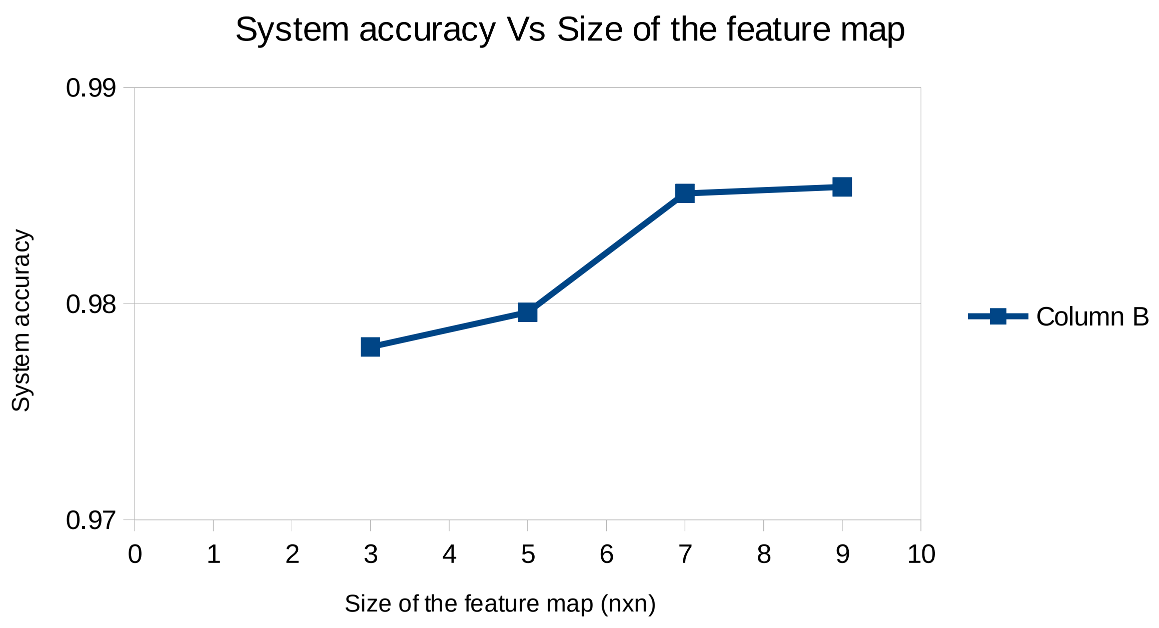
Mohamed ABDUL GAFOOR

Exercise 6: Modify the size of the feature map within the convolutional layer as specified in Table 3. How is the system accuracy influenced by this parameter?

The following table shows system accuracy that varies with the size of the feature map. This was tested for few reading and found the average.

Size of the feature map	1x1	3x3	5x5	7x7	9x9
System accuracy	0.9495	0.9780	0.9796	0.9851	0.9854
CNN Error	5.06	4.90	2.48	1.62	1.38

The graph below shows the variation of the system accuracy with feature map size. We can see it increases in this case. However, larger kernel size also degrade the performance. So usually 3x3 is a very popular choice in most application.



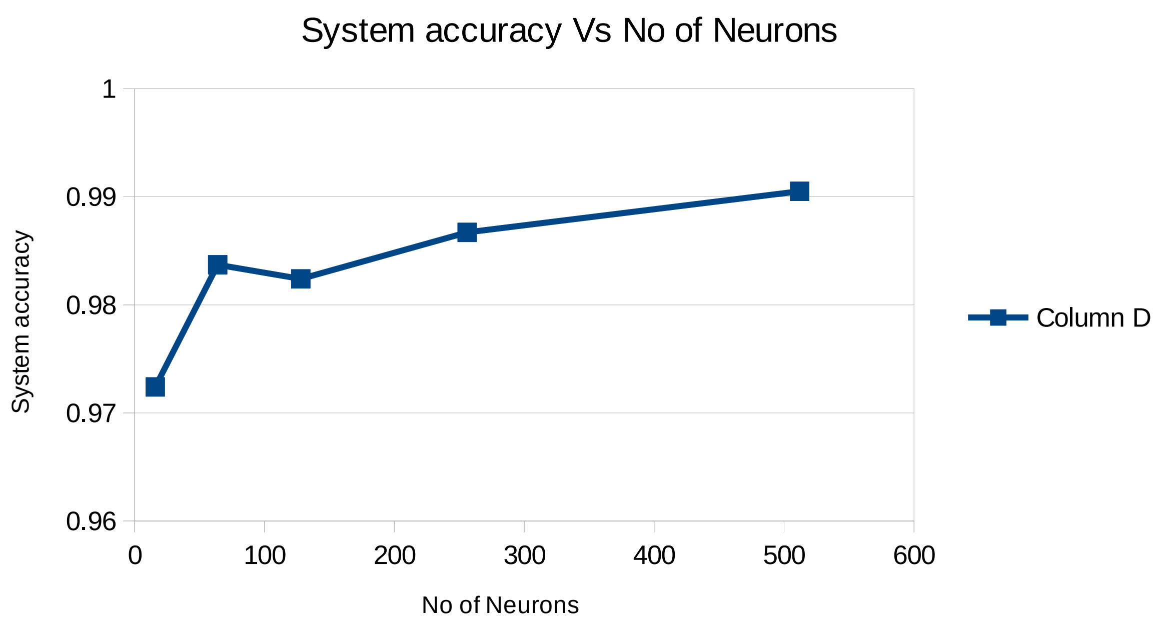
Mohamed ABDUL GAFOOR

Exercise 7: Modify the number of neurons on the dense hidden layer as specified in Table 4. Select a value of 3x3 neurons for convolutional kernel. What can be observed regarding the system accuracy? How about the convergence time necessary for the system to train a model?

The table below summarize information regarding Number of neurons, System accuracy, Convergence time, Min epoch and CNN Error.

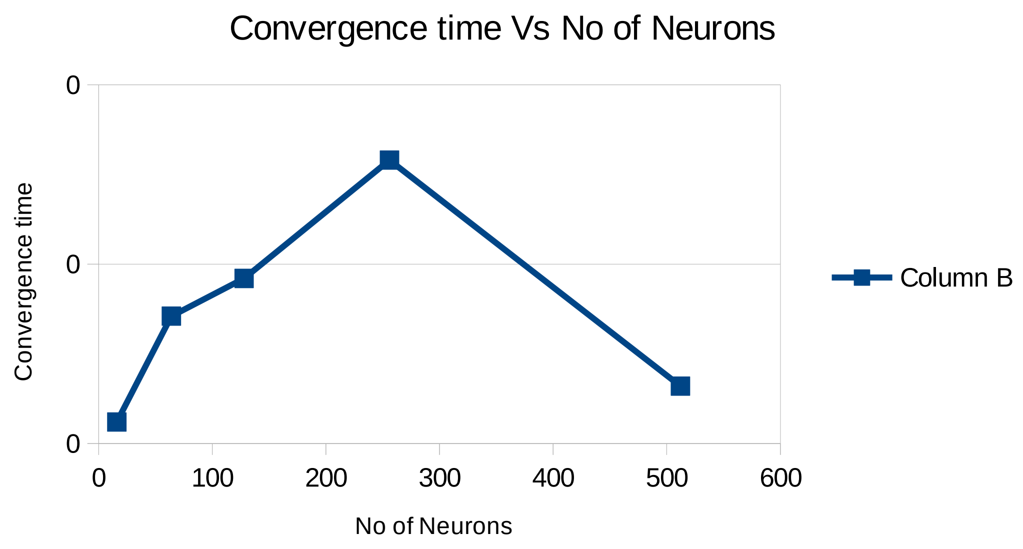
No of neurons	16	64	128	256	512
System accuracy	0.9724	0.9837	0.9824	0.9867	0.9905
Convergence time	0.00012	0.00071	0.00092	0.00158	0.00032
Min epoch	13	12	7	8	8
CNN Error	13.60	6.12	9.8	3.90	2.62

The system accuracy increases if we increase the number of neurons. Similarly, the convergence time increases.



Mohamed ABDUL GAFOOR

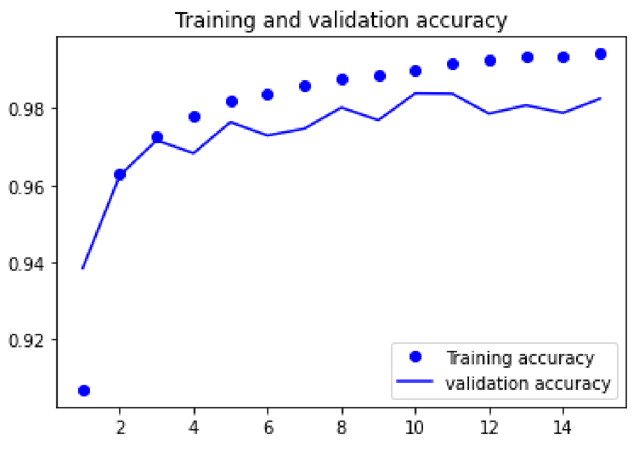
The below figure shows how the convergence time increases over the number of neurons, then it decreases afterward. So it converge faster if the number of neurons in the hidden layer increases. For example, even though I set the epoch size to 20, the model converges at 8 epoch if we set the neuron size to 512 in the hidden layer.

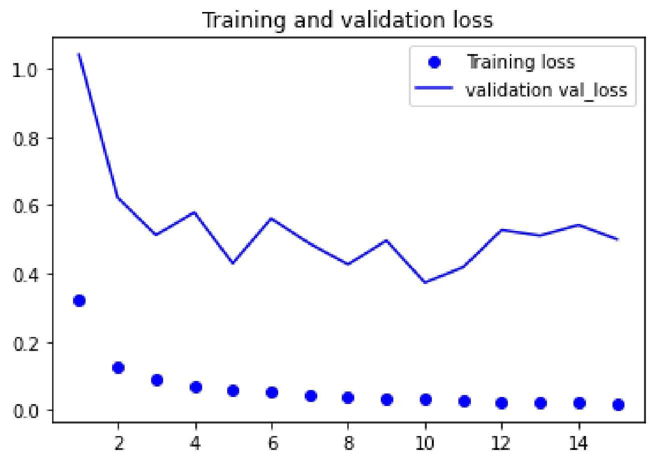


The below snippet shows how the code looks like if we create a callback functions in order to implement the early stopping;

```
# call early stopping
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
                                        mode ="min", patience = 5,
                                        restore_best_weights = True)


start = timeit.timeit()
#TODO - Application 2 - Step 8 - Train the model
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=20, batch_size=200, callbacks =[earlystopping])
end = timeit.timeit()
```

The following figures show the training loss & validation loss, training accuracy & validation accuracy if I set the hidden layer neuron to 256. This says our model is not good. It is over-fitting.



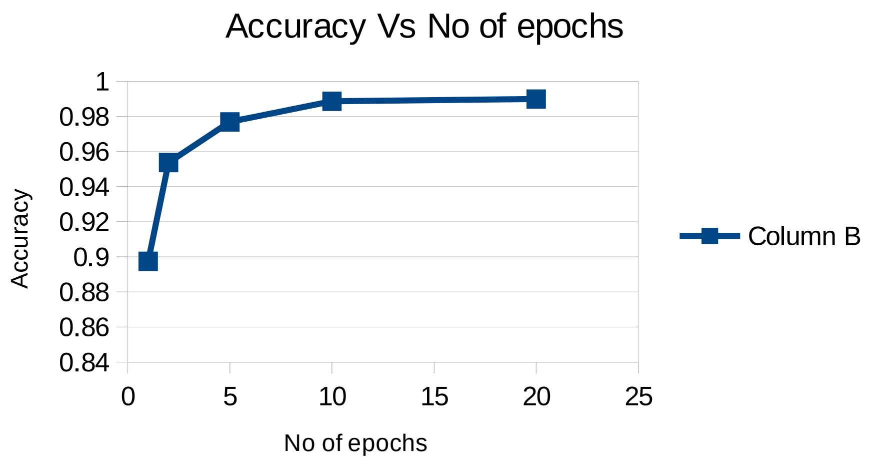


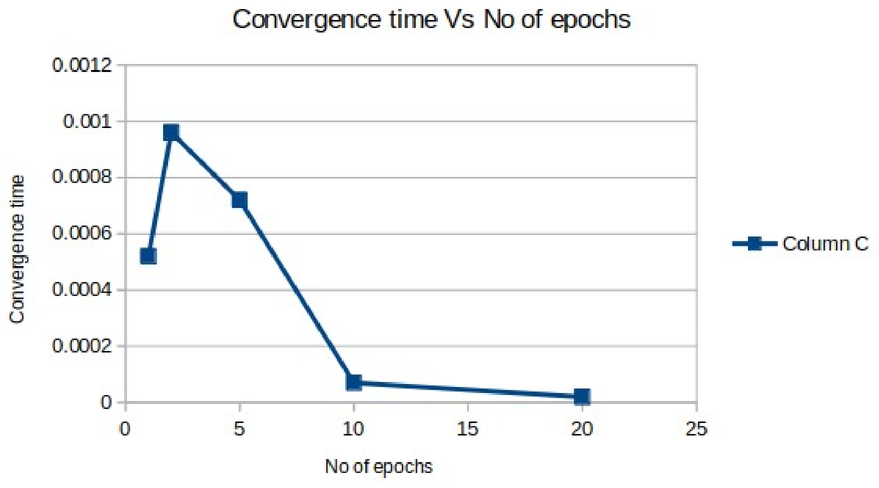
Mohamed ABDUL GAFOOR

Exercise 8: Modify the number of epochs used to train the model as specified in Table 5. Select a value of 128 neurons in the dense hidden layer. What can be observed regarding the system accuracy? How about the convergence time?

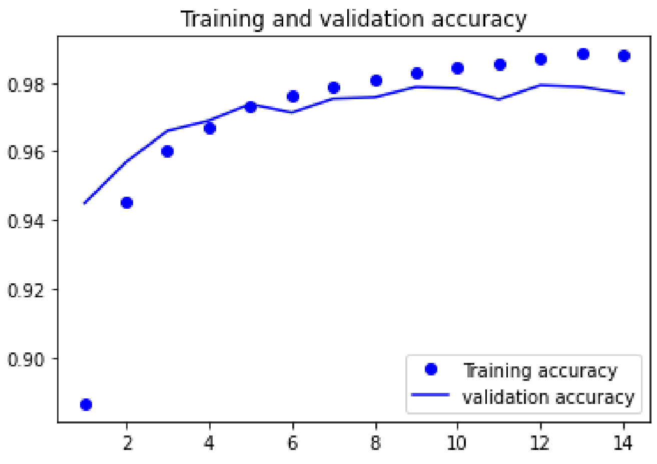
No of epochs	1	2	5	10	20
System accuracy	0.8974	0.9537	0.9769	0.9886	0.9899
Time of execuution	0.00052	0.00096	0.00072	7e-05	2e-05

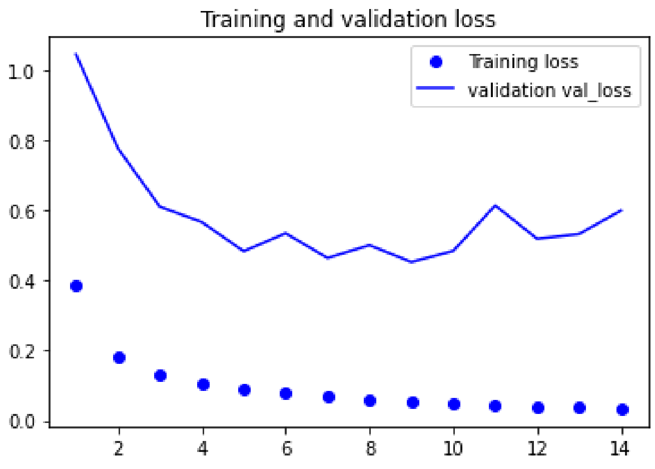
The following figure shows how the accuracy changes with the number of epochs. It is evidence that the accuracy is good if we increase the epoch size. The model is not converging at 1, 2 and 5 epochs. After 10 epochs we can observe that the performance of the model remains same more of less.



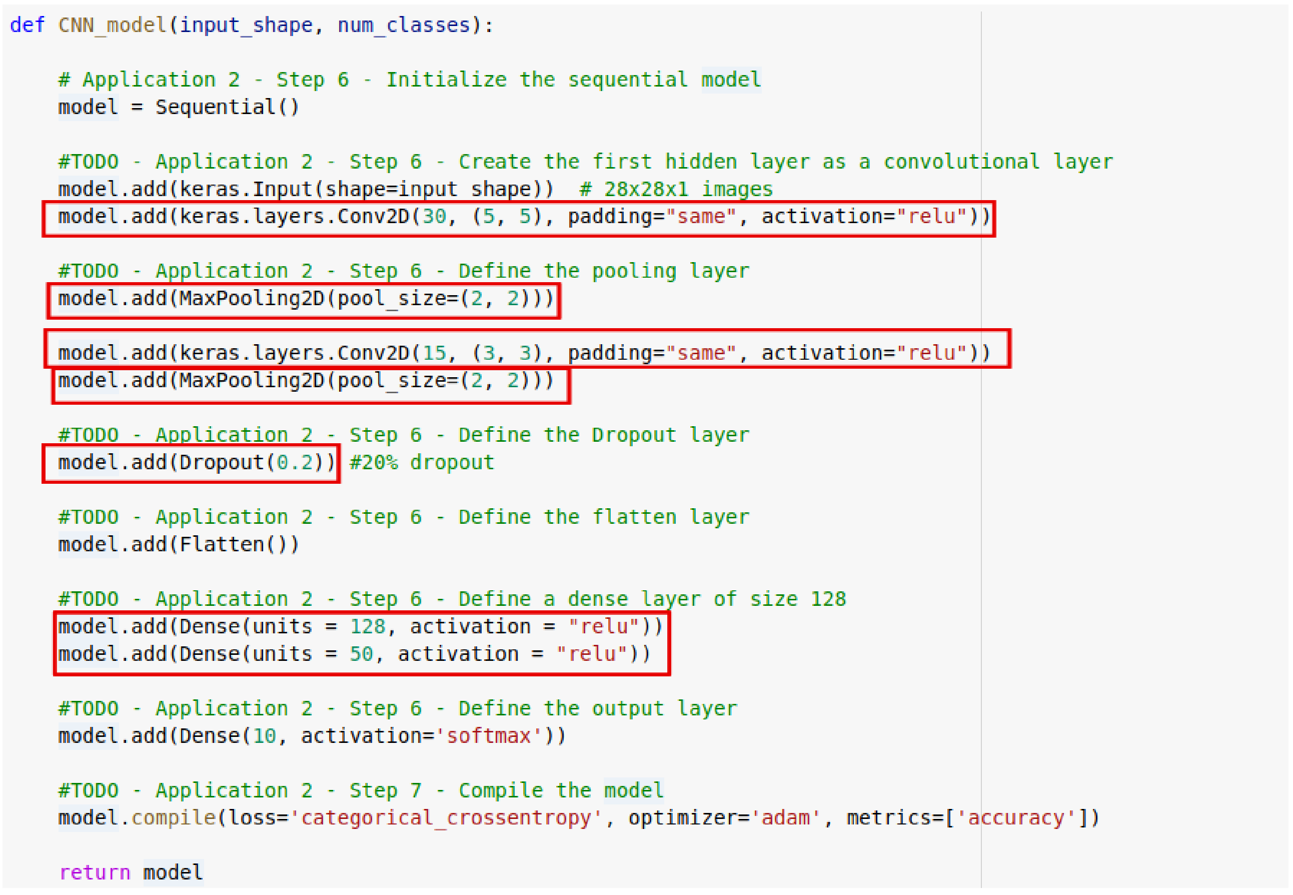


Again if we notice the training loss and the validation loss, it is very clear that our model is over-fitting.





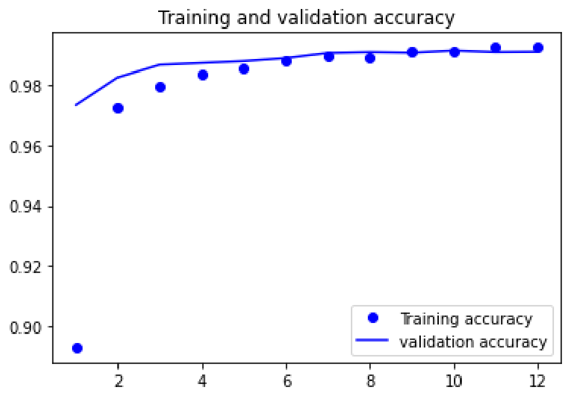
Exercise 9: Modify the CNN architecture with additional convolutional, max pooling layers and fully connected layers. The network topology can be summarized as follows:

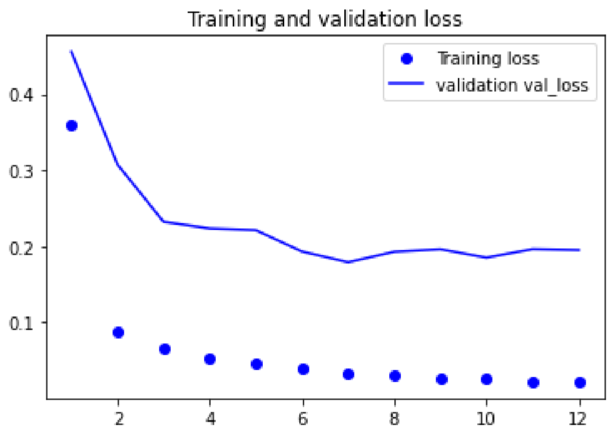


```
def CNN_model(input_shape, num_classes):

    # Application 2 - Step 6 - Initialize the sequential model
    model = Sequential()

    #TODO - Application 2 - Step 6 - Create the first hidden layer as a convolutional layer
    model.add(keras.Input(shape=input_shape))  # 28x28x1 images
    model.add(keras.layers.Conv2D(30, (5, 5), padding="same", activation="relu"))

    #TODO - Application 2 - Step 6 - Define the pooling layer
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(keras.layers.Conv2D(15, (3, 3), padding="same", activation="relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    #TODO - Application 2 - Step 6 - Define the Dropout layer
    model.add(Dropout(0.2)) #20% dropout

    #TODO - Application 2 - Step 6 - Define the flatten layer
    model.add(Flatten())

    #TODO - Application 2 - Step 6 - Define a dense layer of size 128
    model.add(Dense(units = 128, activation = "relu"))
    model.add(Dense(units = 50, activation = "relu"))

    #TODO - Application 2 - Step 6 - Define the output layer
    model.add(Dense(10, activation='softmax'))

    #TODO - Application 2 - Step 7 - Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

The above code is the novel implementation.

So for this specific configuration, the accuracy is 0.9927. The training and the validation loss/accuracy as follow. This model performance is not good as we can observe the over-fitting of the model.





We have tested with many configuration such as changing dropout values, changing number of neurons in the hidden layers etc. However, we could not find a good model.