In [3]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
```

In [5]:
```
X_test.shape
```

Out[5]: (10000, 32, 32, 3)

In [6]:
```
X_train.shape
```

Out[6]: (50000, 32, 32, 3)

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

Out[7]: (10000, 1)

In [8]:
```
y_train.shape
```

Out[8]: (50000, 1)

In [9]:
```
y_train[:5]
```

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

In [10]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[10]: array([6, 9, 9, 4, 1], dtype=uint8)

In [11]:
```
y_test = y_test.reshape(-1,)
```

In [12]:
```
classes = ['airplane','automobile','bird','cat','deer', 'dog','frog','horse','ship','truck']
```

In [13]:
```
def plot_sample(X, y, index):
    plt.figure(figsize=(15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [14]:
```
plot_sample(X_train, y_train, 5)
```

In [15]:
```
plot_sample(X_train, y_train, 5)
```

In [16]:
```
plot_sample(X_train, y_train, 501)
```

In [17]:
```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [18]:
```
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)
```

```
E:\Anaconda ML\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)
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 55s 33ms/step - accuracy: 0.3040 - loss: 1.9352
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 55s 35ms/step - accuracy: 0.4225 - loss: 1.6385
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 51s 33ms/step - accuracy: 0.4531 - loss: 1.5545
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 50s 32ms/step - accuracy: 0.4737 - loss: 1.4950
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 30ms/step - accuracy: 0.4953 - loss: 1.4420
```

Out[18]: <keras.src.callbacks.history.History at 0x1e6ed7663d0>

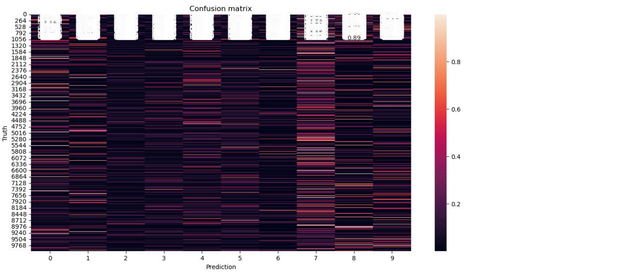
In [19]:
```
from sklearn.metrics import confusion_matrix, classification_report
    import numpy as np
    y_pred = ann.predict(X_test)
    y_pred_classes = [np.argmax(element) for element in y_pred]

    print("classification report: \n", classification_report(y_test, y_pred_classes))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
classification report:
               precision    recall  f1-score   support

           0       0.52      0.60      0.56      1000
           1       0.66      0.53      0.58      1000
           2       0.49      0.12      0.19      1000
           3       0.38      0.19      0.25      1000
           4       0.39      0.45      0.42      1000
           5       0.45      0.32      0.37      1000
           6       0.59      0.38      0.46      1000
           7       0.32      0.79      0.45      1000
           8       0.58      0.64      0.61      1000
           9       0.52      0.60      0.56      1000

    accuracy                           0.46     10000
   macro avg       0.49      0.46      0.45     10000
weighted avg       0.49      0.46      0.45     10000
```

In [20]:
```
import seaborn as sns
```

In [21]:
```
plt.figure(figsize=(14,7))
    sns.heatmap(y_pred, annot = True)
    plt.ylabel('Truth')
    plt.xlabel('Prediction')
    plt.title('Confusion matrix')
    plt.show
```

Out[21]: <function matplotlib.pyplot.show(close=None, block=None)>



In [22]:
```
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'),

    ])
```

```
E:\Anaconda ML\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: 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__(activity_regularizer=activity_regularizer, **kwargs)
```

In [23]:
```
cnn.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
```

In [24]:
```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.3983 - loss: 1.6582
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.6119 - loss: 1.1124
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.6695 - loss: 0.9481
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.7075 - loss: 0.8428
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.7288 - loss: 0.7750
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.7536 - loss: 0.7110
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.7706 - loss: 0.6543
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.7877 - loss: 0.6031
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.8054 - loss: 0.5548
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.8267 - loss: 0.5020
```

Out[24]: <keras.src.callbacks.history.History at 0x1e7049849d0>

In [25]:
```
cnn.evaluate(X_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7152 - loss: 0.9070
```

Out[25]: [0.9251238107681274, 0.7091000080108643]

In [26]:
```
y_pred = cnn.predict(X_test)
    y_pred[:5]
```

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

Out[26]:
```
array([[3.1308160.4e-04, 4.4474894e-05, 5.50113909e-04, 7.99367249e-01,
        8.86134439e-05, 4.0329918e-02, 1.07765093e-01, 6.09774315e-06,
        5.11943251e-02, 3.41743929e-04],
       [5.92875573e-02, 1.165216730e-03, 1.352674644-05, 9.33361566e-09,
        8.25674178e-08, 2.40147768e-09, 1.99592201e-08, 1.20279198e-09,
        9.38898563e-01, 1.47697487e-04],
       [5.96160175e-02, 1.738127102e-01, 2.15590112e-02, 7.67891994e-03,
        1.13394135e-03, 3.51405656e-03, 2.92896270e-03, 2.28178990e-03,
        7.09434986e-01, 1.80395674e-02],
       [9.83872950e-01, 4.68066632e-04, 6.39343983e-04, 1.89955153e-05,
        4.20502515e-05, 7.97551638e-06, 4.19331955e-05, 9.58110468e-06,
        1.47862770e-02, 1.12778129e-04],
       [7.27530676e-07, 4.49761183e-06, 7.50960410e-03, 1.68715734e-02,
        4.27279890e-01, 6.38672639e-04, 5.47628284e-01, 1.12274364e-07,
        8.65199254e-05, 7.82810829e-08]], dtype=float32)
```

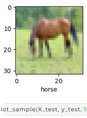
In [27]:
```
y_classes = [np.argmax(element) for element in y_pred]
    y_classes[:5]
```

Out[27]: [3, 8, 8, 0, 6]

In [28]:
```
y_test[:5]
```

Out[28]: array([3, 8, 8, 0, 6], dtype=uint8)

In [29]:
```
plot_sample(X_test, y_test, 60)
```



In [30]:
```
plot_sample(X_test, y_test, 100)
```

In [37]:
```
plot_sample(X_test, y_test, 79)
```

In [38]:
```
classes[y_classes[79]]
```

Out[38]: 'ship'

In [ ]:
```

```