Problem Statement

Convolutional neural network (CNN):-Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Basic classification: 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__)
```
```
2.16.1
```

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

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_da
```
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 3us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

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

Explore the data

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
```

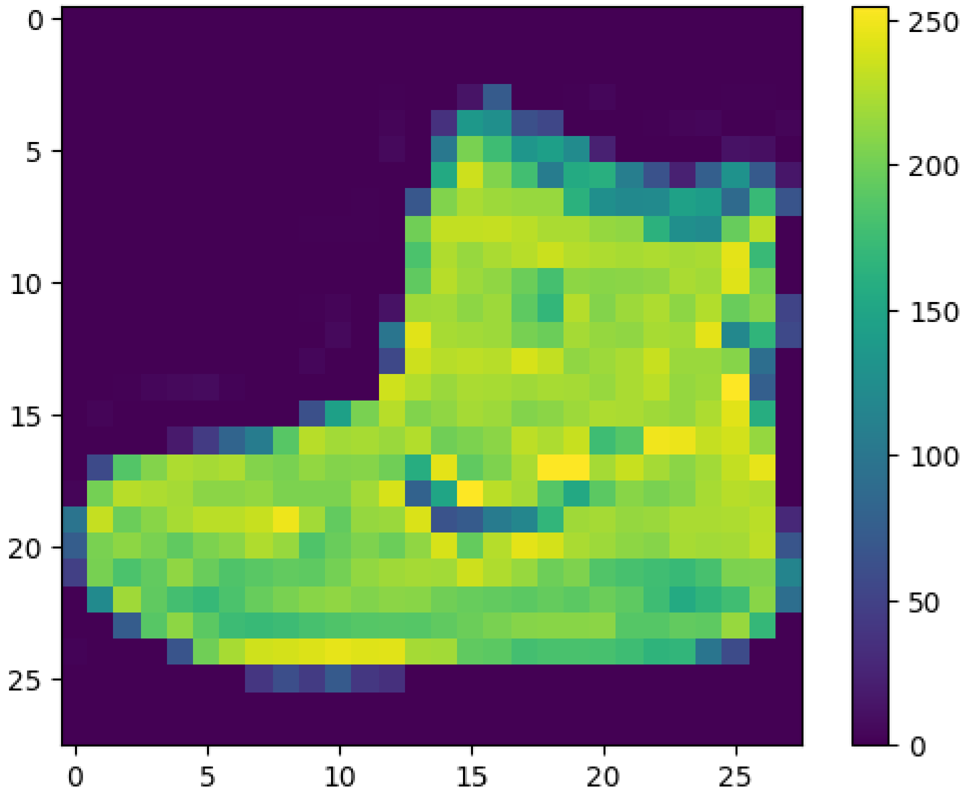
Out[7]: (10000, 28, 28)

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

Out[8]: 10000

Preprocess the data

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)
])
```

```
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras
\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an `input_shape`/
`input_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Compile the model

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

In [16]:
```
model.fit(train_images, train_labels, epochs=15)
```

```
Epoch 1/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9642 - loss: 0.0941
Epoch 2/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9665 - loss: 0.0882
Epoch 3/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9673 - loss: 0.0884
Epoch 4/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9675 - loss: 0.0897
Epoch 5/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9687 - loss: 0.0861
Epoch 6/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9678 - loss: 0.0865
Epoch 7/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9683 - loss: 0.0835
Epoch 8/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9681 - loss: 0.0859
Epoch 9/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9703 - loss: 0.0801
Epoch 10/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9695 - loss: 0.0806
Epoch 11/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9694 - loss: 0.0823
Epoch 12/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9699 - loss: 0.0809
Epoch 13/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9704 - loss: 0.0776
Epoch 14/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9723 - loss: 0.0752
Epoch 15/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9709 - loss: 0.0799
```

Out[16]: <keras.src.callbacks.history.History at 0x23aad214750>

In [17]:

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

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

```
313/313 - 1s - 4ms/step - accuracy: 0.8899 - loss: 0.5718

Test accuracy: 0.8899000287055969
```

In [18]:

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

In [19]:

```
predictions = probability_model.predict(test_images)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

In [20]:

```
predictions[0]
```

Out[20]:

```
array([1.2963246e-29, 1.1097593e-33, 3.0771771e-20, 1.3870973e-29,
       5.9499563e-26, 4.8554261e-07, 7.0353786e-20, 1.4308742e-07,
       2.9225679e-24, 9.9999940e-01], dtype=float32)
```

In [21]:

```
np.argmax(predictions[0])
```

Out[21]: 9

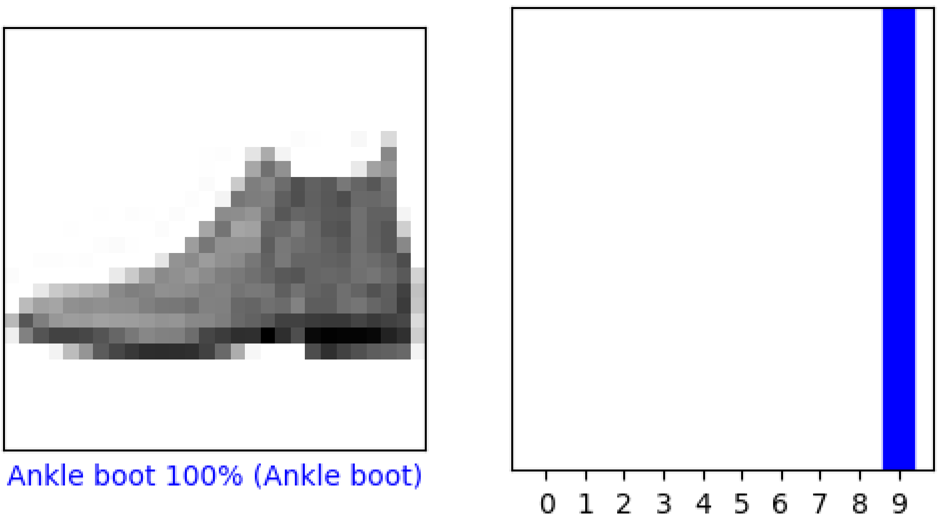
In [22]:

```
test_labels[0]
```

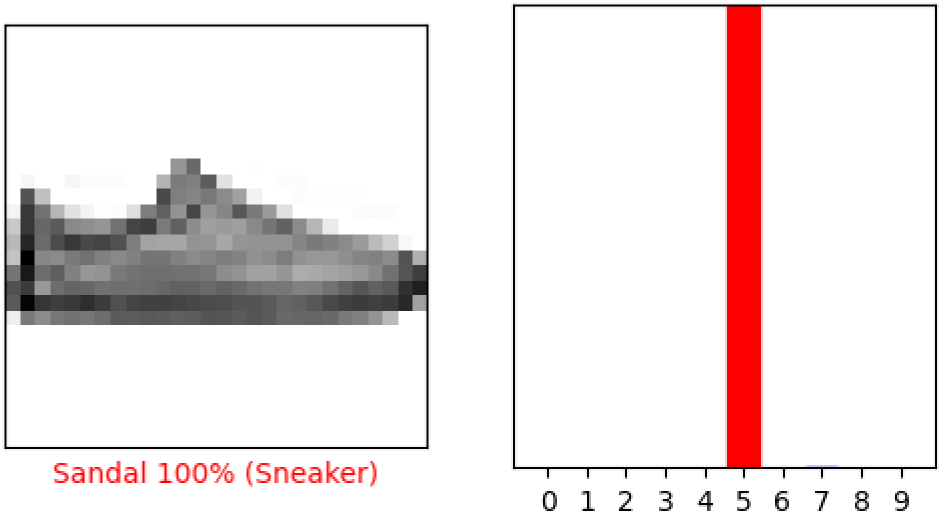
Out[22]: 9

In [23]:
```
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')
```

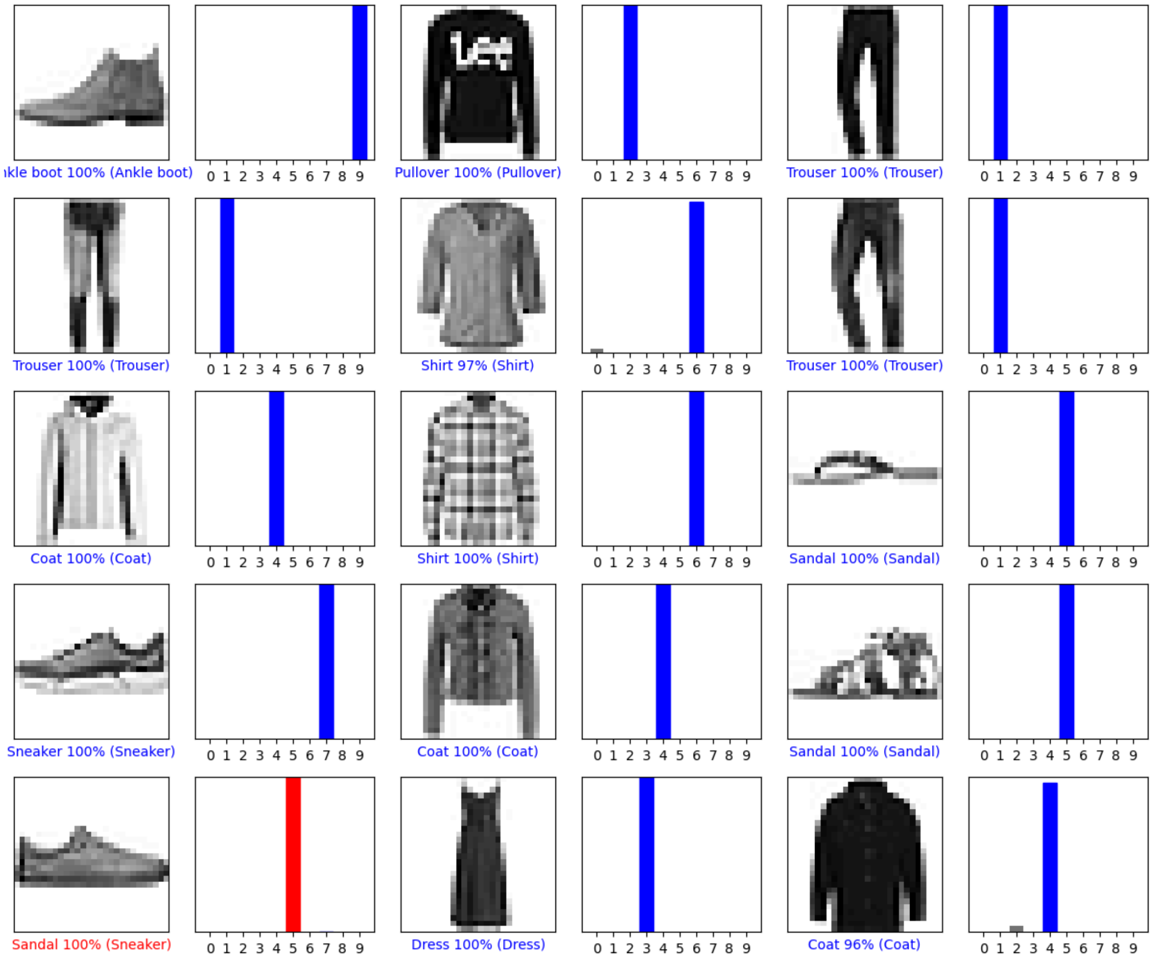
In [24]:
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



In [25]:
```
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 [26]:
```
# 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 [27]:
```
# Grab an image from the test dataset.
img = test_images[1]

print(img.shape)
```

```
(28, 28)
```

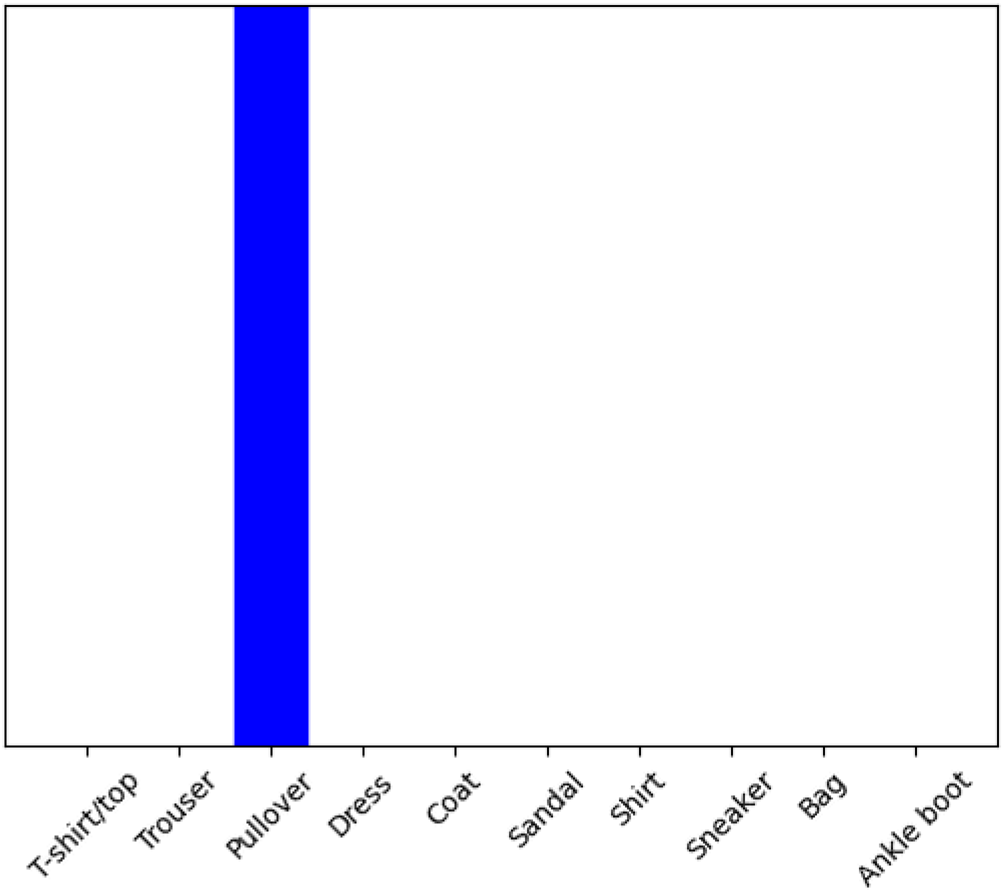
In [28]:
```
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)
```

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

In [29]:
```
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
[[1.8872024e-04 4.2620731e-21 9.9981123e-01 8.5900930e-16 1.0239879e-07
  6.9458333e-32 3.3591638e-09 0.0000000e+00 2.1812813e-24 1.7385899e-33]]
```

In [30]:
```
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```



In [31]:
```
np.argmax(predictions_single[0])
```

Out[31]: 2

In [ ]:
```
```