Facial Expression Recognition

Chinmay Rao and Suraj Pai

May 22, 2020

1 Introduction

In this report we present our design, implementation and evaluation of Convolutional Neural Networks for classifying facial expressions. We highlight our methods and implementation in section 2 followed by our experiments, results and interpretations in section 3. Section 4 describes our final model with an analysis of its performance and interpretability.

2 Methods and Materials

2.1 Dataset

FER13 Facial Expression dataset consists of 48x48 grayscale images of human faces. Each image corresponds to one of the seven expressions - angry, disgust, fear, happy, sad, surprise and neutral. The training set comprises of 28,709 instances. The test set is split into - PublicTest and PrivateTest - each having 3,589 instances.

Category	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Number of instances	3995	436	4097	7215	4830	3171	4965
Percent of the total	13.91	1.52	14.27	25.13	16.82	11.04	17.29

Table 1: Distribution of the FER13 training data. It is seen here that the training set is imbalanced since in a perfectly balanced case, each class would represent 14.27% of the samples

2.2 Model

We selected an initial model loosely inspired by AlexNet[1]. This was heavily cut short and modified because of small image sizes in the dataset and incompatible sizes of feature maps as we move to deeper layers. The final *baseline* architecture is a simple 4-layer architecture shown in table 2.

Layer	**Sizes**	**Activation**	**Regularization**
Conv2d	kernel size=3x3, kernels=32	ReLU	-
Conv2d	kernel size=3x3, kernels=32	ReLU	-
Maxpool2d	kernel size=2x2, stride=2	-	-
Conv2d	kernel size=3x3, kernels=64	ReLU	-
Conv2d	kernel size=3x3, kernels=64	ReLU	-
Maxpool2d	kernel size=2x2, stride=2	-	-
AdaptiveAvgPool2d	output size=5x5	-	-
Linear	Units=64	ReLU	Dropout
Linear	Units=7	-	-

Table 2: Building blocks of the baseline architecture. Adaptive average pooling is applied before the Linear layers to ensure a fixed size input since feature-maps can vary in size across different sized kernels.

We implement a custom JSON-based model definition structure to allow for easy experimentation and tracking [1]. F1 score and accuracy are used to quantify the performance of our models. Note that, F1 score considers both precision and recall.

[1]Refer to README.md/README.pdf in the code for more detailed information

3 Experiments

Our first experiment involved choosing an appropriate optimizer where we considered SGD and ADAM with learning rates of 0.01, 0.001 and 0.001. SGD showed lacking perfomance across all learning rates while ADAM was able to perform sufficiently well at a learning rate of 0.01. Moving forward, we use ADAM with LR of 0.01 across all experiments.

3.1 Weighted Loss

Class imbalance can significantly impact prediction performance if the testing distribution is balanced. Taking this into account, we investigate weighted loss where each class is assigned a weight corresponding to the inverse of its frequency in the training data. Higher penalties are imposed for wrong predictions of a low frequency class.

Figure 1 shows validation loss and accuracy during training and figure 2 shows validation F1-score with and without weighted cross-entropy.

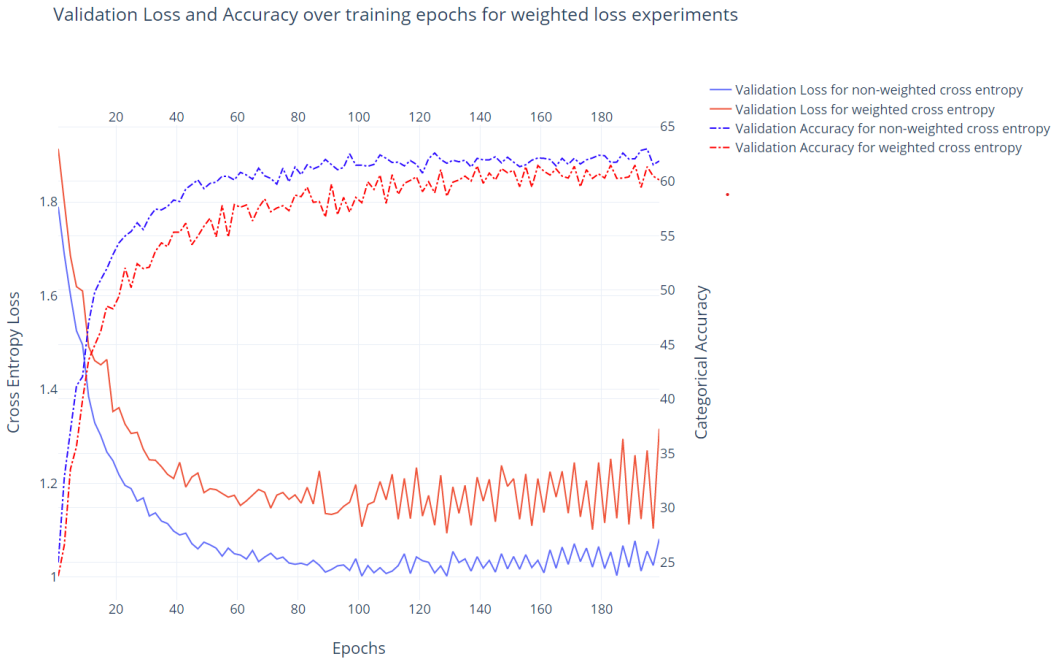


Figure 1: Validation loss and accuracy curves over training epochs for two models trained with and without weighted cross entropy. The weighted loss models are seen in red while the non-weighted loss models are in blue. The accuracies are represented with dotted lines.

Figure 2: F1-score on the validation data for two models trained with and without weighted cross entropy

We observed that, contrary to what one would expect, the validation accuracy and F1-score of the model with weighted loss is lower than the model with uniformly applied cross-entropy. This maybe due to the validation set being sampled randomly from the unbalanced training set which leads to over-penalizing wrong predictions for less-representative classes. As a result, future experiments use non-weighted cross entropy loss.

3.2 Data Augmentation

In this experiment, we analyse the effect of 3 different types of data augmentation: Random Horizontal Flip, Random Affine and Color Jitter on model performance. During training, at every alternate epoch, one of the three transforms is chosen randomly and applied across the entire training dataset. Figure 3 shows the training and validation accuracies with and without data-augmentation.

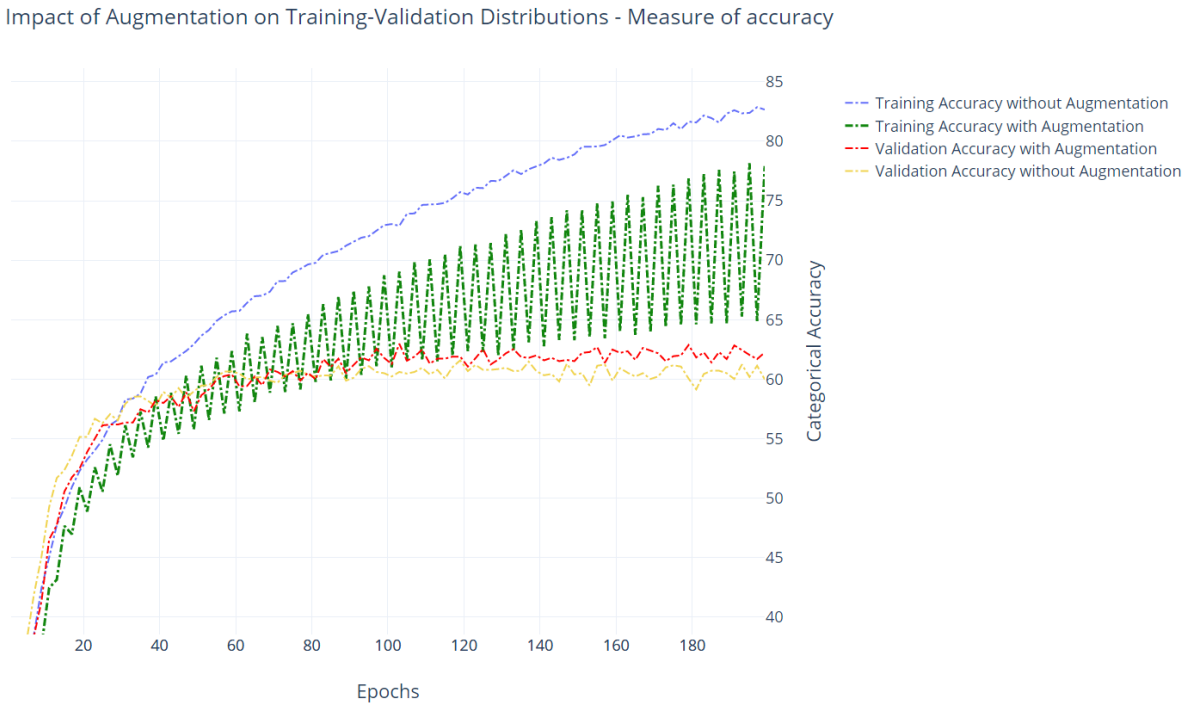


Figure 3: Training and validation accuracy curves for the baseline model and the data-augmented model. Blue and green dotted lines show training accuracies with and without augmentation respectively. The red and yellow dotted lines show the validation accuracies on the same.

An interesting observation is the oscillating nature of the training accuracy when augmentation is applied. This can be attributed to the way data-augmentation is applied since the drop is noticed on epochs when the data is augmented. The training accuracy for augmentation stays lower than the baseline but the validation accuracy shows improvement over the baseline. This can be explained by the fact that the augmented model, due to being trained on more diverse data distributions, has generalized better.

Figure 4 shows the validation losses and accuracies with and without data-augmentation. Figure 5 shows F1-scores across these configurations.

Figure 4: Validation loss and accuracy curves for the baseline model and the data-augmented model. The data-augmented models are seen in red while the non-augmented models are in green. The accuracies are represented with dotted lines.

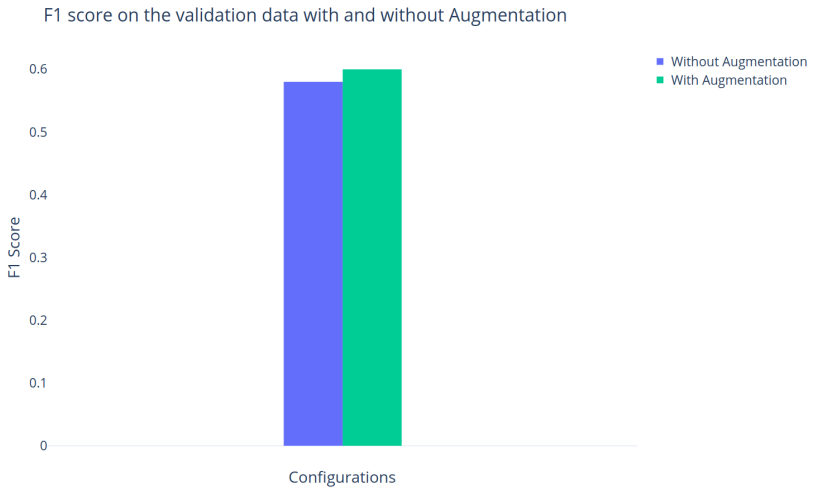


Figure 5: F1-score on the validation data for two models trained with and without data-augmentation

Across both these figures, the model trained with data augmentation performs better in terms of accuracy and F1-score. All models (including baseline) in the subsequent experiments use augmentation as it provides better performance.

3.3 Effect of Network Depth

We compare the baseline model with three new models having 5, 6 and 7 convolutional blocks. In the 5-block model, an additional block with kernel size 3x3 and 128 kernels is introduced. For the 6-block model, this is extended by another block of the same properties. Finally, in the 7-block model, a block with 256 kernels of size 3x3 is added. The validation losses and accuracies for the four models are shown in figure 6.

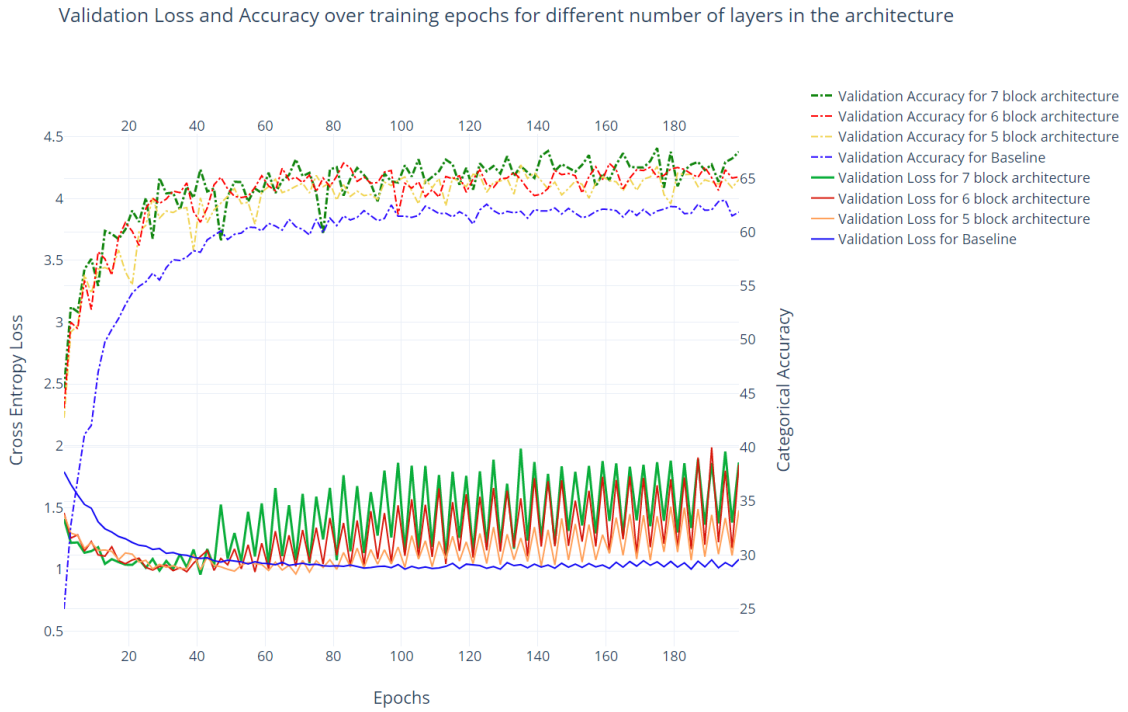


Figure 6: Validation loss and accuracy curves for four models with different number of layers - (data-augmented) baseline, 5-block, 6-block and 7-block

Although the validation losses for the three deeper models are oscillating and generally increasing with the loss for the deepest model (7-block) being largely the highest, this model's accuracy is the highest among the four. The increase in loss accompanied by increase/stagnation in accuracy, according to our hypothesis, is due to the bad predictions becoming slightly worse every epoch. This would increase the loss, but keep the accuracy constant since for computing accuracy, only the maximum probability prediction is considered. The validation F-1 scores are shown in figure 7 with 7-block performing the best.

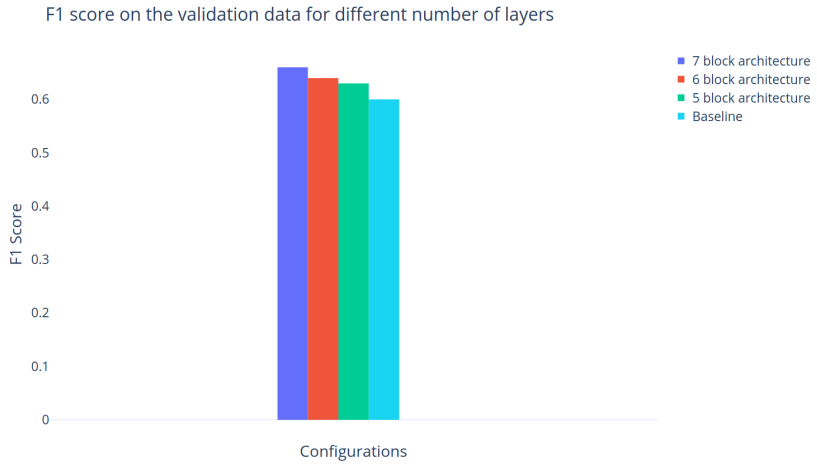


Figure 7: F1-score on the validation data for four models with different number of layers - (data-augmented) baseline, 5-block, 6-block and 7-block

3.4 Impact of Kernel Sizes

For studying the effect of convolution kernel sizes on model performance, we construct models with increased kernel sizes in the first layer or both first and second layers compared to the baseline. The key idea here is to investigate if larger receptive fields lead to better prediction performance and if so, at what layer should the field be increased. Four new models are created with kernel sizes [5x5 and 3x3], [5x5 and 5x5], [7x7 and 3x3], and [7x7 and 7x5] in the first and second layers respectively. For convenience, we refer to them as 5-3-kernelSize, 5-5-kernelSize, 7-3-kernelSize and 7-5-kernelSize models. Figure 8 shows the validation

accuracy and loss curves.

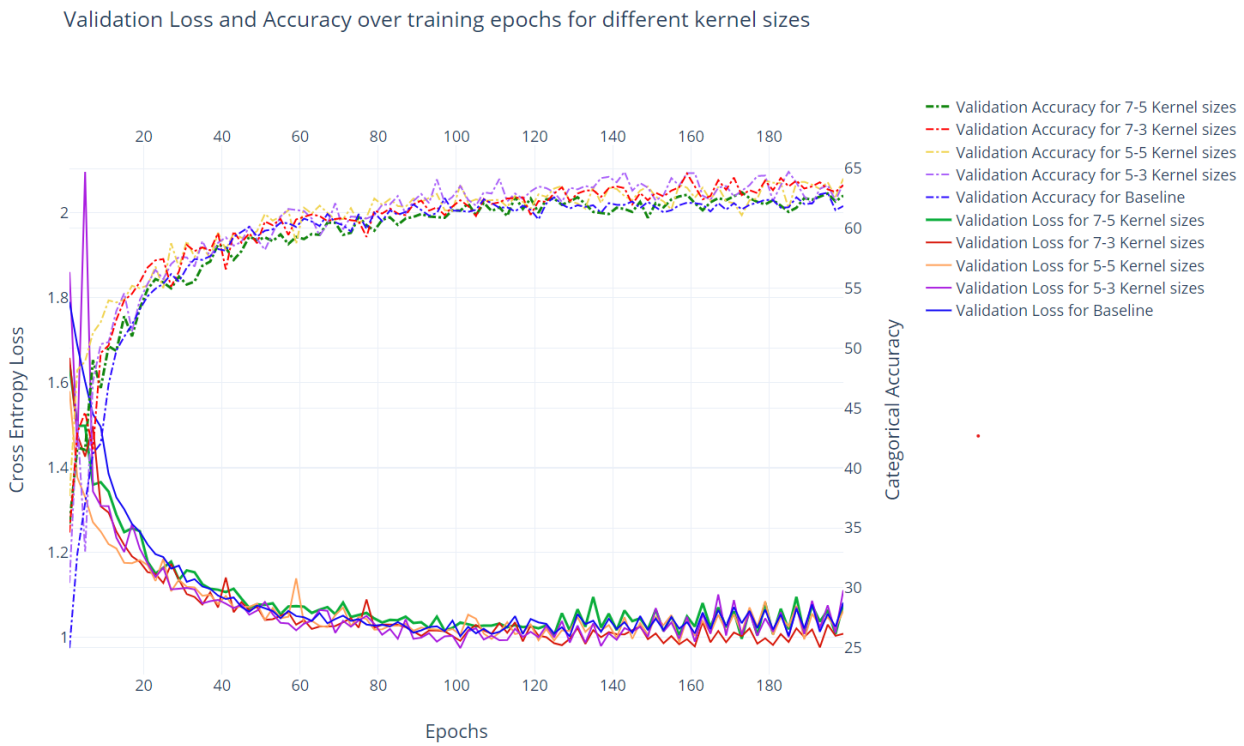


Figure 8: Validation accuracies and losses for the baseline and the 5-3-kernelSize, 5-5-kernelSize, 7-3-kernelSize and 7-5-kernelSize models

The accuracy of the 5-3-kernelSize model is observed to be the highest from epoch 90 to 180 almost consistently, although it dropped to the second position at the end with kernel size 5-5 performing the best. The F-1 scores in figure 9 show comparable performance between 5-5 kernelSize and 5-3 kernelSize with 5-3 kernel slightly better.

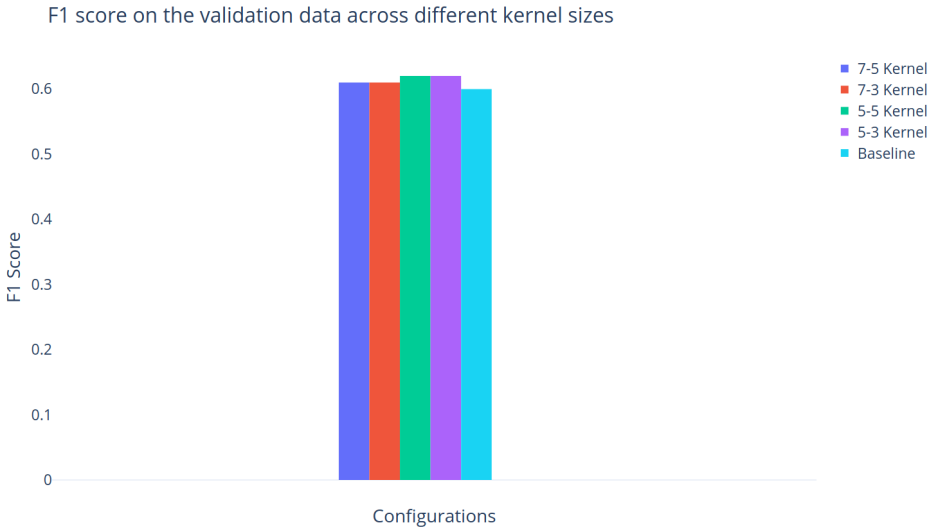


Figure 9: F-1 scores for the baseline and the 5-3-kernelSize, 5-5-kernelSize, 7-3-kernelSize and 7-5-kernelSize models

Our hypothesis is that the filter size 5x5 at the first layer would have been just large enough to extract sufficient local information but increasing it further leads to worse performance. A similar reasoning can be applied to the 3x3 kernels in the second layer which capture features from the first layer's activations.

3.5 Different Normalization Methods

In this experiment, we use two different normalization methods - *batch normalization* and *instance normalization* - to study their influence on the model's performance. Instance normalization differs form batch normalization in that the normalization operation is performed across spatial locations only per instance of a batch as opposed to across both spatial locations and across the entire batch.

The results of training are shown in figure 10.

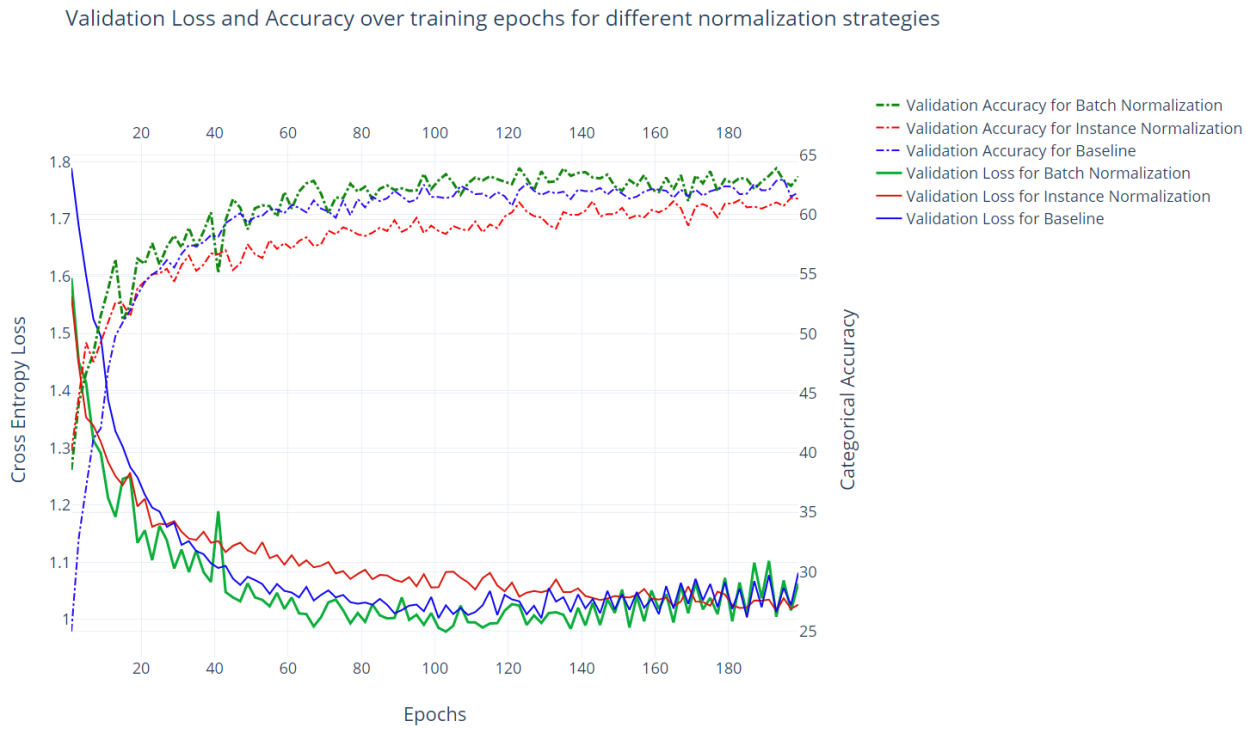


Figure 10: Validation accuracy and loss curves for baseline and models using batch normalization and instance normalization.

It is observed that the validation accuracy of the batch-norm model was consistently the highest while instance norm performed worse than the baseline. The superiority of batch normalization is revalidated with F1-scores in figure 11.

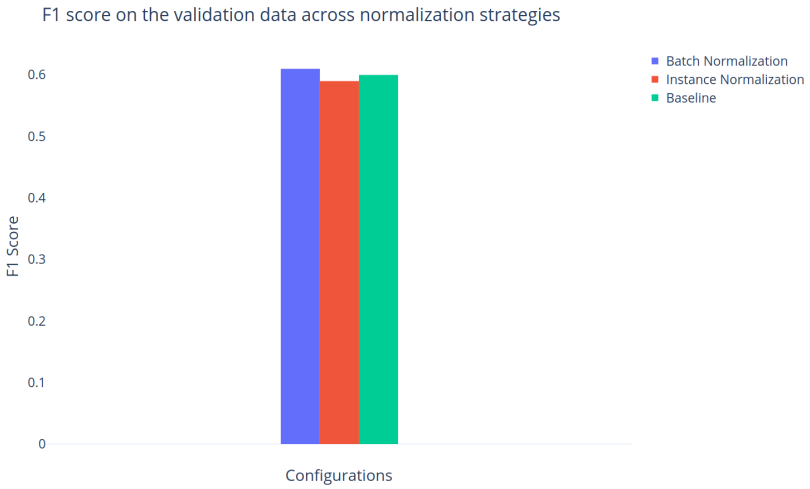


Figure 11: F-1 scores for the batch-norm model, instance-norm model and baseline

4 Final Model Analysis

Results of the ablation studies conducted led us to the final model architecture shown in figure 12.

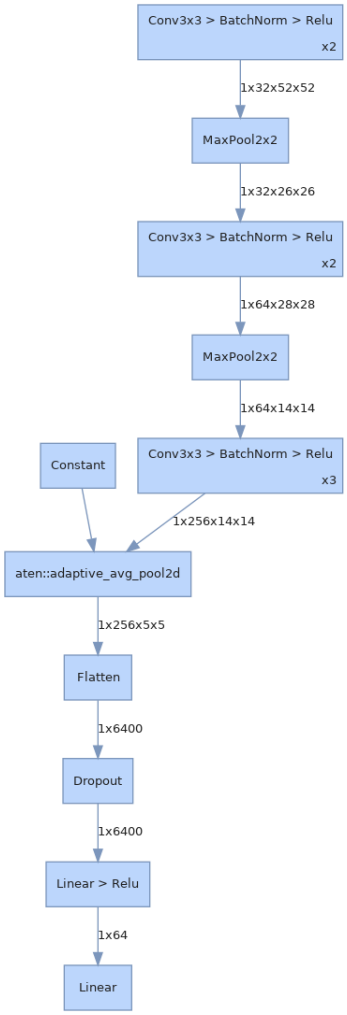
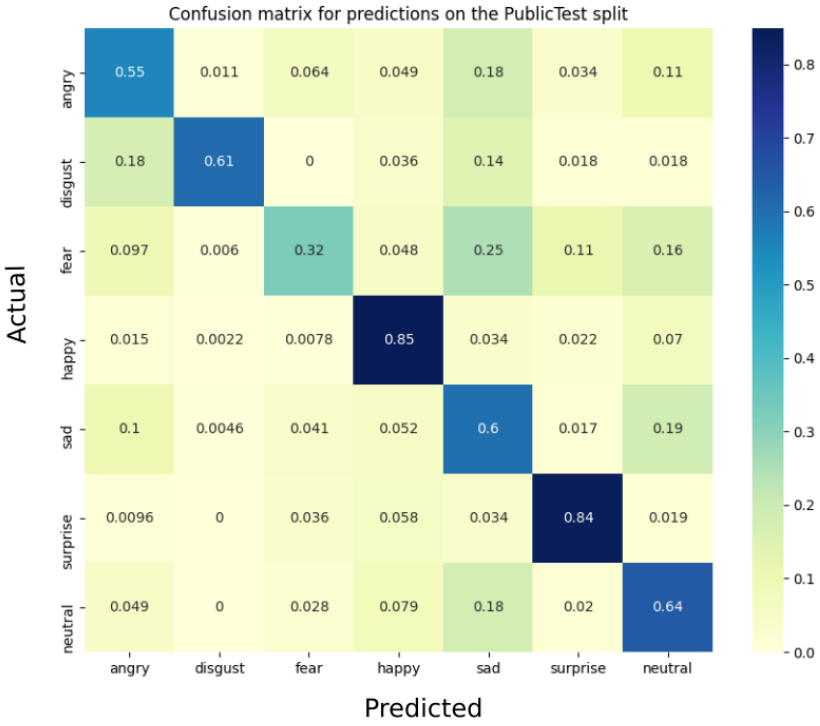


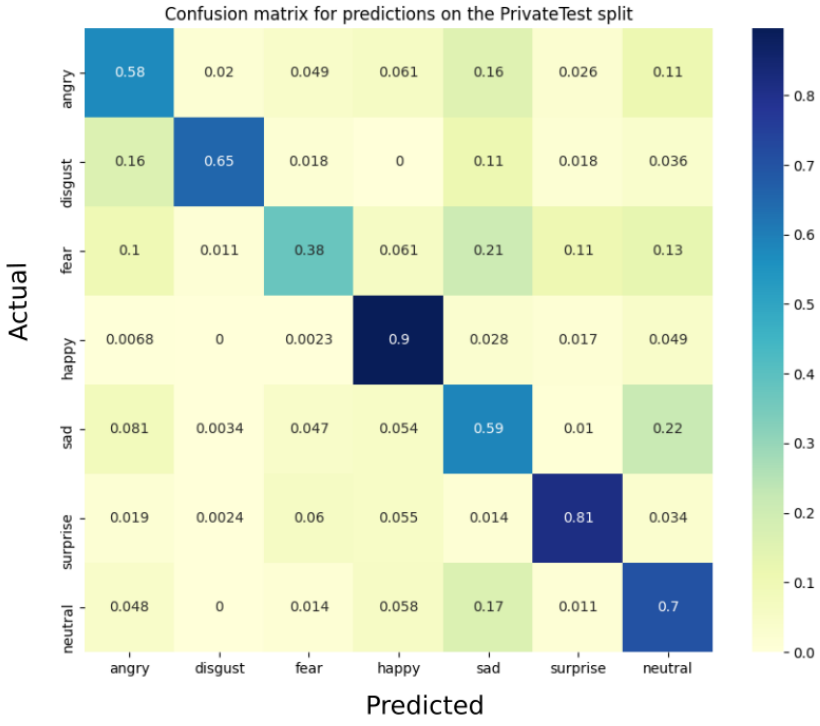
Figure 12: Architecture of the best model

The model consists of 7 convolutional layers with 3×3 kernel sizes across the layers. Despite the 5×3 kernel-size producing the best results in experiment 3.4, better performance with 3×3 kernel-size was found when the number of layers was higher. As determined from experiments 3.5 and 3.2, we apply batch normalization and data-augmentation during training.

Results of the final model on the testing data are shown in figure 13 with a confusion matrix. It can be seen that happy and surprise are predicted accurately across both the splits with low false positives. Fear is predicted with a low accuracy although it forms a significant portion (15%) of the training samples. Interestingly, fear is mis-classified as sad or neutral approximately 34% of the time on PublicTest and 41% on PrivateTest. This could be attributed to lack of easily identifiable features for fear in the training images to distinguish it from the other two emotions.



(a) On PublicTest split



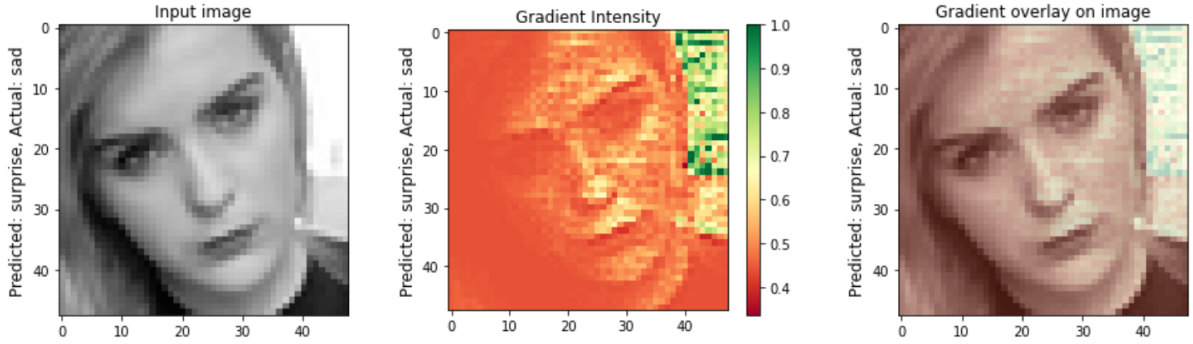
(b) On PrivateTest split

Figure 13: Normalized confusion matrices on the two test data splits. A dark blue value represents high values for a particular element in the matrix. Dark blue values at elements corresponding to the same emotion in the matrix are desired.

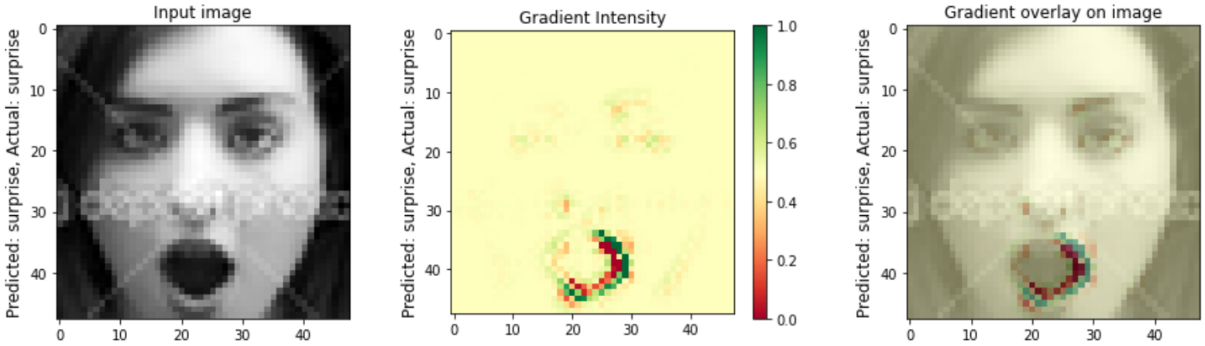
4.1 Visualizations

In this section, we attempt to interpret what our final model has learnt by presenting visualizations in the form of saliency maps, activation maps and learned filter weights.

4.1.1 Saliency Maps



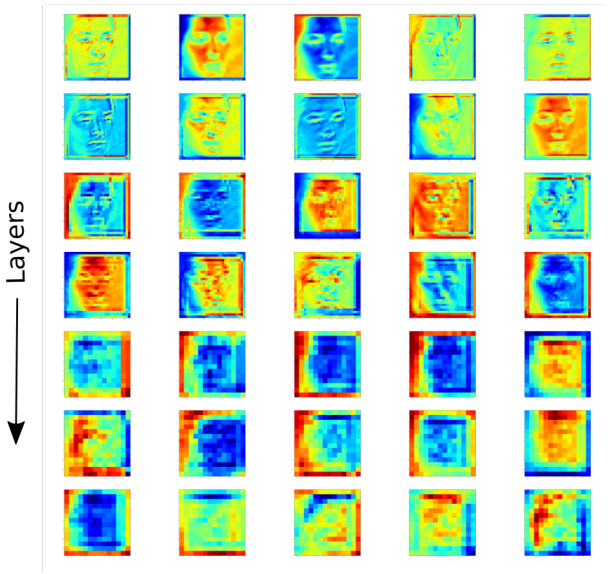
(a) The randomly initialized network incorrectly classifies a "Sad" instance as "Surprise" and cannot attribute specific regions in the face to this prediction.



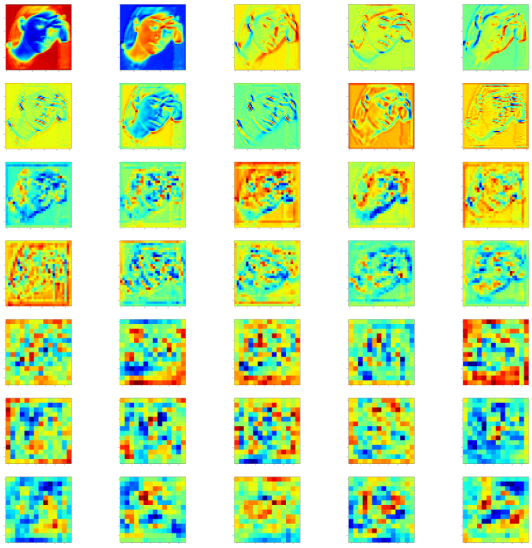
(b) The trained model correctly predicts "Surprise" and attributes the region around the mouth as the most significant pixels contributing to this prediction. Pixels in red mean that changing them will decrease the probability of predicting surprise whereas pixels in green will increase this probability.

Figure 14: Saliency Maps [2] for the model predicting "Surprise" compared between a randomly initialized network (a) and our best model (b). The maps are obtained by calculating gradient of the output class with respect to the input image which tells us how changes in the input image lead to changes in the output. This can help attribute particular pixels to predictions.

4.1.2 Activation Maps



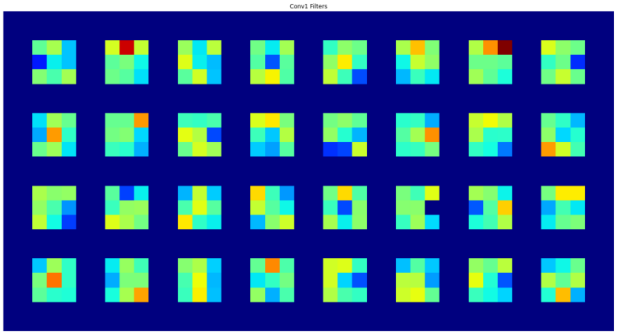
(a) Activation maps for randomly initialized CNN. As we move to deeper layers, the activations are still smooth and intensities are aggregated in regions.



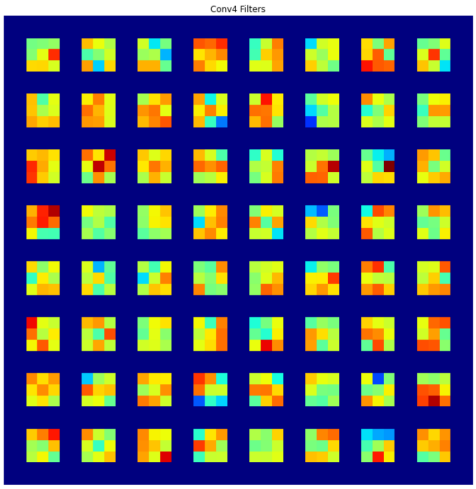
(b) Activation maps for trained CNN. In the deeper layers, specific regions are activated with a high value and there seem to be visible spatial patterns across activation values intensities.

Figure 15: Activation maps for the randomly initialized network (a) and our best model (b). The 5×7 grid shows the first 5 activation maps across 7 layers of the network. The 'jet' colorscale is used with blue and red representing lowest and highest intensities respectively.

4.1.3 Filter Visualizations



(a) 32 filters at first convolutional layer



(b) 64 filters at fourth convolutional layer

Figure 16: 3×3 Filters in the first (a) and 4th layer (b) of the best model. Values are more spread out in terms of intensity in the 4th layer while the intensity range is narrower in the first layer.

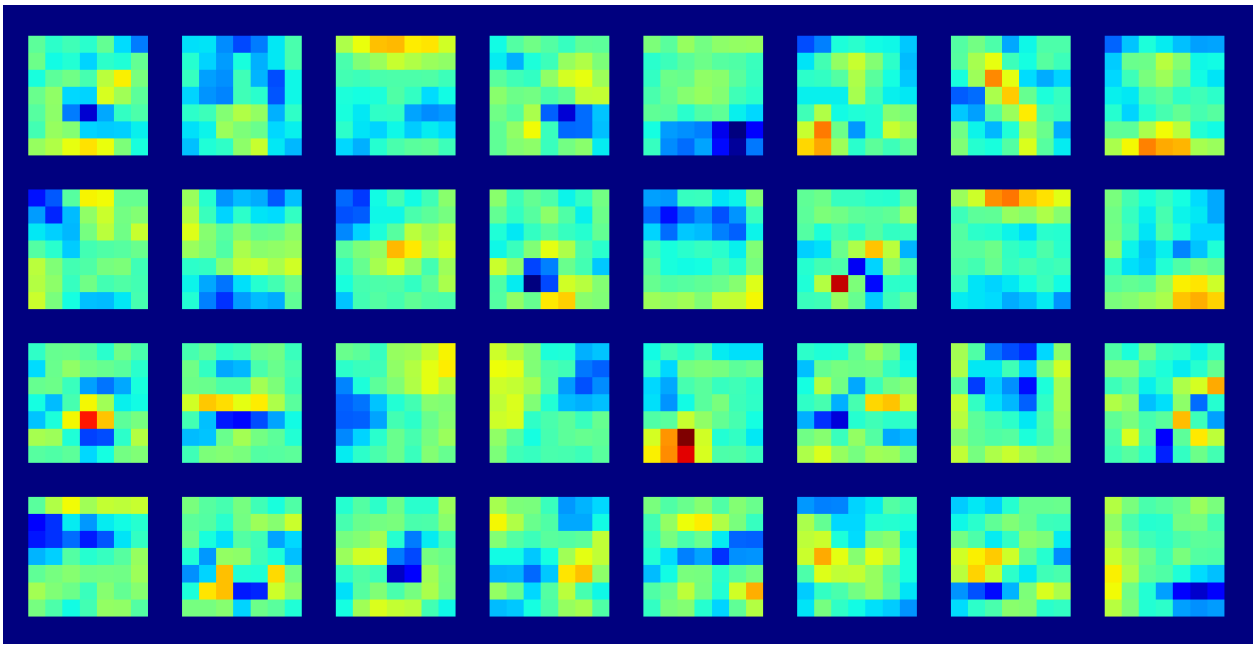


Figure 17: 32 7×7 filters at the first conv layer of the 7-block model from experiment 3.4. It can be seen that these filters are easier to interpret visually. We can observe some interesting properties such as gradients oriented in particular directions and high values for filters accumulated in specific locations.

5 Reflections

Building robust CNN models can be a challenging process since a single best approach does not exist for all tasks. Starting with a simple baseline model allows quick experimentation and an extensive configuration study for a particular problem. For instance, we started with a modified AlexNet but tapered it down to the simple 4 layer baseline considering the nature of the dataset and the prediction task. Once a sufficiently performant baseline is established, a series of ablation studies need to be conducted to determine the effect of each configuration on model performance. These studies can give us strategic directions to investigate while iteratively building our model.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.