Maastricht University

Department of Data Science and Knowledge Engineering

Master's Program in Data Science for Decision Making

Computer Vision Assignment Report

May 25, 2021

Emotion Recognition Task

Authors:
Kirill Shcherbakov
Prakash Gupta

Supervisor:
Mirela Popa

Contents

1 Introduction

An emotion is composed of feelings, thought, and behavioral responses. Recognizing emotions is now possible because of recent advancements in the Deep Learning field which provide the bridge to the Computer Vision to achieve this task fruitfully. The task of this assignment is to accomplish recognize an emotion of the face through an image.

1.1 Dataset

The FER2013 dataset [1] is used to perform various experiments, train and test the models. This dataset is selected for availability and size considerations for this task. It consists of 34k 48x48 pixel grayscale images of faces where 28,709 samples of the training set, 3,589 samples of the public test set, and 3,589 private training set. Every image belongs to one of the seven emotion categories:

- 0=Angry
- 1=Disgust
- 2=Fear
- 3=Happy
- 4=Sad
- 5=Surprise
- 6=Neutral

2 Implementation details

In this section, the implementation details of the model training and data preparation to detect the face emotion from a given image are discussed. The high level description of the models and the intuition behind them are presented in Appendix A.2. A description of the technology stack used is presented in Appendix A.1.

2.1 Data preparation

The data is presented as a CSV file where each pixel value is represented as a column, a separate column with the label of the corresponding image, and a separate column for the dataset flag indication (training, validation, and test sets). We preprocess the data creating training, validation and test dataloaders using the PyTorch functionalities. The training dataset is used for the model training process, and the validation dataset is used as a model performance evaluation. After the resulting model meets the specified target metric values, the model is tested on a test set to perform inference.

As mentioned earlier, we use data augmentation using the Albumentations library in the custom pytorch dataset class, which includes the following transformations:

- Flip the input horizontally around the y-axis with the probability 0.5
- Flip the input vertically around the x-axis with the probability 0.3
- Rotate the input by an angle selected randomly from the uniform distribution with the probability 0.2
- Perform CoarseDropout of the square regions in the image with the probability 0.2
- Apply gaussian noise to the input image with the probability 0.2

The transformations mentioned above are selected after preliminary experiments that revealed the best described combination, which will be discussed in the Experiments section.

Creating the dataloaders, the following batch size values are considered in this work: 64, 128, 256, and 512.

2.2 Model training details

Since every optimization technique has a weakness, the Stochastic Gradient Descent (SGD) and the Adam Optimizer are chosen as the optimization modules to be considered and experimented with. Likewise, the Cross Entropy and The Negative Log Likelihood loss (NLLLoss) are chosen as loss functions for the experiments.

Most obviously the magnitude of the learning rate matters. If it is too large, optimization diverges, if it is too small, it takes too long to train or we end up with a suboptimal result. One way of adjusting the learning rate is to use a scheduler that adjusts it explicitly at each step. Here, we try the following schedulers to improve training procedure:

- *torch.optim.scheduler.OneCycleLR* that sets the learning rate of each parameter group according to the 1 cycle learning rate policy.
- *torch.optim.scheduler.CosineAnnealingLR* that sets the learning rate of each parameter group using a cosine annealing schedule.

2.3 Quantitative and qualitative evaluations

The target metrics are accuracy, precision, recall, and F1 based on validation data. Each training epoch displays the results of running the model on the validation data as well as the training and validation losses.

As a qualitative metric, we visually inspect the results of the model on 7 images checking the predicted classes.

3 Experiments

In this section, we cover the various experiments to check the accuracy of the model by changing the parameters, and layers of the model. All the experiments have been conducted on 10 training epochs. A detailed experiment with changing the number of layers with the third model is presented in Appendix A.3.

3.1 Number of filters and the kernel size

The first experiment is based on the different number of filters in convolution layers and different kernel size. The first architecture is picked to conduct this experiment. We create 3 different models of the first architecture type with the following structure:

1. Filters version 1: a model with 4 Conv2D layers 3-16-32-64-128 with kernel of size 3 or 5 and ReLU activation functions between them. BatchNorm2d is applied after every Conv2d layer. MaxPool2d is applied after each BatchNorm2d layer except the second layer. After that, a Linear layer with BatchNorm1d and ReLU activation followed by a Linear layer with LogSoftmax.

2. Filters version 2: a model with 4 Conv2D layers 3-32-64-128-256 with kernel of size 3 or 5 and ReLU activation functions between them. BatchNorm2d is applied after every Conv2d layer. MaxPool2d is applied after each BatchNorm2d layer except the second layer. After that, a Linear layer with BatchNorm1d and ReLU activation followed by a Linear layer with LogSoftmax.

3. Filters version 3: a model with 4 Conv2D layers 3-64-128-256-512 with kernel of size 3 or 5 and ReLU activation functions between them. BatchNorm2d is applied after every Conv2d layer. MaxPool2d is applied after each BatchNorm2d layer except the second layer. After that, a Linear layer with BatchNorm1d and ReLU activation followed by a Linear layer with LogSoftmax.

We used SGD optimizer with learning rate 0.002, momentum 0.9 and weight decay $9e-4$, the OneCycleLR scheduler, NLLLoss objective, and dropout value 0.1.

The results are shown in Table 1. **One can see that with an increase in the number of filters, the performance of the model on the validation set improves. This is easily explained by the fact that the higher the number of filters, the higher the number of abstractions that the Network is able to extract from image data**. Hence, there are larger combinations of patterns to capture by the model, and this wider understanding of an image allows performing better on the classification task.

However, no conclusions can be drawn about the relationship between accuracy and the kernel size. And this can be explained by the fact that if you start using a larger kernel, you may start losing details in some smaller features (where 3x3 would detect them better) and in other cases, where your dataset has larger features, the 5x5 may start to detect features that 3x3 misses.

Table 1: Comparison of the different number of filters and the kernel size of the architectures of the first model.

Model 1	**Validation Accuracy, %**	**Validation Loss**
Filters version 1, kernel size=3	59	1.076
Filters version 1, kernel size=5	59	1.076
Filters version 2, kernel size=3	63.5	1.008
Filters version 2, kernel size=5	58.9	1.050
Filters version 3, kernel size=3	63.7	1.001
Filters version 3, kernel size=5	61.3	1.014

3.2 Different models

The purpose of this experiment is to identify the best model. After preliminary experiments with each of the models under consideration, the final versions of each of the models are selected and their comparison is presented in Table 2. In this experiment, we used the SGD optimizer with the learning rate 0.002, momentum 0.9 and weight decay $9e-4$, the OneCycleLR scheduler, NLLLoss objective, and without any dropout.

1. Model 1: a model with 4 Conv2D layers 3-16-32-64-128 with the kernel of size 3 and ReLU activation functions between them. BatchNorm2d is applied after every Conv2d layer. MaxPool2d is applied after each BatchNorm2d layer except the second layer. After that, a Linear layer with BatchNorm1d and ReLU activation followed by a Linear layer with LogSoftmax.

2. Model 2: a model with 7 Conv2D layers 8-16-32-64-128-256 with the kernel of size 3 and ReLU activation functions between them. BatchNorm2d is applied after every Conv2d layer. MaxPool2d is applied after each BatchNorm2d layer. After that, 2 Linear layers with BatchNorm1d and ReLU activation of sizes 1024-512-256 followed by a Linear layer 256-7 with LogSoftmax.

3. Model 3: a U-Net like model with the "encoder" part consisting of the blocks with Conv2d layers 32-64-128-256-512-1024-1024, where each convolution layer followed by a BatchNorm2d layer and a ReLU activation layer. Additionally, a MaxPooling layer is applied after the second, fifth, and seventh "encoder" blocks. The "decoder" part consists of the blocks with Conv2d layers 1024-512-256-7, where each block is followed by BatchNorm2d and ReLU layers. An AvgPool2d layer is applied after the ninth in the "decoder" part. After that, the output of the final layer is activated by a LogSoftmax layer.

Figure 1 shows a comparison of the validation losses of the models under consideration. Based on the results obtained, **model 3 was chosen as the final model for further experiments**. As expected, this model is way powerful because of

Table 2: Comparison of the different cherry-picked models.

Model	**Validation Accuracy, %**	**Validation Loss**
1	61.2	1.061
2	60.1	1.074
3	67.4	0.903

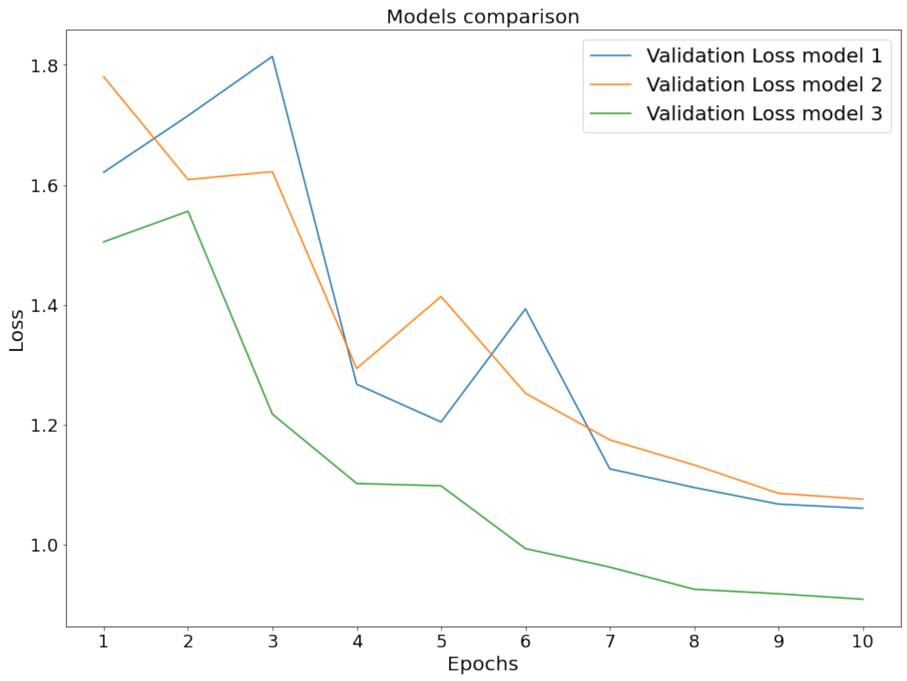


Figure 1: Validation loss comparison of the different models.

the U-Net like architecture in combination with the chosen number of filters and layers, which allows you to get the most information from the image, extracting the more and more complex patterns that help to make a classification decision more accurate.

3.3 Optimizer and learning rate

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. This experiment is based on changing the learning rate of optimizers and analysing the validation accuracy and validation loss for model 3. In this experiment, we fix the batch size equal to 64 and set the dropout value to 0.

The results are shown in Table 3. One can notice that with the increase in the learning rate value of Adam's optimizer, the validation accuracy increases, but after LR=0.001, the validation accuracy drops. On the other hand, in experiments with SGD optimizer, the validation accuracy increases with the increment of the learning rate. Moreover, the validation accuracy of SGD is generally higher than the Adam validation accuracy. Therefore, **SGD with the learning rate** 0.02 **is chosen as the final optimizer for model 3**.

Table 3: Comparison of the different learning rates of the optimizers on model 3

Optimizer	Learning Rate	Val. Accuracy, %	Val. Loss
Adam	0.0001	64.2	0.955
Adam	0.0005	64.1	0.963
Adam	0.001	65.7	0.923
Adam	0.0015	64.0	0.961
Adam	0.02	64.2	0.954
SGD	0.002	66.7	0.897
SGD	0.005	67.8	0.880
SGD	0.01	67.1	0.890
SGD	0.015	67.8	0.892
SGD	0.02	67.8	0.876

3.4 Loss function

In this experiment, we consider two loss functions that are created for multi-class classification: the NLLLoss and CrossEntropy criterions. Just like in the experiment with the optimizer and learning rate, we fix the batch size to 64 and do not use dropout. The results are shown in Table 4. The results on the validation set show that **CrossEntropy loss function is more suitable for model 3 with the SGD optimizer**.

Table 4: Comparison of the different criterions with the optimizers on model 3

Criterion	Optimizer	Val. Accuracy, %	Val. Loss
NLLLoss	SGD	66.7	0.897
NLLLoss	Adam	64.9	0.937
CrossEntropyLoss	SGD	67.1	0.878
CrossEntropyLoss	Adam	63.3	0.994

3.5 Scheduler

This experiment aims to identify the most optimal scheduler. Two options are considered here: OneCycleLR applying it within the batch loop and CosineAnnealingLR applying it every epoch. Here we use model 3 with SGD optimizer with the learning rate 0.02 and CrossEntropy as the objective function. We fix the batch size to 64 and do not use dropout. The results are shown in Table 5. **The results obtained demonstrate that the best result with the selected parameters shows OneCycleLR**.

Table 5: Comparison of the different schedulers on model 3

Scheduler	Val. Accuracy, %	Val. Loss
OneCycleLR	67.7	0.887
CosineAnnealingLR	65.9	0.901

3.6 Batch Size

The final experiment for selecting the final configuration is the analysis of the variation of validation accuracy by choosing different batch sizes on model 3. Here we use model 3 with SGD optimizer with the learning rate 0.02, CrossEntropy as the objective function, and the OneCycleLR scheduler applying after each batch iteration. In addition, we perform experiments with doubling of the size of filters in model 3 and report the results for different batch sizes. The results are shown in the table 6. **The maximum validation accuracy is achieved with the batch size 128 with doubling the channels size**. It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.

Table 6: Comparison of the different batch sizes on model 3

Batch size	**Max. channel size**	**Val. Accuracy, %**	**Val. Loss**
64	1024	66.7	0.897
128	1024	69.2	0.862
256	1024	65.2	0.926
512	1024	62.0	0.987
64	2048	69.0	0.852
128	2048	69.6	0.849

4 Final model results and Conclusion

4.1 Final results

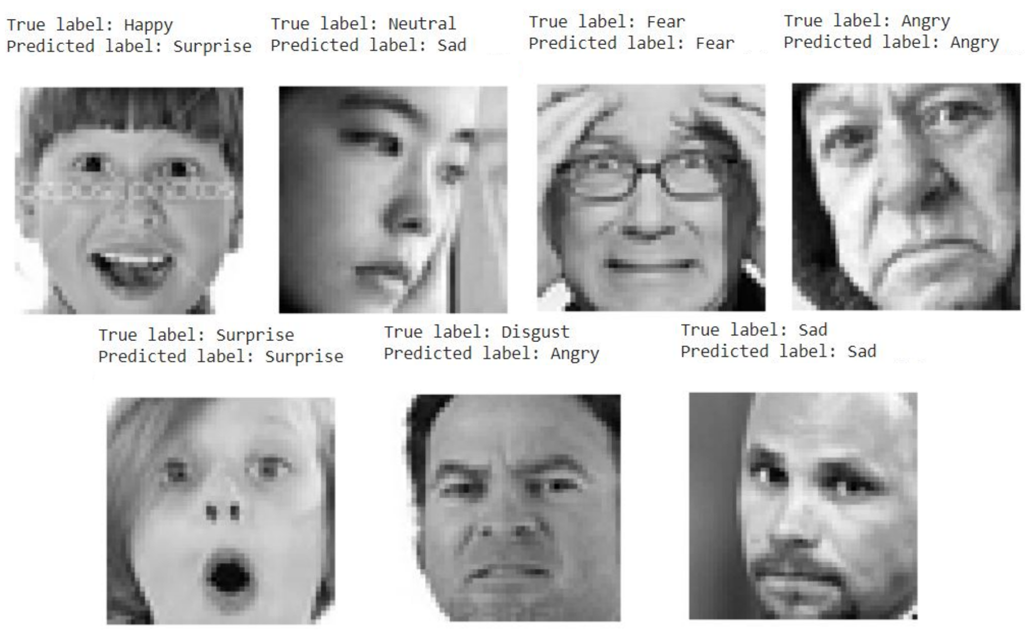
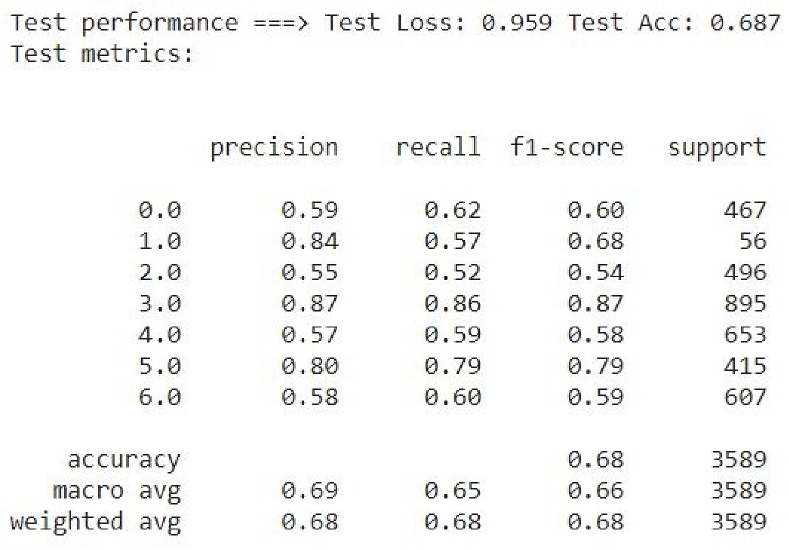


Figure 2: Qualitative evaluation of the model results



```
Test performance ===> Test Loss: 0.959 Test Acc: 0.687
Test metrics:


              precision    recall  f1-score   support

         0.0       0.59      0.62      0.60       467
         1.0       0.84      0.57      0.68        56
         2.0       0.55      0.52      0.54       496
         3.0       0.87      0.86      0.87       895
         4.0       0.57      0.59      0.58       653
         5.0       0.80      0.79      0.79       415
         6.0       0.58      0.60      0.59       607

    accuracy                           0.68      3589
   macro avg       0.69      0.65      0.66      3589
weighted avg       0.68      0.68      0.68      3589
```

Figure 3: Report of the final model accuracy, precision, and recall on the test data

In the experiments listed above, model 3 with twice the number of channels was selected as the final experiment with 15 epochs. Here we use SGD optimizer with the learning rate 0.02, CrossEntropy as the objective function, the OneCycleLR scheduler applying after each batch iteration, and fix the batch size to 128. As shown in Table 6 (last row), the model shows an accuracy of 69.6% on the validation dataset. The results of the quantitative metrics of the model on the test dataset are shown in the Figure 3. One can see that the model does the worst job with the classification of class 2 (fear). This can be explained by the fact that in the dataset, images of class 5 (surprise) and class 2 (fear) are often similar and in some cases indistinguishable even by a human eye. One can also notice that the model is very successful with the classification of class 3 (happy), which can be explained by the fact that the images of this class have easily extracted features (e.g., smiles), which can hardly be mixed up with other emotions, that allow the model to successfully cope with the classification of this emotion.

A qualitative comparison is shown in Figure 2. As mentioned above, our hypotheses were confirmed, that in some images it is difficult to determine the class even to the human eye. Therefore, we can conclude that the model effectively copes even on images where it can predict the wrong class.

4.2 Visualizing filters and weights

Visualizing the filters within a learned CNN can provide insight into how the model works. The visualization of the learned filters of the second convolution layer (randomly chosen) of the final model is shown in the Figure 4. The dark squares indicate small or inhibitory weights and the light squares represent large or excitatory weights. Using this intuition, one can see that the bottom right filter on the last row detects a gradient from light in the top right to dark in the bottom left.

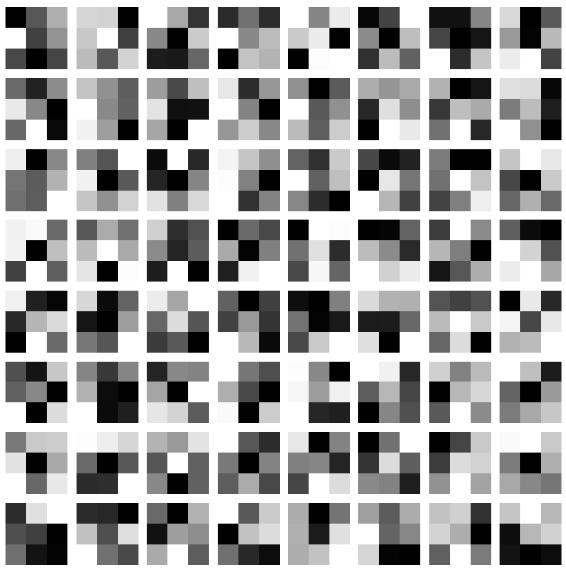


Figure 4: Filters visualization of the second layer of model 3

To show the result of feature maps capturing of applying the filters to a given input image, we pick two images with different emotions: happy (Figure 5) and fear (Figure 6). We can see in Figures 7 and 8 that the result of applying the filters in the second convolutional layer is a lot of versions of the original images with different features highlighted (e.g., the contours of the eyes, eyebrows, and nose). As we progress deeper into the model layers, the feature maps show less and less

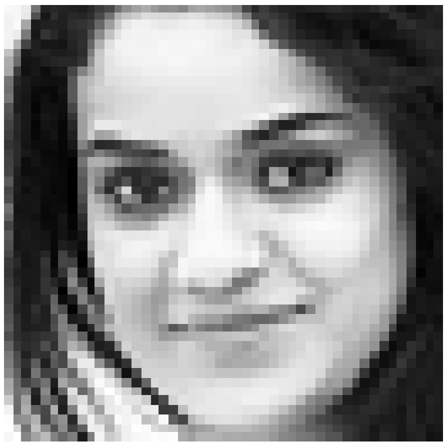


Figure 5: Happy input image.

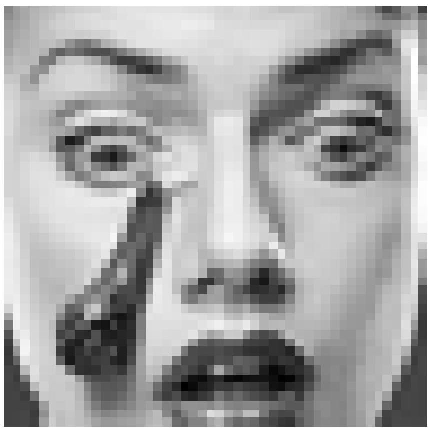


Figure 6: Fear input image.

detail. On the representation of the deeper layers (layers 3, 4, 5, and 6 in Figure 7) of the emotion of happiness, we see that the model focuses on the smile and the eyes with eyebrows, which probably helps in further classifying this class. In a similar manner, one can see how the model on the deeper layers abstracts the contours of the frightened mouth and eyes into higher features in Figure 8. This pattern was to be expected, as the model abstracts the features from the image into more general concepts that can be used to make a classification. We generally lose the ability to interpret these deeper feature maps.

4.3 Conclusion

The experiments have shown that it is essential to select and configure (finding the optimal combination in terms of validation performance) the model architecture, iteratively searching over the values of Kernel/Filter Size and Number of Channels, and adding modules such as BatchNormalization, Dropout, and Pooling layers, strictly following the rule ”one change at a time”, since the effect of a change can be both negative and positive. Having a large dataset is crucial for the performance of the deep learning model. However, the experiments have shown that the performance of the model can be drastically improved by augmenting the data we already have. It also helps the model to generalize on different types of images. Last but not least, the experiments showed that the optimization of hyperparameters plays a huge role, where one also needs to follow the rule ”one parameter change at a time” and use all possible techniques to improve the training process (various combinations of the optimizers and the loss functions, schedulers).

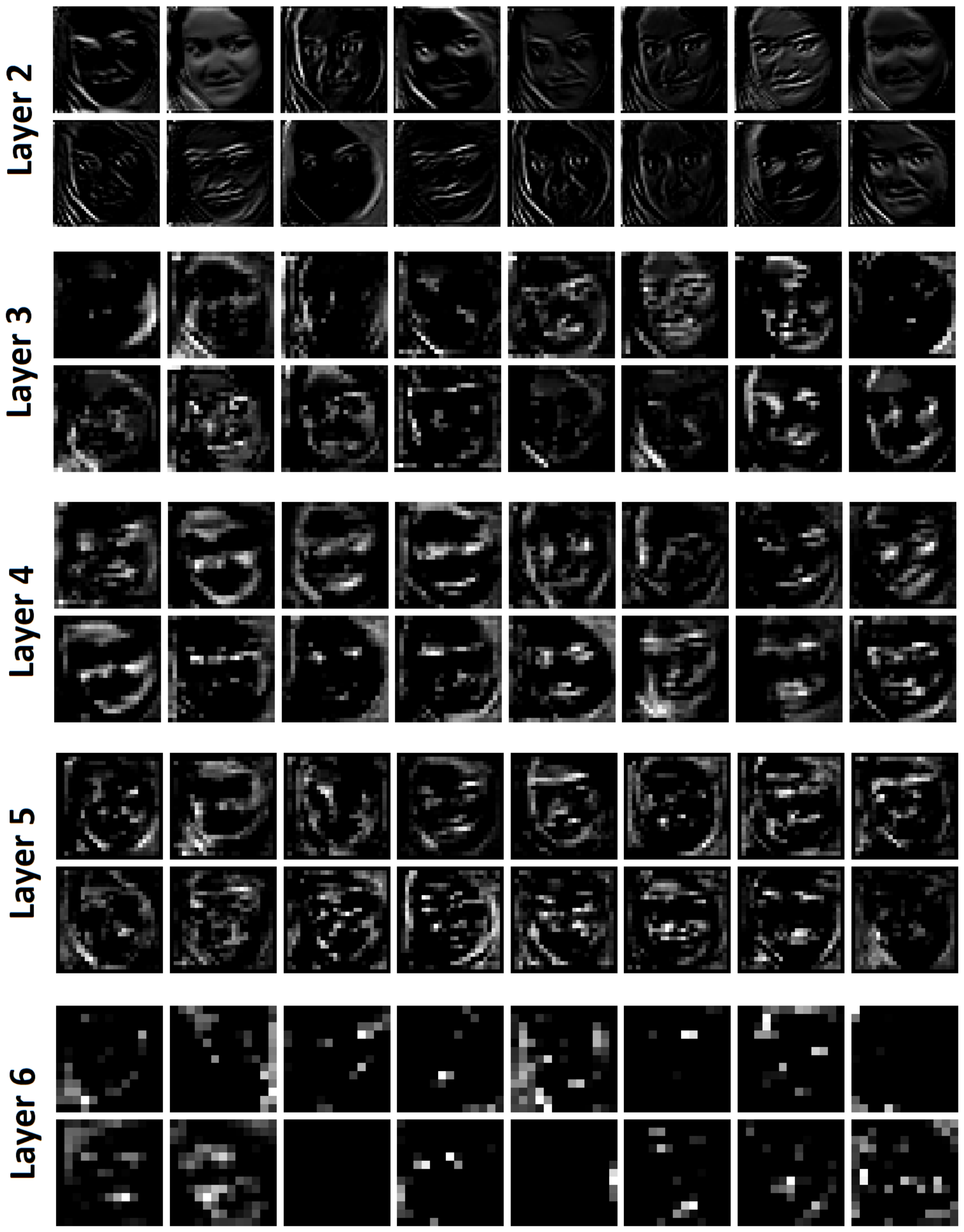


Figure 7: Intermediate representation of the happy input image of the activated convolutional layers

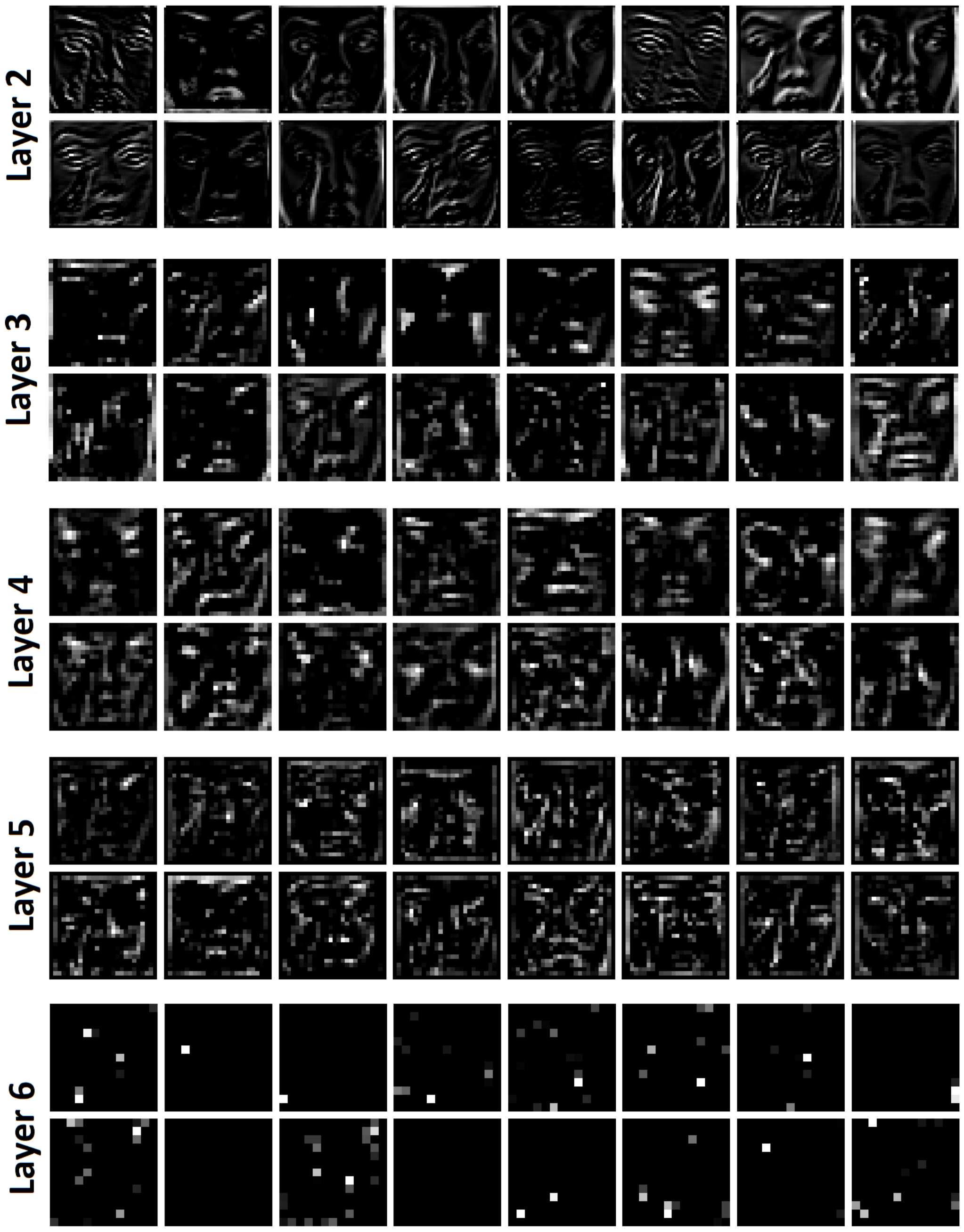


Figure 8: Intermediate representation of the fear input image of the activated convolutional layers

References

[1] Fer2013 dataset. https://www.kaggle.com/ashishpatel26/facial-expression-recognitionferchallenge.

[2] Pytorch. https://pytorch.org/.

[3] Albumentations. https://albumentations.ai/.

[4] Thomas Brox. Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation., 2015.

A Appendix

A.1 Framework stack

In this work, the PyTorch [2] library is used as a deep learning framework on which the experiments are performed since it has a reputation for being more widely used in research than in production which perfectly matches with the scope of this assignment to understand the effect of the different CNN operations on your results. A lot of effort in solving any deep learning problem goes into preparing the data. PyTorch provides many tools to make data loading easy. Here, the Pandas library is used for easier csv parsing in combination with the PyTorch dataset functionalities such as *torch.utils.data.Dataset* to create a custom dataset leveraging the PyTorch dataset APIs and callable custom transforms that can be composable, and *torch.utils.DataLoader* that wraps an iterable around the dataset to enable easy access to the samples (images and labels).

In addition, to increase the quality of trained models and to create new training samples from the existing data, the Albumentations [3] framework is used to effectively implement a wide range of image transformation operations, some of which are not included in the built-in PyTorch transformation methods, that are optimized for performance. The Sklearn library is used to build a text report showing the main classification metrics, and the Matplotlib library is used to build the visualization.

All experiments are performed in Google Colab. All dependencies for running the code and instructions are provided in the README description.

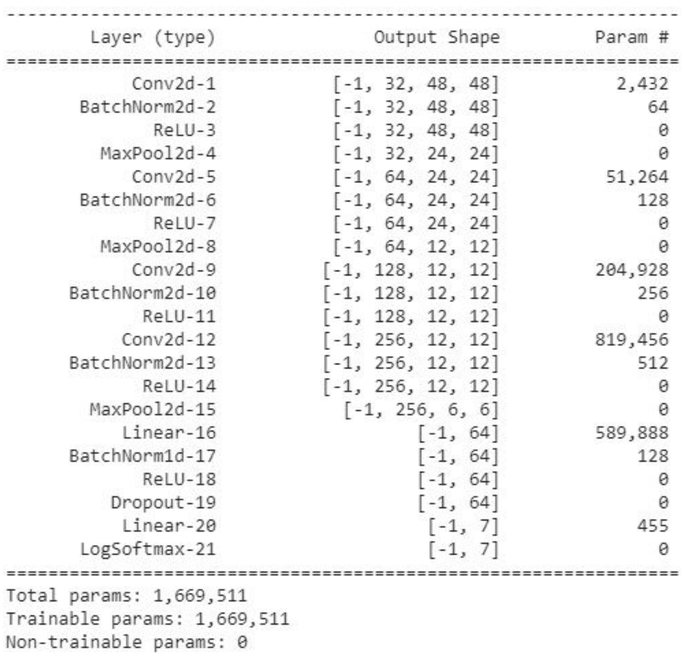
A.2 High level models description

A.2.1 Model 1

The first CNN model is shown in Figure 9. As the first model, a simple architecture is chosen, consisting of 4 layers of convolutions, together with each of which there is a BatchNormalization layer, an activation ReLU function, and a MaxPooling layer. This is followed by 1 fully-connected layer with BatchNorm layers, the ReLU activation function, and a Dropout layer that helps to prevent overfitting by randomly turning off some "neurons", after which we get a probabilistic class distribution. In preliminary experiments, it was found that adding a BatchNormalization layer after a Conv2D layer has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks because it standardizes the inputs to a layer for each mini-batch. Also, another finding is that adding a MaxPooling layer after a combination of Conv2D, BatchNormalization and ReLU activation layers helps and speeds up the learning process, giving some translation invariance and extracting the sharpest features of an image. In this model, we make the kernel size for Conv2D layers flexible, so that it can take values equal to 3 and 5 in order to see the effect of changing this parameter.

A.2.2 Model 2

The second CNN model is shown in Figure 10. This model is ideologically a repetition of the architecture of the first model with a number of modifications: the



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 48, 48]           2,432
       BatchNorm2d-2           [-1, 32, 48, 48]              64
              ReLU-3           [-1, 32, 48, 48]               0
         MaxPool2d-4           [-1, 32, 24, 24]               0
            Conv2d-5           [-1, 64, 24, 24]          51,264
       BatchNorm2d-6           [-1, 64, 24, 24]             128
              ReLU-7           [-1, 64, 24, 24]               0
         MaxPool2d-8           [-1, 64, 12, 12]               0
            Conv2d-9          [-1, 128, 12, 12]         204,928
      BatchNorm2d-10          [-1, 128, 12, 12]             256
             ReLU-11          [-1, 128, 12, 12]               0
           Conv2d-12          [-1, 256, 12, 12]         819,456
      BatchNorm2d-13          [-1, 256, 12, 12]             512
             ReLU-14          [-1, 256, 12, 12]               0
        MaxPool2d-15            [-1, 256, 6, 6]               0
           Linear-16                   [-1, 64]         589,888
      BatchNorm1d-17                   [-1, 64]             128
             ReLU-18                   [-1, 64]               0
          Dropout-19                   [-1, 64]               0
           Linear-20                    [-1, 7]             455
       LogSoftmax-21                    [-1, 7]               0
================================================================
Total params: 1,669,511
Trainable params: 1,669,511
Non-trainable params: 0
```

Figure 9: The first architecture

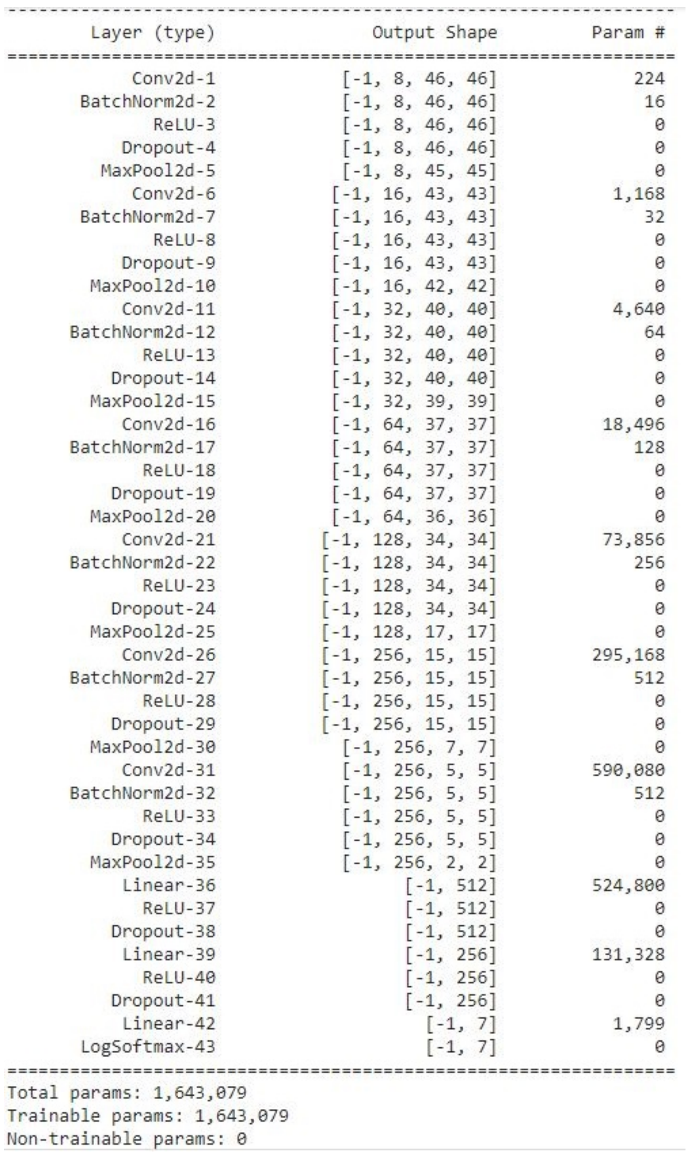
number of convolution layers is increased to 7, the kernel size is fixed to 3, a Dropout layer is placed in each convolution bundle, and the number of fully conntected layers is increased to 3 removing a BatchNormalization module.

A.2.3 Model 3

For the third model (Figure 11), the inspiration of the architecture is based on U-Net model [4]. The adapted description of the work is shown in Figure 12. The architecture contains two paths. The first path is called contraction and second is called expanding path. The contraction path (also called as the encoder) is used to capture the context of the image and the second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. In our case, this decodes the compressed representation of the outcome of the contraction path and allows the model to get a better understanding of the class to which an image belongs. The main advantage of the U-Net is that it can accept an image of any size and solve the bottleneck problem which is present in other auto-encoder architectures.

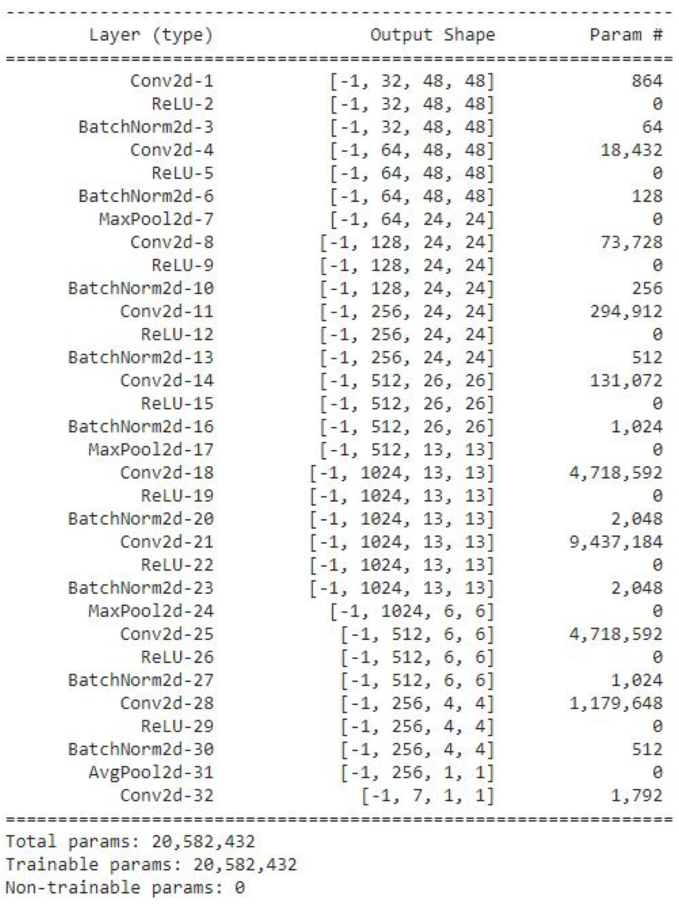
A.3 Experiment with layers

In this experiment, we vary the number of layers with channels and also compare with different criterion and optimizer and check the validation accuracy on model 3. The outcomes of the variation shown in Table 7.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 46, 46]             224
       BatchNorm2d-2            [-1, 8, 46, 46]              16
              ReLU-3            [-1, 8, 46, 46]               0
           Dropout-4            [-1, 8, 46, 46]               0
         MaxPool2d-5            [-1, 8, 45, 45]               0
            Conv2d-6           [-1, 16, 43, 43]           1,168
       BatchNorm2d-7           [-1, 16, 43, 43]              32
              ReLU-8           [-1, 16, 43, 43]               0
           Dropout-9           [-1, 16, 43, 43]               0
        MaxPool2d-10           [-1, 16, 42, 42]               0
           Conv2d-11           [-1, 32, 40, 40]           4,640
      BatchNorm2d-12           [-1, 32, 40, 40]              64
             ReLU-13           [-1, 32, 40, 40]               0
          Dropout-14           [-1, 32, 40, 40]               0
        MaxPool2d-15           [-1, 32, 39, 39]               0
           Conv2d-16           [-1, 64, 37, 37]          18,496
      BatchNorm2d-17           [-1, 64, 37, 37]             128
             ReLU-18           [-1, 64, 37, 37]               0
          Dropout-19           [-1, 64, 37, 37]               0
        MaxPool2d-20           [-1, 64, 36, 36]               0
           Conv2d-21          [-1, 128, 34, 34]          73,856
      BatchNorm2d-22          [-1, 128, 34, 34]             256
             ReLU-23          [-1, 128, 34, 34]               0
          Dropout-24          [-1, 128, 34, 34]               0
        MaxPool2d-25          [-1, 128, 17, 17]               0
           Conv2d-26          [-1, 256, 15, 15]         295,168
      BatchNorm2d-27          [-1, 256, 15, 15]             512
             ReLU-28          [-1, 256, 15, 15]               0
          Dropout-29          [-1, 256, 15, 15]               0
        MaxPool2d-30            [-1, 256, 7, 7]               0
           Conv2d-31            [-1, 256, 5, 5]         590,080
      BatchNorm2d-32            [-1, 256, 5, 5]             512
             ReLU-33            [-1, 256, 5, 5]               0
          Dropout-34            [-1, 256, 5, 5]               0
        MaxPool2d-35            [-1, 256, 2, 2]               0
           Linear-36                  [-1, 512]         524,800
             ReLU-37                  [-1, 512]               0
          Dropout-38                  [-1, 512]               0
           Linear-39                  [-1, 256]         131,328
             ReLU-40                  [-1, 256]               0
          Dropout-41                  [-1, 256]               0
           Linear-42                    [-1, 7]           1,799
       LogSoftmax-43                    [-1, 7]               0
================================================================
Total params: 1,643,079
Trainable params: 1,643,079
Non-trainable params: 0
```

Figure 10: The second architecture



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 48, 48]             864
              ReLU-2           [-1, 32, 48, 48]               0
       BatchNorm2d-3           [-1, 32, 48, 48]              64
            Conv2d-4           [-1, 64, 48, 48]          18,432
              ReLU-5           [-1, 64, 48, 48]               0
       BatchNorm2d-6           [-1, 64, 48, 48]             128
         MaxPool2d-7           [-1, 64, 24, 24]               0
            Conv2d-8          [-1, 128, 24, 24]          73,728
              ReLU-9          [-1, 128, 24, 24]               0
      BatchNorm2d-10          [-1, 128, 24, 24]             256
           Conv2d-11          [-1, 256, 24, 24]         294,912
             ReLU-12          [-1, 256, 24, 24]               0
      BatchNorm2d-13          [-1, 256, 24, 24]             512
           Conv2d-14          [-1, 512, 26, 26]         131,072
             ReLU-15          [-1, 512, 26, 26]               0
      BatchNorm2d-16          [-1, 512, 26, 26]           1,024
        MaxPool2d-17          [-1, 512, 13, 13]               0
           Conv2d-18         [-1, 1024, 13, 13]       4,718,592
             ReLU-19         [-1, 1024, 13, 13]               0
      BatchNorm2d-20         [-1, 1024, 13, 13]           2,048
           Conv2d-21         [-1, 1024, 13, 13]       9,437,184
             ReLU-22         [-1, 1024, 13, 13]               0
      BatchNorm2d-23         [-1, 1024, 13, 13]           2,048
        MaxPool2d-24           [-1, 1024, 6, 6]               0
           Conv2d-25            [-1, 512, 6, 6]       4,718,592
             ReLU-26            [-1, 512, 6, 6]               0
      BatchNorm2d-27            [-1, 512, 6, 6]           1,024
           Conv2d-28            [-1, 256, 4, 4]       1,179,648
             ReLU-29            [-1, 256, 4, 4]               0
      BatchNorm2d-30            [-1, 256, 4, 4]             512
        AvgPool2d-31            [-1, 256, 1, 1]               0
           Conv2d-32              [-1, 7, 1, 1]           1,792
================================================================
Total params: 20,582,432
Trainable params: 20,582,432
Non-trainable params: 0
```

Figure 11: The third architecture

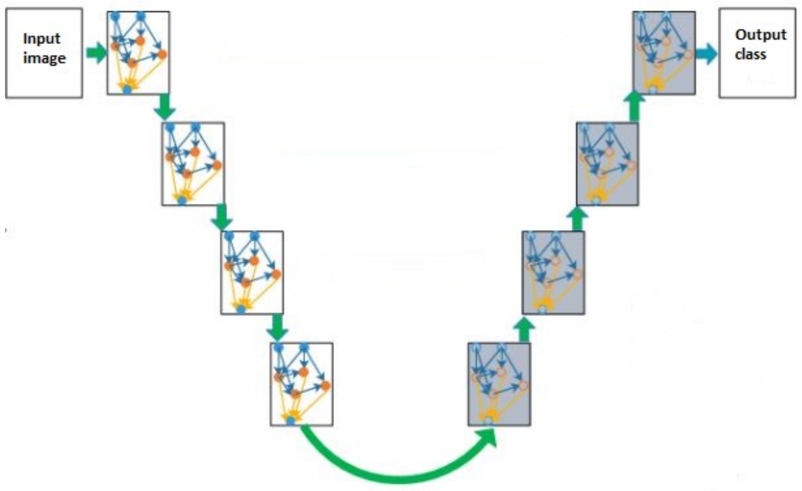


Figure 12: The third model

Table 7: Comparison of the different number of layers on model 3

No of layers	**Max. channel size**	**Criterion**	**Optimizer**	**Val. Accuracy, %**	**Val. Loss**
4	32	CrossEntropyLoss	Adam	51.2	1.262
4	32	NLLLoss	Adam	52.0	1.286
4	32	NLLLoss	SGD	50.8	1.297
4	32	CrossEntropyLoss	SGD	50	1.307
5	64	CrossEntropyLoss	Adam	59	1.080
5	64	NLLLoss	Adam	56	1.116
5	64	NLLLoss	SGD	58.3	1.105
5	64	CrossEntropyLoss	SGD	58	1.083
6	128	CrossEntropyLoss	Adam	61.7	1.013
6	128	NLLLoss	Adam	62.1	1.007
6	128	NLLLoss	SGD	62.2	1.013
6	128	CrossEntropyLoss	SGD	62.2	1.013
7	256	CrossEntropyLoss	Adam	60.2	1.052
7	256	NLLLoss	Adam	62.1	1.007
7	256	NLLLoss	SGD	62.2	1.013
7	256	CrossEntropyLoss	SGD	57.1	1.117
10	2048	NLLoss	SGD	69.0	0.852