**Evaluate image recognition with confusión matrix and Clasification report**

**from** sklearn.metrics **import** confusion\_matrix , classification\_report

**import** numpy **as** np

y\_pred **=** ann**.**predict(X\_test)

y\_pred\_classes **=** [np**.**argmax(element) **for** element **in** y\_pred]

print("Classification Report: \n", classification\_report(y\_test, y\_pred\_classes))

Classification Report:

precision recall f1-score support

0 0.63 0.45 0.53 1000

1 0.72 0.46 0.56 1000

2 0.33 0.46 0.39 1000

3 0.36 0.25 0.29 1000

4 0.44 0.37 0.40 1000

5 0.34 0.46 0.39 1000

6 0.56 0.47 0.51 1000

7 0.39 0.67 0.50 1000

8 0.64 0.60 0.62 1000

9 0.59 0.53 0.55 1000

accuracy 0.47 10000

macro avg 0.50 0.47 0.47 10000

weighted avg 0.50 0.47 0.47 10000

**Now let us build a convolutional neural network to train our images**

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 [16]:

cnn**.**compile(optimizer**=**'adam',

loss**=**'sparse\_categorical\_crossentropy',

metrics**=**['accuracy'])

In [17]:

cnn**.**fit(X\_train, y\_train, epochs**=**10)

Epoch 1/10

1563/1563 [==============================] - 2s 2ms/step - loss: 1.4407 - accuracy: 0.4810

Epoch 2/10

1563/1563 [==============================] - 2s 2ms/step - loss: 1.1084 - accuracy: 0.6109

Epoch 3/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.9895 - accuracy: 0.6574

Epoch 4/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.9071 - accuracy: 0.6870

Epoch 5/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.8416 - accuracy: 0.7097

Epoch 6/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.7847 - accuracy: 0.7262

Epoch 7/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.7350 - accuracy: 0.7448

Epoch 8/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.6941 - accuracy: 0.7574

Epoch 9/10

1563/1563 [==============================] - 2s 1ms/step - loss: 0.6516 - accuracy: 0.7731

Epoch 10/10

1563/1563 [==============================] - 2s 2ms/step - loss: 0.6187 - accuracy: 0.7836

Out[17]:

**With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features**

cnn**.**evaluate(X\_test,y\_test)

313/313 [==============================] - 0s 1ms/step - loss: 0.9022 - accuracy: 0.7028

[0.9021560549736023, 0.7027999758720398]

In [19]:

y\_pred **=** cnn**.**predict(X\_test)

y\_pred[:5]

Out[19]:

array([[4.3996371e-04, 3.4844263e-05, 1.5558505e-03, 8.8400185e-01,

1.9452239e-04, 3.5314459e-02, 7.2777577e-02, 6.9044131e-06,

5.6417785e-03, 3.2224660e-05],

[8.1062522e-03, 5.0841425e-02, 1.2453231e-07, 5.3348430e-07,

9.1728407e-07, 1.0009186e-08, 2.8985988e-07, 1.7532484e-09,

9.4089705e-01, 1.5346886e-04],

[1.7055811e-02, 1.1841061e-01, 4.6799007e-05, 2.7727904e-02,

1.0848254e-03, 1.0896578e-03, 1.3575243e-04, 2.8652203e-04,

7.8895986e-01, 4.5202184e-02],

[3.1300801e-01, 1.1591638e-02, 1.1511055e-02, 3.9592334e-03,

7.7280165e-03, 5.6289224e-05, 2.3531138e-04, 9.4204297e-06,

6.5178138e-01, 1.1968113e-04],

[1.3230885e-05, 2.1221960e-05, 9.2594400e-02, 3.3585075e-02,

4.4722903e-01, 4.1028224e-03, 4.2241842e-01, 2.8064171e-05,

6.6392668e-06, 1.0745022e-06]], dtype=float32)

In [20]:

y\_classes **=** [np**.**argmax(element) **for** element **in** y\_pred]

y\_classes[:5]

Out[20]:

[3, 8, 8, 8, 4]

In [21]:

y\_test[:5]

Out[21]:

array([3, 8, 8, 0, 6], dtype=uint8)

In [22]:

plot\_sample(X\_test, y\_test,3)

A picture containing text, screen, display, screenshot

Description automatically generated

In [23]:

classes[y\_classes[3]]

Out[23]:

'ship'

In [24]:

classes[y\_classes[3]]

Out[24]:

'ship'