Import Libraries

In [2]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras.utils import to_categorical
from keras.datasets import cifar10
import sys
import matplotlib.pyplot as pyplot
import numpy as np
```

```
Using TensorFlow backend.
```

Import Dataset

In [3]:
```
(trainX, trainY), (testX, testY) = cifar10.load_data()
# one hot encode target values
trainY1=trainY
testY1=testY
trainY = to_categorical(trainY)
testY = to_categorical(testY)
```

Normalise the Data

In [4]:
```
fig, ax = pyplot.subplots(5, 5)
k = 0

for i in range(5):
    for j in range(5):
        ax[i][j].imshow(trainX[k], aspect='auto')
        k += 1

pyplot.show()
```

In [5]:
```
# convert from integers to floats
train_n= trainX.astype('float32')
test_n = testX.astype('float32')
# normalize to range 0-1
train_n = train_n / 255.0
test_n= test_n / 255.0
```

Making a sequential model

In [6]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_unif
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_unif
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_unif
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_unif
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uni
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uni
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))
# compile model
opt = tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy']
```

```
WARNING:tensorflow:From D:\conda\envs\env\lib\site-packages\tensorflow_core\pyt
hon\ops\resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (f
rom tensorflow.python.ops.resource_variable_ops) with constraint is deprecated
and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

Fitting the model

In [7]:
```
history = model.fit(train_n, trainY, epochs=20, batch_size=64, validation_data=(t
```

```
4 - acc: 0.6617 - val_loss: 0.8902 - val_acc: 0.6861
Epoch 15/20
50000/50000 [==============================] - 14s 277us/sample - loss: 0.924
7 - acc: 0.6715 - val_loss: 0.8453 - val_acc: 0.7041
Epoch 16/20
50000/50000 [==============================] - 14s 275us/sample - loss: 0.894
3 - acc: 0.6817 - val_loss: 0.8325 - val_acc: 0.7046
Epoch 17/20
50000/50000 [==============================] - 14s 271us/sample - loss: 0.876
2 - acc: 0.6880 - val_loss: 0.7979 - val_acc: 0.7229
Epoch 18/20
50000/50000 [==============================] - 14s 275us/sample - loss: 0.847
1 - acc: 0.7005 - val_loss: 0.7880 - val_acc: 0.7247
Epoch 19/20
50000/50000 [==============================] - 14s 276us/sample - loss: 0.825
4 - acc: 0.7079 - val_loss: 0.7812 - val_acc: 0.7272
Epoch 20/20
50000/50000 [==============================] - 15s 296us/sample - loss: 0.814
9 - acc: 0.7120 - val_loss: 0.7501 - val_acc: 0.7374
```

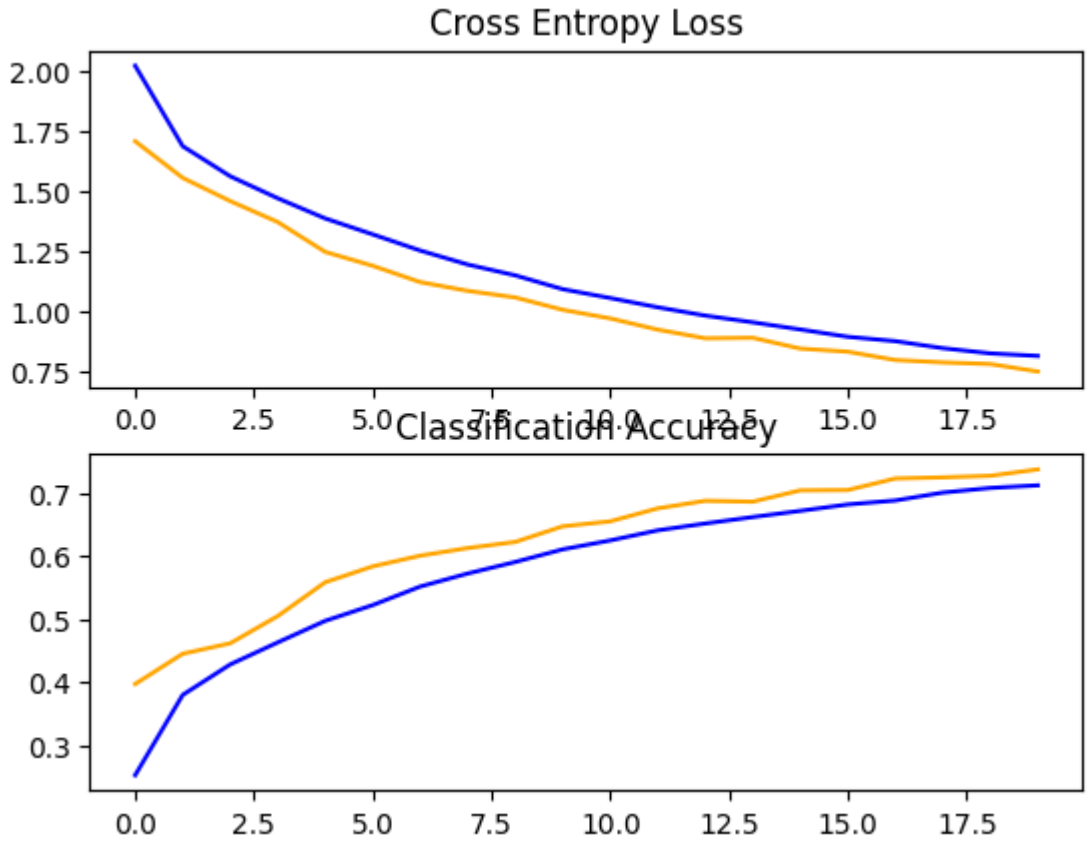
Checking the accuracy

In [8]:
```
_, acc = model.evaluate(test_n, testY, verbose=1)
```

```
10000/10000 [==============================] - 2s 209us/sample - loss: 0.7501 -
acc: 0.7374
```

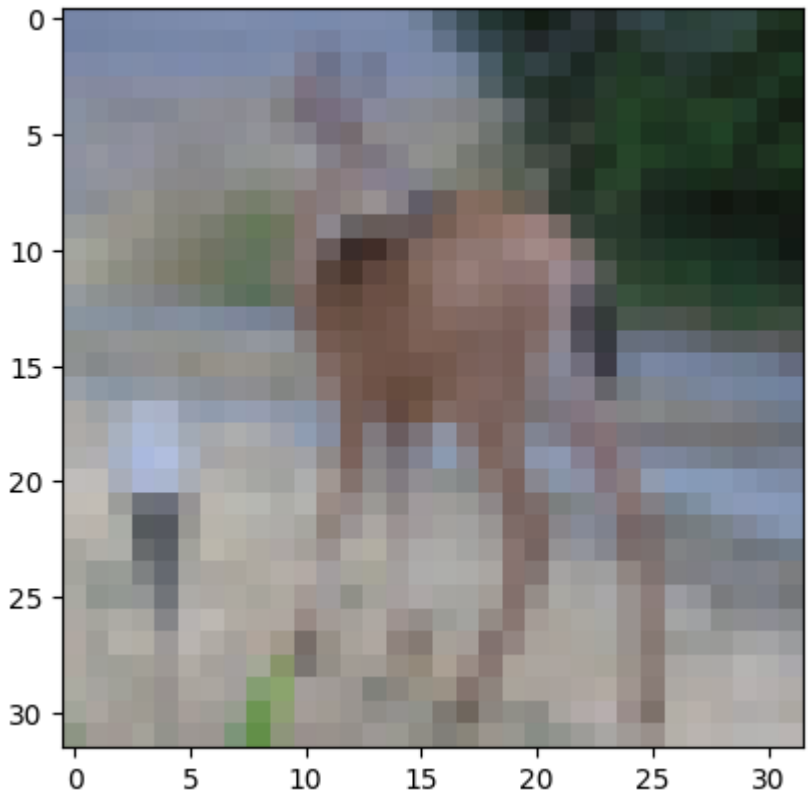
Plotting the accuracy and loss

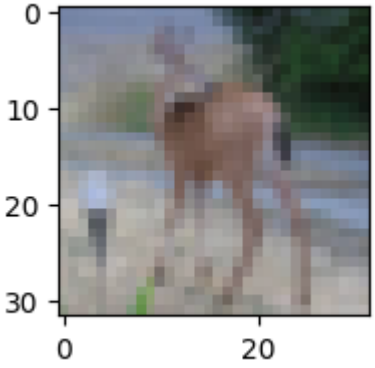
In [9]:
```
# plot loss
pyplot.subplot(211)
pyplot.title('Cross Entropy Loss')
pyplot.plot(history.history['loss'], color='blue', label='train')
pyplot.plot(history.history['val_loss'], color='orange', label='test')
# plot accuracy
pyplot.subplot(212)
pyplot.title('Classification Accuracy')
pyplot.plot(history.history['acc'], color='blue', label='train')
pyplot.plot(history.history['val_acc'], color='orange', label='test')
# save plot to file
filename = sys.argv[0].split('/')[-1]
pyplot.savefig(filename + '_plot.png')
pyplot.show()
```



In [15]:
```
labels = '''airplane automobile bird cat deerdog frog horseship truck'''.split()
trainY1, testY1 = trainY1.flatten(), testY1.flatten()
# select the image from our test dataset
image_number = 100

# display the image
pyplot.imshow(test_n[image_number])

# load the image in an array
n = np.array(test_n[image_number])

# reshape it
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[testY1[image_number]]

# display the result
print("Original label is {} and predicted label is {}".format(
    original_label, predicted_label))
pyplot.figure(figsize=(2,2))
pyplot.imshow(test_n[image_number], aspect='auto')
pyplot.show()
```

```
Original label is deerdog and predicted label is deerdog
```





In [ ]: