SU Group 15 Report

Aykut Tükel, Turan Akay, Tuğcan Barbin, Özgür Nuhoğlu

We used ImageDataGenerator to rescale the inputs in the [0, 255] range to be in the [0, 1] range and resize the images from 200x200 to 64x64. This massively decreased training time.

flow from directory creates batches of augmented data based on the path of a directory. The Keras generator, on the other hand, always looks for subfolders, and images within those subfolders are connected with a class. Test set 2 not having a subfolder caused a problem here but making the code cells for test set1 and test set2 separate resolved the issue.

Shuffle is now being used for training and validation. Its core goal is to add some additional stochasticity to the training process. It's off for testing.

Validation Split: During the first few days of training and testing, we used 25% and 20% of our training data as our validation data. Changing this to 15% gave us a significant improvement in our validation and test accuracy. While training, we tried a 10% split but that turned out to be a bad idea and gave us a low validation accuracy compared to 15%.

Hyperparameters:

Batch size: Both for training, validation and testing, we used a batch size of 64. The number of images in each epoch is still equal to the number of images in their respective sets. We already know that a too big batch size will lead to bad generalization. In the first few days of the project, the batch size was set at 128 and 256, changing frequently while testing. This leads to faster training times without any significant validation or test accuracy loss. Only when trying to go from .99678 to .99954 test score on kaggle, we tried a batch size of 64 and observed some better results.

Number of epochs and early stopping patience: Early stopping patience is not a hyperparameter, however it was very helpful in optimizing our number of epochs. We decided to include the number of epochs and early stopping patience together because in our project they closely affected each other's results. When starting out, we didn't use early stopping and used an epoch of 10, 15, 20. After settling on 20, because it gave the best validation accuracy, we added early stopping with patience 5. We observed that early stopping didn't stop training. We then increased our epoch to 100. At 100, the training would usually stop around 30 to 35 epochs. We then tested a patience of 10 and with 10, the training would stop between 50 to 70 epochs and this gave us the best validation score and test score.

Learning rate: We used Adam as our optimizer. For learning rate values, we tried the default 0.001 first. However it was too high. 0.0001 turned out to be a bit better but it still wasn't giving out great (99% +) validation accuracy. 0.0006 turned out to be ideal given our current model.

Activation function: In both lectures and recitation, it was emphasized that we definitely should use ReLU for everything except the output layer, for which we should use softmax. We used them and didn't try any other activation function.

Dropout Rate: We used a dropout rate of 0.5 initially. However we got marginally better results with a dropout rate of 0.4

Weight initialization: We used the default value for this.

Filters: Filters stayed at the standard 32, 64, 128 values.

Kernel Size: We used kernel sizes of 5 and 3. These were never changed.

Stride: We used the default stride value of 1.

Our final model:

Convolution2D + Convolution2D + MaxPool2D + Dropout) x 2 with relu as the activation function.

Convolution2D + Convolution2D + MaxPool2D + Flatten + Dense + Dropout with relu as the activation function.

batch size = 64, validation split = 0.15, es patience = 10, target size = (64, 64), input shape = (64, 64, 3), epochs = 100 , filters: 32, 64, 128, kernel size 1 = (5, 5), kernel size 2 = (3, 3), pool size = (2, 2) , learning rate = 0.0006, dropout probability = 0.4

Initially we had Convolution2D + Convolution2D + MaxPool2D + Dropout) x1 but after some testing, we found that x 2 gave out better results. We expected this to increase the training time significantly however it didn't increase the training time in any noticeable way.

Project flow: The project went very differently from what we expected, because from the first models we used, we already had a very high validation data accuracy. Most of the time spent on this project was optimizing our hyperparameters. After trying out many different combinations of them we ended up with the model above.

We didn't change our initial model structure after the aforementioned change. Using our validation scores, we improved the model bit by bit with each iteration.

Once we reached the test accuracy of 0.99954 at kaggle leaderboards, and saw that 3 more groups were stuck at the same accuracy, we stopped trying to improve our model. We theorized that either there were some faulty labels in the test data comparaison or the handful of predictions we made wrong were much harder than the rest.

One observation that we can't quite explain is that our training accuracy is lower than validation accuracy which is lower than test accuracy. Our training accuracy, at its highest, is 0.995, validation accuracy is at 0.9991 and test accuracy is at 0.99954. We speculate that this might be due to one of two explanations:

1) The test data set is easier than the training data set.
2) Dropout might be the cause of this. Since it disables some neurons, there is some information loss. This might lead to bigger training losses than validation because for validation all of the units are available.

Other than this, there are no abnormalities with our results that we can observe. Initially we planned to use transfer learning with ResNet50 however since our accuracy was good enough, we decided to improve on it by optimizing our hyperparameters for the last few percentages we needed for 99%.

Final Remarks: We made our daily three submissions every day on kaggle to see how much better our model was, compared to the previous day. Each day we managed to make incremental improvements until getting stuck at our current accuracy. Unfortunately we didn't keep a track of which hyperparameters resulted in which results but as written above, we kept a record of the hyperparameters we used.

Thank you for reading.