

Computer Vision Assignment-3

Image Classification challenge

Hriday Nagrani
AU1841042
hriday.n@ahduni.edu.in

Panth Patel
AU1841020
panth.p@ahduni.edu.in

Abstract—This is a report for Image Classification Performed on colorectal histology data set using CNN. First we discuss the model that we used to train and the reasoning behind the selection of hyper-parameters. We then discuss the results and comparisons with different models using different set of hyper parameters

Index Terms—CNN, Deep Learning, Hyper-Parameters, Optimiser

I. INTRODUCTION

Image classification is the task of identifying images and categorizing them in one of several predefined distinct classes. It has wide range of applications in every industry from gaming to medical.

Image classification can be done by extracting good features out of image and using them to predict the class of test image. These features can be surf, sift kind of features.

Here we have done image classification through CNNs. CNNs learn filter values automatically to learn features and follow a feature hierarchy.

A. Data set Information

Classification of textures in colorectal cancer histology. Each example is a 150 x 150 x 3 RGB image of one of 8 classes. Data was split into 80-20 ratio

Number of TRAIN examples: 3500
Number of TEST examples: 1500
Number of LABEL classes: 8

First Before using the Data, we had to pre-process. We reduced the size of each and every image to 128x128x3. Then we normalized the pixel values for each image by dividing every pixel value with 255 and bringing it between 0 and 1.

II. MODEL INFORMATION

We built a rather simple Convolutional Neural Network for this dataset. It just has 4 Convolutional Layers with Maxpooling Layers in between. The first Layer Takes input of 128x128x3 image and has 16 filters of kernel size 3x3 and hence produces an output volume of 128x128x16 which is then passed on to the Maxpooling Layer to reduce the dimension to 64x64x16.

The Second Convolutional layer has 32 Filters of kernel size 3x3 and takes an input of 64x64x16 and produces

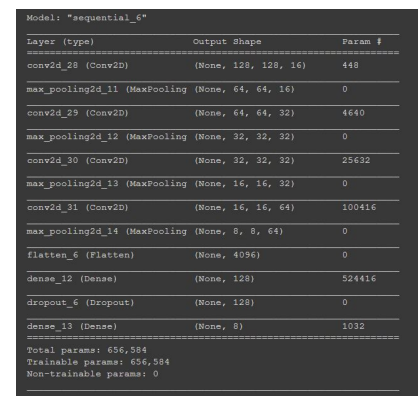
and output of 64x64x32 which is passed through a maxpooling layer and reducing the output volume to 32x32x32.

The Third layer is same as the Second one except the fact that it has a kernel size of 5x5 and the output volume after maxpooling layer obtained is 16x16x32 which is served as input to the Fourth Layer.

Fourth Layer consists of 64 filters of kernel size 7x7 and takes an input volume of 16x16x32 and produces an output volume of 8x8x64 after maxpooling which is then flattened and passed to the Fully Connected layer. All the Convolutional Layers discussed above uses the activation function "Relu".

The fully-connected Layer has 2 layers with 128 neurons and 8 neurons serving as classifiers. The Dense layer with 128 neurons use "Relu" as activation function and the second layer with 8 neurons uses "Softmax" as Activation function.

The above model was trained using "Adam" optimizer and "sparse categorical crossentropy" loss function with adaptive learning rate using callbacks on the model. The highest validation accuracy we achieved was of 86.80 % with training accuracy of 93.12 % showing that model is neither under-fitted nor over-fitted. And the graphs for the same are shown below. 17 epochs were used for calculating the same and the callback stopped the training as we the validation loss kept on increasing afterwards to stop overfitting.



Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 128, 128, 16)	448
max_pooling2d_11 (MaxPooling)	(None, 64, 64, 16)	0
conv2d_29 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d_12 (MaxPooling)	(None, 32, 32, 32)	0
conv2d_30 (Conv2D)	(None, 32, 32, 32)	25632
max_pooling2d_13 (MaxPooling)	(None, 16, 16, 32)	0
conv2d_31 (Conv2D)	(None, 16, 16, 64)	100416
max_pooling2d_14 (MaxPooling)	(None, 8, 8, 64)	0
flatten_6 (Flatten)	(None, 4096)	0
dense_12 (Dense)	(None, 128)	524416
dropout_6 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 8)	1032
Total params: 656,584		
Trainable params: 656,584		
Non-trainable params: 0		

Fig. 1. Model Summary

III. MODEL PERFORMANCE

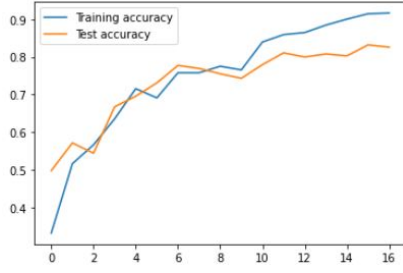


Fig. 2. Accuracy

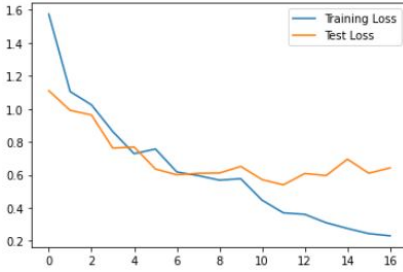


Fig. 3. Loss

IV. HYPER-PARAMETER INFORMATION AND DISCUSSIONS

A. Number of Conv layers:

Adding more conv layers resulted in increment of number of parameters, which led to more over fitting. 4 conv layers with 16,32,32,64 number of filters in each layer respectively was the sweet spot between very high over fitting and and very low accuracy. Also the number of parameters are not very high, we have less than 680000 parameters, adding more layers and going up to 20 million parameters didnt result in drastic increase of accuracy, more or less accuracy was either less or nearly equal to what we got from out model.

Here are the accuracy and loss comparisons with different number of conv layers:

1) 3 Conv Layers(16,32,64)

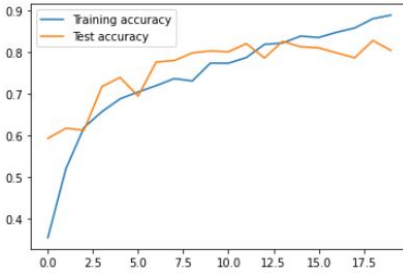


Fig. 4. Accuracy with 3 conv layers

Here we can see the decrement in validation accuracy and sudden increase of validation loss after 12th epoch,

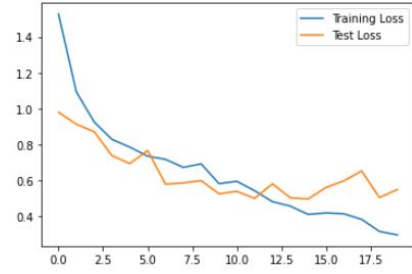


Fig. 5. Loss with 3 conv layers

this reflects the over-fitting happening in the model as clearly the difference between train and test accuracy is quite high. This happened because removing one layer resulted in increment of number of parameters, because the last layer gives an volume of 16x16x64 and hence due to large image size the number of connections with dense layer increases resulting an increase in parameters. Notice a very high validation loss after 10 epochs in the loss curve.

2) 5 Conv Layers(16,32,32,64,128)

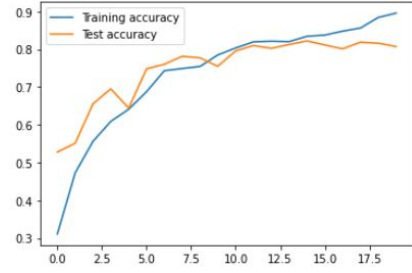


Fig. 6. Accuracy with 5 conv layers

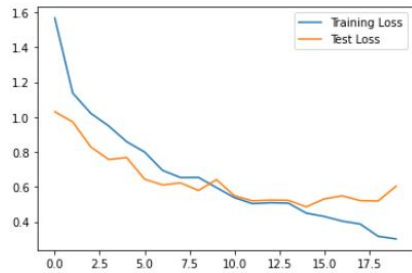


Fig. 7. Loss with 5 conv layers

Here too the number of parameters are higher than that of 4 conv layers, the performance is better than 3 conv layers case but not better than 4 conv layers, Clearly 4 conv layers are preferred over this case as it gives a little better accuracy with lesser parameters.

B. Filter Sizes

We have used kernel sizes of 3,3,5,7 respectively for each conv layer. The simple reasoning behind the selection of

these kernel sizes is that as we go deeper into the network, we want to extract more coarser features, and that is also the reason why we have increased the number of kernels after each layer.

Here are the accuracy and loss comparisons with different kernel size:

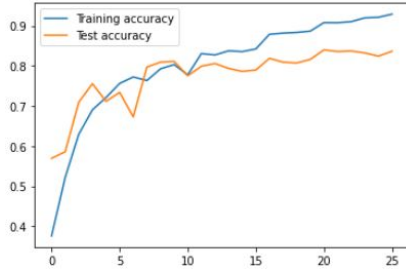


Fig. 8. Accuracy with same kernel size of 3 over all layers

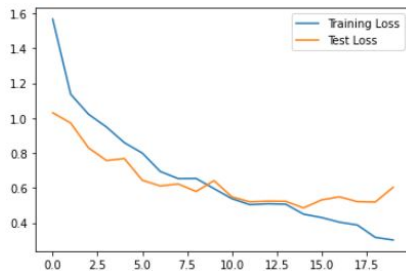


Fig. 9. Loss with same kernel size of 3 over all layers

Note: Keeping the same kernel size of 3 across all layers led to lesser validation accuracy. Validation loss is pretty high too, among the highest of all the models trained

C. Dense Layer:

We added only 2 dense layers, one with 128 neurons and other one with 8 as we have 8 categories defined in the model. Changing the 128 neurons to higher value resulted in high number of parameters which in turn led to more over-fitting. The validation loss started to increase after a fraction of epochs, suggesting the scenario of over-fitting happening.

Here are the accuracy comparisons with different Dense layer structures:

We can see that adding an extra dense layers increases the number of parameters drastically, leading to over-fit scenario.

D. Dropout:

As discussed in above points, we were constantly hindered by the over-fitting problem, so we decided to use the dropout fraction of "0.5". It seems illogical to go beyond the fraction of 0.5 of dropout as that would only lead to lower training accuracy.

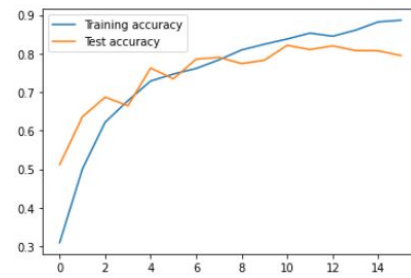


Fig. 10. Accuracy with added dense layer of 256

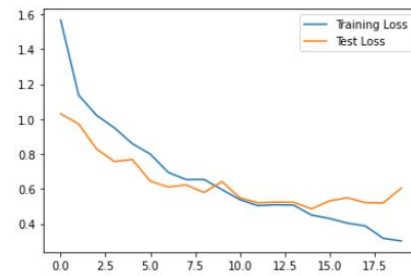


Fig. 11. Loss with added dense layer of 256

Here are the accuracy comparisons with different dropout fraction:

1) Dropout = 0.1

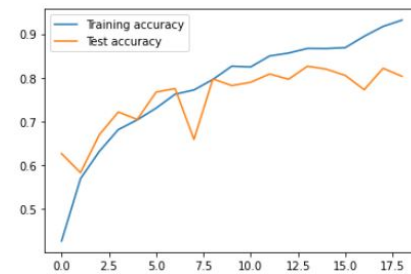


Fig. 12. Accuracy with dropout = 0.1

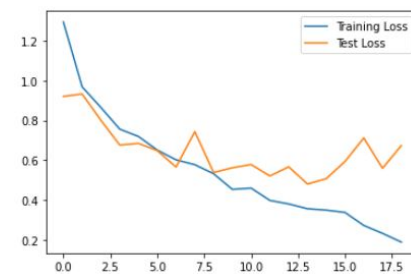


Fig. 13. Loss with dropout = 0.1

With dropout only 0.1, clear case of overfitting is observed. Regularization done is very low resulting in the worst validation loss curve where loss reached around 0.8 after last epoch. Also a very high difference between train and test accuracy is seen, almost 0.2 difference.

2) Dropout = 0.8

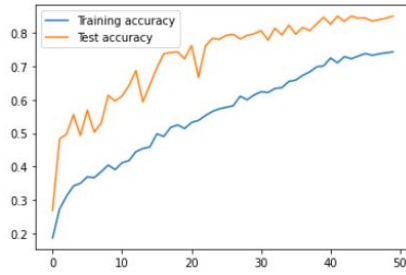


Fig. 14. Accuracy with dropout = 0.8

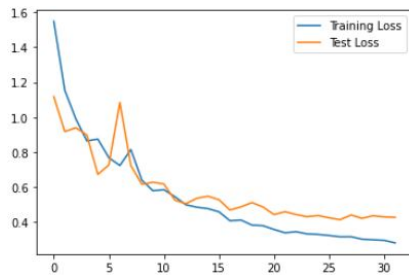


Fig. 15. Loss with dropout = 0.8

With dropout only 0.8, clear case of under-fitting is observed. Notice how the test accuracy is higher than train accuracy at each point and also the final train accuracy obtained is 0.7, which is pretty lower indicating the train set has not fit the model properly.

E. Optimizers:

We have used Adam Optimizer i.e Adaptive Moment Estimation Optimizer which computes learning rates for each parameter. It keeps both exponentially square of gradients as well as gradients, similar to momentum. Momentum can be seen as a ball running down the slope but Adam behaves as a heavy ball with friction, hence preferring flat minima in the error surface. Hence we can say that Adam works very well in practice and is favoured more over others.

REFERENCES

- [1] <https://www.tensorflow.org/datasets/catalog/colorectalhistology>, "Data-Set"
- [2] <https://www.tensorflow.org/apidocs/python/tf/keras/Model>, "Tensorflow Model Documentation"
- [3] <https://www.tensorflow.org/datasets/apidocs/python/tfds>, "Tensorflow tfds documentation"