In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np                                  # Harikannan M (no group,individual project)
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]:
```
(50000, 32, 32, 3)
```

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [4]:
```
y_train[:5]
```

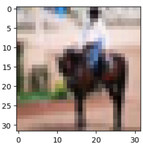
Out[4]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [5]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[5]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [6]:
```
plt.figure(figsize = (16,3))
plt.imshow(X_train[11])
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x242e012d590>
```



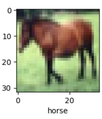
In [7]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
classes[7]
```

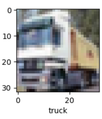
Out[8]:
```
'horse'
```

In [9]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (14,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

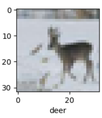
In [10]:
```
plot_sample(X_train, y_train, 7)
```



In [11]:
```
plot_sample(X_train, y_train, 1)
```



In [12]:
```
plot_sample(X_train, y_train, 20)
```



In [13]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [14]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 153s 97ms/step - loss: 1.8125 - accuracy: 0.3519
Epoch 2/5
1563/1563 [==============================] - 155s 99ms/step - loss: 1.6253 - accuracy: 0.4257
Epoch 3/5
1563/1563 [==============================] - 114s 73ms/step - loss: 1.5422 - accuracy: 0.4545
Epoch 4/5
1563/1563 [==============================] - 117s 75ms/step - loss: 1.4828 - accuracy: 0.4789
Epoch 5/5
1563/1563 [==============================] - 116s 74ms/step - loss: 1.4319 - accuracy: 0.4980
```

Out[14]:
```
<keras.src.callbacks.History at 0x242ed475610>
```

In [16]:
```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

In [17]:
```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [18]:
```
cnn.fit(X_train, y_train,epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 45s 28ms/step - loss: 1.4218 - accuracy: 0.4903
Epoch 2/10
1563/1563 [==============================] - 47s 30ms/step - loss: 1.0769 - accuracy: 0.6226
Epoch 3/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.9495 - accuracy: 0.6706
Epoch 4/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.8634 - accuracy: 0.7021
Epoch 5/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.7943 - accuracy: 0.7230
Epoch 6/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.7289 - accuracy: 0.7473
Epoch 7/10
1563/1563 [==============================] - 45s 29ms/step - loss: 0.6706 - accuracy: 0.7676
Epoch 8/10
1563/1563 [==============================] - 50s 32ms/step - loss: 0.6191 - accuracy: 0.7841
Epoch 9/10
1563/1563 [==============================] - 45s 29ms/step - loss: 0.5745 - accuracy: 0.7987
Epoch 10/10
1563/1563 [==============================] - 45s 29ms/step - loss: 0.5271 - accuracy: 0.8149
```

Out[18]:
```
<keras.src.callbacks.History at 0x242ed4cdf50>
```

In [19]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.9300 - accuracy: 0.7031
```

Out[19]:
```
[0.9300476312637329, 0.7031000256538391]
```

In [20]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 3s 9ms/step
```

Out[20]:
```
array([[3.68794659e-04, 6.26935298e-06, 1.10766778e-04, 9.00848866e-01,
        5.05031203e-05, 7.19681755e-03, 6.15275651e-03, 3.98469565e-05,
        8.51434171e-02, 8.20190689e-05],
       [1.04129958e-05, 3.86907649e-03, 1.54653563e-07, 4.48303705e-09,
        2.92635027e-10, 2.69052752e-10, 1.03890195e-11, 2.75074963e-11,
        9.96116638e-01, 3.66536210e-06],
       [1.92961898e-02, 4.09921080e-01, 1.01969221e-04, 6.58256351e-04,
        3.01635264e-05, 1.01558171e-05, 2.19621325e-06, 1.07541518e-05,
        5.68232596e-01, 1.73660577e-03],
       [9.40942407e-01, 1.44250356e-04, 1.13118570e-02, 1.15071700e-04,
        9.86444298e-04, 9.29737354e-08, 4.89417175e-07, 1.21561141e-06,
        4.64951806e-02, 2.94914071e-06],
       [8.69468238e-07, 1.01004622e-03, 1.40884444e-02, 1.70375835e-02,
        9.42941666e-01, 2.85674492e-03, 2.19858158e-02, 1.47268383e-05,
        1.73738663e-05, 4.67078062e-05]], dtype=float32)
```

In [21]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

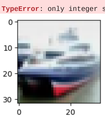
Out[21]:
```
[3, 8, 8, 0, 4]
```

In [22]:
```
y_test[:5]
```

Out[22]:
```
array([[3],
       [8],
       [8],
       [0],
       [6]], dtype=uint8)
```

In [26]:
```
plot_sample(X_test, y_test,1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 plot_sample(X_test, y_test,1)

Cell In[9], line 4, in plot_sample(X, y, index)
      2 plt.figure(figsize = (14,2))
      3 plt.imshow(X[index])
----> 4 plt.xlabel(classes[y[index]])

TypeError: only integer scalar arrays can be converted to a scalar index
```



In [28]:
```
classes[y_classes[1]]
```

Out[28]:
```
'ship'
```

In [31]:
```
plot_sample(X_test, y_test,10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[31], line 1
----> 1 plot_sample(X_test, y_test,10)

Cell In[9], line 4, in plot_sample(X, y, index)
      2 plt.figure(figsize = (14,2))
      3 plt.imshow(X[index])
----> 4 plt.xlabel(classes[y[index]])

TypeError: only integer scalar arrays can be converted to a scalar index
```

In [32]:
```
classes[y_classes[10]]
```

Out[32]: 'airplane'

In [ ]:

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras                                    # Harikannan M (no group,individual project)
```

In [4]:
```
fashion_mnist = keras.datasets.fashion_mnist
```

In [6]:
```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

In [22]:
```
class_names = ['T-shirt', 'Trouser', 'Dress', 'Coat', 'Shirt', 'Sneakers', 'Bag', 'Sandel', 'Ankle boot', 'Pullover']
```

In [23]:
```
train_images.shape
```

Out[23]:
```
(60000, 28, 28)
```

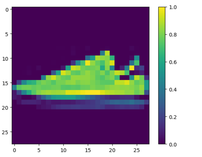
In [24]:
```
len(train_labels)
```

Out[24]:
```
60000
```

In [25]:
```
test_images.shape
```

Out[25]:
```
(10000, 28, 28)
```

In [26]:
```
plt.figure()
plt.imshow(train_images[6])
plt.colorbar()
plt.grid(False)
plt.show()
```



In [27]:
```
train_images = train_images / 255

test_images = test_images / 255
```

In [28]:
```
plt.figure(figsize=(7,7))
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 [32]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

In [34]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [35]:
```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 1.0003 - accuracy: 0.6600
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.6395 - accuracy: 0.7678
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5643 - accuracy: 0.7982
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5222 - accuracy: 0.8157
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4946 - accuracy: 0.8261
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4747 - accuracy: 0.8331
Epoch 7/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.4593 - accuracy: 0.8384
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4471 - accuracy: 0.8427
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4378 - accuracy: 0.8454
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4295 - accuracy: 0.8490
```

Out[35]:
```
<keras.src.callbacks.History at 0x24b66c120d0>
```

In [36]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.4605 - accuracy: 0.8348
Test accuracy: 0.8348000049591064
```

In [37]:
```
predictions = model.predict(test_images)
```

```
313/313 [==============================] - 0s 858us/step
```

In [43]:
```
predictions[0]
```

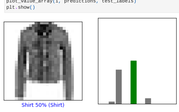
Out[43]:
```
array([5.3714081e-07, 5.1864415e-08, 4.0368827e-06, 2.1542639e-06,
       4.0933084e-06, 1.7572316e-01, 8.9583621e-06, 1.9404456e-01,
       2.8627858e-01, 6.2654984e-01], dtype=float32)
```

In [44]:
```
np.argmax(predictions[0])
```

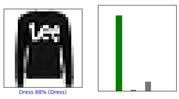
Out[44]:
```
9
```

In [45]:
```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i], 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):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  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('green')
```

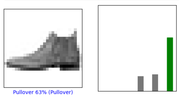
In [66]:
```
i = 10
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



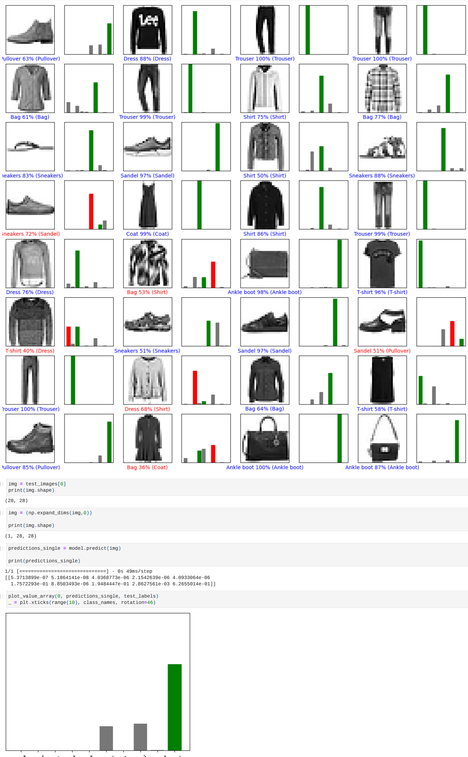
In [69]:
```
i = 7
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



In [72]:
```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



In [79]:
```
num_rows = 8
num_cols = 4
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, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)
plt.show()
```



In [80]:
```
img = test_images[0]
print(img.shape)
```

```
(28, 28)
```

In [84]:
```
img = (np.expand_dims(img,0))

print(img.shape)
```

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

In [85]:
```
predictions_single = model.predict(img)

print(predictions_single)
```

```
1/1 [==============================] - 0s 49ms/step
[[5.3713809e-07 5.1864141e-08 4.0368772e-06 2.1542639e-06 4.0933064e-06
  1.7572293e-01 8.9583493e-06 1.9404447e-01 2.8627561e-03 6.2655014e-01]]
```

In [86]:
```
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```


In [90]:
```
np.argmax(predictions_single[0])
```

Out[86]: 9

In [ ]: