features. The CNN layers have a stride of 1, padding of 1, kernel of size 3, and the maxpool layers have a stride of 2 and a kernel size of 2. It expects an image of depth 3 and goes upto 128 depth with image dimensions 12x12. It ends with a fully connected layer to get the final scores of each output class and it's best prediction.

You can see the whole summary in figure 3 and it's respective loss plot for train and test. You can see how the train loss reduces over many epochs but the test loss is still quite high. There is a big gap between train and test loss values.



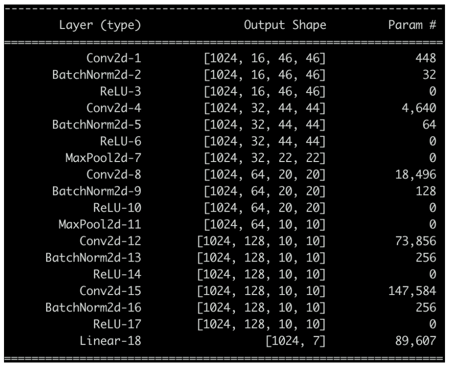
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [1024, 16, 48, 48]             448
       BatchNorm2d-2          [1024, 16, 48, 48]              32
              ReLU-3          [1024, 16, 48, 48]               0
            Conv2d-4          [1024, 32, 48, 48]           4,640
       BatchNorm2d-5          [1024, 32, 48, 48]              64
              ReLU-6          [1024, 32, 48, 48]               0
         MaxPool2d-7          [1024, 32, 24, 24]               0
            Conv2d-8          [1024, 64, 24, 24]          18,496
       BatchNorm2d-9          [1024, 64, 24, 24]             128
             ReLU-10          [1024, 64, 24, 24]               0
        MaxPool2d-11          [1024, 64, 12, 12]               0
           Conv2d-12         [1024, 128, 12, 12]          73,856
      BatchNorm2d-13         [1024, 128, 12, 12]             256
             ReLU-14         [1024, 128, 12, 12]               0
           Conv2d-15         [1024, 128, 12, 12]         147,584
      BatchNorm2d-16         [1024, 128, 12, 12]             256
             ReLU-17         [1024, 128, 12, 12]               0
           Linear-18                  [1024, 7]          129,031
================================================================
```

(a) EmoCNN summary



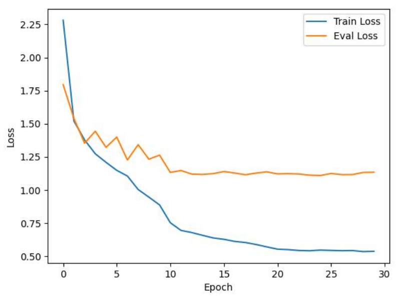
(b) EmoCNN Loss Train-Test Plot

Figure 3: EmoCNN



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [1024, 16, 46, 46]             448
       BatchNorm2d-2          [1024, 16, 46, 46]              32
              ReLU-3          [1024, 16, 46, 46]               0
            Conv2d-4          [1024, 32, 44, 44]           4,640
       BatchNorm2d-5          [1024, 32, 44, 44]              64
              ReLU-6          [1024, 32, 44, 44]               0
         MaxPool2d-7          [1024, 32, 22, 22]               0
            Conv2d-8          [1024, 64, 20, 20]          18,496
       BatchNorm2d-9          [1024, 64, 20, 20]             128
             ReLU-10          [1024, 64, 20, 20]               0
        MaxPool2d-11          [1024, 64, 10, 10]               0
           Conv2d-12         [1024, 128, 10, 10]          73,856
      BatchNorm2d-13         [1024, 128, 10, 10]             256
             ReLU-14         [1024, 128, 10, 10]               0
           Conv2d-15         [1024, 128, 10, 10]         147,584
      BatchNorm2d-16         [1024, 128, 10, 10]             256
             ReLU-17         [1024, 128, 10, 10]               0
           Linear-18                  [1024, 7]           89,607
================================================================
```

(a) Emo2CNN summary



(b) Emo2CNN Loss Train-Test Plot

Figure 4: Emo2CNN

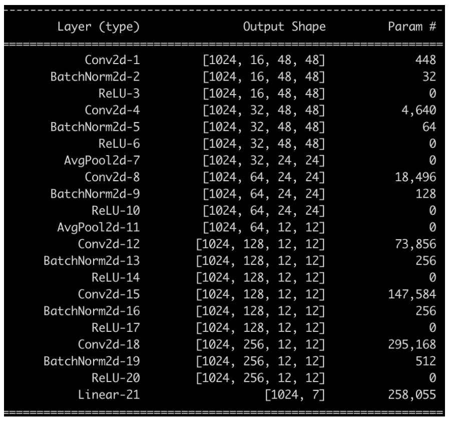
This model gives accuracy of 59.6% which is a pretty decent accuracy given the small size of the model.

4.2 Emo2CNN

The Emo2CNN has the same architecture as EmoCNN but the only differencce is in the padding value. Here there is no padding given to the architecture. So the final layer here is 10x10.

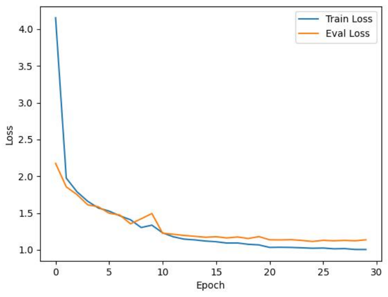
You can see the whole summary in figure 4 and it's respective loss plot for train and test. You can see how the train loss reduces over many epochs but the test loss is still quite high and it fluctuates, similar to EmoCNN.

This model gives accuracy of 58.1%. This reduction in the accuracy might be due to the features that we lose for each layer.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [1024, 16, 48, 48]             448
       BatchNorm2d-2          [1024, 16, 48, 48]              32
              ReLU-3          [1024, 16, 48, 48]               0
            Conv2d-4          [1024, 32, 48, 48]           4,640
       BatchNorm2d-5          [1024, 32, 48, 48]              64
              ReLU-6          [1024, 32, 48, 48]               0
         AvgPool2d-7          [1024, 32, 24, 24]               0
            Conv2d-8          [1024, 64, 24, 24]          18,496
       BatchNorm2d-9          [1024, 64, 24, 24]             128
             ReLU-10          [1024, 64, 24, 24]               0
        AvgPool2d-11          [1024, 64, 12, 12]               0
           Conv2d-12         [1024, 128, 12, 12]          73,856
      BatchNorm2d-13         [1024, 128, 12, 12]             256
             ReLU-14         [1024, 128, 12, 12]               0
           Conv2d-15         [1024, 128, 12, 12]         147,584
      BatchNorm2d-16         [1024, 128, 12, 12]             256
             ReLU-17         [1024, 128, 12, 12]               0
           Conv2d-18         [1024, 256, 12, 12]         295,168
      BatchNorm2d-19         [1024, 256, 12, 12]             512
             ReLU-20         [1024, 256, 12, 12]               0
           Linear-21                   [1024, 7]         258,055
================================================================
```

(a) Emo3CNN summary



(b) Emo3CNN Loss Train-Test Plot

Figure 5: Emo3CNN

4.3 Emo3CNN

The Emo3CNN has the same architecture as EmoCNN but the only differencce is that insteaed of the maxpool layer, the average pooling layer is used. The number of parameters are the same.

You can see the whole summary in figure 5 and it's respective loss plot for train and test. The train and test loss are very similar here but they are still quite high.

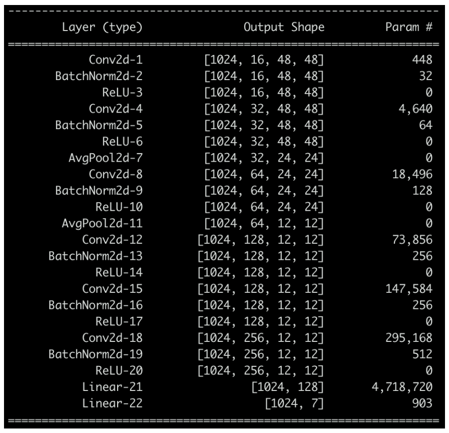
This model gives accuracy of 51%. This is because we want to only choose the most important features, which the maxpool gives. The AvgPool will smooth the image. We will see an example in the visualization part which will explain a bit more about this.

4.4 Emo4CNN

The Emo4CNN has the same architecture as Emo3CNN but the only difference is that this goes to a depth of 256 while the Emo3CNN goes upto 128 depth. This also has 2 fully connected layers in the end to gradually reduce the vector size to 7 classes rather to reduce at once from 256.

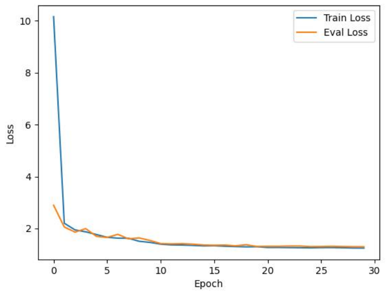
You can see the whole summary in figure 6 and it's respective loss plot for train and test. The train and test loss are very similar here but they are still very high.

This model gives accuracy of 53%. This is a little higher than Emo3CNN because it has a deeper model and more ability to extract features. The AvgPool here is the reason why it is still lower than the other models that we have seen which use MaxPool. There is also a possibility of losing information in the fully connected layers whichc reduce the number of features very drastically.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [1024, 16, 48, 48]             448
       BatchNorm2d-2         [1024, 16, 48, 48]              32
              ReLU-3         [1024, 16, 48, 48]               0
            Conv2d-4         [1024, 32, 48, 48]           4,640
       BatchNorm2d-5         [1024, 32, 48, 48]              64
              ReLU-6         [1024, 32, 48, 48]               0
         AvgPool2d-7         [1024, 32, 24, 24]               0
            Conv2d-8         [1024, 64, 24, 24]          18,496
       BatchNorm2d-9         [1024, 64, 24, 24]             128
             ReLU-10         [1024, 64, 24, 24]               0
        AvgPool2d-11         [1024, 64, 12, 12]               0
           Conv2d-12        [1024, 128, 12, 12]          73,856
      BatchNorm2d-13        [1024, 128, 12, 12]             256
             ReLU-14        [1024, 128, 12, 12]               0
           Conv2d-15        [1024, 128, 12, 12]         147,584
      BatchNorm2d-16        [1024, 128, 12, 12]             256
             ReLU-17        [1024, 128, 12, 12]               0
           Conv2d-18        [1024, 256, 12, 12]         295,168
      BatchNorm2d-19        [1024, 256, 12, 12]             512
             ReLU-20        [1024, 256, 12, 12]               0
           Linear-21                [1024, 128]       4,718,720
           Linear-22                  [1024, 7]             903
================================================================
```

(a) Emo4CNN summary



(b) Emo4CNN Loss Train-Test Plot

Figure 6: Emo4CNN

4.5 Emo5CNN

The Emo5CNN has 9 sequential layers with 2 AvgPool layers. Instead of having 2 fully connected layers, here I have tried to reduce the dimensions with CNN layers after going upto 256 depth. Then the architecture comes down to a depth of 64 when there is a fully connected layer with the number of output classes.

You can see the whole summary in figure 7 and it's respective loss plot for train and test.

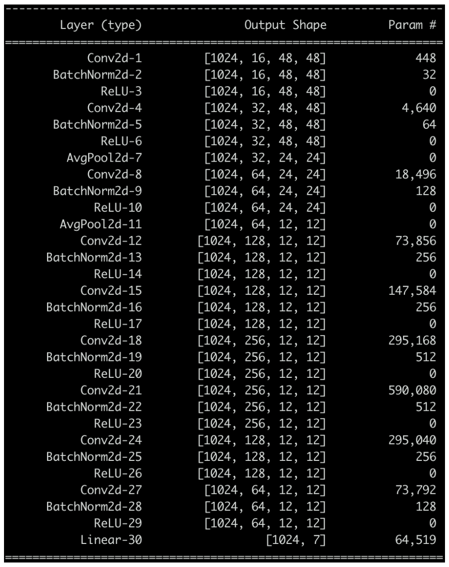
This model gives accuracy of 62%. This is the best model that I have trained. Although it uses an AvgPool layer, the multiple convolutional layers to gradually decrease the depth has helped retain more features at the end instead of having a fully connected layer to drastically reduce the number of layers. Intuitively it feels like having this model with a MaxPool layer should give better results and lower loss value.

4.6 Emo6CNN

This is a very different architecture which has a filter size of 5 and padding of 2 so that it goes through every pixel in the image. The input dimension of the image is 48x48 which is already very small. It has very few features to begin with and a filter size of 5 will smooth over it even more and have very poor accuracy.

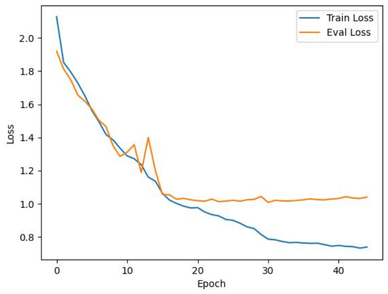
You can see the whole summary in figure 8 and it's respective loss plot for train and test.

The loss here is very high compared to every other model because it has learnt the least here. This is because of the simple model and a very big filter size. As the dimensions reduce and the filter size increases, there is lesser and lesser for the model to run.



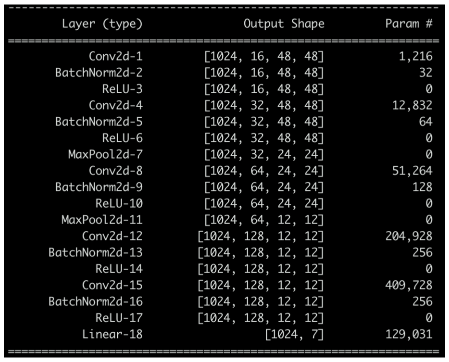
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [1024, 16, 48, 48]             448
       BatchNorm2d-2         [1024, 16, 48, 48]              32
              ReLU-3         [1024, 16, 48, 48]               0
            Conv2d-4         [1024, 32, 48, 48]           4,640
       BatchNorm2d-5         [1024, 32, 48, 48]              64
              ReLU-6         [1024, 32, 48, 48]               0
         AvgPool2d-7         [1024, 32, 24, 24]               0
            Conv2d-8         [1024, 64, 24, 24]          18,496
       BatchNorm2d-9         [1024, 64, 24, 24]             128
             ReLU-10         [1024, 64, 24, 24]               0
        AvgPool2d-11         [1024, 64, 12, 12]               0
           Conv2d-12        [1024, 128, 12, 12]          73,856
      BatchNorm2d-13        [1024, 128, 12, 12]             256
             ReLU-14        [1024, 128, 12, 12]               0
           Conv2d-15        [1024, 128, 12, 12]         147,584
      BatchNorm2d-16        [1024, 128, 12, 12]             256
             ReLU-17        [1024, 128, 12, 12]               0
           Conv2d-18        [1024, 256, 12, 12]         295,168
      BatchNorm2d-19        [1024, 256, 12, 12]             512
             ReLU-20        [1024, 256, 12, 12]               0
           Conv2d-21        [1024, 256, 12, 12]         590,080
      BatchNorm2d-22        [1024, 256, 12, 12]             512
             ReLU-23        [1024, 256, 12, 12]               0
           Conv2d-24        [1024, 128, 12, 12]         295,040
      BatchNorm2d-25        [1024, 128, 12, 12]             256
             ReLU-26        [1024, 128, 12, 12]               0
           Conv2d-27         [1024, 64, 12, 12]          73,792
      BatchNorm2d-28         [1024, 64, 12, 12]             128
             ReLU-29         [1024, 64, 12, 12]               0
           Linear-30                 [1024, 7]           64,519
================================================================
```

(a) Emo5CNN summary



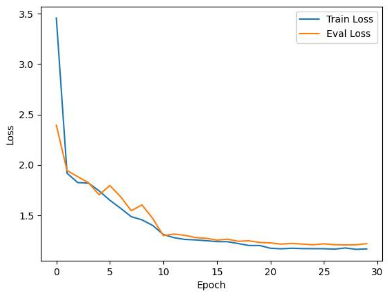
(b) Emo5CNN Loss Train-Test Plot

Figure 7: Emo5CNN



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [1024, 16, 48, 48]           1,216
       BatchNorm2d-2         [1024, 16, 48, 48]              32
              ReLU-3         [1024, 16, 48, 48]               0
            Conv2d-4         [1024, 32, 48, 48]          12,832
       BatchNorm2d-5         [1024, 32, 48, 48]              64
              ReLU-6         [1024, 32, 48, 48]               0
         MaxPool2d-7         [1024, 32, 24, 24]               0
            Conv2d-8         [1024, 64, 24, 24]          51,264
       BatchNorm2d-9         [1024, 64, 24, 24]             128
             ReLU-10         [1024, 64, 24, 24]               0
        MaxPool2d-11         [1024, 64, 12, 12]               0
           Conv2d-12        [1024, 128, 12, 12]         204,928
      BatchNorm2d-13        [1024, 128, 12, 12]             256
             ReLU-14        [1024, 128, 12, 12]               0
           Conv2d-15        [1024, 128, 12, 12]         409,728
      BatchNorm2d-16        [1024, 128, 12, 12]             256
             ReLU-17        [1024, 128, 12, 12]               0
           Linear-18                 [1024, 7]          129,031
================================================================
```

(a) Emo6CNN summary



(b) Emo6CNN Loss Train-Test Plot

Figure 8: Emo6CNN

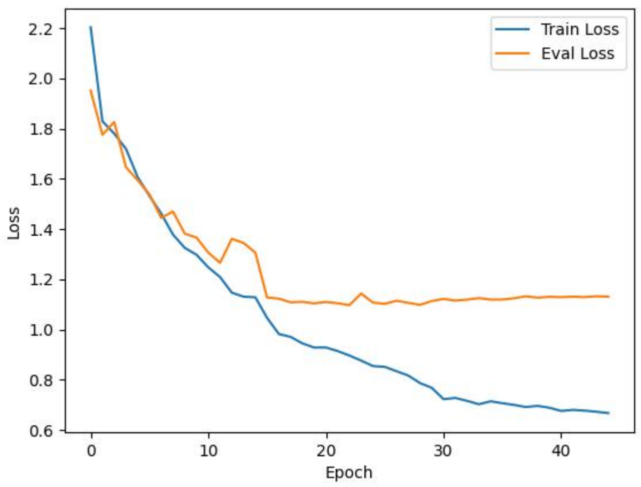


Figure 9: Emo5CNN without augmentation

5 Results

The best results we saw were with the Emo5CNN which had 9 sequential convolutional layers. This model also had 2 AvgPool layers. I did other tests with this model to identify the other points of importance in a CNN model. The figure 7 has the configurable parameters of adaptive learning rate which starts at 0.001 and reduces by a factor of 10 in every 15 epochs. The model was trained for 45 epochs because it needed more time to plateau to the local minima.

Figure 9 shows the train test loss graph without augmentations applied to the dataset. It only converts the image to a tensor and does normalization and scaling. Because of this, we can see the effect of the test loss which is higher than training done with augmentation.

This just shows the importance of augmentation in testing and not having an over-fitted model.

Figure 10 shows the training done with a fixed learning rate of 0.0001. The graph shows a hugely over-fitted model. It also shows a big fluctuation in the test loss. This is because the learning rate is not able to find the global minima. The minima reduces and increases with every epoch. So having an adaptive learning rate which gradually decreases in every few epochs is very important.

VGG16 model for these datasets has a maximum accuracy of 72%. We have achieved an accuracy of 62% with these methoda which have less than half the layers which is a pretty good accuracy given the number of parameters.

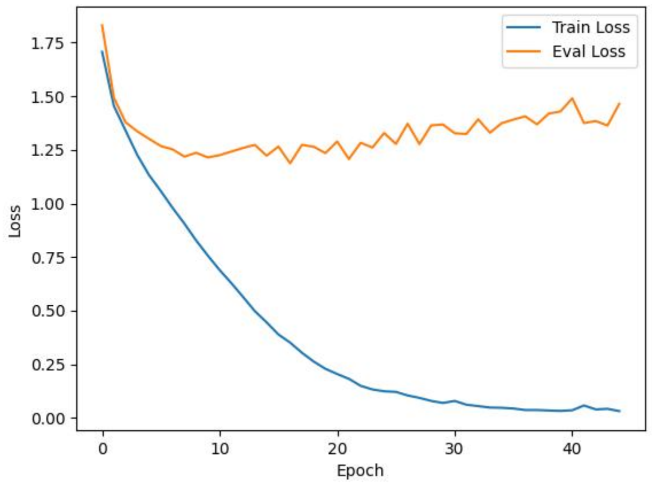


Figure 10: Emo5CNN without adaptive learniing rate

6 Visualization

Figure 11 shows the model activations of each layer of Emo5CNN model. We can clearly see the features that it is learning in each layer. This gives a bit more insight on how each model learns and which features it captures.

Figure 12 gives another example of learning features.

7 Conclusion

The main conclusions of the exercise are outlined as -

- MaxPool works better than AvgPool in cases where we want to learn particular features of an image. AvgPool are used more where we need a smoother output rather than a Max output.
- A deeper model has more features and works better.
- Adaptive learning rate helps the model to reach a global minima better than fixed learning rate. It also helps the model not be over-fitted to the data.
- Augmentations help in creating robust data for training which helps avoid data over-fitting.
- A drastic drop in features using fully connected layers loses important features. Instead using CNNs to reduce the depth gradually retains more information of the image and reduces the loss, giving better accuracy.

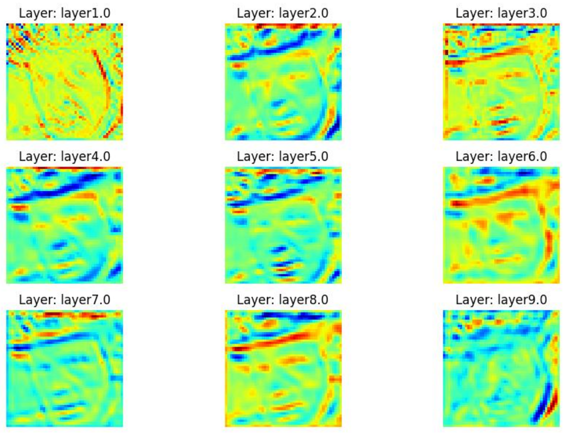


Figure 11: Emo5CNN model activations of each layer

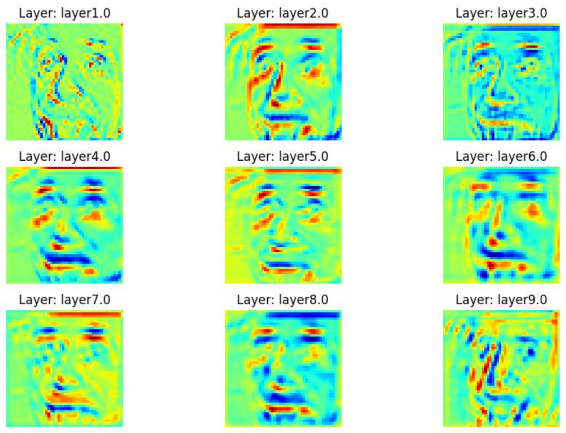


Figure 12: Emo5CNN model activations of each layer image 2

These practices will give better accuracy to our model training in the future.
You can find the whole code base in `https://github.com/manyu252/EmotionRecognitionCV`