Computer Vision

Assignment 2 Report

Jannik Wirtz - i6292051, Lukas Santing - i6298143

1. Model Optimization Experiments

In order to construct our model, we used the Keras TensorFlow library (Chollet et. al., 2015). We experimented with a variety of structures, as can be found below. While we tested the final model on accuracy, precision and recall, we based the hyperparameter choices on just the accuracy of the predefined validation set.

We combine every convolutional layer in our feature extractor with batch-normalization, max-pooling and dropout. We skip the batch-normalization for the input layer, since the data is already standardized. We also limit the number of pooling layers to a maximum of 3, since the network's internal data representation would become too small to function.

1.0 Initial Setup (Base model)

Optimizer:
learning_rate = 1e-3
batchsize = 64
epochs = 20
loss='categorical_crossentropy'

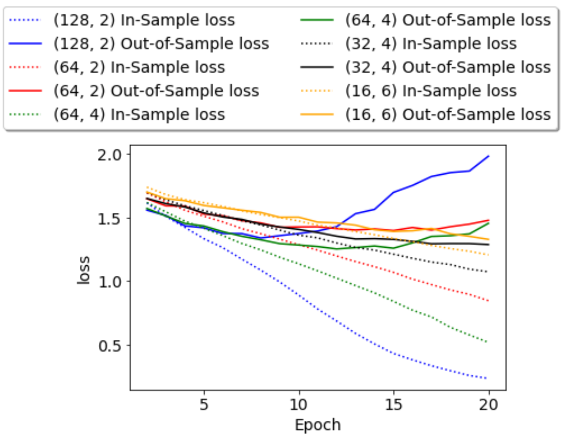
Model:
initial_units = 64 // Units per layer double every conv layer and halve every dense layer
conv_layers = 2
dense_layers = 3 // 2 dense layers as classifier + 1 dense output layer
zero-padding='same'
kernel-sizes = 3x3
dilation-rate=(1, 1) // ie. None
dropout-rate=0.25
activations='relu' (except output-layer; 'softmax')
Validation Accuracy – Base Model: 51%

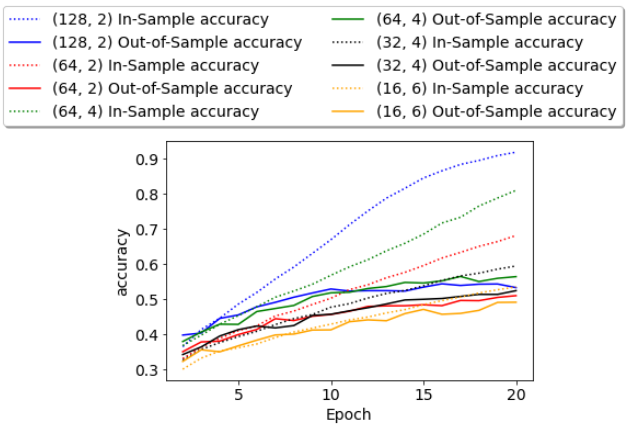
1.1 Layers and units/filters

Tuple description := (initial #-Units/Filters, #-layers) // Units per layer double every layer

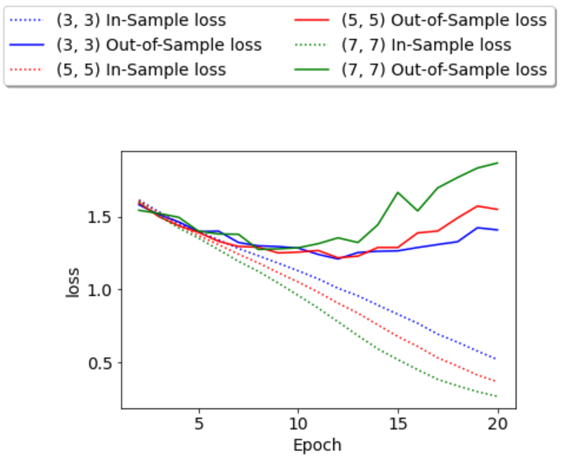
We can see that most models exhibit a tendency to overfit between epochs 10 to 15, recognisable by the increasing out-of-sample loss (validation error). Exceptions to this are models with the least number of parameters, these however provided much worse accuracies.

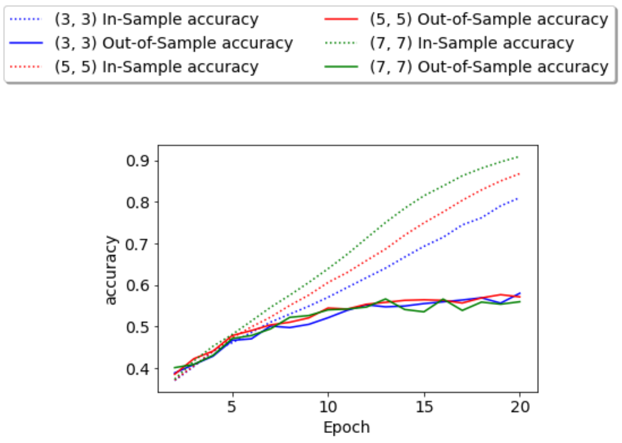
We chose to proceed with the (64, 4) model, because it made the most accurate predictions on the validation data after 20 epochs. **Choice:** Feature extractor with 4 Convolutional Layers with 64, 128, 256 and 512 filters respectively.





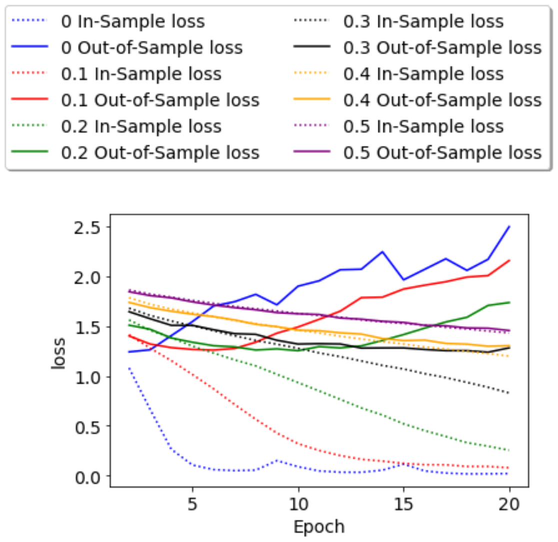
1.2 Kernel sizes for every layer

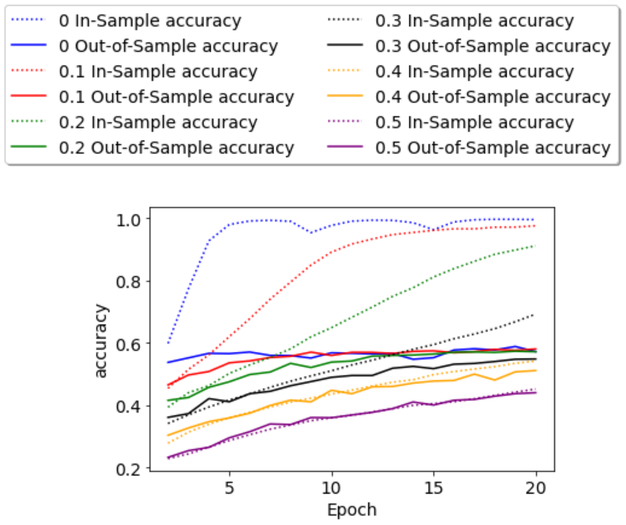




The second series of experiments focused on the kernel sizes of all Conv-layers. We did not find a significant difference in performance, such that we decided to continue with the smallest of the kernel-sizes, since it is the most computational efficient. **Choice:** 3x3 kernel sizes

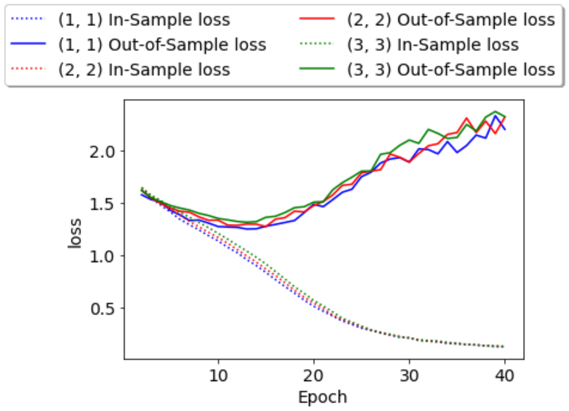
1.3 Dropout rates for all layers

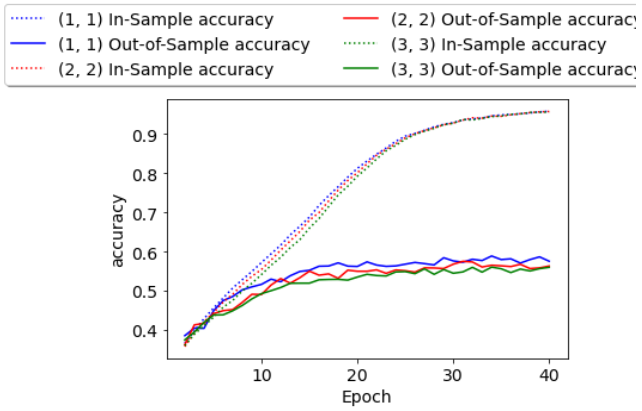




In order to determine a good strength of dropout regularization for our model, we tested different dropout-rates. The best tradeoff between overfitting mitigation and generalizability was achieved with a dropout rate of 0.25. **Choice:** 25% Dropout for all layers (including dense layers)

1.4 Dilation Layers for first layer





No significant difference in performance was observable for different dilation rates for the CNN-Layers. **Choice:** default(1,1) dilation rate

1.5 Best parameters

The best parameters that we found from these experiments are as follows.

Optimizer:
learning_rate = 1e-3
batchsize = 64

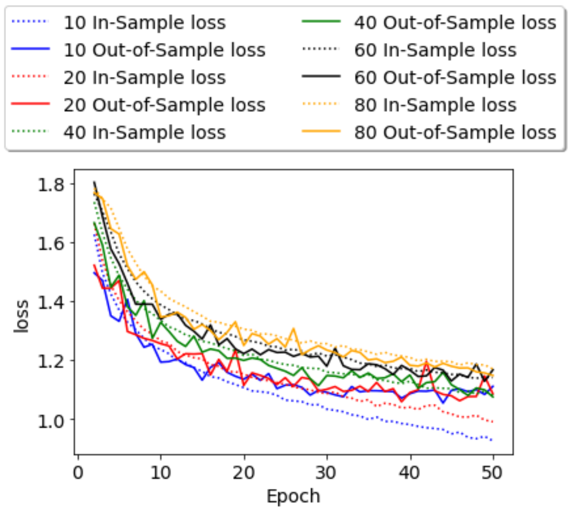
epochs = 60
loss='categorical_crossentropy'

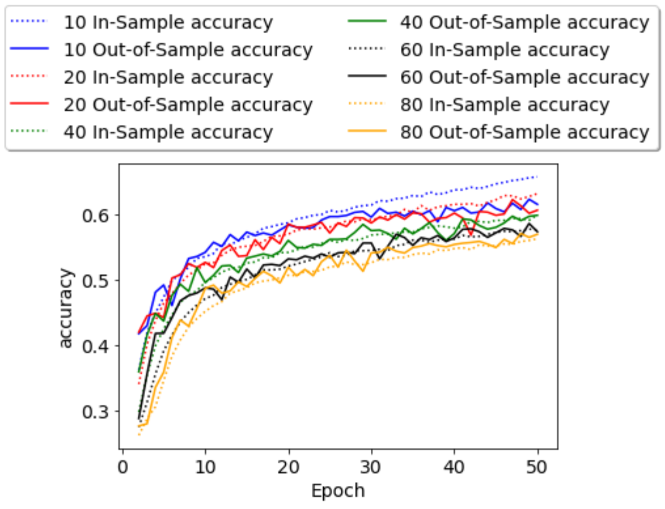
Model:
initial_units = 64 // Units per layer double every conv layer and halve every dense layer
conv_layers = 4
Dense_layers = 3
zero-padding='same'
kernel-sizes = 3x3
dilation-rate=(1, 1) // ie. None
dropout-rate=0.25
activations='relu' (except output-layer; 'softmax')

2. Data Augmentation Experiments

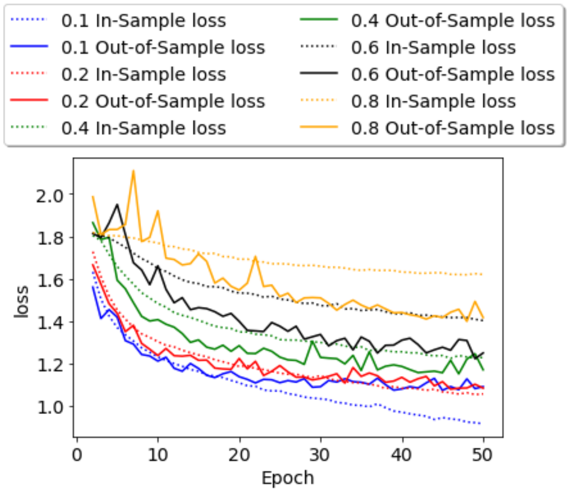
Once we had the results for the model structure, we further experimented data augmentation. We took our best model from the previous section, and used the Keras data augmentation function (Chollet et. al., 2015), and experimented with various augmentation parameters to find the optimal setup. Augmenting the training data before training the model can help it generalize to different data. We therefore expected to see an increase in performance, and saw an increase from about 56% in accuracy without data augmentation to roughly ~62%. Just as with the model optimization, we tested on the accuracy when tested on the predefined validation set.

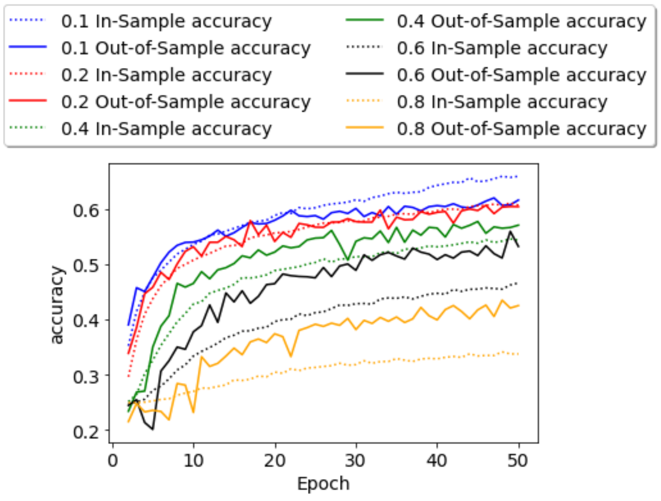
2.1 Rotation Range



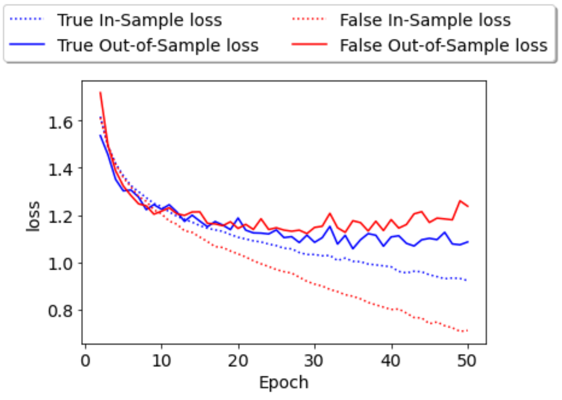


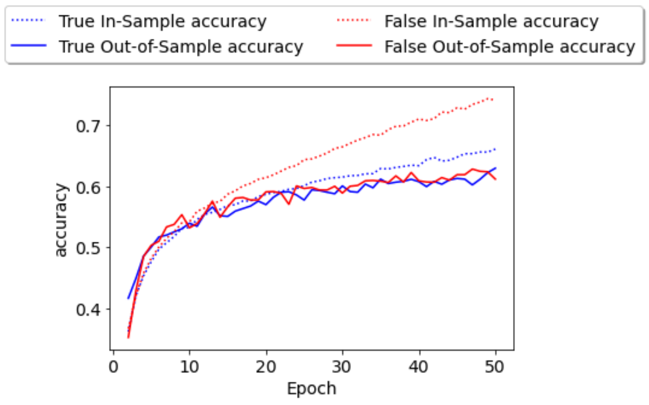
2.2 Horizontal and vertical shift range





2.3 Horizontal Flip (Boolean)





2.3 Best parameters

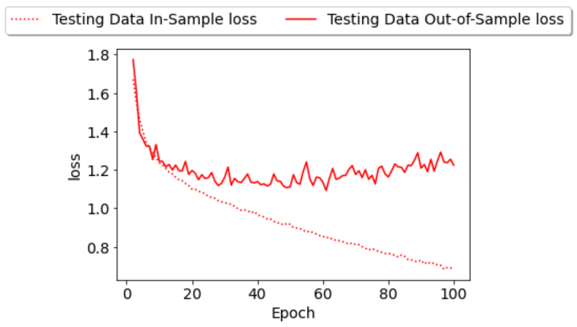
The best parameters from our experiments show that, while data augmentation does help the model to perform better on unseen images, too large values for these operations decrease this advantage. Slight data augmentation is better than heavy data augmentation.

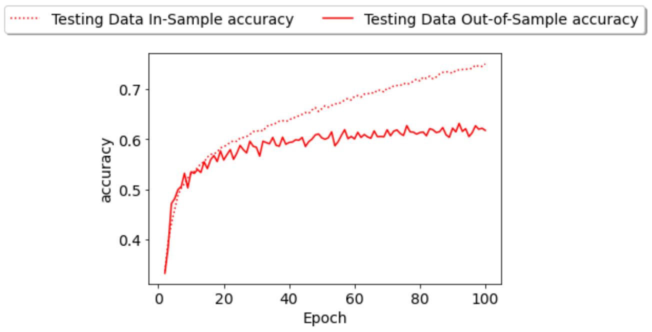
Data Augmentation best parameters
Rotation range (angle): 10
Horizontal and vertical shift: 0.1
Horizontal flip (boolean value): True

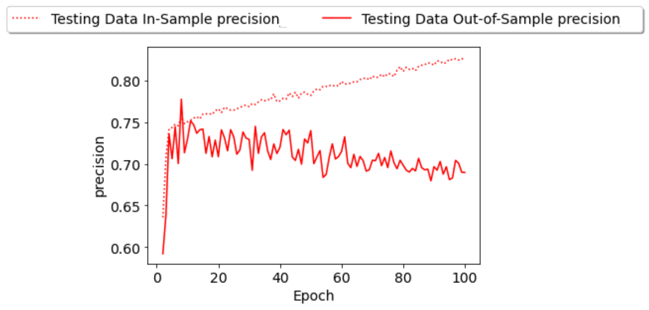
3. Final Model Architecture

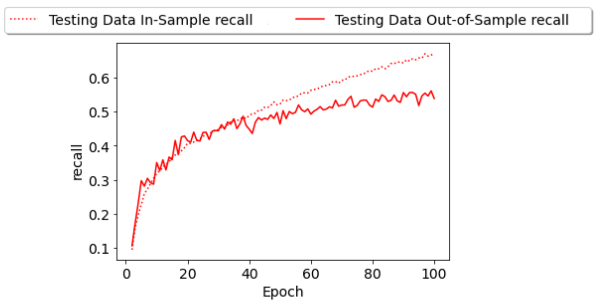
The final that we used for the remaining experiments was a combined result of the best parameters as discussed in sections 1 and 2. We trained a model for 100 epochs with these parameters and tested them on accuracy, precision, and recall on the predefined test set, as seen in section 3.1. We then further trained a 60 epoch version of the model which we tested with 5-fold cross validation, as seen in section 3.2.

3.1 Evaluation on Test data









3.2 Evaluation on 5-fold cross validation

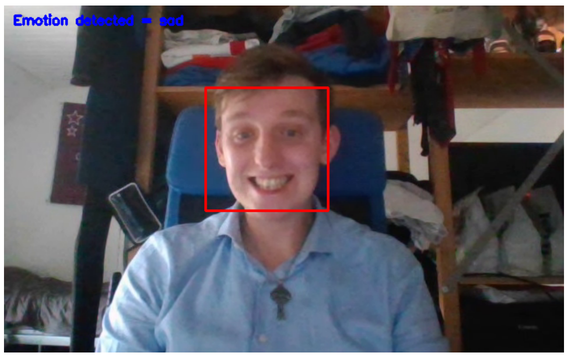
Below are the results, in tabular format, of performing 5-fold cross validation on our final model.

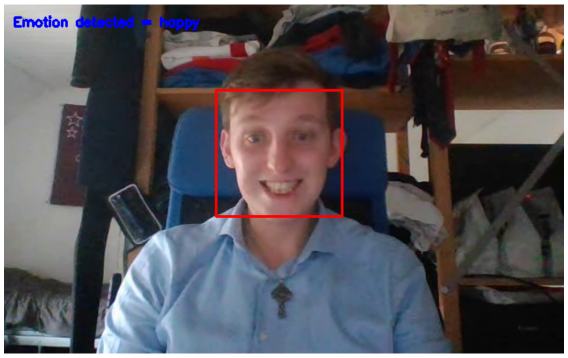
	Training Loss	**Training Accuracy**	**Training Precision**	**Training Recall**	**Test Loss**	**Test Accuracy**	**Test Precision**	**Test Recall**
1	0.8376	0.6888	0.7993	0.5707	1.1326	0.6114	0.7137	0.5033
2	0.8772	0.6784	0.7971	0.5466	1.0869	0.6237	0.7296	0.5172
3	0.8334	0.6937	0.7982	0.5760	1.0967	0.6220	0.7293	0.5161
4	0.8442	0.6845	0.7973	0.5656	1.1214	0.6186	0.7148	0.5128
5	0.8536	0.6857	0.7996	0.5591	1.1239	0.6092	0.7160	0.4914

4. Real-life webcam test

We also used a webcam to record a video, on which we did classification of emotions. For this we used the facial detection software from OpenCV (Bradski, 2000). Below are some examples of classifications of frames representing a variety of emotions. A wide variety of emotions were tested, but for sake of brevity we show examples for “happy” and “neutral”.

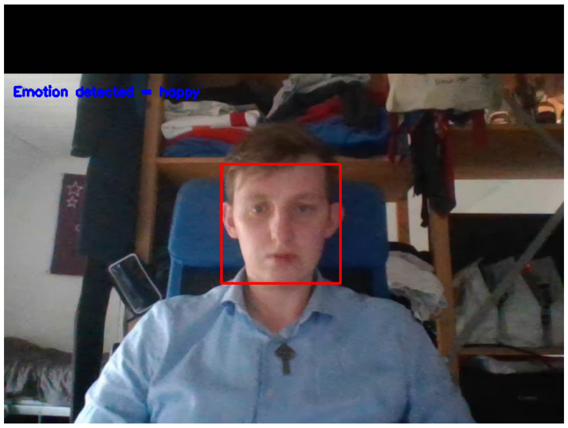
Predictions for: “Happy” emotion:

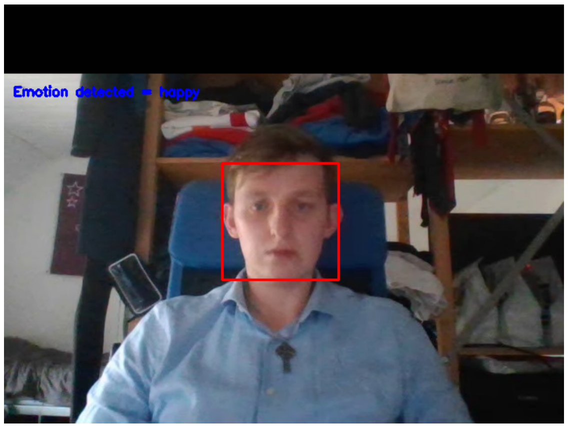




This emotion sequence lasted roughly 34 frames, and was (in our estimation, correctly) classified as either “surprise” or “happy” 18 out of 34 times. The other frames were predicted to be either “fear”, or surprisingly “sad”. Fear could to a certain extent be expected due to the baring of teeth, but we are not sure what justification it could have for the identification of sadness in these frames.

Predictions for: “Neutral” emotion





At least, these were my attempts to create a “neutral” expression. None of these frames were classified as neutral, most were classified as “sad”, which is understandable, with a surprising number of “happy” and even “surprise”.

The model seems to identify all emotions or very similar ones, and over the course of multiple frames tends to identify the emotion correctly at least once.

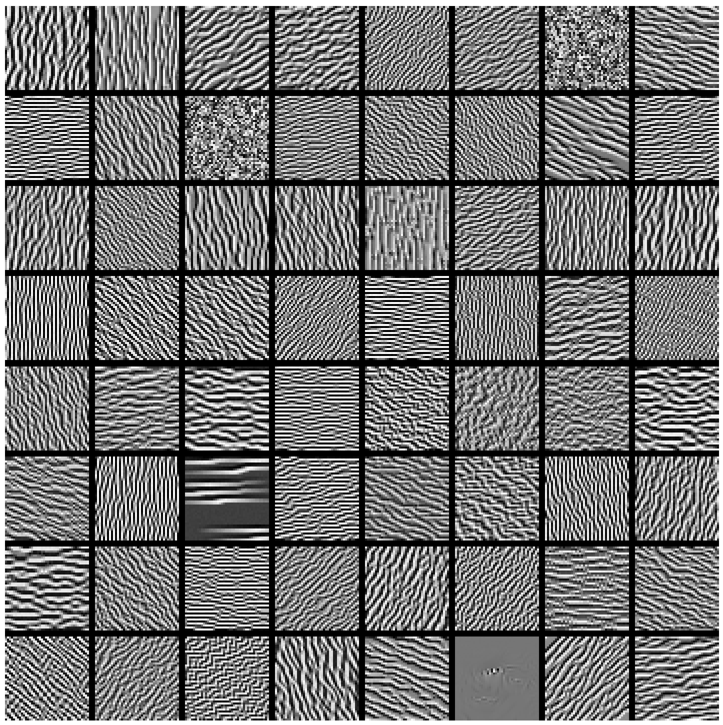
However, as an emotion in a video tends to last for multiple frames and each frame gets a unique prediction, there are also many chances for it to predict the emotion incorrectly. It often does this, and seems to tend towards the ‘sad’ classification. This is interesting, as the ‘sad’ emotion does not appear more often than other emotions in the dataset.

5. Filter Visualization

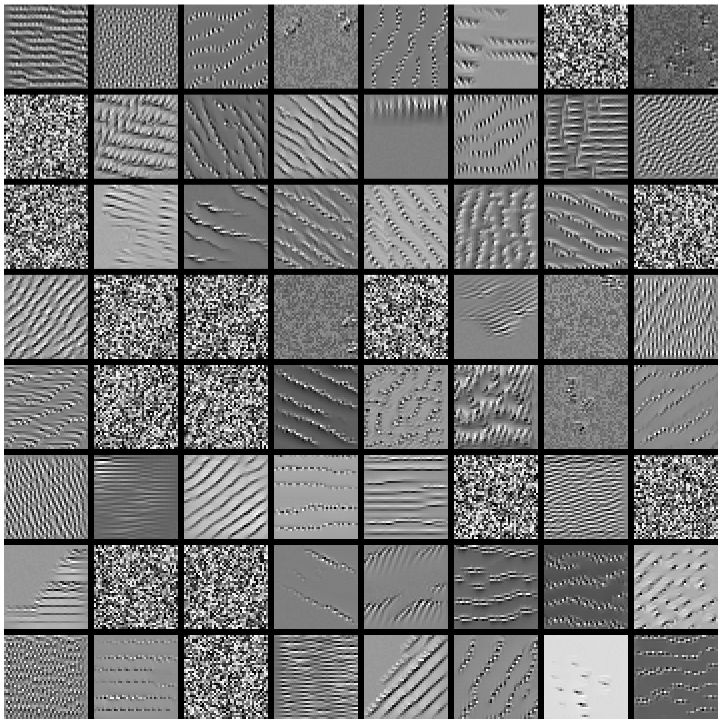
Disclaimer: Due to the optimization based visualization approach chosen here, there are some filters which do not converge to something other than black/white noise. Hyperparameters for this visualization approach include a number of iterations and a learning rate. In order to generate the images below 100 iterations were used with a learning rate of 10.

As we look at the filters of deeper layers, we see how the features become more complex and even seem to consist of combinations of features from previous layers. The earlier layers are clearly specialized on simple lines and textures. The third convolutional Layer of our architecture clearly learned to identify eyes and noses, while the fourth and last conv. Layer seems to combine these features into abstract face-like structures.

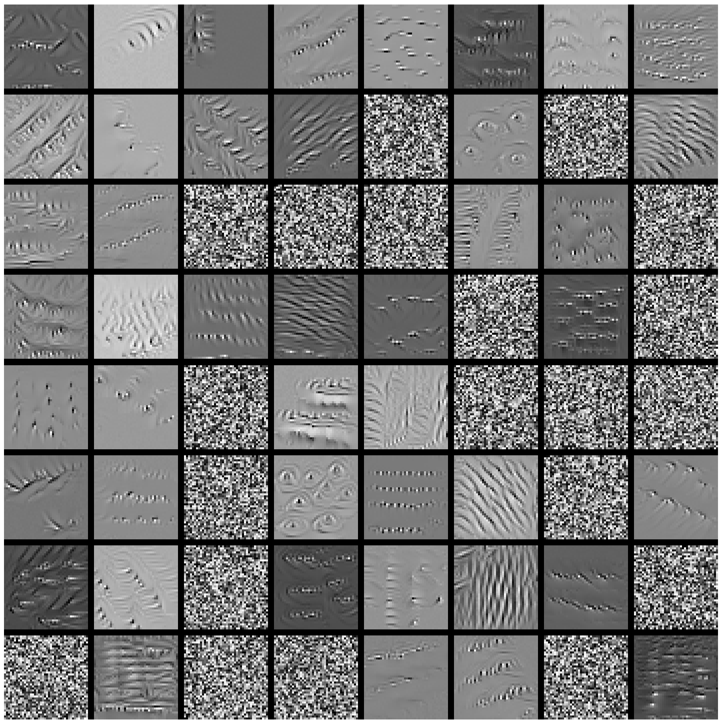
Conv Layer 1: (all 64 filters)



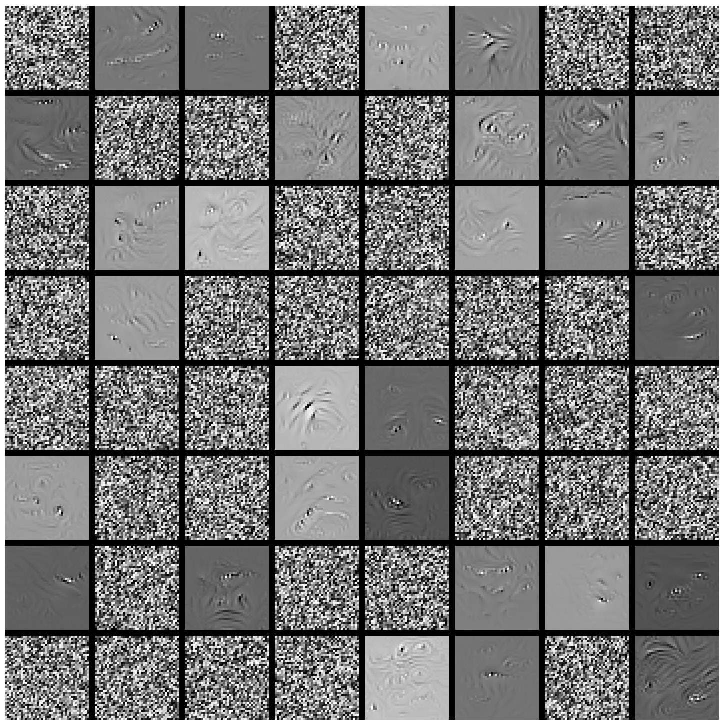
Conv Layer 2: (First 64 filters)



Conv Layer 3: (First 64 filters)



Conv Layer 4: (First 64 filters)



6. Conclusion

As with any other experiment it is crucial to only ever vary a single or at most two variables of the model when performing a hyperparameter optimization via grid-search. It is the only way to achieve meaningful results which lead to improvements in prediction accuracies and generalizability of the model. We managed to improve our initial model from a 51% validation accuracy to around 56%. With additional training data generated with the use of data augmentation we were able to improve our models performance even further and were able to achieve a validation accuracy of 62%.

References

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb&#x27;s Journal of Software Tools*.

Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras