CV Assignment 2

Diego Di Benedetto, Julia Hindel

May 2021

1 Introduction

The following report outlines the configuration of a CNN model for the classification of the FER2013 dataset which contains grayscale images of 7 different facial expressions. In the following, the applied pre-processing, tested architecture variants, visualization and hyperparameter-tuning are outlined.

2 Pre-processing

The data labelled with 'PrivateTest' and 'PublicTest' are used for testing, whereas the 'Training' split is used for training and validation. In order to increase the quantity of data and add variety to the training set, we augment the data with flipping, random cropping, contrast stretching, varying of saturation, brightness adaption and zoom. Additionally, augmentation also balances the number of instances for each class to ensure that the neural network is not biased towards a specific class. In total, we obtain 50505 samples for training and validation. We normalize all data by dividing by the fixed scalar 255 as it represents the maximum intensity range of grayscale images. To calculate robust evaluation scores of the models, we use 5-fold cross validation and save the best performing variant for further visualizations.

3 Architecture

We test two different architecture variants. The first being sequential convolutional layers alternated with batch normalization, max pooling and dropout followed by dense layers (model 1, final architecture is shown in Figure 7). The second architecture (model 2, final architecture displayed in Figure 7) incorporates four convolutions of different kernel sizes which are assembled in parallel. Thereby, we simplified the inception module as proposed in GoogleNet (1). The last layer of our architecture is always a dense layer with 7 neurons to which a softmax function has been applied. Thereby, the value of each neuron represents the probability that the input belongs to this specific class. As a consequence of this representation, we use CrossEntropyLoss as a loss function. Additionally,

we use the Adam optimizer with a learning rate of 0.0001 as it was found to provide the best results. A batch size of 10 is used as a default. After each convolution we apply a ReLU activation as it is commonly used in convolutional networks.

4 Test Results and Discussion

We record the average accuracy, recall and precision of the folds for 5 epochs. Then the final configuration is trained for 10 or 50 epochs due to time constrains. Moreover, more training iterations do not provide substantial improvements. For model 1, we start with 3 blocks consisting of a convolutional layer with kernel size 3x3 and padding followed by batch normalization, ReLU activation and max pooling with size of 2. The number of channels in the convolutional layers are 8, 16, and 32. Then we add two dense layers with 32 and 7 neurons respectively. We observe that the model is slightly overfit as the training loss continues to decrease while the validation stagnates. Consequently, we apply dropout and decrease the complexity by attaching an additional convolutional layer to increase the size of the vector space before the dense layers. We observe that adding this additional layer improved the result. Additionally, we argue that batch normalization (i.e. normalizing the input of layers) significantly helps the generalization of the model as the accuracy amounts to two thirds, if this computation is omitted. In another test, we observed that removing the max pooling after early layers of convolution improves the result. We argue this behaviour with the increased capacity of the network to learn a good representation of the image. At the same time, dropout helped in this case to smooth the validation loss curve. Applying larger kernel sizes in first convolutional or smaller kernel sizes in the last convolutional layers did not improve the performance but also didn't proof to be that determinant for the achieved performance. However, using the preferred activation function ELU (2) resulted in higher accuracy, precision and recall. Applying dilation to the kernel or group convolutions did not show useful for this application. We reason this behaviour as the features in the image are spatial closely correlated and group convolution also reduce the computational complexity. In our case, we should rather increase the complexity. Consequently, we added more dense layers at the end of the network. Thereby, we reach an accuracy of 60% after 50 epochs.

For model 2, we have performed similar experiments. The complete results are shown in the appendix. We carefully tuned dropout and again observed the beneficial effect of applying the ELU activation. Additionally, using a larger kernel size at the top of the network as inspired by ResNet improved the performance. Replacing the inner layers of 3x3 convolution with 1x1 convolutions resulted in a deterioration of the results. Additionally, adding residual connections did not help the networks performance. We reason this observation with the depth of the network. It appears that the network needs to reach a certain depth so this additional input is beneficial. We also increased the batch size to 32, but it did not positively affect the performance. Increasing it to higher

numbers was not possible due to hardware constraints. Additionally, the effect of a higher (1e-3) and lower (1e-5) learning rate are estimated. Even if the higher learning rate shows a faster decrease in loss at the beginning, it doesn't improve the overall accuracy after 5 epochs. The loss of the final configuration after 10 epochs is displayed in Figure 1.

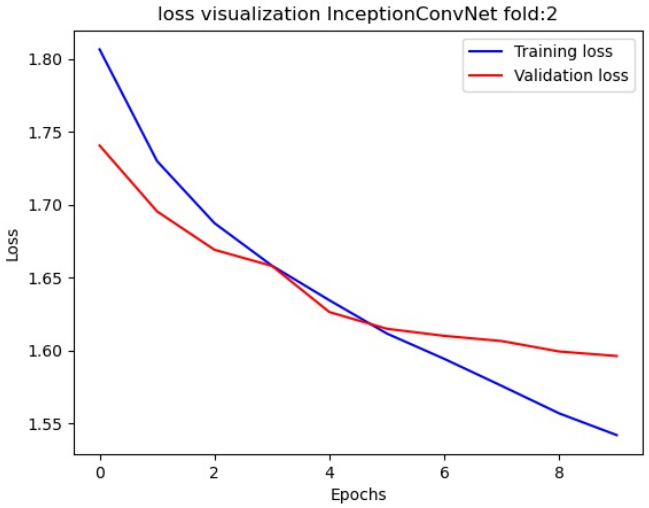


Figure 1: Loss visualization of the best performing fold

5 Visualization

To reason about the learned behaviour of the model, we have implemented various visualization techniques of weights, activations and occlusion. For model 2, the weights of the parallel convolutions of the inception module give indication about which filter has a higher importance. As displayed in Figure 2, layer c and d have a higher number of black pixels indicating low weights. Therefore, layer b (with convolutions of 1x1 followed by 3x3) seem to influence the classification task more strongly. This observation matches our findings of model 1 that kernels of 3x3 are most suitable for these input images.

The activations show which features of the face excite the neurons of a specific filter the most. As can be seen in Figure 2, each kernel focuses on different aspects of the face. For example in layer 1 of model 2, kernel 5 detects the eye region and the background as shown in Figure 3. This observation is made for all tested input images.

On the other hand, occlusion demonstrates which area of the image is particularly distinct for a specific class. In Figure 4, an occlusion analysis is applied for a sample of each class on model 2. One can reason that the network mostly

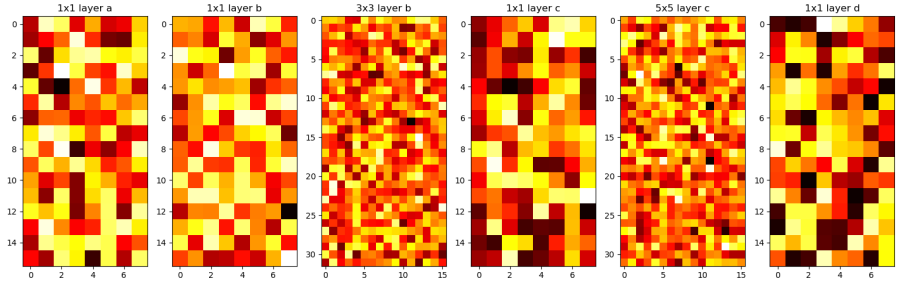


Figure 2: weights of inception module

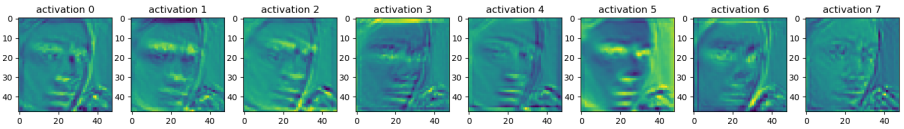


Figure 3: visualization of the activations for each filter

focuses on the bottom part of the face for the class "disgust". In comparison, the mouth seems to be the most distinct feature for the classification of the emotion happiness. For fear, the network focuses on the whole image, however, as can be seen in the confusion matrix shown in Figure 5, the network generalizes badly on this class and mostly miss-classifies samples. In summary, we reason that the network learns to concentrate on regions of the face that also humans would use for the classification of these emotions.

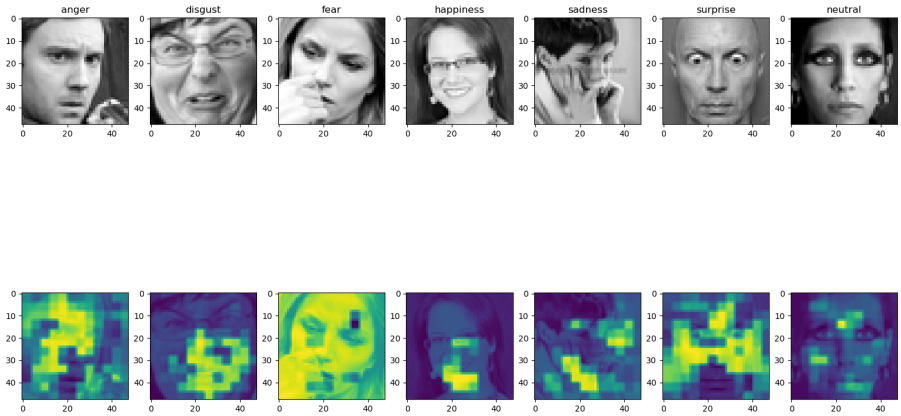


Figure 4: occlusion visualization

6 Conclusion

To conclude, improving the performance of a CNN model for the classification of the FER2013 dataset is a very challenging task as it was difficult to balance effects of over- and underfit. We tested a sequential and parallel CNN architecture and varied a high amount of hyperparameters. It's difficult to find the right combination of number of layers and regularization techniques such as dropout in addition to the most suitable hyperparameters. Even if we only had a relatively small dataset, we could not test all possible combinations. Therefore, we reason that in a real application, it makes sense to exploit popular architectures and also consider transfer learning with pre-trained models. Additionally, we argue that the depth of the network had the biggest effect on the performance compared to parameters such as learning rate, batch and kernel size. The inception-inspired network, thereby, achieved a higher accuracy after a small number of epochs but seemed to easily overfit when trained on a higher number of epochs. Nevertheless, visualialization of weights, activations and occlusion enabled us to reason the plausible learning of the network. Thereby, badly performing classes matched with the networks inability to find a focus area of this specific class.

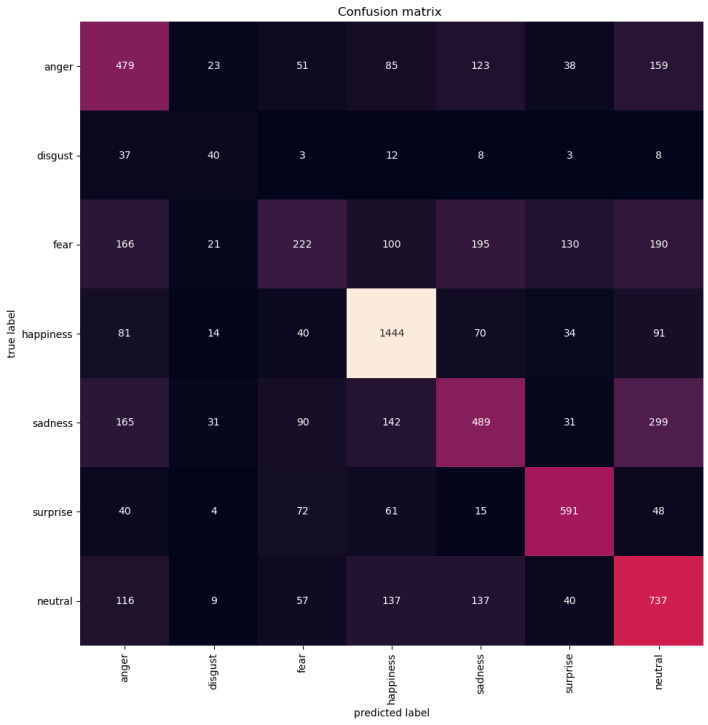
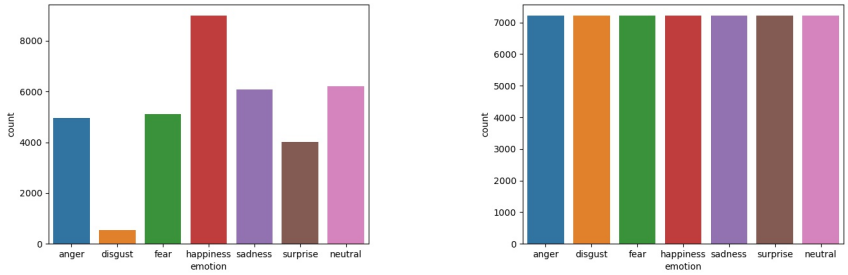


Figure 5: confusion matrix



(a) initial distribution

(b) augmented distribution

Figure 6: augmentation of the training dataset

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol: "O'Reilly Media, Inc.", 2017.

7 Appendix

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 48, 48]	80
BatchNorm2d-2	[-1, 8, 48, 48]	16
ELU-3	[-1, 8, 48, 48]	0
Conv2d-4	[-1, 16, 48, 48]	1,168
BatchNorm2d-5	[-1, 16, 48, 48]	32
ELU-6	[-1, 16, 48, 48]	0
MaxPool2d-7	[-1, 16, 24, 24]	0
Dropout-8	[-1, 16, 24, 24]	0
Conv2d-9	[-1, 32, 24, 24]	4,640
BatchNorm2d-10	[-1, 32, 24, 24]	64
ELU-11	[-1, 32, 24, 24]	0
MaxPool2d-12	[-1, 32, 12, 12]	0
Dropout-13	[-1, 32, 12, 12]	0
Conv2d-14	[-1, 64, 12, 12]	18,496
BatchNorm2d-15	[-1, 64, 12, 12]	128
ELU-16	[-1, 64, 12, 12]	0
Dropout-17	[-1, 64, 12, 12]	0
Conv2d-18	[-1, 128, 12, 12]	73,856
BatchNorm2d-19	[-1, 128, 12, 12]	256
ELU-20	[-1, 128, 12, 12]	0
MaxPool2d-21	[-1, 128, 6, 6]	0
Dropout-22	[-1, 128, 6, 6]	0
Conv2d-23	[-1, 256, 6, 6]	295,168
BatchNorm2d-24	[-1, 256, 6, 6]	512
ELU-25	[-1, 256, 6, 6]	0
MaxPool2d-26	[-1, 256, 3, 3]	0
Dropout-27	[-1, 256, 3, 3]	0
Flatten-28	[-1, 2304]	0
Linear-29	[-1, 128]	295,040
ELU-30	[-1, 128]	0
Linear-31	[-1, 64]	8,256
ELU-32	[-1, 64]	0
Linear-33	[-1, 32]	2,080
ELU-34	[-1, 32]	0
Linear-35	[-1, 16]	528
ELU-36	[-1, 16]	0
Linear-37	[-1, 7]	119

Total params: 700,439
Trainable params: 700,439
Non-trainable params: 0

Figure 7: model 1

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 48, 48]	400
BatchNorm2d-2	[-1, 8, 48, 48]	16
ELU-3	[-1, 8, 48, 48]	0
MaxPool2d-4	[-1, 8, 24, 24]	0
Conv2d-5	[-1, 16, 24, 24]	144
ELU-6	[-1, 16, 24, 24]	0
Conv2d-7	[-1, 16, 24, 24]	144
ELU-8	[-1, 16, 24, 24]	0
Conv2d-9	[-1, 32, 24, 24]	4,640
ELU-10	[-1, 32, 24, 24]	0
Conv2d-11	[-1, 16, 24, 24]	144
ELU-12	[-1, 16, 24, 24]	0
Conv2d-13	[-1, 32, 24, 24]	12,832
ELU-14	[-1, 32, 24, 24]	0
MaxPool2d-15	[-1, 8, 24, 24]	0
Conv2d-16	[-1, 16, 24, 24]	144
ELU-17	[-1, 16, 24, 24]	0
Conv2d-18	[-1, 32, 24, 24]	3,104
BatchNorm2d-19	[-1, 32, 24, 24]	64
ELU-20	[-1, 32, 24, 24]	0
Dropout-21	[-1, 32, 24, 24]	0
Conv2d-22	[-1, 64, 24, 24]	2,112
ELU-23	[-1, 64, 24, 24]	0
Conv2d-24	[-1, 64, 24, 24]	2,112
ELU-25	[-1, 64, 24, 24]	0
Conv2d-26	[-1, 128, 24, 24]	73,856
ELU-27	[-1, 128, 24, 24]	0
Conv2d-28	[-1, 64, 24, 24]	2,112
ELU-29	[-1, 64, 24, 24]	0
Conv2d-30	[-1, 128, 24, 24]	204,928
ELU-31	[-1, 128, 24, 24]	0
MaxPool2d-32	[-1, 32, 24, 24]	0
Conv2d-33	[-1, 64, 24, 24]	2,112
ELU-34	[-1, 64, 24, 24]	0
Conv2d-35	[-1, 64, 24, 24]	24,640
BatchNorm2d-36	[-1, 64, 24, 24]	128
ELU-37	[-1, 64, 24, 24]	0
MaxPool2d-38	[-1, 64, 12, 12]	0
Dropout-39	[-1, 64, 12, 12]	0
Conv2d-40	[-1, 128, 12, 12]	8,320
ELU-41	[-1, 128, 12, 12]	0
Conv2d-42	[-1, 128, 12, 12]	8,320
ELU-43	[-1, 128, 12, 12]	0
Conv2d-44	[-1, 256, 12, 12]	295,168
ELU-45	[-1, 256, 12, 12]	0
Conv2d-46	[-1, 128, 12, 12]	8,320
ELU-47	[-1, 128, 12, 12]	0
Conv2d-48	[-1, 256, 12, 12]	819,456
ELU-49	[-1, 256, 12, 12]	0
MaxPool2d-50	[-1, 64, 12, 12]	0
Conv2d-51	[-1, 128, 12, 12]	8,320
ELU-52	[-1, 128, 12, 12]	0
Conv2d-53	[-1, 256, 12, 12]	196,864
BatchNorm2d-54	[-1, 256, 12, 12]	512
ELU-55	[-1, 256, 12, 12]	0
MaxPool2d-56	[-1, 256, 6, 6]	0
Conv2d-57	[-1, 64, 6, 6]	16,448
BatchNorm2d-58	[-1, 64, 6, 6]	128
ELU-59	[-1, 64, 6, 6]	0
MaxPool2d-60	[-1, 64, 3, 3]	0
Flatten-61	[-1, 576]	0
Linear-62	[-1, 32]	18,464
ELU-63	[-1, 32]	0
Dropout-64	[-1, 32]	0
Linear-65	[-1, 7]	231

Total params: 1,714,183
Trainable params: 1,714,183

complex CNN											
l1	l2	l3	l4	d4	d5	d6	dropout	act.	acc.	prec.	recall
3x3x8	3x3x16	3x3x32		32	7			ReLU	0.47	0.36	0.42
3x3x8	3x3x16	3x3x32		32	7		yes	ReLU	0.43	0.34	0.35
3x3x8	3x3x16	3x3x32	3x3x64	32	7			ReLU	0.46	0.34	0.38
3x3x8	3x3x16	3x3x32	3x3x64	32	7		no batch_n	ReLU	0.33	0.22	0.22
3x3x8 no mp	3x3x16	3x3x32	3x3x64	32	7			ReLU	0.51	0.45	0.46
3x3x8 no mp	3x3x16	3x3x32	3x3x64	32	7		yes	ReLU	0.50	0.43	0.44
5x5x8 no mp	3x3x16	3x3x32	3x3x64	32	7		yes	ReLU	0.48	0.43	0.42
3x3x8 no mp	3x3x16	3x3x32	1x1x64	32	7		yes	ReLU	0.44	0.38	0.38
3x3x8 no mp	3x3x16	3x3x32	3x3x64	32	7		yes	ELU	0.50	0.47	0.45
3x3x8 no mp	3x3x16, dil	3x3x32	3x3x64	32	7		yes	ELU	0.49	0.42	0.41
3x3x8 no mp	3x3x16, gc	3x3x32	3x3x64	32	7		yes	ELU	0.49	0.44	0.42
3x3x8 no mp	3x3x16, gc	3x3x32	3x3x64	32	7		yes	ELU	0.50	0.43	0.45
3x3x8 no mp	3x3x16	3x3x32	3x3x64	32	16	7	yes	ELU	0.46	0.36	0.37

inception model							
l1	l2	l3	dropout	act.	acc.	prec.	recall
inc	inc	inc	no	ReLU	0.55	0.52	0.49
inc	inc	inc	yes (0.2)	ReLU	0.53	0.48	0.48
inc	inc	inc	yes (0.2)	ELU batch=32	0.53	0.41	0.49
inc	inc	inc	yes (0.2)	ELU batch=64	no fit in memory		
inc	inc	inc	yes (0.2)	ELU lr=1e-5	0.43	0.37	0.38
inc	inc	inc	yes (0.2)	ELU lr=1e-3	0.47	0.33	0.38
inc	inc	inc	yes (0.1)	ReLU	0.53	0.49	0.46
inc	inc	inc	yes (0.1)	ELU	0.54	0.42	0.49
7x7 inc	inc	inc	yes (0.1)	ELU	0.54	0.49	0.50
7x7 inc	3x3 inc	3x3 inc	yes (0.1)	ELU	0.53	0.48	0.47
7x7 inc	inc + sc	inc	yes (0.1)	ELU	0.53	0.49	0.48
7x7 inc	inc	inc	yes (0.1)	ELU, more dense	0.53	0.47	0.48