209355P , Nuwanthi Maldini
Git Repository: https://github.com/NMaldini/MachineLearning_ImageClassification_CNN

1)

a) **Explanation to your model, design decisions, training-test dataset descriptions and what other factors were considered to improve your model.**

Data preparation

Using the 'tensorflow.keras.datasets' method, the dataset has been divided into training and test.

```
x_train : (60000, 28, 28, 1)
y_train : (60000, 10)
x_test : (10000, 28, 28, 1)
y_test : (10000, 10)
```

Dataset contains 10 classes.

Labels have been one hot encoded for both training and test.

A one hot encoding is a representation of categorical variables as binary vectors.The Keras library offers a function called **to_categorical()** that you can use to one-hot encode integer data.

Model designing

CNN layers have been used to build the model

Convolutional layer, followed by pooling layer. Then finally I used a dropout layer for regularization. Pooling layer is used with pool size 2. Pooling layer helps to reduce rows and columns.

Different Convolutional blocks with different filter sizes have been created. Then the final Convolutional block will be flatten using a flatten layer. Finally a dense layer is used with 'softmax' activation function in order to distribute probability across 10 classes in output layer.

Train Model

History object has been saved which has training history along with training loss and accuracy. X_train value has been normalized before fitting to the model. (255 is used because pixel values range from 0-255). Because it's better to normalize -1 to 1 or 0-1.Training running time also defined using epochs along with batch size.

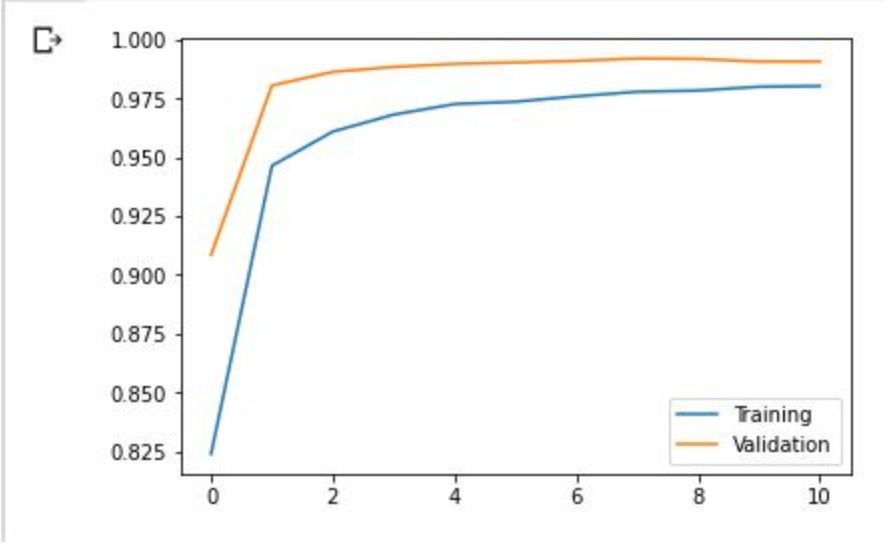
I have used EarlyStopping callback, because I wanted to monitor validation accuracy parameter. This callback will save time if the training is not going in right direction.

b) Accuracy of the modelat the end of each epoch

Max accuracy :

`validation_accuracy: 0.9920`

```
Epoch 1/20
536/536 [==============================] - 114s 212ms/step - loss: 0.6385 - accuracy: 0.8240 - val_loss: 0.2670 - val_accuracy: 0.9085
Epoch 2/20
536/536 [==============================] - 115s 214ms/step - loss: 0.1752 - accuracy: 0.9464 - val_loss: 0.0594 - val_accuracy: 0.9805
Epoch 3/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1254 - accuracy: 0.9609 - val_loss: 0.0396 - val_accuracy: 0.9863
Epoch 4/20
536/536 [==============================] - 116s 217ms/step - loss: 0.1037 - accuracy: 0.9682 - val_loss: 0.0353 - val_accuracy: 0.9884
Epoch 5/20
536/536 [==============================] - 117s 218ms/step - loss: 0.0895 - accuracy: 0.9726 - val_loss: 0.0283 - val_accuracy: 0.9897
Epoch 6/20
536/536 [==============================] - 113s 212ms/step - loss: 0.0845 - accuracy: 0.9736 - val_loss: 0.0278 - val_accuracy: 0.9903
Epoch 7/20
536/536 [==============================] - 114s 212ms/step - loss: 0.0784 - accuracy: 0.9759 - val_loss: 0.0255 - val_accuracy: 0.9910
Epoch 8/20
536/536 [==============================] - 117s 218ms/step - loss: 0.0707 - accuracy: 0.9779 - val_loss: 0.0260 - val_accuracy: 0.9920
Epoch 9/20
536/536 [==============================] - 116s 217ms/step - loss: 0.0692 - accuracy: 0.9784 - val_loss: 0.0254 - val_accuracy: 0.9918
Epoch 10/20
536/536 [==============================] - 114s 212ms/step - loss: 0.0645 - accuracy: 0.9800 - val_loss: 0.0308 - val_accuracy: 0.9907
Epoch 11/20
536/536 [==============================] - 115s 214ms/step - loss: 0.0637 - accuracy: 0.9803 - val_loss: 0.0267 - val_accuracy: 0.9907
```



2) Add random noise to the training and test datasets and report the accuracy. You may use the following sample code to add random noise. You may vary the noise_factorto control the amount of noise added to the datasets and report the results.

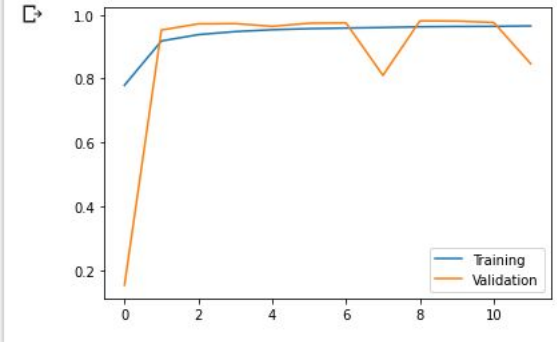
Max accuracy :

`validation_accuracy: 0.9809`

```
Epoch 1/20
536/536 [==============================] - 118s 219ms/step - loss: 0.7892 - accuracy: 0.7796 - val_loss: 2.2794 - val_accuracy: 0.1541
Epoch 2/20
536/536 [==============================] - 117s 218ms/step - loss: 0.2651 - accuracy: 0.9180 - val_loss: 0.1449 - val_accuracy: 0.9524
Epoch 3/20
536/536 [==============================] - 115s 215ms/step - loss: 0.2012 - accuracy: 0.9379 - val_loss: 0.0879 - val_accuracy: 0.9715
Epoch 4/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1685 - accuracy: 0.9474 - val_loss: 0.0787 - val_accuracy: 0.9724
Epoch 5/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1477 - accuracy: 0.9532 - val_loss: 0.1018 - val_accuracy: 0.9637
Epoch 6/20
536/536 [==============================] - 115s 214ms/step - loss: 0.1392 - accuracy: 0.9566 - val_loss: 0.0856 - val_accuracy: 0.9734
Epoch 7/20
536/536 [==============================] - 117s 218ms/step - loss: 0.1340 - accuracy: 0.9585 - val_loss: 0.0705 - val_accuracy: 0.9745
Epoch 8/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1253 - accuracy: 0.9602 - val_loss: 0.8044 - val_accuracy: 0.8103
Epoch 9/20
536/536 [==============================] - 117s 218ms/step - loss: 0.1214 - accuracy: 0.9624 - val_loss: 0.0586 - val_accuracy: 0.9809
Epoch 10/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1140 - accuracy: 0.9634 - val_loss: 0.0605 - val_accuracy: 0.9803
Epoch 11/20
536/536 [==============================] - 114s 213ms/step - loss: 0.1132 - accuracy: 0.9637 - val_loss: 0.0720 - val_accuracy: 0.9759
Epoch 12/20
536/536 [==============================] - 118s 219ms/step - loss: 0.1112 - accuracy: 0.9651 - val_loss: 0.6016 - val_accuracy: 0.8471
```

```
accs = h.history['accuracy']
val_accs = h.history['val_accuracy']

plt.plot(range(len(accs)), accs, label='Training')
plt.plot(range(len(accs)), val_accs, label='Validation')
plt.legend()
plt.show()
```



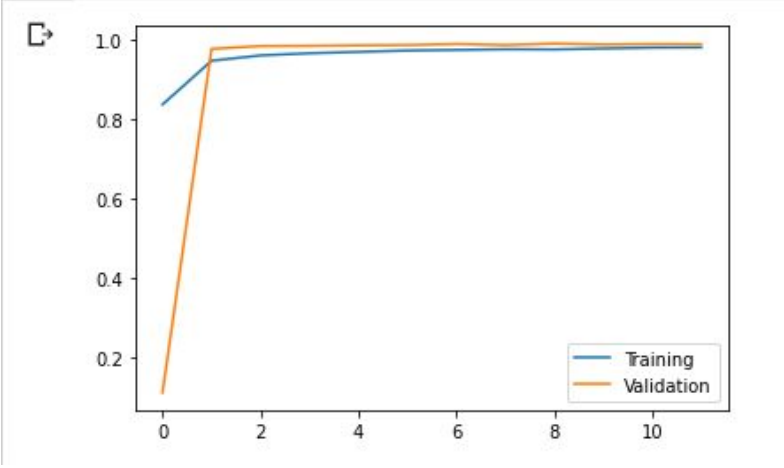
Training and validation accuracy seems similar with noise factor but there are some variations.

3) Explain how the accuracy of the image classifier can be improved for the scenario where the dataset includes noise as in part 2 above. You may implement a new model with the improvements.

I have added another Convolutional layer to the model. Altogether there are 2 Convolutional 2D layers.
After that modification results showed some improvements.

Max accuracy :
validation_accuracy: 0.9910

```
Epoch 1/20
536/536 [==============================] - 222s 415ms/step - loss: 0.5239 - accuracy: 0.8378 - val_loss: 10.1290 - val_accuracy: 0.1135
Epoch 2/20
536/536 [==============================] - 217s 406ms/step - loss: 0.1734 - accuracy: 0.9477 - val_loss: 0.0721 - val_accuracy: 0.9776
Epoch 3/20
536/536 [==============================] - 219s 408ms/step - loss: 0.1319 - accuracy: 0.9612 - val_loss: 0.0515 - val_accuracy: 0.9847
Epoch 4/20
536/536 [==============================] - 221s 411ms/step - loss: 0.1111 - accuracy: 0.9666 - val_loss: 0.0436 - val_accuracy: 0.9853
Epoch 5/20
536/536 [==============================] - 216s 404ms/step - loss: 0.1004 - accuracy: 0.9700 - val_loss: 0.0458 - val_accuracy: 0.9865
Epoch 6/20
536/536 [==============================] - 218s 407ms/step - loss: 0.0910 - accuracy: 0.9734 - val_loss: 0.0416 - val_accuracy: 0.9873
Epoch 7/20
536/536 [==============================] - 220s 411ms/step - loss: 0.0840 - accuracy: 0.9749 - val_loss: 0.0339 - val_accuracy: 0.9898
Epoch 8/20
536/536 [==============================] - 217s 406ms/step - loss: 0.0787 - accuracy: 0.9767 - val_loss: 0.0430 - val_accuracy: 0.9873
Epoch 9/20
536/536 [==============================] - 218s 408ms/step - loss: 0.0764 - accuracy: 0.9762 - val_loss: 0.0316 - val_accuracy: 0.9910
Epoch 10/20
536/536 [==============================] - 221s 413ms/step - loss: 0.0726 - accuracy: 0.9787 - val_loss: 0.0371 - val_accuracy: 0.9886
Epoch 11/20
536/536 [==============================] - 219s 408ms/step - loss: 0.0650 - accuracy: 0.9807 - val_loss: 0.0347 - val_accuracy: 0.9895
Epoch 12/20
536/536 [==============================] - 219s 408ms/step - loss: 0.0633 - accuracy: 0.9813 - val_loss: 0.0356 - val_accuracy: 0.9893
```



Training and validation accuracy lines show better alignment which means the model has been improved successfully.