1. Run https://www.tensorflow.org/tutorials/keras/classification . This classifier works on fashion items.

1.1. Result

I compiled the code and made it run.
The model was trained using 10 epochs and reached an accuracy of 91.08% on the training data.
The accuracy of the test dataset reached 88%.

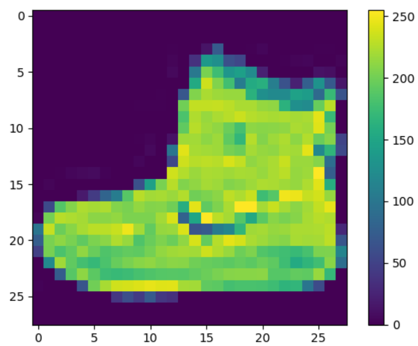
1.2. Output (The full output is available in "Output.zip"):

```
Epoch 1/10
   32/60000 [..............................] - ETA: 3:29 - loss: 2.3689 - acc: 0.1562
  832/60000 [..............................] - ETA: 11s - loss: 1.3873 - acc: 0.5373
…
58528/60000 [============================>.] - ETA: 0s - loss: 0.4948 - acc: 0.8251
59392/60000 [============================>.] - ETA: 0s - loss: 0.4940 - acc: 0.8255
60000/60000 [==============================] - 4s 58us/sample - loss: 0.4938 - acc: 0.8256

Epoch 2/10
   32/60000 [..............................] - ETA: 9s - loss: 0.1532 - acc: 0.9375
  960/60000 [..............................] - ETA: 3s - loss: 0.4037 - acc: 0.8635
…
59552/60000 [============================>.] - ETA: 0s - loss: 0.3697 - acc: 0.8673
60000/60000 [==============================] - 3s 57us/sample - loss: 0.3697 - acc: 0.8671

Epoch 3/10
   32/60000 [..............................] - ETA: 9s - loss: 0.3800 - acc: 0.8750
  960/60000 [..............................] - ETA: 3s - loss: 0.3207 - acc: 0.8917
…
59936/60000 [============================>.] - ETA: 0s - loss: 0.3338 - acc: 0.8787
60000/60000 [==============================] - 4s 59us/sample - loss: 0.3338 - acc: 0.8788

Epoch 4/10
   32/60000 [..............................] - ETA: 9s - loss: 0.2278 - acc: 0.9375
  960/60000 [..............................] - ETA: 3s - loss: 0.2901 - acc: 0.8979
…
59488/60000 [============================>.] - ETA: 0s - loss: 0.3105 - acc: 0.8873
60000/60000 [==============================] - 3s 57us/sample - loss: 0.3102 - acc: 0.8873

Epoch 5/10
   32/60000 [..............................] - ETA: 9s - loss: 0.1193 - acc: 0.9375
  992/60000 [..............................] - ETA: 3s - loss: 0.3120 - acc: 0.8831
…
59264/60000 [============================>.] - ETA: 0s - loss: 0.2949 - acc: 0.8916
60000/60000 [==============================] - 3s 57us/sample - loss: 0.2947 - acc: 0.8918

Epoch 6/10
   32/60000 [..............................] - ETA: 9s - loss: 0.2225 - acc: 0.9375
  960/60000 [..............................] - ETA: 3s - loss: 0.3150 - acc: 0.8823
…
59776/60000 [============================>.] - ETA: 0s - loss: 0.2814 - acc: 0.8949
60000/60000 [==============================] - 3s 56us/sample - loss: 0.2812 - acc: 0.8950

Epoch 7/10
   32/60000 [..............................] - ETA: 29s - loss: 0.1169 - acc: 0.9688
 1152/60000 [..............................] - ETA: 3s - loss: 0.2344 - acc: 0.9115
…
59200/60000 [============================>.] - ETA: 0s - loss: 0.2678 - acc: 0.9009
60000/60000 [==============================] - 3s 57us/sample - loss: 0.2681 - acc: 0.9009

Epoch 8/10
   32/60000 [..............................] - ETA: 9s - loss: 0.1449 - acc: 0.9375
  960/60000 [..............................] - ETA: 3s - loss: 0.2596 - acc: 0.9042
```

```
…
59456/60000 [============================>.] - ETA: 0s - loss: 0.2575 - acc: 0.9035
60000/60000 [==============================] - 3s 58us/sample - loss: 0.2579 - acc: 0.9032

Epoch 9/10
   32/60000 [..............................] - ETA: 7s - loss: 0.1876 - acc: 0.9688
  928/60000 [..............................] - ETA: 3s - loss: 0.2276 - acc: 0.9159
…
58688/60000 [============================>.] - ETA: 0s - loss: 0.2477 - acc: 0.9079
59648/60000 [============================>.] - ETA: 0s - loss: 0.2473 - acc: 0.9081
60000/60000 [==============================] - 3s 57us/sample - loss: 0.2472 - acc: 0.9082

Epoch 10/10

   32/60000 [..............................] - ETA: 9s - loss: 0.3489 - acc: 0.8438
  928/60000 [..............................] - ETA: 3s - loss: 0.2369 - acc: 0.9084
…
59904/60000 [============================>.] - ETA: 0s - loss: 0.2411 - acc: 0.9108
60000/60000 [==============================] - 3s 58us/sample - loss: 0.2410 - acc: 0.9108

10000/10000 - 0s - loss: 0.3494 - acc: 0.8800
Test accuracy: 0.88

(28, 28)
(1, 28, 28)
[[5.5457942e-05 1.6383477e-14 9.9598467e-01 3.2473368e-10 1.3082309e-03
  2.7307448e-16 2.6517855e-03 2.6463840e-17 1.4227564e-10 4.2015943e-15]]
```

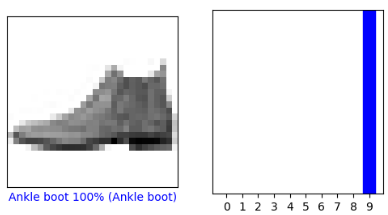
1.3. Graphs (The full output is available in "Output.zip"):

Training Image example

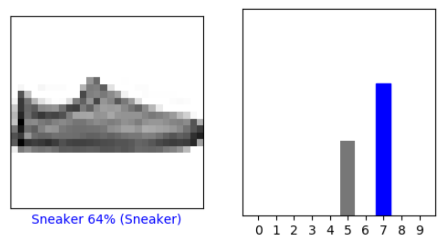


First 25 images from training set and class names

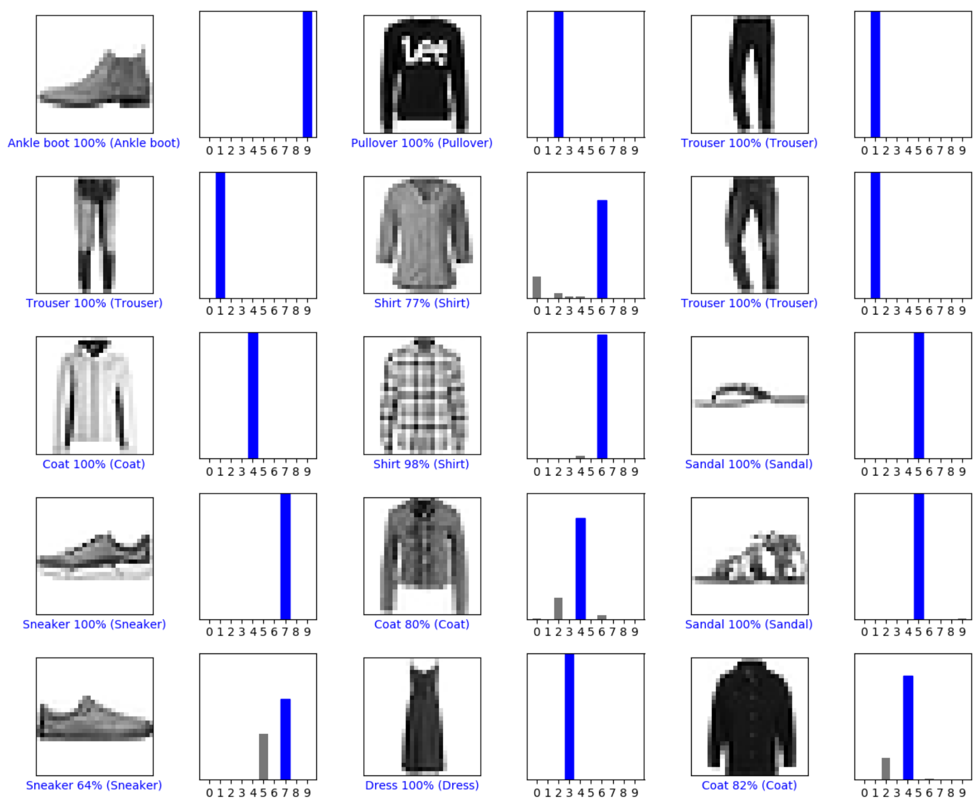
Prediction label of image [i=0]



Prediction label of image [i=12]



Prediction label of 15 test images [0<i<15]



2. Convert it to recognize digits. There is also an minst dataset for digits.

2.1. Result

I changed the training dataset into dataset of digit images, and the class names accordingly.
The model was trained using 10 epochs and reached an accuracy of 99.56% on the training data.
The accuracy of the test dataset reached 97.87%.

2.2. Code (The full code is available in "Python.zip"):

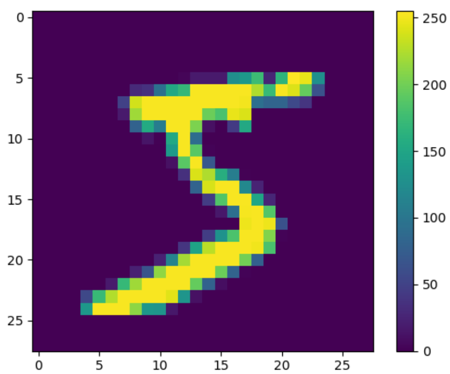
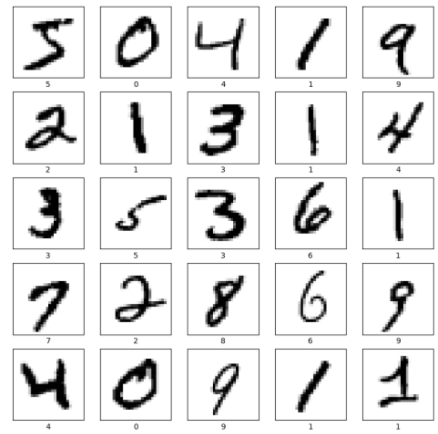
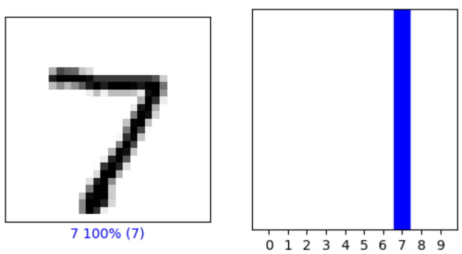
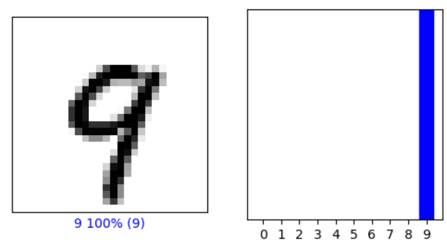
```
digit_minst = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = digit_minst.load_data()
class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

2.3. Output (The full output is available in "Output.zip"):

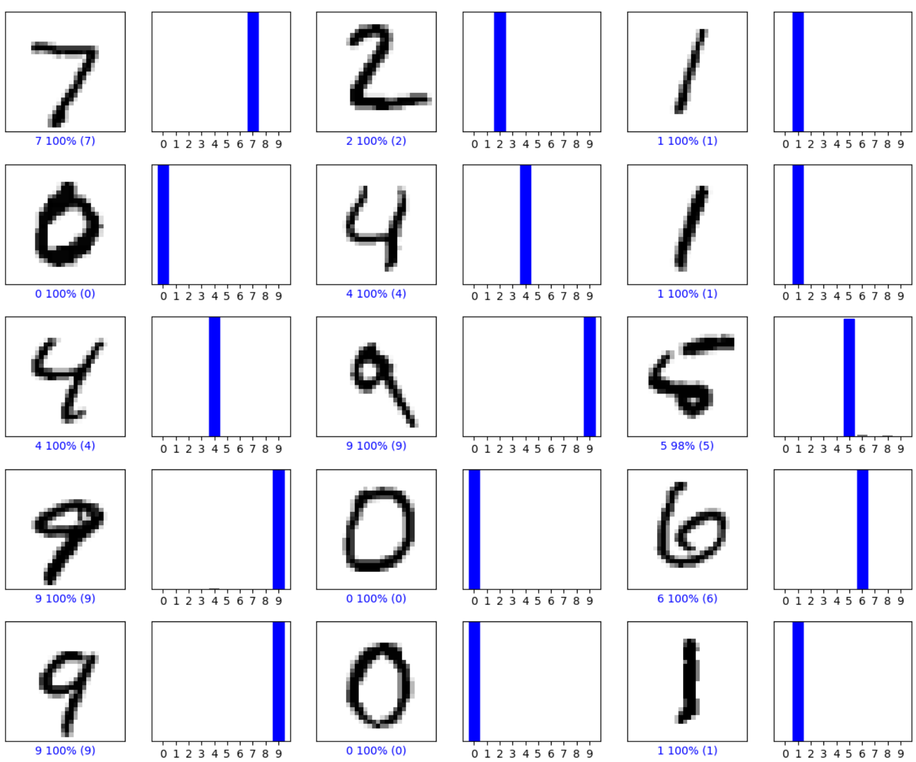
```
Epoch 1/10
   32/60000 [..............................] - ETA: 3:12 - loss: 2.5639 - acc: 0.0625
  672/60000 [..............................] - ETA: 13s - loss: 1.7868 - acc: 0.4658
59808/60000 [============================>.] - ETA: 0s - loss: 0.2558 - acc: 0.9255
60000/60000 [==============================] - 4s 64us/sample - loss: 0.2554 - acc: 0.9256

Epoch 2/10
   32/60000 [..............................] - ETA: 11s - loss: 0.0126 - acc: 1.0000
  768/60000 [..............................] - ETA: 4s - loss: 0.1200 - acc: 0.9714
59680/60000 [============================>.] - ETA: 0s - loss: 0.1144 - acc: 0.9668
60000/60000 [==============================] - 5s 79us/sample - loss: 0.1143 - acc: 0.9668

Epoch 3/10
   32/60000 [..............................] - ETA: 11s - loss: 0.0470 - acc: 0.9688
  896/60000 [..............................] - ETA: 3s - loss: 0.0740 - acc: 0.9821
59840/60000 [============================>.] - ETA: 0s - loss: 0.0785 - acc: 0.9763
60000/60000 [==============================] - 4s 61us/sample - loss: 0.0786 - acc: 0.9763

Epoch 4/10
   32/60000 [..............................] - ETA: 7s - loss: 0.0256 - acc: 1.0000
  832/60000 [..............................] - ETA: 3s - loss: 0.0508 - acc: 0.9844
59712/60000 [============================>.] - ETA: 0s - loss: 0.0588 - acc: 0.9818
60000/60000 [==============================] - 4s 59us/sample - loss: 0.0590 - acc: 0.9817

Epoch 5/10
   32/60000 [..............................] - ETA: 9s - loss: 0.0391 - acc: 1.0000
  928/60000 [..............................] - ETA: 3s - loss: 0.0423 - acc: 0.9860
59424/60000 [============================>.] - ETA: 0s - loss: 0.0462 - acc: 0.9855
60000/60000 [==============================] - 4s 59us/sample - loss: 0.0462 - acc: 0.9855

Epoch 6/10
   32/60000 [..............................] - ETA: 9s - loss: 0.0202 - acc: 1.0000
  960/60000 [..............................] - ETA: 3s - loss: 0.0510 - acc: 0.9854
59840/60000 [============================>.] - ETA: 0s - loss: 0.0355 - acc: 0.9891
60000/60000 [==============================] - 4s 63us/sample - loss: 0.0355 - acc: 0.9890

Epoch 7/10
   32/60000 [..............................] - ETA: 9s - loss: 0.0013 - acc: 1.0000
  800/60000 [..............................] - ETA: 4s - loss: 0.0199 - acc: 0.9950
 1504/60000 [..............................] - ETA: 4s - loss: 0.0249 - acc: 0.9920
59840/60000 [============================>.] - ETA: 0s - loss: 0.0301 - acc: 0.9903
60000/60000 [==============================] - 4s 66us/sample - loss: 0.0301 - acc: 0.9903

Epoch 8/10
   32/60000 [..............................] - ETA: 18s - loss: 0.0074 - acc: 1.0000
  576/60000 [..............................] - ETA: 6s - loss: 0.0193 - acc: 0.9965
59840/60000 [============================>.] - ETA: 0s - loss: 0.0229 - acc: 0.9932
```

```
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0229 - acc: 0.9932

Epoch 9/10
   32/60000 [..............................] - ETA: 14s - loss: 0.0062 - acc: 1.0000
  608/60000 [..............................] - ETA: 5s - loss: 0.0136 - acc: 0.9934
59872/60000 [============================>.] - ETA: 0s - loss: 0.0198 - acc: 0.9942
60000/60000 [==============================] - 4s 65us/sample - loss: 0.0197 - acc: 0.9942

Epoch 10/10
   32/60000 [..............................] - ETA: 9s - loss: 0.0058 - acc: 1.0000
  896/60000 [..............................] - ETA: 3s - loss: 0.0167 - acc: 0.9978
59680/60000 [============================>.] - ETA: 0s - loss: 0.0152 - acc: 0.9956
60000/60000 [==============================] - 4s 62us/sample - loss: 0.0152 - acc: 0.9956

10000/10000 - 0s - loss: 0.0812 - acc: 0.9787

Test accuracy: 0.9787
(28, 28)
(1, 28, 28)
[[9.7758135e-11 1.4205011e-07 9.9999821e-01 4.6705611e-07 4.9148202e-18
  2.0141092e-10 4.5865227e-09 1.7331948e-16 1.1550956e-06 4.3926822e-16]]
```

2.4. Graphs (The full output is available in “Output.zip”):

Training Image example	First 25 images from training set and class names
	
Prediction label of image [i=0]	**Prediction label of image [i=12]**
	

Prediction label of 15 test images [0<i<15]



3. **Make modifications to the networks and report the train and test accuracies as you modify the networks. You can also change the activation function.**
4. **Plot a histogram of the accuracies of the test examples for correctly and incorrectly classified objects.**

4.1. Result

I changed the network by enlarging the training: instead of 10 epochs - I used 200 epochs.
In addition, instead of 128 layers, I used 300 layers.
The model was reached an accuracy of 100% on the training data (In comparison to the accuracy of 99.56% reached in previous model).
The accuracy of the test dataset reached 98.41% (In comparison to the accuracy of 97.87% reached in previous model).
In addition, I plotted an histogram that represents the model accuracy. According to the histogram of the accuracies, 98.41% of the test images were correct and 1.59% were incorrect.

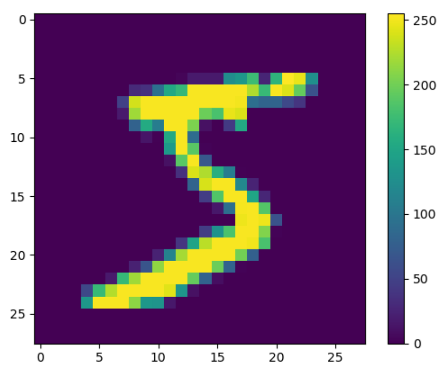
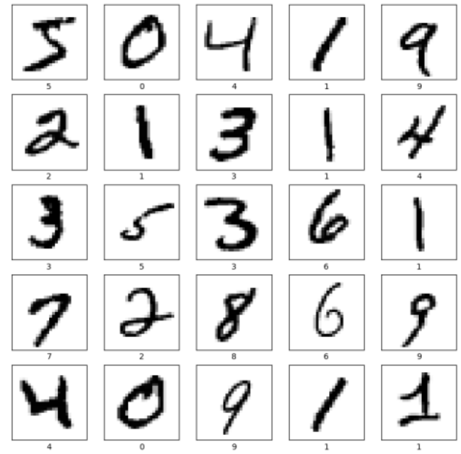
4.2. Code (The full code is available in "Python.zip"):

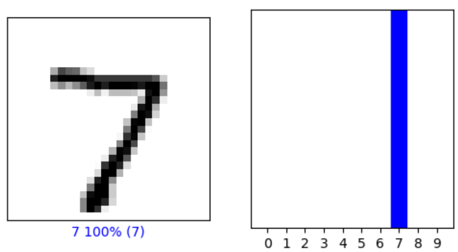
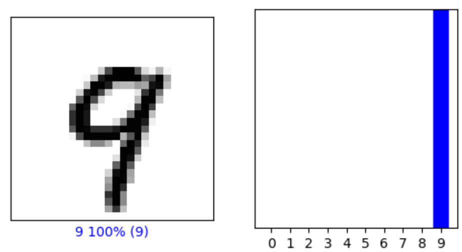
```
# Build the model
 model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(300, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.fit(train_images, train_labels, epochs=200)

# Plot accuracy
x = np.arange(2)
plt.bar(x, height= [test_acc , 1-test_acc])
plt.xticks(x, ['Correct', 'Incorrect'])
```

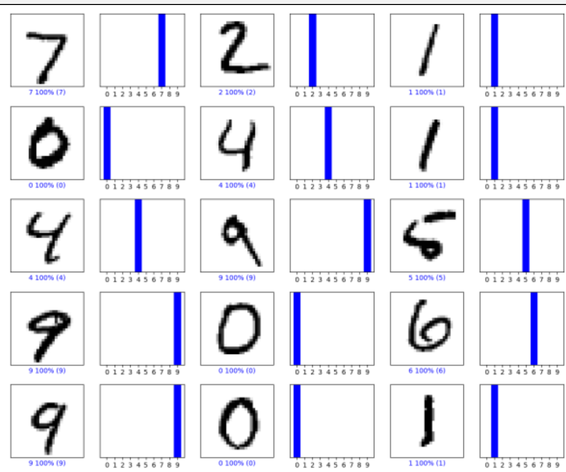
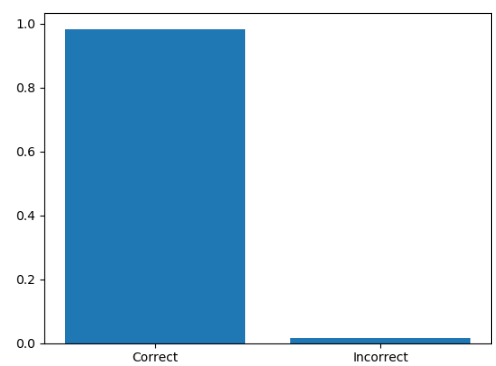
4.3. Output (The full output is available in "Output.zip"):

```
Epoch 1/200
   32/60000 [..............................] - ETA: 2:18 - loss: 2.4784 - acc: 0.0625
  832/60000 [..............................] - ETA: 8s - loss: 1.5719 - acc: 0.5541
 1600/60000 [..............................] - ETA: 6s - loss: 1.1595 - acc: 0.6888
 2464/60000 [>.............................] - ETA: 5s - loss: 0.9609 - acc: 0.7317
…
Epoch 200/200

   32/60000 [..............................] - ETA: 9s - loss: 0.0000e+00 - acc: 1.0000
  640/60000 [..............................] - ETA: 5s - loss: 0.0000e+00 - acc: 1.0000
 1216/60000 [..............................] - ETA: 5s - loss: 0.0000e+00 - acc: 1.0000
…
59072/60000 [============================>.] - ETA: 0s - loss: 1.2108e-11 - acc: 1.0000
59584/60000 [============================>.] - ETA: 0s - loss: 1.2004e-11 - acc: 1.0000
60000/60000 [==============================] - 7s 119us/sample - loss: 1.1921e-11 - acc: 1.0000
10000/10000 - 1s - loss: 0.2675 - acc: 0.9841

Test accuracy: 0.9841
(28, 28)
(1, 28, 28)
[[0.0000000e+00 1.9053744e-33 1.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]]
```

4.4. Graphs (The full output is available in "Output.zip"):

Training Image example	First 25 images from training set and class names
	

Prediction label of image [i=0]	Prediction label of image [i=12]
	

Prediction label of 15 test images [0<i<15]	Histogram of accuracies
	

5. Change the test images and use the other test set (for fashion network test on the digits test dataset and vice versa). Here again plot histograms. How sure are the classifiers when run on the wrong type of data?

5.1. Part A:

5.1.1. Result

I trained the model on the dataset of images of digits. Then, I tested the model over dataset of images of fashion.
As the previous model, I used 200 epochs and 300 layers.
The model was reached an accuracy of 99.97% on the training data.
However, the accuracy of the test dataset reached 7.75%
According to the histogram of the accuracies, 7.75% of the test images were correct and 92.25% were incorrect.

Because the model was trained on one dataset and tested on another dataset - the model does not really know how to distinguish the different classifications of each image in the test dataset. The accuracy obtained seems to be random. Which means, the results the model was "correct" were random. In another run of the model, we may get different accuracy.

5.1.2. Code (The full code is available in "Python.zip"):

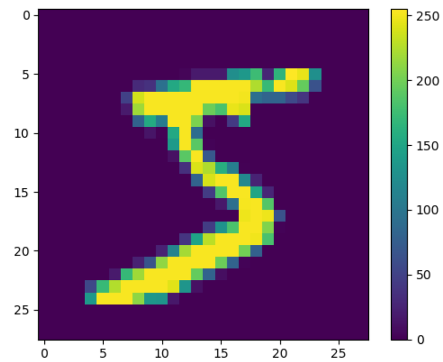
```
# Train the model
model.fit(digit_train_images, digit_train_labels, epochs=200)

# Test the model
test_loss, test_acc = model.evaluate(fashion_test_images,  fashion_test_labels, verbose=2)
```

5.1.3. Output (The full output is available in "Output.zip"):

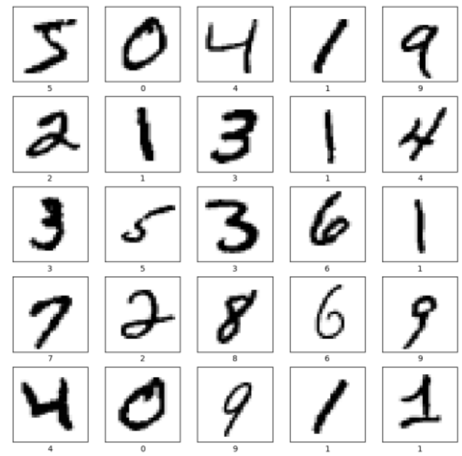
```
Epoch 1/200
   32/60000 [..............................] - ETA: 4:08 - loss: 2.5637 - acc: 0.0000e+00
  608/60000 [..............................] - ETA: 17s - loss: 1.6100 - acc: 0.5477
 1216/60000 [..............................] - ETA: 11s - loss: 1.1905 - acc: 0.6743
 1792/60000 [..............................] - ETA: 9s - loss: 0.9850 - acc: 0.7305
…
Epoch 200/200
   32/60000 [..............................] - ETA: 16s - loss: 0.0000e+00 - acc: 1.0000
  480/60000 [..............................] - ETA: 7s - loss: 6.9631e-04 - acc: 1.0000
  928/60000 [..............................] - ETA: 7s - loss: 3.6016e-04 - acc: 1.0000
 1408/60000 [..............................] - ETA: 6s - loss: 2.3738e-04 - acc: 1.0000
…
58880/60000 [============================>.] - ETA: 0s - loss: 0.0019 - acc: 0.9997
59360/60000 [============================>.] - ETA: 0s - loss: 0.0019 - acc: 0.9997
59872/60000 [============================>.] - ETA: 0s - loss: 0.0018 - acc: 0.9997
60000/60000 [==============================] - 6s 106us/sample - loss: 0.0018 - acc: 0.9997

Test size:  10000
10000/10000 - 0s - loss: 23955.7466 - acc: 0.0775

Test accuracy (original): 0.0775
Test accuracy (manual): 0.07750000000000001
(28, 28)
(1, 28, 28)
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

5.1.4. Graphs (The full output is available in “Output.zip”):

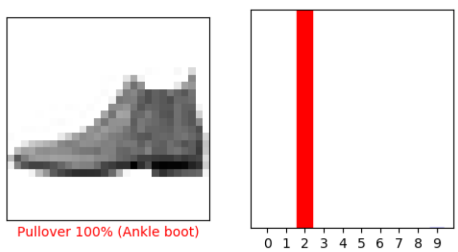
Training Image example



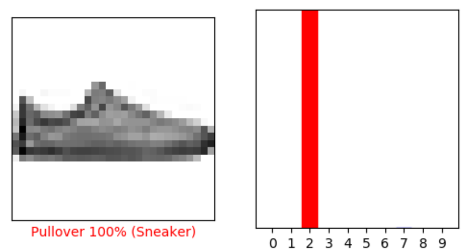
First 25 images from training set and class names



Prediction label of image [i=0]

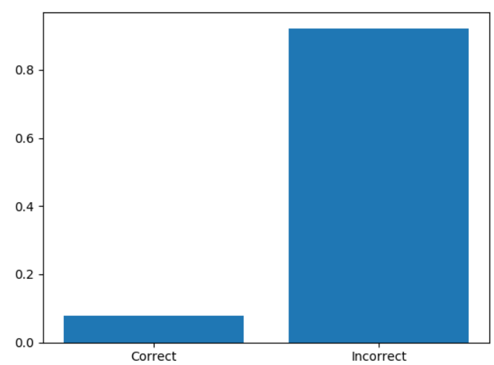


Prediction label of image [i=12]



Prediction label of 15 test images [0<i<15]

Histogram of accuracies



5.2. Part B:

5.2.1. Result

I trained the model on the dataset of images of fashion. Then, I tested the model over dataset of images of digits.
As the previous model, I used 200 epochs and 300 layers.
The model was reached an accuracy of 99.25% on the training data.
However, the accuracy of the test dataset reached 19.73%.
According to the histogram of the accuracies, 19.73% of the test images were correct and 80.27% were incorrect.

Because the model was trained on one dataset and tested on another dataset - the model does not really know how to distinguish the different classifications of each image in the test dataset. The accuracy obtained seems to be random. Which means, the results the model was "correct" were random. In another run of the model, we may get different accuracy.

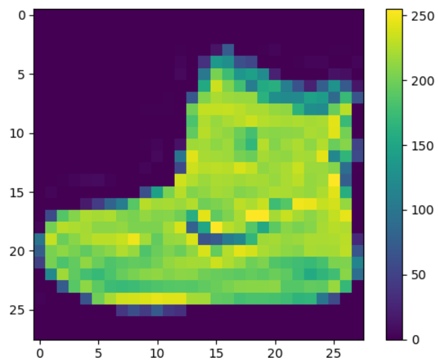
5.2.2. Code (The full code is available in “Python.zip”):

```
# Train the model¶
model.fit(fashion_train_images, fashion_train_labels, epochs=200)

# Test the model
test_loss, test_acc = model.evaluate(digit_test_images,  digit_test_labels, verbose=2)
```

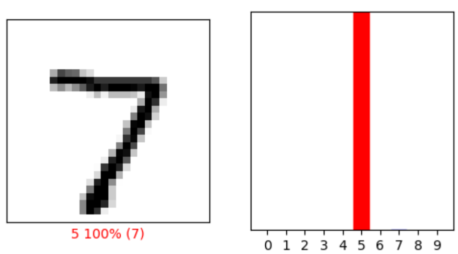
5.2.3. Output (The full output is available in “Output.zip”):

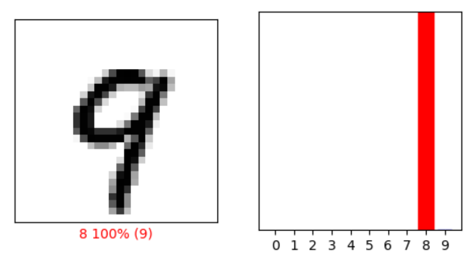
```
Epoch 1/200
   32/60000 [..............................] - ETA: 3:25 - loss: 2.3026 - acc: 0.1875
  544/60000 [..............................] - ETA: 17s - loss: 1.5222 - acc: 0.4908
 1056/60000 [..............................] - ETA: 12s - loss: 1.2227 - acc: 0.5900
 1600/60000 [..............................] - ETA: 9s - loss: 1.0494 - acc: 0.6494
…
Epoch 200/200
   32/60000 [..............................] - ETA: 11s - loss: 4.4882e-04 - acc: 1.0000
  704/60000 [..............................] - ETA: 4s - loss: 0.0422 - acc: 0.9830
 1312/60000 [..............................] - ETA: 4s - loss: 0.0436 - acc: 0.9840
 1952/60000 [..............................] - ETA: 4s - loss: 0.0410 - acc: 0.9867
…

58944/60000 [============================>.] - ETA: 0s - loss: 0.0225 - acc: 0.9924
59616/60000 [============================>.] - ETA: 0s - loss: 0.0224 - acc: 0.9925
60000/60000 [==============================] - 5s 82us/sample - loss: 0.0223 - acc: 0.9925
Test size:  10000
10000/10000 - 0s - loss: 10212.9262 - acc: 0.1973

Test accuracy (original): 0.1973
Test accuracy (manual): 0.19730000000000003
(28, 28)
(1, 28, 28)
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

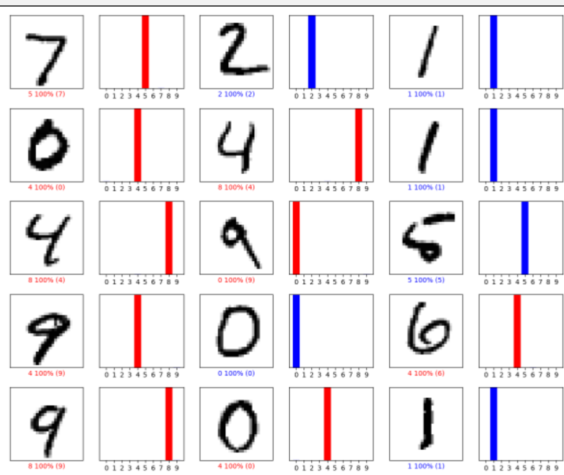
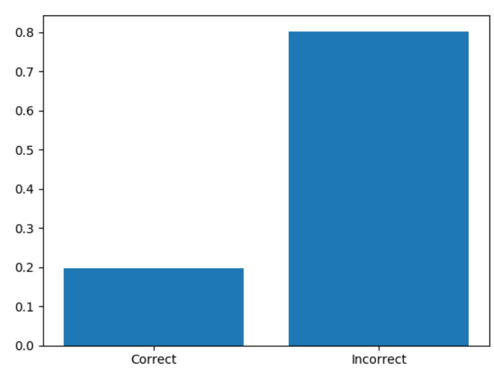
5.2.4. Graphs (The full output is available in “Output.zip”):

<table>
<tr><th>Training Image example</th><th>First 25 images from training set and class names</th></tr>
<tr><td>


</td><td>


</td></tr>
<tr><th>Prediction label of image [i=0]</th><th>Prediction label of image [i=12]</th></tr>
<tr><td>


</td><td>


</td></tr>
<tr><th>Prediction label of 15 test images [0<i<15]</th><th>Histogram of accuracies</th></tr>
<tr><td>

</td><td>


</td></tr>
</table>

6. Given a test image for one digit and a test image for another digit generate a set of images I(alpha) = alpha * I1 + (1-alpha) * I2 for 100 alphas between 0 and 1. Run the digit classifier on these images and plot the results both the classification and the probability of the class. What are your conclusions from that?

6.1. Result

I trained the model on the dataset of images of digits.
Then, I generated a new test dataset of 100 images, where each image was generated from a weighted average of 2 images in the training dataset: $I(\alpha) = \alpha \cdot I_1 + (1 - \alpha) \cdot I_2, \quad \alpha \in [0,1)$
The labels of the generated test images were set according to the following rule: for each generated average-weighted image : $I(\alpha)$ – if alpha is less than 0.5, then the label of $I(\alpha)$ will be the label of I_2. Otherwise, it will be the label of I_1.
Both of the images chosen randomly – I_1 chosen as an image of the digit '2' and I_2 chosen as an image of the digit '1'.
$I(\alpha)$ was an image of the combination of the images of digit '1' and '2'.

The model was reached an accuracy of 99.56% on the training data.
However, the accuracy of the test dataset reached 71%.
According to the histogram of the accuracies, 71% of the test images were correct and 29% were incorrect.

It is possible that in images with approximately alpha = 50%, it was difficult to categorize which class the image belongs to.

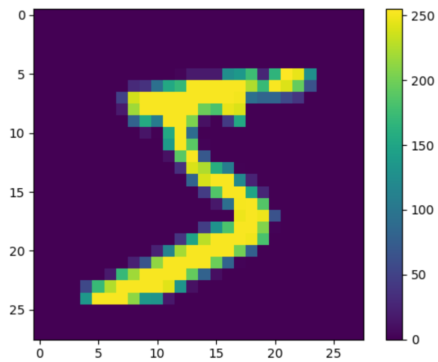
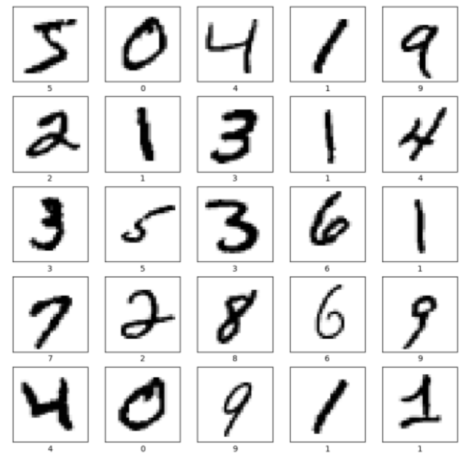
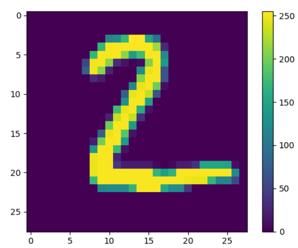
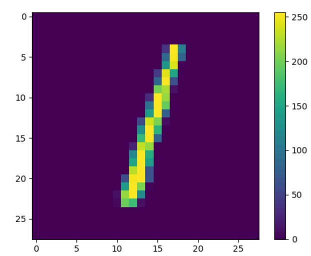
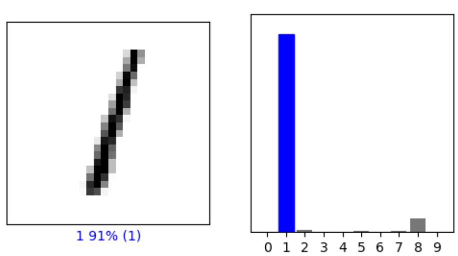
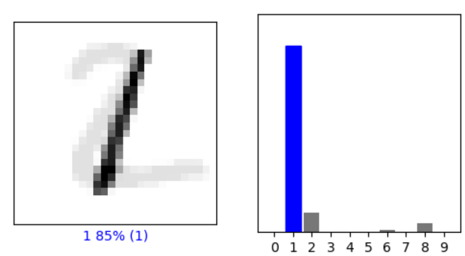
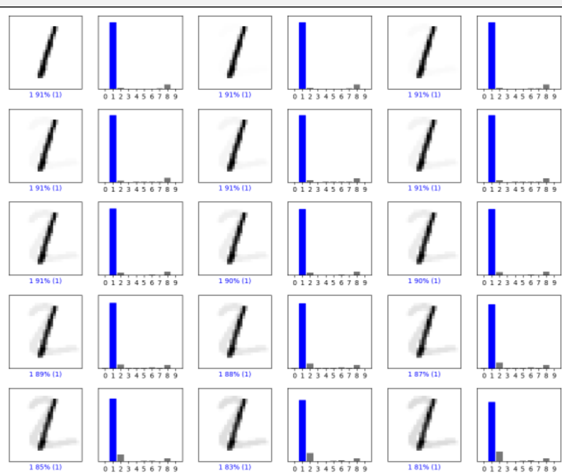
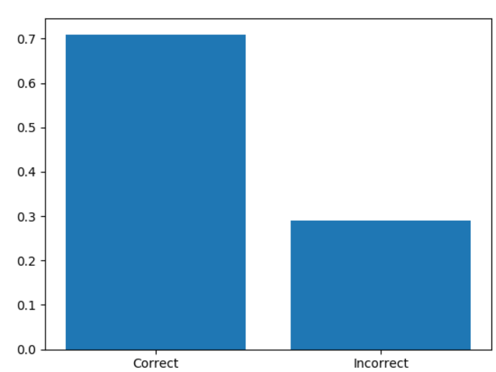
6.2. Code (The full code is available in "Python.zip"):

```
# Create new array
test_images_with_alpha_temp=[]
test_labels_with_alpha_temp=[]

test_images_with_alpha = np.array([])
test_labels_with_alpha = np.array([])

# Alpha is a number between 0 and 1
for number in range(0, 100, 1):

    # Create alpha
    alpha = (number/100)

    # Create new image
    img_new = ((img1 * alpha) + (img2 * (1 - alpha))) / 2

    print(img_new)

    if (number < 50):
        label_new = label2
    else:
        label_new = label1

    # Add new image to the list
    test_images_with_alpha_temp.append(img_new)

    # Add new image label to the array
    test_labels_with_alpha_temp.append(label_new)

# Create array of the generated images
test_images_with_alpha = np.stack(test_images_with_alpha_temp, axis=0)
test_labels_with_alpha = np.stack(test_labels_with_alpha_temp, axis=0)

test_loss, test_acc = model.evaluate(test_images_with_alpha,  test_labels_with_alpha, verbose=2)
```

6.3. <u>Output</u> (The full output is available in "Output.zip"):

```
Epoch 1/10
   32/60000 [..............................] - ETA: 2:14 - loss: 2.2323 - acc: 0.0938
  896/60000 [..............................] - ETA: 8s - loss: 1.5415 - acc: 0.5614
 1728/60000 [..............................] - ETA: 5s - loss: 1.1578 - acc: 0.6840
…
Epoch 10/10
   32/60000 [..............................] - ETA: 7s - loss: 0.0039 - acc: 1.0000
  992/60000 [..............................] - ETA: 3s - loss: 0.0132 - acc: 0.9940
…
58848/60000 [============================>.] - ETA: 0s - loss: 0.0144 - acc: 0.9956
59744/60000 [============================>.] - ETA: 0s - loss: 0.0146 - acc: 0.9956
60000/60000 [==============================] - 4s 73us/sample - loss: 0.0146 - acc: 0.9956

Test images [[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]
 ...
 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]

Test labels [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

100/100 - 0s - loss: 0.9968 - acc: 0.7100

Test accuracy (original): 0.71
Test accuracy (manual): 0.71
(28, 28)
(1, 28, 28)
[[2.1226342e-04 9.0937620e-01 1.3782789e-02 5.6325016e-04 1.8312243e-03
  3.9818413e-03 4.1574440e-03 5.1381667e-03 6.0741700e-02 2.1514372e-04]]
```

6.4. Graphs (The full output is available in “Output.zip”):

<table>
<tr><th>Training Image example</th><th>First 25 images from training set and class names</th></tr>
<tr><td></td><td></td></tr>
<tr><th>Source image of I1 for generation test dataset</th><th>Source image of I2 for generation test dataset</th></tr>
<tr><td></td><td></td></tr>
<tr><th>Prediction label of image [i=0]</th><th>Prediction label of image [i=12]</th></tr>
<tr><td></td><td></td></tr>
<tr><th>Prediction label of 15 test images [0<i<15]</th><th>Histogram of accuracies</th></tr>
<tr><td></td><td></td></tr>
</table>

7. Taking the two test images mentioned above, extract the results of the second layer (128 numbers) V1 and V2. Print them. These are the internal representation of the images.

7.1. Result

Using the previous model, I ran the algorithm once again and printed the images via representation in an array.
The first image was taken from the test dataset and was an image of the digit '2'.
The second image was taken from the test dataset and was an image of the digit '1'.

Then, I extracted the results of the second layer for each of them.

7.2. Code (The full code is available in "Python.zip"):

```
img1 = test_images[1]
label1 = test_labels[1]

img2 = test_images[2]
label2 = test_labels[2]

# Extract the second layer of each image from the test dataset
my_input_data = test_images

new_temp_model = K.Model(model.input, model.layers[1].output) #replace 3 with index of desired layer
output_of_2nd_layer = new_temp_model.predict(my_input_data) #this is what you want
output_of_2nd_layer_flatten = output_of_2nd_layer.flatten()

np.set_printoptions(threshold=np.inf) #Print all results
print('output_of_2nd_layer len=', len(output_of_2nd_layer))  # print test size

for i in range (len(output_of_2nd_layer)):
    print('\n','\n','output_of_2nd_layer, i=', i, ' output =' ,output_of_2nd_layer[i]) #print layers
```

7.3. <u>Output</u> (The full output is available in "Output.zip"):

```
img1 =  [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 116 125 171 255 255 150  93   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 169 253 253 253 253 253 253 218  30
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 169 253 253 253 213 142 176 253 253 122
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  52 250 253 210  32  12   0   6 206 253 140
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  77 251 210  25   0   0   0 122 248 253  65
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  31  18   0   0   0   0 209 253 253  65
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 117 247 253 198  10
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  76 247 253 231  63   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 128 253 253 144   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 176 246 253 159  12   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  25 234 253 233  35   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 198 253 253 141   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  78 248 253 189  12   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  19 200 253 253 141   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 134 253 253 173  12   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  25   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  43  20  20  20  20   5   0
    5  20  20  37 150 150 150 147  10   0]
 [  0   0   0   0   0   0   0   0 248 253 253 253 253 253 253 253 168 143
  166 253 253 253 253 253 253 253 123   0]
 [  0   0   0   0   0   0   0   0 174 253 253 253 253 253 253 253 253 253
  253 253 249 247 247 169 117 117  57   0]
 [  0   0   0   0   0   0   0   0   0 118 123 123 123 166 253 253 253 155
  123 123  41   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]

label1 =  2
```

```
img2 =  [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  38 254
  109   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  87 252
   82   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 135 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 244 150
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  84 254  63
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 202 223  11
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  32 254 216   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  95 254 195   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 140 254  77   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  57 237 205   8   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 124 255 165   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 171 254  81   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  24 232 215   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 120 254 159   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 151 254 142   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 228 254  66   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  61 251 254  66   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 141 254 205   3   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  10 215 254 121   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   5 198 176  10   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]

label2 =  1
```

```
output_of_2nd_layer, i= 1  output =
[ 939.62476      0.           0.        2271.7932       0.
   0.           0.        1374.6206       0.         817.9914
   0.         669.91986      0.           3.7656143    0.
   0.           0.           0.          98.77909      0.
 545.8703    1639.1353       0.           0.           0.
   0.         869.58264      0.           0.           0.
2011.0665       0.        1844.323        0.           0.
   0.           0.          67.688965     0.           0.
   0.         304.4722       0.           0.           0.
   0.           0.         566.1507       0.           0.
 220.194      235.00769      0.           0.         649.9116
   0.        1392.4863       0.           0.           0.
  79.09892      0.           0.           0.           0.
   0.         519.59564      0.           0.           0.
 579.05383   1064.2676       0.         626.804        0.
   0.         972.13617      0.           0.          17.06022
   0.           0.           0.           0.           0.
 751.52686      0.           0.          65.27749    972.2796
 184.36916    217.7673       0.           0.           0.
1013.7535       0.           0.           0.         181.12967
 414.6407    1936.0737    1989.6023       0.           0.
  85.19098      0.           0.         573.9432       0.
1167.6256     252.02185      0.           0.         570.8768
1663.2925    1146.774        0.         657.4305       0.
   0.           0.           0.         378.42084    330.3594
   0.           0.        1475.9553   ]


output_of_2nd_layer, i= 2  output =
[  0.         0.       177.77248  604.0104     0.       603.65845
  0.       557.0574   160.29947  471.85132    0.        48.41715
  0.       423.2381     0.         0.         0.       619.0772
  0.         0.        49.37328  184.11423    0.       384.31854
283.62143    0.         0.         0.       783.46924    0.
759.8518    34.8955     0.         0.         0.         0.
  0.         0.       155.04541  257.78717    0.        73.29929
  0.       635.4768     0.         0.         0.       631.0795
  0.       754.39124    0.         0.       819.6793     0.
  0.         0.       743.5557     0.        81.968155   0.
 12.061419   0.         0.       531.71844    0.       229.62984
482.28757   43.361137   0.         0.       276.67972    0.
  0.         0.         0.         0.       143.92305  186.16173
  0.        94.20784  347.33072  653.84503  560.9479     0.
  0.       568.3852     0.         0.         0.         0.
  0.         0.         0.       243.79457  811.06934    0.
 54.205692   0.       830.5772   564.8198    25.6098     0.
 35.071133   0.        93.640594   0.         0.       430.67285
142.52277    0.        68.63363  498.9651     0.         0.
  0.       215.61148    0.         0.       716.0542    32.31893
  0.         0.         0.       184.78534    0.         0.
108.83217  396.6044  ]
```

8. Predict the results of the network using these vectors. That means run the final level only on them. You should get the same results.

8.1. Result

Using the previous model, I ran the algorithm once again and extracted the results of the last layer for each of the test images.
In each image, the last layer contains an array of probabilities - where each index represents the class possibilities of what the image can be, and each value in the array represents the probability that what is shown in the image is that class.
Then, I compared the classified results of the last layer to the true results of the test images.
The calculated accuracy and the accuracy of the model were equal. The accuracy that was obtained was 69%.

8.2. Code (The full code is available in "Python.zip"):

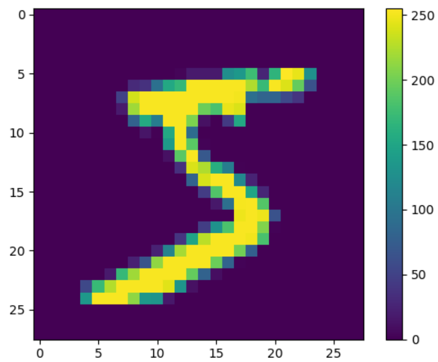
```
# Extract the second layer of each image from the test dataset
my_input_data = test_images_with_alpha

last_layer = len(model.layers)-1

new_temp_model = K.Model(model.input, model.layers[last_layer].output) #replace 3 with index of desired layer
output_of_last_layer = new_temp_model.predict(my_input_data) #this is what you want
output_of_last_layer_flatten = output_of_last_layer.flatten()

np.set_printoptions(threshold=np.inf) #Print all results

list_of_max_index = []
for i in range (len(output_of_last_layer)):
    output_of_last_layer_of_img = output_of_last_layer[i]
    max_value = np.amax(output_of_last_layer_of_img)
    for j in range (len(output_of_last_layer_of_img)):
        if output_of_last_layer_of_img[j] == max_value:
            max_index = j
            list_of_max_index.append(j)
            break

    print('\n','\n','output_of_last_layer, i=', i, '\n',' output ='
,output_of_last_layer_of_img,'\n','max_value',max_value,'\n','max_index',max_index) #print layers


# Evaluate accuracy
test_labels_with_alpha_list = np.array(test_labels_with_alpha)
test_loss, test_acc = model.evaluate(test_images_with_alpha,  test_labels_with_alpha, verbose=2)

p = model.predict(test_images_with_alpha)
p = np.argmax(p,axis=1)
test_acc1 = 1-np.count_nonzero(p-test_labels_with_alpha)/len(test_labels_with_alpha)

check_acc = 0
for k in range (len(test_labels_with_alpha_list)):
    if list_of_max_index[k] == test_labels_with_alpha_list[k]:
        check_acc += 1
test_acc_from_last_layer = check_acc / len(test_labels_with_alpha_list)


print('\nTest accuracy (original):', test_acc, '\nTest accuracy (manual):', test_acc1, '\nTest accuracy calculated
from last layer :', test_acc_from_last_layer)
```

8.3. Output (The full output is available in "Output.zip"):

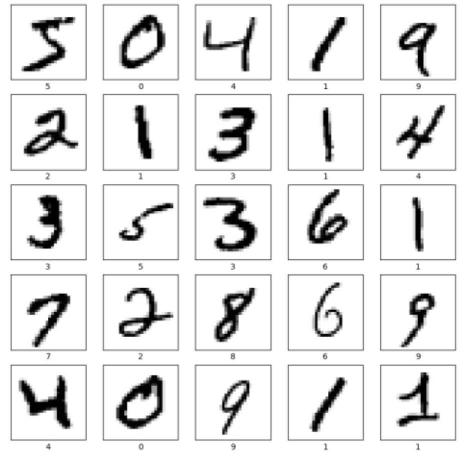
```
test_images_with_alpha [[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
…
test_labels_with_alpha [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

Epoch 1/10
   32/60000 [..............................] - ETA: 2:18 - loss: 2.3738 - acc: 0.0312
  896/60000 [..............................] - ETA: 8s - loss: 1.6885 - acc: 0.5033
 1664/60000 [..............................] - ETA: 6s - loss: 1.2850 - acc: 0.6394
…
Epoch 10/10
…
59936/60000 [============================>.] - ETA: 0s - loss: 0.0162 - acc: 0.9951
60000/60000 [==============================] - 4s 71us/sample - loss: 0.0163 - acc: 0.9951

output_of_last_layer, i= 0
  output = [6.9685047e-05 9.2785943e-01 7.5802724e-03 3.6837559e-04 6.9567957e-03
 1.4146540e-03 4.8387828e-04 1.1721648e-02 4.3481901e-02 6.3396154e-05]
 max_value 0.9278594
 max_index 1

output_of_last_layer, i= 1
  output = [7.8523102e-05 9.2608994e-01 9.4182445e-03 3.9074241e-04 6.7476197e-03
 1.3486085e-03 5.3330365e-04 1.1593652e-02 4.3733556e-02 6.5767621e-05]
 max_value 0.92608994
 max_index 1

 output_of_last_layer, i= 2
  output = [8.9377652e-05 9.2298049e-01 1.1829171e-02 4.2266774e-04 6.5958980e-03
 1.3053172e-03 5.8274908e-04 1.1664173e-02 4.4460990e-02 6.9110501e-05]
 max_value 0.9229805
 max_index 1
…

list_of_max_index [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2]
Test images [[[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
   0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
…]]

Test labels [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

100/100 - 0s - loss: 0.9574 - acc: 0.6900

Test accuracy (original): 0.69
Test accuracy (manual): 0.69
Test accuracy calculated from last layer : 0.69

(28, 28)
(1, 28, 28)
[[7.8522884e-05 9.2609006e-01 9.4182501e-03 3.9074247e-04 6.7476141e-03
  1.3486061e-03 5.3330371e-04 1.1593665e-02 4.3733548e-02 6.5767628e-05]]
```

8.4. Graphs (The full output is available in “Output.zip”):

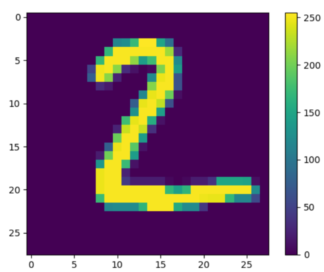
Training Image example



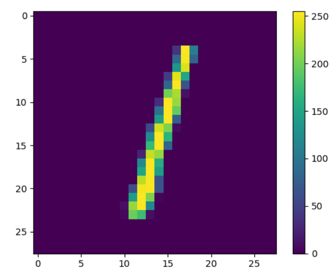
First 25 images from training set and class names



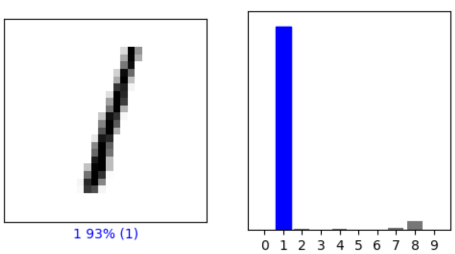
Source image of I1 for generation test dataset



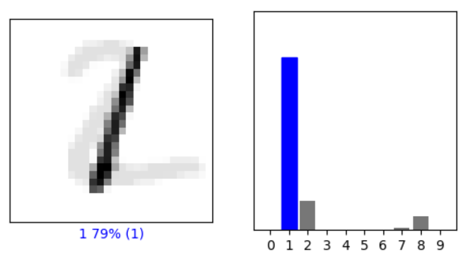
Source image of I2 for generation test dataset



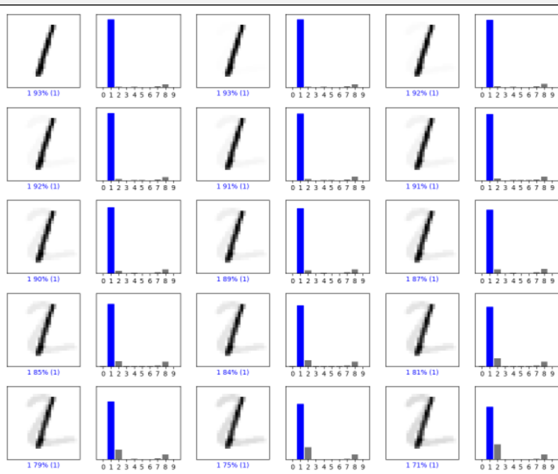
Prediction label of image [i=0]



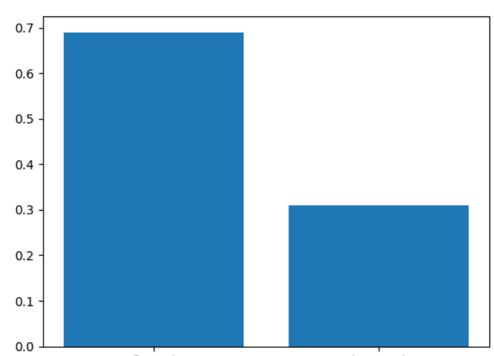
Prediction label of image [i=12]



Prediction label of 15 test images [0<i<15]



Histogram of accuracies



9. **Take V1 and V2 and generate 50 vectors, where V(alpha) = alpha * V1 + (1-alpha) *V2, where alpha is between -1 and 2. Plot the probabilities for these two classes for all these values. See what values you get when alpha is far from 0 (digit 1) and 1 (digit 2).**

9.1. Result

I trained the model on the dataset of images of digits.
Then, I generated a new test dataset of 50 images, where each image was generated from a weighted average of 2 images in the training dataset: $V(\alpha) = \alpha \cdot I_1 + (1-\alpha) \cdot I_2, \quad \alpha \in [-1,2)$
The labels of the generated test images were set according to the following rule: for each generated average-weighted image : $I(\alpha)$ – if alpha is less than 0.5 ((-1+2)/2 = 0.5), then the label of $I(\alpha)$ will be the label of I_2. Otherwise, it will be the label of I_1.
Both of the images chosen randomly – I_1 chosen as an image of the digit '2' and I_2 chosen as an image of the digit '1'.
$I(\alpha)$ was an image of the combination of the images of digit '1' and '2'.

The model was reached an accuracy of 99.49% on the training data.
However, the accuracy of the test dataset reached 66%.
According to the histogram of the accuracies, 66% of the test images were correct and 34% were incorrect.

According to the results, when alpha is far from 0, which means closer to I_1 and the image of the digit '2' is more prominent but white - then the model recognizes most of the test images as an image of the digit '7'.
However, when alpha is closer to from 0, which means closer to I_2 and the image of the digit '1' is more prominent but white - then the model recognizes most of the test images as an image of the digit '2'.

9.2. Code (The full code is available in "Python.zip"):

```
# Create new array
test_images_with_alpha_temp=[]
test_labels_with_alpha_temp=[]

test_images_with_alpha = np.array([])
test_labels_with_alpha = np.array([])

# Alpha is a number between -1 and 2
alpha=-1
while alpha<2:

    # Create new image
    img_new = ((img1 * alpha) + (img2 * (1 - alpha))) / 2
    print(img_new)

    if (alpha < 0.5):
        label_new = label2
    else:
        label_new = label1

    # Add new image to the list
    test_images_with_alpha_temp.append(img_new)

    # Add new image label to the array
    test_labels_with_alpha_temp.append(label_new)
    #test_labels_with_alpha = np.append(test_labels_with_alpha, label_new)

    alpha += 0.06

# Create array of the generated images
test_images_with_alpha = np.stack(test_images_with_alpha_temp, axis=0)
test_labels_with_alpha = np.stack(test_labels_with_alpha_temp, axis=0)

test_loss, test_acc = model.evaluate(test_images_with_alpha,  test_labels_with_alpha, verbose=2)
```

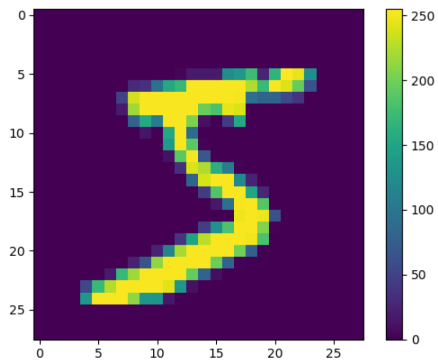
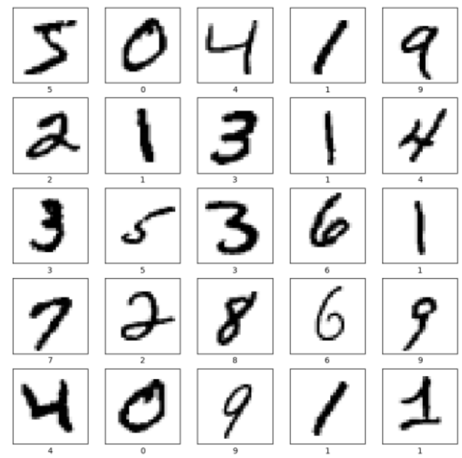
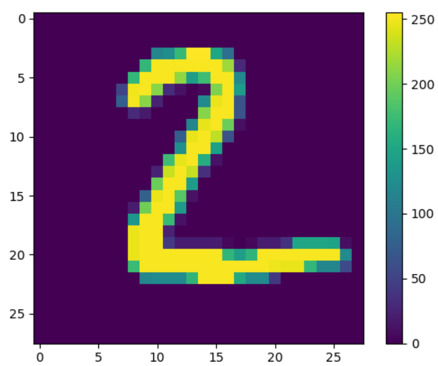
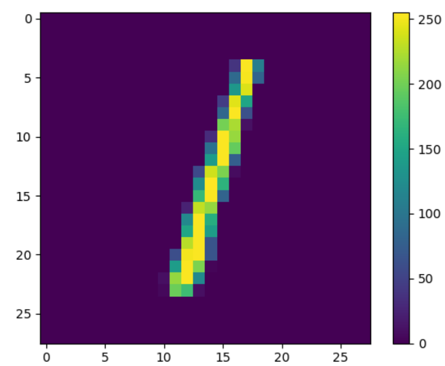
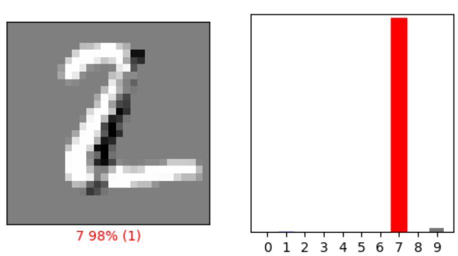
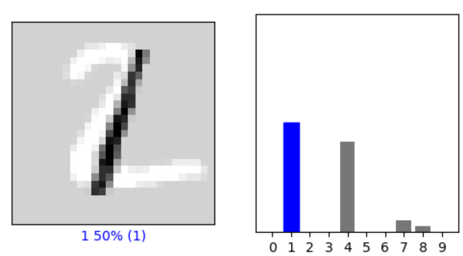
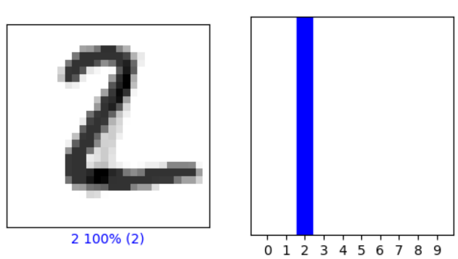
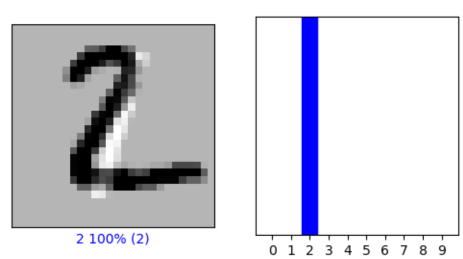
9.3. <u>Output</u> (The full output is available in “Output.zip”):

```
img1 =  [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 116 125 171 255 255 150  93   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 169 253 253 253 253 253 253 218  30
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 169 253 253 253 213 142 176 253 253 122
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  52 250 253 210  32  12   0   6 206 253 140
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  77 251 210  25   0   0   0 122 248 253  65
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  31  18   0   0   0   0 209 253 253  65
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 117 247 253 198  10
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  76 247 253 231  63   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 128 253 253 144   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 176 246 253 159  12   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  25 234 253 233  35   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 198 253 253 141   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  78 248 253 189  12   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  19 200 253 253 141   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 134 253 253 173  12   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  25   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 248 253 253  43  20  20  20  20   5   0
    5  20  20  37 150 150 150 147  10   0]
 [  0   0   0   0   0   0   0   0 248 253 253 253 253 253 253 253 168 143
  166 253 253 253 253 253 253 253 123   0]
 [  0   0   0   0   0   0   0   0 174 253 253 253 253 253 253 253 253 253
  253 253 249 247 247 169 117 117  57   0]
 [  0   0   0   0   0   0   0   0   0 118 123 123 123 166 253 253 253 155
  123 123  41   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
label1 =  2
```

```
img2 =  [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  38 254
  109   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  87 252
   82   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 135 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 244 150
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  84 254  63
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 202 223  11
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  32 254 216   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  95 254 195   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 140 254  77   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  57 237 205   8   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 124 255 165   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 171 254  81   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  24 232 215   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 120 254 159   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 151 254 142   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 228 254  66   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  61 251 254  66   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 141 254 205   3   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  10 215 254 121   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   5 198 176  10   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
label2 =  1
```

```
test_labels_with_alpha [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2]

Train on 60000 samples

Epoch 1/10
   32/60000 [..............................] - ETA: 2:40 - loss: 2.4426 - acc: 0.0625
  832/60000 [..............................] - ETA: 9s - loss: 1.7527 - acc: 0.4952
 1664/60000 [..............................] - ETA: 6s - loss: 1.2789 - acc: 0.6454
…
Epoch 10/10
…
57952/60000 [===========================>..] - ETA: 0s - loss: 0.0158 - acc: 0.9948
58816/60000 [============================>.] - ETA: 0s - loss: 0.0158 - acc: 0.9948
59328/60000 [============================>.] - ETA: 0s - loss: 0.0159 - acc: 0.9947
60000/60000 [==============================] - 5s 79us/sample - loss: 0.0160 - acc: 0.9949
Test images [[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]
Test labels [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2]
50/50 - 0s - loss: 3.0280 - acc: 0.6600

Test accuracy (original): 0.66
Test accuracy (manual): 0.6599999999999999
(28, 28)
(1, 28, 28)
[[1.1649617e-16 2.1574893e-08 6.3789146e-16 1.1028823e-11 2.9255222e-07
  1.9048621e-10 1.5124934e-14 9.9999785e-01 5.2894117e-10 1.7355468e-06]]
```

9.4. Graphs (The full output is available in “Output.zip”):

Training Image example	First 25 images from training set and class names
	
Source image of I1 for generation test dataset	**Source image of I2 for generation test dataset**
	
Prediction label of image [i=0]	**Prediction label of image [i=12]**
	
Prediction label of image [i=30]	**Prediction label of image [i=45]**
	

Prediction label of 15 test images [0<i<15]	Histogram of accuracies