ANN in Deep Learning (MNIST with TensorFlow/Keras)

Muhammad Zaid Kamil

Installation

```
In [5]: import tensorflow.compat.v1 as tf
```

```
In [6]: tf.disable_v2_behavior()
```

```
WARNING:tensorflow:From C:\Users\zombi\AppData\Roaming\Python\Python39\site-packages\tensorflow\python\compat\v2_compat.py:107:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

```
In [7]: import tensorflow as tf
```

```
In [8]: hello = tf.constant('Hello, TensorFlow!')
```

```
In [9]: sess = tf.compat.v1.Session()
```

```
In [10]: print(sess.run(hello))
```

```
b'Hello, TensorFlow!'
```

MNIST with TensorFlow

Read the description of the task in https://www.tensorflow.org/tutorials/quickstart/beginner.

Step 1:

```
In [2]: import tensorflow as tf

        mnist = tf.keras.datasets.mnist

        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        x_train, x_test = x_train / 255.0, x_test / 255.0

        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
        model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
        model.add(tf.keras.layers.Dropout(0.2))
        model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

Step 1: Setting up Tensor Flow

Step 2: Loading the dataset

Step 3: Building a ML model

```
In [3]: predictions = model(x_train[:1]).numpy()
        predictions

Out[3]: array([[0.09971632, 0.10760017, 0.16850476, 0.13726926, 0.05614685,
                0.11357612, 0.0781882 , 0.05929397, 0.11647072, 0.0632336 ]],
              dtype=float32)
```

Step 5: Predictions to return logits one for each class

```
In [4]: tf.nn.softmax(predictions).numpy()

Out[4]: array([[0.09991138, 0.10070218, 0.10702603, 0.10373469, 0.09565176,
                0.10130578, 0.09778346, 0.09595326, 0.10159943, 0.09633204]],
              dtype=float32)
```

Step 6: The tf.nn.softmax is used to convert logits → probabilities for each class

```
In [5]: loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Step 7: Define Loss function

```
In [6]: loss_fn(y_train[:1], predictions).numpy()

Out[6]: 2.2896118
```

Step 8: Untrained model, initial loss close to 2.8

```
In [8]: model.fit(x_train, y_train, epochs=5)

Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 1.5796 - accuracy: 0.9007
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.5224 - accuracy: 0.9446
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.5090 - accuracy: 0.9565
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.5020 - accuracy: 0.9625
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.4971 - accuracy: 0.9668

Out[8]: <keras.callbacks.History at 0x1b7c32989a0>
```

Step 9: Train and evaluate the model. Model.fit to adjust model parameters and adjust loss

```
In [9]: model.evaluate(x_test,  y_test, verbose=2)

313/313 - 1s - loss: 1.4923 - accuracy: 0.9717 - 654ms/epoch - 2ms/step

Out[9]: [1.4923325777053833, 0.9717000126838684]
```

Step 9: The Model.evaluate method checks the models performance, usually on a "Validation-set" or "Test-set".

Step 10: The image classifier is trained to 97% accuracy in this dataset

```
In [10]: probability_model = tf.keras.Sequential([
           model,
           tf.keras.layers.Softmax()
         ])
```

Step 11: Wrap the trained model and attach to softmax

```
In [11]: probability_model(x_test[:5])

Out[11]: <tf.Tensor: shape=(5, 10), dtype=float32, numpy=
         array([[0.08533674, 0.08533674, 0.08533674, 0.08533674, 0.08533674,
                 0.08533674, 0.08533674, 0.23196931, 0.08533674, 0.08533674],
                [0.08533674, 0.08533674, 0.23196931, 0.08533674, 0.08533674,
                 0.08533674, 0.08533674, 0.08533674, 0.08533674, 0.08533674],
                [0.0853368 , 0.23196824, 0.08533701, 0.0853368 , 0.0853368 ,
                 0.0853368 , 0.08533687, 0.08533693, 0.08533686, 0.0853368 ],
                [0.23196931, 0.08533674, 0.08533674, 0.08533674, 0.08533674,
                 0.08533674, 0.08533674, 0.08533674, 0.08533674, 0.08533674],
                [0.08539622, 0.08539622, 0.08539622, 0.08539622, 0.23102519,
                 0.08539622, 0.08539622, 0.08539643, 0.08539622, 0.08580474]],
               dtype=float32)>
```

Step 12: The Model to return probability

Create a function that is a model for recognizing digits, based on looking at every pixel in the image

```
In [12]: import tensorflow as tf
         tf.__version__

Out[12]: '2.9.1'

In [13]: mnist = tf.keras.datasets.mnist

In [14]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

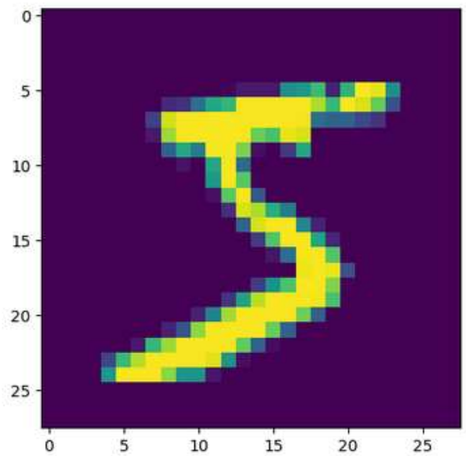
Step 1: Retreiving Data set of 28x28 images of handwritten digits of 0-9. Image.

#. Feed through the pictures in neural network and neural network check which number image

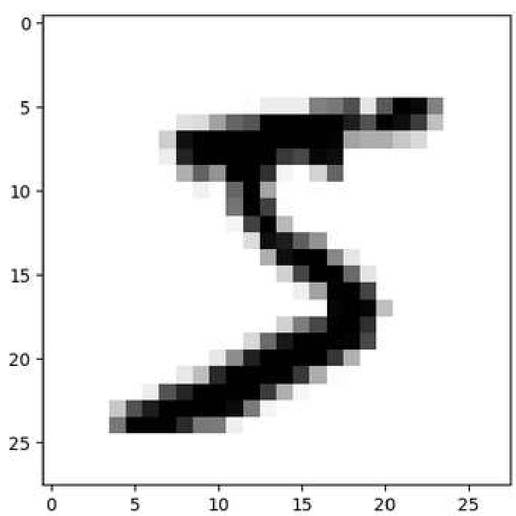
```
In [15]: import matplotlib.pyplot as plt
         print(x_train[0])
         plt.show()

[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

Step 2: The number plot shown represents pixels range from 0 to 255 pixels. Pixels > 0 represents the digit

```
In [17]: import matplotlib.pyplot as plt
         plt.imshow(x_train[0])
         plt.show()
```



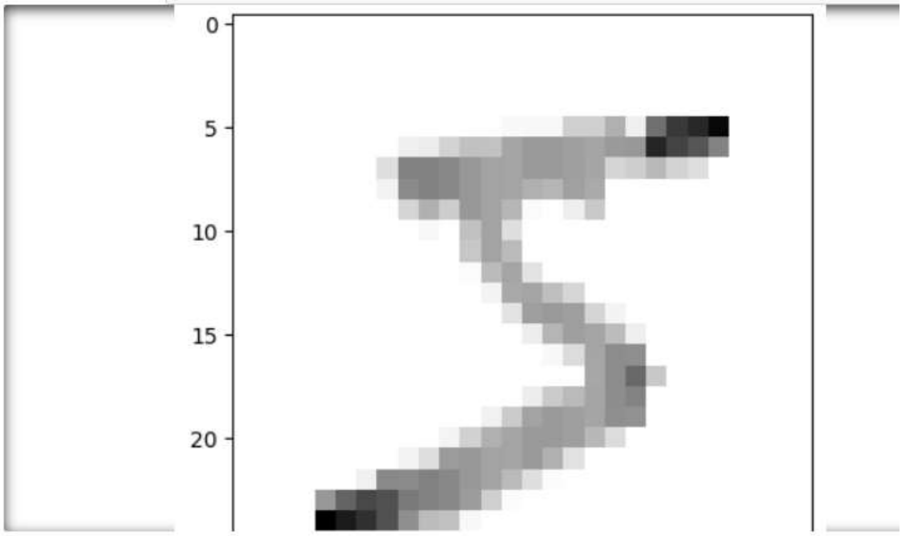
```
In [18]: plt.imshow(x_train[0], cmap = plt.cm.binary)
Out[18]: <matplotlib.image.AxesImage at 0x236460b4f10>
```



Step 3: The actual digit representation image in Black and White

```
In [21]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
         x_train = tf.keras.utils.normalize(x_train, axis = 1)
         x_test = tf.keras.utils.normalize(x_test, axis = 1)
```

```
In [22]: plt.imshow(x_train[0], cmap = plt.cm.binary)
         plt.show()
         print(x_train[0])
```




```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
 0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
 0.         0.         0.         0.         ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.05780486 0.06524513 0.16128198 0.22713296
 0.22277047 0.32790981 0.36833534 0.3689874  0.34978968 0.32678448
 0.368094   0.3747499  0.79066747 0.67980478 0.61494005 0.45002403
 0.         0.         0.         0.         ]
[0.         0.         0.         0.         0.         0.
 0.         0.12250613 0.45858525 0.45852825 0.43408872 0.37314701
 0.33153488 0.32790981 0.36833534 0.3689874  0.34978968 0.32420121
 0.15214552 0.17865984 0.25626376 0.1573102  0.12298801 0.
```

Step 4: Normalizing the data. The pixels range from 0 to 1.

```
In [1]: import tensorflow as tf

        mnist = tf.keras.datasets.mnist

        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        x_train = tf.keras.utils.normalize(x_train, axis = 1)
        x_test = tf.keras.utils.normalize(x_test, axis = 1)

        model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
        model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
        model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

        model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
        model.fit(x_train, y_train, epochs=3)

        Epoch 1/3
        1875/1875 [==============================] - 8s 3ms/step - loss: 0.2613 - accuracy: 0.9227
        Epoch 2/3
        1875/1875 [==============================] - 6s 3ms/step - loss: 0.1103 - accuracy: 0.9654
        Epoch 3/3
        1875/1875 [==============================] - 7s 4ms/step - loss: 0.0755 - accuracy: 0.9764

Out[1]: <keras.callbacks.History at 0x20722261d90>
```

Average of 96% Accuracy of the Neural network after 3 EPOCHS. This was in a sample. – model generalized

```
In [2]: val_loss, val_acc = model.evaluate(x_test, y_test)
        print(val_loss, val_acc)

        313/313 [==============================] - 1s 2ms/step - loss: 0.1000 - accuracy: 0.9678
        0.10002856701612473 0.9678000211715698
```

Calculating validation Loss = 0.1 and accuracy = 0.9678

```
In [3]: predictions = model.predict([x_test])

        WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor. Received: inputs=(<tf.Tensor 'IteratorG
        etNext:0' shape=(None, 28, 28) dtype=float32>,). Consider rewriting this model with the Functional API.
        313/313 [==============================] - 1s 3ms/step
```

```
In [4]: print(predictions)

        [[3.0582585e-09 3.4056779e-07 1.1552959e-06 ... 9.9999136e-01
          3.9908922e-08 1.8685662e-06]
         [5.1894166e-11 3.5538431e-03 9.9644476e-01 ... 1.4328446e-09
          1.3937509e-07 4.0627085e-10]
         [8.1326927e-09 9.9993402e-01 1.7088509e-05 ... 1.3252626e-05
          3.1097064e-05 9.1889092e-08]
         ...
         [1.7900863e-10 2.9408066e-06 5.2566147e-09 ... 1.5657708e-05
          4.7391813e-06 2.0817497e-04]
         [1.7899889e-05 2.9049856e-06 4.4617636e-07 ... 1.9866290e-07
          2.0838568e-03 9.6705639e-07]
         [6.4102355e-09 2.7654480e-07 1.0818975e-07 ... 8.1363150e-13
          6.4531747e-08 2.4956769e-08]]
```

In [6]:
```
import numpy
print(numpy.argmax(predictions[0]))
```

```
7
```

Classify Images of Clothing

In [1]:
```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.10.0
```

In [2]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

In [3]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [4]:
```
train_images.shape
```

Out[4]:
```
(60000, 28, 28)
```

In [5]:
```
len(train_labels)
```

Out[5]:
```
60000
```

In [6]:
```
train_labels
```

Out[6]:
```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

In [7]:
```
test_images.shape
```

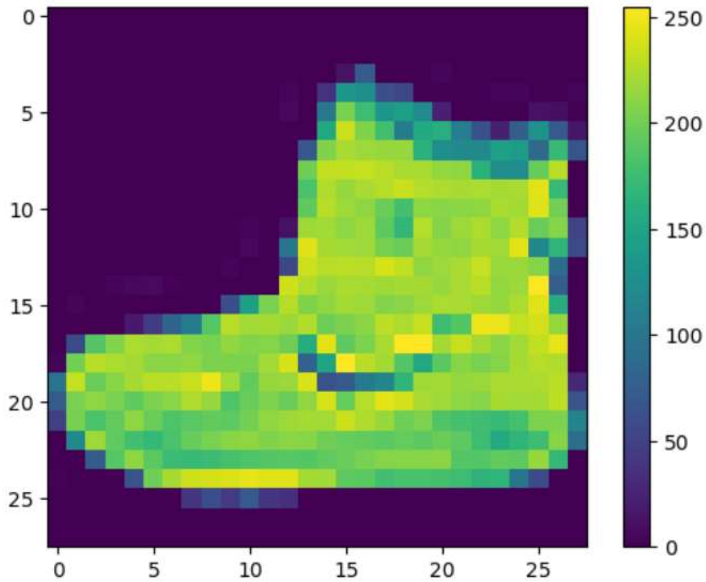
Out[7]:
```
(10000, 28, 28)
```

In [8]:
```
len(test_labels)
```

Out[8]:
```
10000
```

In [9]:

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

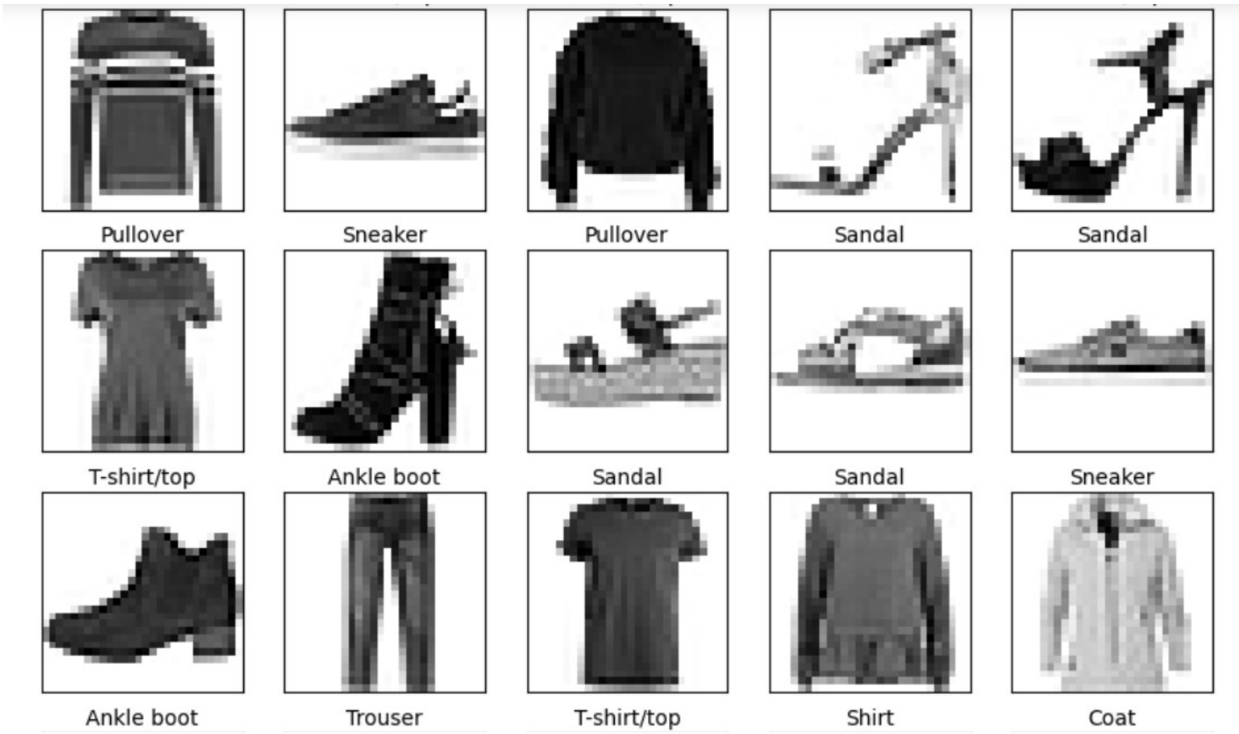


In [10]:

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

In [11]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



In [12]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

In [13]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [13]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [14]:
```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5034 - accuracy: 0.8244
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3828 - accuracy: 0.8610
Epoch 3/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3400 - accuracy: 0.8773
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3157 - accuracy: 0.8843
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2961 - accuracy: 0.8898
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2795 - accuracy: 0.8961
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2694 - accuracy: 0.8996
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2588 - accuracy: 0.9028
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2498 - accuracy: 0.9067
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2412 - accuracy: 0.9097
```

Out[14]:
```
<keras.callbacks.History at 0x2221ae7e0d0>
```

In [15]:
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 0.3257 - accuracy: 0.8885 - 501ms/epoch - 2ms/step

Test accuracy: 0.8884999752044678
```

In [16]:
```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

In [17]:
```
predictions = probability_model.predict(test_images)
```

```
313/313 [==============================] - 0s 1ms/step
```

In [18]:
```
predictions[0]
```

Out[18]:
```
array([1.80547886e-05, 1.80332869e-08, 1.09664903e-07, 2.56643684e-08,
       1.06003222e-06, 5.15667023e-04, 3.11436634e-06, 1.26864975e-02,
       1.07358716e-07, 9.86775398e-01], dtype=float32)
```

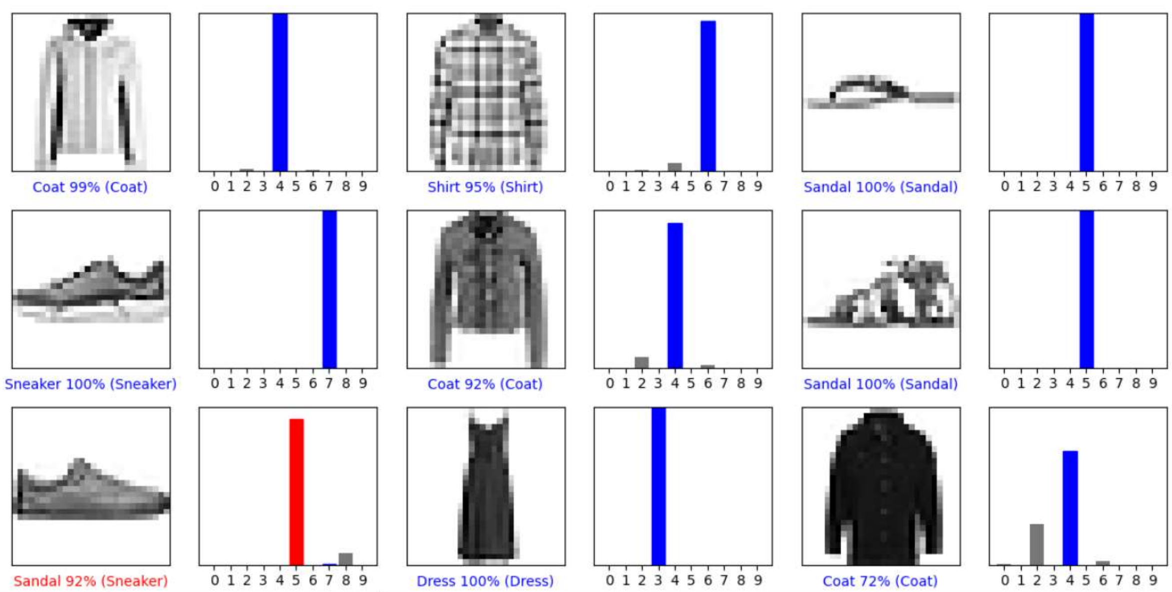
In [19]:
```
np.argmax(predictions[0])
```

Out[19]:
```
9
```

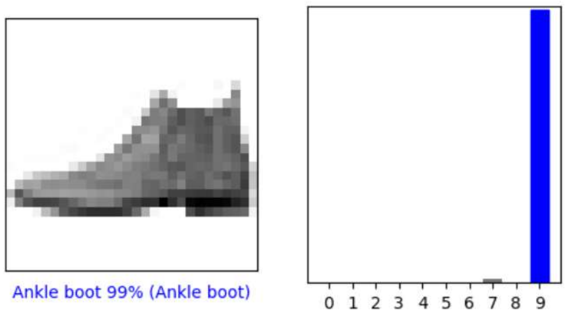
In [20]:
```
test_labels[0]
```

Out[20]:
```
9
```

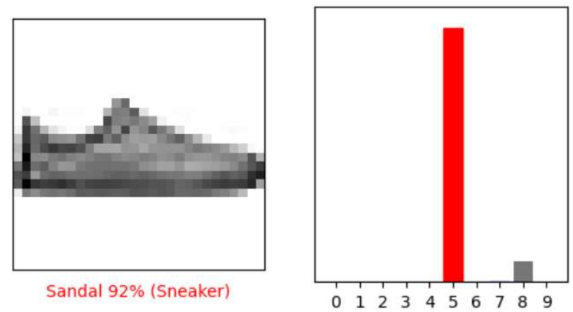
In [21]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```



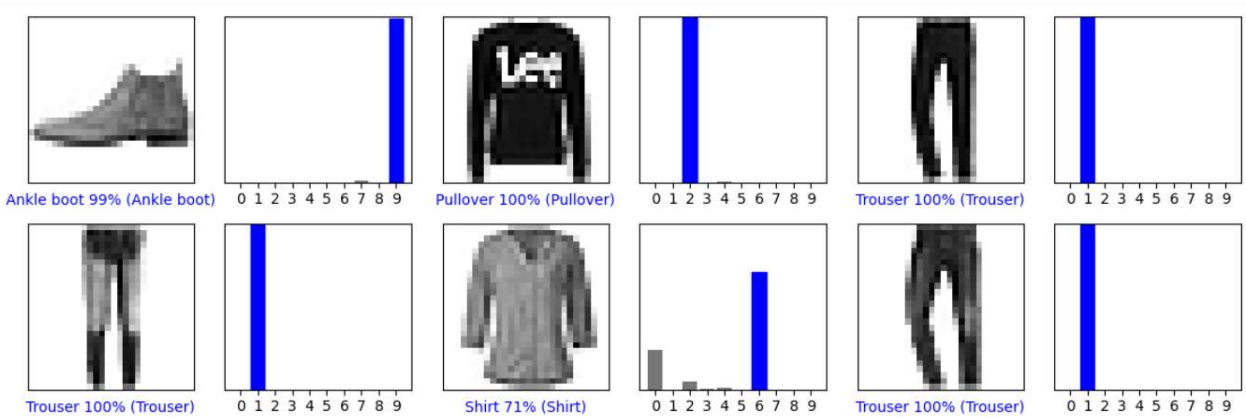
In [22]:
```python
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



In [23]:
```python
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



In [24]:
```python
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



In [25]:
```python
# Grab an image from the test dataset.
img = test_images[1]

print(img.shape)
```

```
(28, 28)
```

In [26]:
```python
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)
```

```
(1, 28, 28)
```

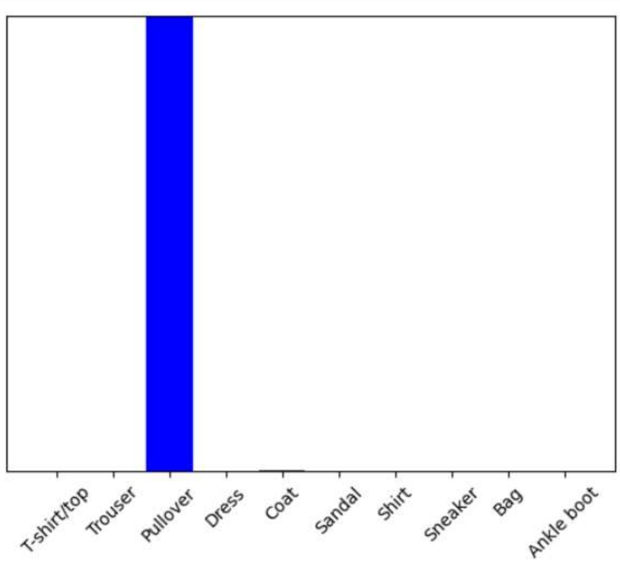
In [27]:
```python
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)
```

```
(1, 1, 28, 28)
```

In [28]:
```python
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
1/1 [==============================] - 0s 46ms/step
[[1.1324369e-04 4.5422674e-13 9.9893767e-01 4.1311107e-11 7.7292504e-04
  3.1105825e-13 1.7616725e-04 8.3188371e-18 2.0409157e-09 7.8077664e-17]]
```

In [29]:
```python
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```



In [30]:
```python
np.argmax(predictions_single[0])
```

Out[30]: 2