

Introduction

This report presents the results of a series of experiments aimed at developing a facial expression recognition model. Five different experiments were conducted, each comprising of a different model architecture, dataset, preprocessing techniques, and experiment performance. The goal of these experiments was to achieve the best possible model performance for recognizing facial expressions in images.

Experiment 1:

- **Model**

Model approach for this experiment is implementation of a baseline convolutional neural network (CNN) model, The model architecture consisted of a series of convolutional and max pooling layers, followed by a fully connected layer for the final classification, The model was trained using the Adam optimizer with default parameter's and a categorical cross-entropy loss function.

- **Dataset**

original dataset.

- **Preprocessing**

Preprocessing technique used in this experiment is rescaling the data and utilization of all available images during the training phase.

- **Performance**

One of the key insights from Experiment 1 is that the model overfit the data heavily. This is indicated by the large gap between the training accuracy (95%) and the validation accuracy (60%).

Epoch 20/20

898/898 [=====] - 12s 13ms/step - loss: 0.1212 - accuracy: 0.9593 - val_loss: 2.0855 - val_accuracy: 0.6042

Experiment 2:

- **Model**

The initial experiment revealed that the model tended to overfit the data, as evidenced by the significant difference between the training accuracy and the validation accuracy. To address this issue, the model architecture was modified in the following ways:

- Dropouts were added to the model at various layers to reduce overfitting

- **Dataset**

original dataset.

- **Preprocessing**

Preprocessing technique used in this experiment is same as the previous experiment, which is rescaling the data and utilization of all available images during the training phase.

- **Performance**

Experiment 2 showed an improvement in the model's performance as compared to Experiment 1. The model was able to achieve an accuracy of 65.10% on the original dataset, in 20 epochs, whereas in experiment 1, the accuracy was 60%. This improvement in accuracy suggests that the modifications made to the model architecture by adding dropouts in Experiment 2, were effective in reducing overfitting and improving the generalization ability of the model.

Epoch 20/20

898/898 [=====] - 12s 13ms/step - loss: 0.5776 - accuracy: 0.8035 - val_loss: 1.1250 - val_accuracy: 0.6510

Experiment 3:

- **Model**

The second experiment showed that the model generalized better by adding dropouts, but still some signs of overfitting, to address this issue I will be increasing the training dataset using augmentation techniques.

- **Dataset**

original dataset + augmented data.

- **Preprocessing**

Preprocessing technique used in this experiment is various transformations to the existing images in the dataset. Some examples of data augmentation techniques include rotation, zoom, shear.

- **Performance**

Experiment 3 aimed to improve the model's performance, but the results showed that these techniques did not result in an improvement in the model's performance within 20 epochs. The model achieved an accuracy of 64% on the augmented dataset, which is not an improvement compared to the results from Experiment 2 where the model achieved more accuracy on the original dataset. However, the model hasn't overfit and looks to generalize very well with the augmented data, which could lead to better results if the model trained for more epochs.

Epoch 20/20

898/898 [=====] - 25s 28ms/step - loss: 1.0347 - accuracy: 0.6268 - val_loss: 0.9462 - val_accuracy: 0.6467

Experiment 4:

- **Model**

In the fourth experiment, I aim to improve the model performance by fine-tuning the learning rate. I used the original dataset and preprocessed the data by rescaling. By identifying the optimal learning rate for the model aiming to achieve faster and more stable convergence during training. The results showed that the model ideal value of the learning rate is **0,0014**.

- **Dataset**

original dataset.

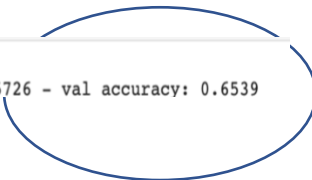
- **Preprocessing**

Preprocessing technique used in this experiment is the same as experiment 1&2, which is rescaling the data and utilization of all available images during the training phase.

- **Performance**

By increasing the learning rate from 0.001 (default value) to 0.0014, the model was able to learn faster and more efficiently, resulting in better performance with accuracy of 65.39% in the first epoch. However, the increased learning rate also led to the model becoming too specialized to the training data, resulting in overfitting.

Epoch 1/20
898/898 [=====] - 13s 13ms/step - loss: 0.2457 - accuracy: 0.9202 - val loss: 1.5726 - val accuracy: 0.6539



Experiment 5:

- **Model**

In this experiment, we aimed to improve the performance of the model by addressing the overfitting issue observed in the previous experiment. To do this, we increased the dropout ratio. Additionally, we also used a learning rate of 0.0014, which we had previously determined to be optimal for the model.

- **Dataset**

original dataset.

- **Preprocessing**

Preprocessing technique used in this experiment is the same as experiment 1&2&3, which is rescaling the data and utilization of all available images during the training phase.

- **Performance**

the results showed that our efforts to prevent overfitting by increasing the dropout ratio and using a tuned learning rate of 0.0014 were successful. The model achieved the highest accuracy so far, with a score of 65.5%.

Epoch 19/20

898/898 [=====] - 19s 21ms/step - loss: 0.4419 - accuracy: 0.8525 - val_loss: 1.2859 - val_accuracy: 0.6555

Future work

- Use selective data augmentation techniques on specific classes rather than all classes
- Implement transfer learning to leverage knowledge from pre-trained models
- Apply PCA to images to reduce dimensionality and improve computational efficiency
- Clean the data images to remove outliers or irrelevant information
- Experiment with different optimizers such as RMSprop, Adagrad to see which one performs better for the model.