Deep Learning based on CNNs: Facial Emotion Recognition

Github user: F-Riv

1 Preprocessing

The Face Emotion Recognition dataset is split in training (28709 images, 75%), validation (3589 images, 12.5%) and test (3589 images, 12.5%), each image is 48x48. The dataset is divided in seven class ('Angry', 'Disgusted', 'Fear', 'Happy', 'Sad', 'Surprised', 'Neutral'). There is a strong class imbalance (in the training set 437 images for disgusted class, 7.216. images for happy class), for this reason class weights have been calculated but they are not used as they do not increase accuracy during training.

The the dataset splits are divided in batches of size 128 to allow a faster training of the models.

2 Hyperparameters

The following hyperparameters are shared between all the models:

- Adam optimizer
- Learning rate 0.001, learning decay: 98% every 1000 steps)
- 50 epochs training (each consists in 225 steps)
- Accuracy as metric, for calculating the loss the SparseCategoricalCrossentropy() function from Keras has been used

3 Model implementation

All the models have been implemented using Keras (in order to allow GPU usage on Mac). The first model has the following structure:

```
 +-----------------------------------------+
 Input: 48x48x1
+-----------------------------------------+
 Conv2D: 32 filters, 3x3 kernel, relu
 BatchNormalization
 Conv2D: 32 filters, 3x3 kernel, relu
 MaxPooling2D: 2x2
 Dropout: 25%
+-----------------------------------------+
 Conv2D: 64 filters, 3x3 kernel, relu
 BatchNormalization
 Conv2D: 64 filters, 3x3 kernel, relu
 MaxPooling2D: 2x2
 Dropout: 25%
+-----------------------------------------+
 Conv2D: 128 filters, 3x3 kernel, relu
 BatchNormalization
 Conv2D: 128 filters, 3x3 kernel, relu
 MaxPooling2D: 2x2
 Dropout: 25%
```

```
+----------------------------------------+
 Flatten
 Dense: 256 units, relu
 Dropout: 50%
 Dense: 7 units, softmax
+----------------------------------------+
```

Techniques such as Batch Normalization and Dropout have been used in order to regularize and prevent overfitting. Every convolutional layer uses ReLU as activation function and the last dense layer uses softmax to output the class probabilities.

Several experiments has been done in order find the best structure for improving accuracy specifically modifying the the following ways the above mentioned model:

- Model 2 involves using AveragePooling2D() instead of MaxPooling2D() in the second and third blocks. The reason behind this change is that while max pooling selects the maximum value from each patch of the feature map, highlighting the most prominent features, average pooling computes the average of the elements in each patch, providing a smoother and more generalized feature map. The expected impact of using average pooling is that it might help the model to focus on more generalized patterns rather than the most dominant features. Since this experiment shows an increased validation accuracy the the AveragePooling2d layers are kept for the further models.
- Model 3 doubles the number of filters in each convolutional layer. Increasing the number of filters allows the model to learn a larger number of features at each layer. This should enable the model to capture more complex and detailed features from the images.
- Model 4 increases the kernel size in convolutional layers from (3,3) to (5,5). A larger kernel size means that each convolutional filter covers a larger area of the input image, which can help capture more spatial information. This could improve the model's ability to recognize larger patterns in the images.
- Model 5 adds one additional block of convolutional and pooling layers to the model. Adding more layers increases the depth of the network, allowing it to learn more complex features.
- Model 6 uses "valid" padding instead of "same" padding in half of the convolutional layers. "Same" padding ensures that the output feature map has the same spatial dimensions as the input by padding the input, while "valid" padding means no padding is applied, resulting in a smaller output feature map after convolution. Using "valid" padding can lead to a reduction in spatial dimensions, which might help the model focus on more localized features and potentially reduce overfitting by avoiding boundary effects. However, it could also lead to a loss of some edge information.

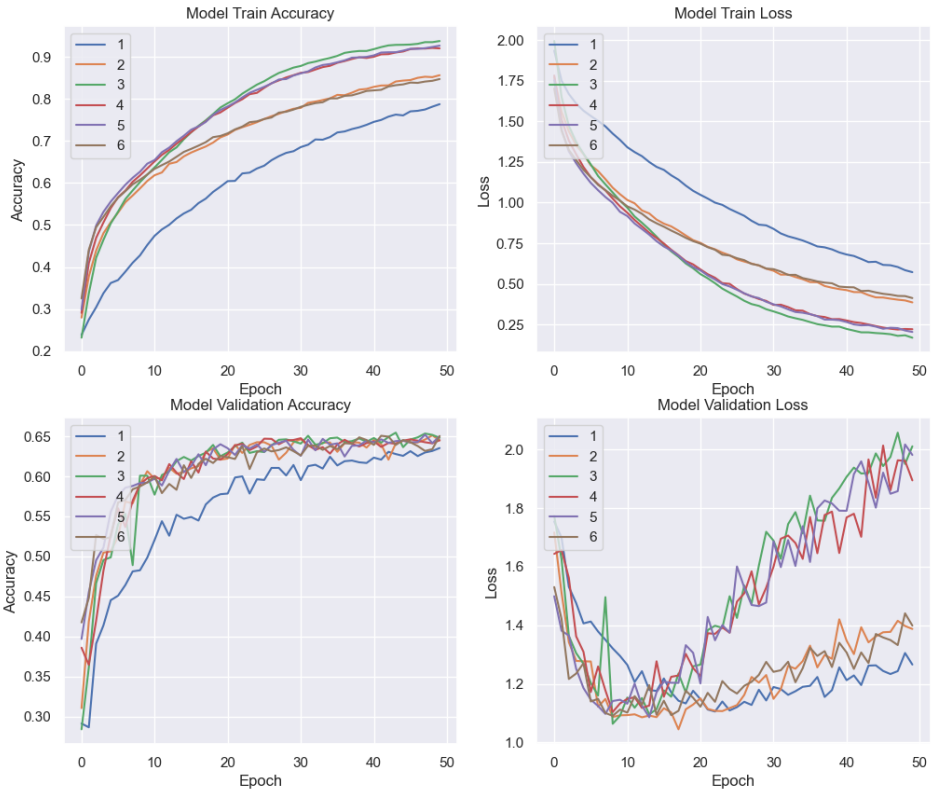
4 Training results

Table 1: Training and Validation Performance of Six Models for 50 epochs

Model	**Training Accuracy**	**Training Loss**	**Val Accuracy**	**Val Loss**
Model 1	0.7877	0.5726	0.6353	1.2657
Model 2	0.8521	0.3988	0.6512	1.3973
Model 3	0.9289	0.2025	0.6545	1.9193
Model 4	0.8623	0.3711	0.6475	1.5991
Model 5	0.9210	0.2285	0.6520	1.8573
Model 6	0.8219	0.4792	0.6506	1.2509

As it is possible to notice from Figure 1 and Table 1 all the variations from the first model shows better validation accuracy despite having an higher validation loss (apart for model 6 that shows improved results in both validation metrics). A larger kernel size (model 4) and a additional block of convolutional layers (model 5) do not improve the accuracy meaningfully.

Figure 1: Plots for training/validation accuracy and loss for each model



Model 3 appears to be the best during training but it shows the highest validation loss. Single model plots can be found in the notebook. Despite implementing techniques to prevent overfitting, such as Dropout and Batch Normalization, the significantly higher training accuracy compared to the validation accuracy at the end of the training epochs suggests that the models are still overfitting to the training set. Moreover, L2 regularization was tried but did not yield significant improvement thus it was not applied and, when set too high, stopped the learning process of the model.

5 Test results

Model 6 and 3 are tried on the test set with the following output: model 3 shows overall slightly better results on the test set.

Model 6

```
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.53      0.57       491
           1       0.70      0.64      0.67        55
           2       0.59      0.44      0.51       528
           3       0.87      0.86      0.87       879
           4       0.48      0.60      0.53       594
           5       0.78      0.77      0.78       416
           6       0.59      0.66      0.62       626

    accuracy                           0.66      3589
```

macro avg	0.66	0.64	0.65	3589
weighted avg	0.67	0.66	0.66	3589

Model 3

Classification Report:

	precision	recall	f1-score	support
0	0.55	0.62	0.58	491
1	0.88	0.65	0.75	55
2	0.54	0.49	0.51	528
3	0.88	0.85	0.86	879
4	0.52	0.55	0.53	594
5	0.82	0.80	0.81	416
6	0.64	0.64	0.64	626
accuracy			0.67	3589
macro avg	0.69	0.66	0.67	3589
weighted avg	0.68	0.67	0.67	3589

For both models the f1-score is the highest in the "Happy" (the one with more instances), the lowest scores are achieved in "Fear" and "Sad" classes. Despite being underrepresented, both models show good scores for the "Disgusted" class.

6 Comparison with a pre-trained model

"EfficientNetB0 [1], initialized with pre-trained ImageNet weights, was utilized as a comparative model for the FER dataset. The model was trained for 10 epochs. However, considering the fine-tuning for this purpose (probably requires more epochs), it did not demonstrate a performance improvement compared to the other custom models designed for this task. At the end of the 10 epochs it shows Validation Accuracy: 0.56 and Validation Loss: 1.18. Here are the results on the test set:

Classification Report:

	precision	recall	f1-score	support
0	0.48	0.28	0.35	491
1	0.33	0.02	0.03	55
2	0.36	0.35	0.36	528
3	0.68	0.75	0.71	879
4	0.38	0.46	0.41	594
5	0.59	0.69	0.64	416
6	0.47	0.45	0.46	626
accuracy			0.51	3589
macro avg	0.47	0.43	0.42	3589
weighted avg	0.50	0.51	0.50	3589

References

[1] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.