In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
(x_train,y_train),(x_test,y_test)= datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 315s 2us/step
```

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

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

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

Out[4]:
```
(10000, 32, 32, 3)
```

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

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

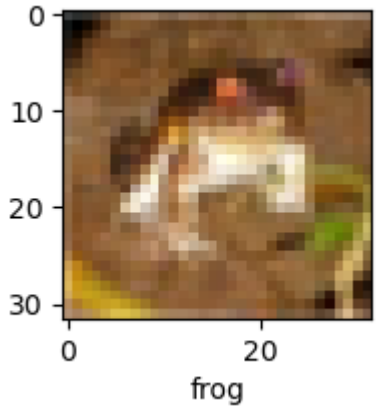
In [6]:
```
y_train = y_train.reshape(-1,)
y_train[:5] #converting the 2D array to 1D array
```

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

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

In [12]:
```
def plot_sample(x,y,index):
    plt.figure(figsize=(15,2))
    plt.imshow(x[index])
    plt.xlabel(classes[y[index]])
```

In [34]:
```
plot_sample(x_train,y_train,0)
```



In [35]:
```
x_train = x_train/255
x_test= x_test/255
```

ANN

In [37]:
```
#building simple ANN
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 [==============================] - 92s 57ms/step - loss: 1.8157 - accuracy: 0.3538
Epoch 2/5
1563/1563 [==============================] - 90s 57ms/step - loss: 1.6238 - accuracy: 0.4250
Epoch 3/5
1563/1563 [==============================] - 89s 57ms/step - loss: 1.5432 - accuracy: 0.4553
Epoch 4/5
1563/1563 [==============================] - 90s 58ms/step - loss: 1.4824 - accuracy: 0.4766
Epoch 5/5
1563/1563 [==============================] - 88s 56ms/step - loss: 1.4305 - accuracy: 0.4965
```

Out[37]:
```
<keras.callbacks.History at 0x2482f354bb0>
```

In [39]:
```
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 [==============================] - 13s 34ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.29      0.42      1000
           1       0.46      0.69      0.56      1000
           2       0.39      0.34      0.36      1000
           3       0.47      0.08      0.14      1000
           4       0.56      0.19      0.29      1000
           5       0.31      0.59      0.41      1000
           6       0.48      0.64      0.54      1000
           7       0.71      0.35      0.47      1000
           8       0.66      0.46      0.54      1000
           9       0.32      0.75      0.45      1000

    accuracy                           0.44     10000
   macro avg       0.51      0.44      0.42     10000
weighted avg       0.51      0.44      0.42     10000
```

convolutional neural network

In [40]:
```
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 [41]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

In [43]:
```
cnn.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 67s 41ms/step - loss: 1.4439 - accuracy: 0.4810
Epoch 2/10
1563/1563 [==============================] - 63s 41ms/step - loss: 1.1092 - accuracy: 0.6104
Epoch 3/10
1563/1563 [==============================] - 52s 34ms/step - loss: 0.9864 - accuracy: 0.6579
Epoch 4/10
1563/1563 [==============================] - 52s 33ms/step - loss: 0.9095 - accuracy: 0.6844
Epoch 5/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.8470 - accuracy: 0.7048
Epoch 6/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.7966 - accuracy: 0.7246
Epoch 7/10
1563/1563 [==============================] - 59s 38ms/step - loss: 0.7545 - accuracy: 0.7372
Epoch 8/10
1563/1563 [==============================] - 56s 36ms/step - loss: 0.7139 - accuracy: 0.7534
Epoch 9/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.6792 - accuracy: 0.7644
Epoch 10/10
1563/1563 [==============================] - 57s 37ms/step - loss: 0.6497 - accuracy: 0.7736
```

Out[43]:
```
<keras.callbacks.History at 0x2487daf3820>
```

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

In [44]:
```
cnn.evaluate(x_test,y_test)
```

Out[44]:
```
313/313 [==============================] - 8s 19ms/step - loss: 0.8947 - accuracy: 0.6952
[0.8946532011032104, 0.6952000260353088]
```

In [46]:
```
y_pred = cnn.predict(x_test)
y_pred[:5]
```

Out[46]:
```
313/313 [==============================] - 6s 18ms/step
array([[8.6582376e-04, 9.6337964e-05, 9.1885403e-03, 7.7531719e-01,
        5.0463285e-03, 1.4090341e-01, 5.7866763e-02, 5.9320498e-04,
        1.0098786e-02, 2.3688361e-05],
       [2.5957930e-03, 3.5851523e-03, 8.3531522e-06, 1.5660592e-07,
        6.6297241e-07, 2.4314772e-07, 1.7936643e-09, 6.5737311e-08,
        9.9366719e-01, 1.4239218e-04],
       [1.5283962e-01, 2.5253248e-01, 1.3910078e-02, 1.1305760e-02,
        6.5608554e-02, 8.4295142e-03, 1.5197422e-03, 1.0968619e-02,
        4.2085728e-01, 6.2028464e-02],
       [7.8053248e-01, 5.0015403e-03, 1.2277883e-02, 2.4265736e-04,
        3.3981010e-02, 5.0957206e-05, 1.4714994e-04, 4.6338231e-04,
        1.6700850e-01, 2.9446735e-04],
       [2.2115216e-06, 3.9828559e-05, 1.2313562e-02, 7.0376739e-02,
        2.1943299e-01, 2.2646736e-03, 6.9538403e-01, 9.6228978e-06,
        1.7106501e-04, 5.2274099e-06]], dtype=float32)
```

In [51]:
```
y_classes=[np.argmax(element)for element in y_pred]
y_classes[:5]
```

Out[51]:
```
[3, 8, 8, 0, 6]
```

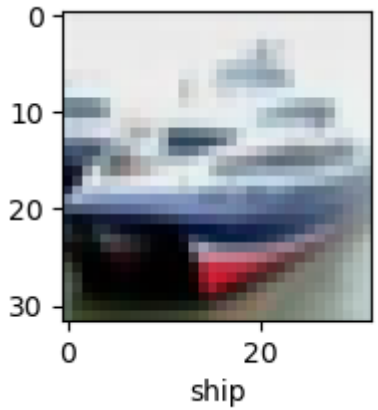
In [49]:
```
y_test
```

Out[49]:
```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

In [50]:
```
y_test=y_test.reshape(-1,)
y_test
```

Out[50]:
```
array([3, 8, 8, ..., 5, 1, 7], dtype=uint8)
```

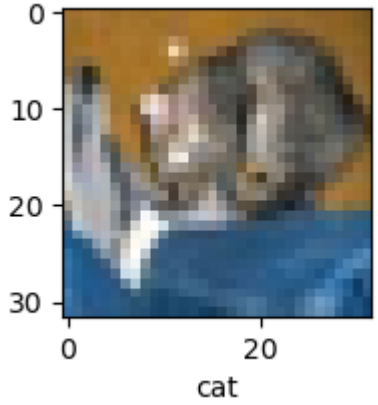
In [52]:
```
plot_sample(x_test,y_test,1)
```



In [54]:
```
classes[y_classes[1]]
```

Out[54]:
```
'ship'
```

In [55]:
```
plot_sample(x_test,y_test,0)
```



In [56]:
```
classes[y_classes[0]]
```

Out[56]:
```
'cat'
```

In [57]:
```
print("Classification report: \n", classification_report(y_test,y_classes))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.72      0.74      0.73      1000
           1       0.85      0.78      0.81      1000
           2       0.57      0.61      0.59      1000
           3       0.52      0.49      0.50      1000
           4       0.60      0.71      0.65      1000
           5       0.64      0.54      0.58      1000
           6       0.67      0.85      0.75      1000
           7       0.81      0.67      0.73      1000
           8       0.80      0.81      0.80      1000
           9       0.81      0.78      0.79      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.70      0.70      0.69     10000
```

In [ ]: