Name: Darshan H S

Student: 20669886

Three different Architecture of Neural Networks as follows

1. The following Architecture is Built using Kera’s Sequential API:



* Here I am using two Conv2d Layers with 32 and 64 filters using relu activation function, Kernel size 3.
* Every Conv2D layer is followed immediately by a MaxPooling2D layer with a pool size= (2,2)
* Dropout has been used to overcome overfitting used with different filters in between flatter
* BatchNormalization()-Batch normalization enables the use of higher learning rates, greatly accelerating the learning process.
* Then we follow with Dense layer with 128 nodes and sigmoid activation
* Finally, output layer is a Dense layer with 10 nodes and activation function = SoftMax for multi-class classification.

Layer (type) Output Shape Param # ================================================================= conv2d\_60 (Conv2D) (None, 26, 26, 32) 320 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_55(MaxPooling)(None, 13, 13, 32) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_61 (Conv2D) (None, 11, 11, 64) 18496 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_56 (MaxPooling)(None, 5, 5, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_38 (Dropout) (None, 5, 5, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten\_27 (Flatten) (None, 1600) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_54 (Dense) (None, 128) 204928 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ batch\_normalization\_10 (Batc)(None, 128) 512 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_39 (Dropout) (None, 128) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_55 (Dense) (None, 128) 16512 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_56 (Dense) (None, 10) 1290 ================================================================

Total params: 242,058

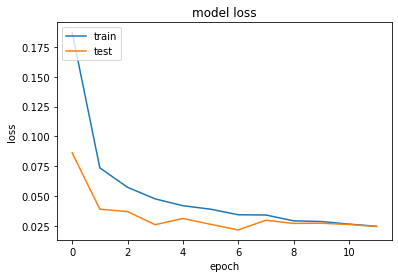
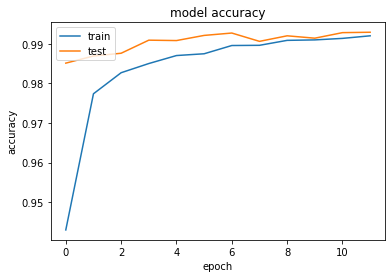
Trainable params: 241,802

Non-trainable params: 256

Test loss: 0.024450226167993243

Test accuracy: 0.992900013923645

dict\_keys(['val\_loss', 'val\_accuracy', 'loss', 'accuracy'])



For the above architecture we have 99.29% accuracy, we see that test and train accuracy above 99% near after 10 epochs’, this model for 12 epochs. Train and Test loss is reaching nearly 0.025 after 10 epochs.

Note: on the above model, for Optimization we are using “keras.optimizers.Adadelta()”

Architecture 2:

*Conv2D + Relu*

*MaxPooling2D*

*Fully Connected Layers;*

*Dropout*

*Flatter*

*Output Layer*

*INPUT*

*Conv2D + Relu*

*MaxPooling2D*

*Conv2D + Relu*

*MaxPooling2D*

1. The following Architecture is Built using Kera’s Sequential API:

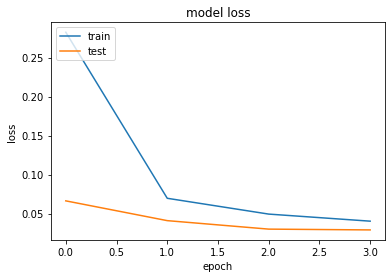
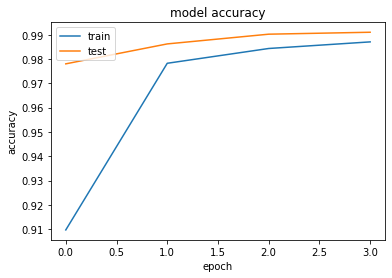
* Here I am using three Conv2d Layers with 32 and two 64 filters using relu activation function, Kernel size 3.
* Every Conv2D layer is followed immediately by a MaxPooling2D layer with a pool size= (2,2)
* Dropout has been used to overcome overfitting used with different filters in between flatter
* Then we follow with Dense layer with 128 nodes and relu activation
* Finally, output layer is a Dense layer with 10 nodes and activation function = SoftMax for multi-class classification.

Layer (type) Output Shape Param # ================================================================= conv2d\_109 (Conv2D) (None, 26, 26, 32) 320 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_104 (MaxPoolin(None, 13, 13, 32) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_110 (Conv2D) (None, 13, 13, 64) 18496 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_105 (MaxPoolin(None, 6, 6, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_111 (Conv2D) (None, 6, 6, 64) 36928 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_106 (MaxPoolin(None, 3, 3, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dropout\_70 (Dropout) (None, 3, 3, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten\_44 (Flatten) (None, 576) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_97 (Dense) (None, 128) 73856 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_98 (Dense) (None, 10) 1290

Test loss: 0.029157119183929173

Test accuracy:0.9909999966621399

dict\_keys(['val\_loss', 'val\_accuracy', 'loss', 'accuracy'])



For the above architecture we have 99.1% accuracy, we see that test and train accuracy almost to 99% accuracy after 4 epochs’, this model for 4 epochs. Train and Test loss is reaching nearly 0.005 after 4 epochs.

Note: on the above model, for Optimization we are using “keras.optimizers.Adadelta()”

Architecture 3:

*Conv2D + Relu*

*MaxPooling2D*

*Fully Connected Layers*

*Output Layer*

*INPUT*

*Conv2D + Relu*

*MaxPooling2D*

*Conv2D + tanh*

*MaxPooling2D*

The following Architecture is Built using Kera’s Sequential API:

* Here I am using three Conv2d Layers with 32 with relu activation, Conv2d Layers with 64 using tanh activation function and Conv2d Layers 64 filter with relu activation, Kernel size 3.
* Every Conv2D layer is followed immediately by a MaxPooling2D layer with a pool size= (2,2)
* The difference here is there no Dropout used and we can see in the below graph the accuracy is reduce and fluctuating.
* Then we follow with Dense layer with 128 nodes and relu activation
* Finally, output layer is a Dense layer with 10 nodes and activation function = SoftMax for multi-class classification.
* With the learning rate of 0.1

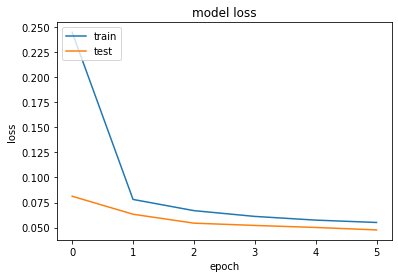
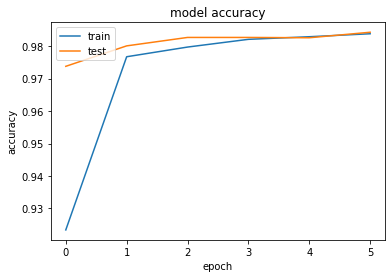
Layer (type) Output Shape Param # ================================================================= conv2d\_188 (Conv2D) (None, 26, 26, 32) 320 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_183 (MaxPoolin (None, 13, 13, 32) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_189 (Conv2D) (None, 13, 13, 64) 18496 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_184 (MaxPoolin (None, 6, 6, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ conv2d\_190 (Conv2D) (None, 6, 6, 64) 36928 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ max\_pooling2d\_185 (MaxPoolin (None, 3, 3, 64) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ flatten\_70 (Flatten) (None, 576) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_153 (Dense) (None, 128) 73856 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dense\_154 (Dense) (None, 10) 1290 ================================================================= Total params: 130,890

Trainable params: 130,890

Non-trainable params: 0

Test loss: 0.0504283158945851

Test accuracy: 0.983299970626831



For the above architecture we have 98% accuracy, we see that test and train accuracy is slightly a 98% accuracy after 6 epochs’, this model for 6 epochs. Train and Test loss is reaching nearly 0.05 after 5 epochs. In this model we are using customized learning rate i.e. 0.1

Comparing all the three Models, first model has better accuracy rate with 12 epochs, in second architecture accuracy is all most same has first one with 4 epochs. In the last model we can see the accuracy is low, because we have not used any dropout layers and its inefficient.

References: <https://keras.io/api/> , <https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/>

*“This is my own work. I have not copied any of it from anyone else.”*

*Darshan H S*