IMPORTING THE REQUIRED

In [1]:
```python
import tensorflow as tf
from keras.utils import np_utils
from matplotlib import pyplot as plt
import numpy as np
```

In [2]:
```python
import keras
```

LOADING AND SPLITTING THE DATA

In [3]:
```python
(X_train, y_train), (X_test, y_test) = keras.datasets.cifar10.load_data()
```

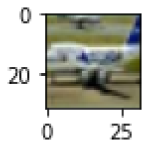
In [4]:
```python
print('Training Images: {}'.format(X_train.shape))
print('Testing Images: {}'.format(X_test.shape))
```

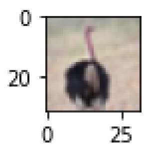
```
Training Images: (50000, 32, 32, 3)
Testing Images: (10000, 32, 32, 3)
```

In [5]:
```python
print(X_train[0].shape)
```

```
(32, 32, 3)
```

In [6]:
```python
for i in range(332,336):
    plt.subplot(120+ 1 + i)
    img = X_train[i]
    plt.imshow(img)
    plt.show()
```





PREPROCESSING THE DATA

In [7]:

```
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')


X_train /= 255
X_test=X_test/255
n_classes = 10


print("Shape before one-hot encoding: ", y_train.shape)
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_test = np_utils.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
```

```
Shape before one-hot encoding:  (50000, 1)
Shape after one-hot encoding:  (50000, 10)
```

BUILDING THE MODEL

In [8]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten



model = Sequential()
#convolutional layers
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu

model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))
# output layer
model.add(Dense(10, activation='softmax'))

# compiling
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# training the model
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test)
```

```
Epoch 1/10
391/391 [==============================] - 238s 609ms/step - loss: 1.6132 - accuracy:
0.4057 - val_loss: 1.1868 - val_accuracy: 0.5725
Epoch 2/10
391/391 [==============================] - 236s 603ms/step - loss: 1.1267 - accuracy:
0.5973 - val_loss: 0.9726 - val_accuracy: 0.6537
Epoch 3/10
391/391 [==============================] - 238s 609ms/step - loss: 0.9458 - accuracy:
0.6693 - val_loss: 0.8126 - val_accuracy: 0.7153
Epoch 4/10
391/391 [==============================] - 224s 572ms/step - loss: 0.8321 - accuracy:
0.7104 - val_loss: 0.7832 - val_accuracy: 0.7344
Epoch 5/10
391/391 [==============================] - 252s 643ms/step - loss: 0.7524 - accuracy:
0.7384 - val_loss: 0.7224 - val_accuracy: 0.7529
Epoch 6/10
391/391 [==============================] - 229s 586ms/step - loss: 0.6854 - accuracy:
0.7598 - val_loss: 0.6891 - val_accuracy: 0.7622
Epoch 7/10
391/391 [==============================] - 217s 556ms/step - loss: 0.6326 - accuracy:
0.7798 - val_loss: 0.6596 - val_accuracy: 0.7685
Epoch 8/10
391/391 [==============================] - 276s 707ms/step - loss: 0.5868 - accuracy:
0.7950 - val_loss: 0.6563 - val_accuracy: 0.7758
Epoch 9/10
391/391 [==============================] - 217s 554ms/step - loss: 0.5397 - accuracy:
0.8101 - val_loss: 0.6478 - val_accuracy: 0.7765
Epoch 10/10
391/391 [==============================] - 218s 558ms/step - loss: 0.5065 - accuracy:
0.8221 - val_loss: 0.6460 - val_accuracy: 0.7871
```

Out[8]: <tensorflow.python.keras.callbacks.History at 0x2650da48340>

PREDICTING

In [9]:
```
classes = range(0,10)

names = ['airplane',
        'automobile',
        'bird',
        'cat',
        'deer',
        'dog',
        'frog',
        'horse',
        'ship',
        'truck']

# zip the names and classes to make a dictionary of class_labels
class_labels = dict(zip(classes, names))

# generate batch of 9 images to predict
batch = X_test[100:109]
labels = np.argmax(Y_test[100:109],axis=-1)

# make predictions
predictions = model.predict(batch, verbose = 1)
```

```
1/1 [==============================] - 0s 999us/step
```

In [10]:
```
print (predictions)
```

```
[[6.62275299e-04 2.08798701e-05 1.73586644e-02 3.20912302e-02
  6.77367747e-01 1.17048189e-01 7.34696339e-04 1.54376432e-01
  1.94164153e-04 1.45675003e-04]
 [4.06096544e-04 1.56691589e-04 3.13600451e-02 5.25065899e-01
  5.34385592e-02 1.22463003e-01 2.39344224e-01 1.75825227e-02
  7.69958540e-04 9.41297878e-03]
 [3.73336320e-06 1.04613673e-06 1.43254618e-03 1.75538324e-02
  1.50167853e-05 1.13022770e-03 9.79834020e-01 2.78436783e-05
  8.20663388e-07 9.03053547e-07]
 [3.42060157e-05 4.57784472e-06 2.12435774e-03 8.76361609e-01
  7.17466697e-03 2.76180394e-02 8.62549841e-02 2.29424055e-04
  4.44794023e-05 1.53614499e-04]
 [6.16439320e-06 9.95885193e-01 5.15927612e-09 5.35385070e-09
  1.87039065e-10 5.68557770e-11 4.49718618e-09 3.64494657e-10
  1.44155987e-04 3.96451680e-03]
 [5.52665824e-07 9.98052359e-01 3.28795302e-08 3.55554050e-07
  4.23005658e-10 5.70011096e-08 4.68998422e-07 1.94774835e-10
  4.60435149e-06 1.94154901e-03]
 [4.61585447e-02 1.23705007e-02 3.77463289e-02 3.59680206e-01
  5.87779470e-03 2.04456538e-01 2.16899171e-01 2.21284642e-03
  1.00540623e-01 1.40575105e-02]
 [7.47478452e-08 5.41297052e-09 1.33436784e-04 1.88222519e-04
  5.14796520e-05 2.47998719e-06 9.99624252e-01 9.42762135e-09
  4.65350913e-09 1.49992090e-08]
 [1.49097177e-04 4.77945694e-04 1.19821789e-06 8.67436313e-07
  6.30787467e-09 1.05506395e-07 1.79891913e-07 1.31341329e-08
  9.99369681e-01 9.62393983e-07]]
```

In [11]:
```python
for image in predictions:
    print(np.sum(image))
```

```
1.0
1.0
1.0
0.99999994
1.0
1.0
1.0000001
0.99999994
1.0000001
```

In [12]:
```python
class_result = np.argmax(predictions,axis=-1)
print (class_result)
```

```
[4 3 6 3 1 1 3 6 8]
```

FINAL OBJECT DETECTION

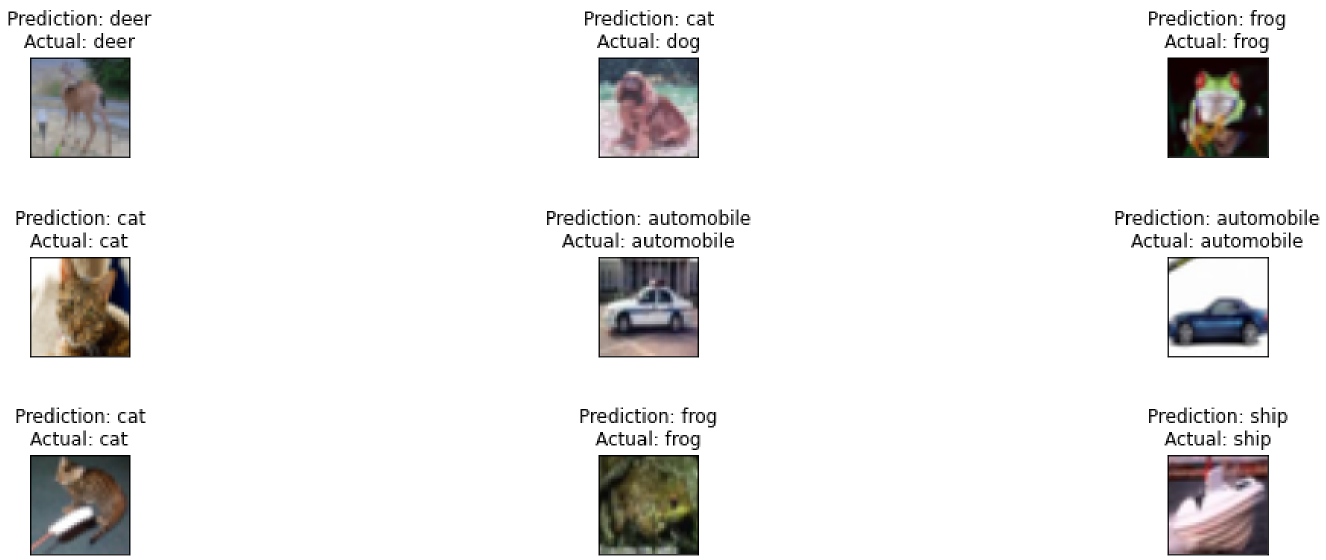
In [13]:
```python

fig, axs = plt.subplots(3, 3, figsize = (19,6))
fig.subplots_adjust(hspace = 1)
axs = axs.flatten()

for i, img in enumerate(batch):
    for key, value in class_labels.items():
        if class_result[i] == key:
            title = 'Prediction: {}\nActual: {}'.format(class_labels[key], class_labels[
            axs[i].set_title(title)
            axs[i].axes.get_xaxis().set_visible(False)
            axs[i].axes.get_yaxis().set_visible(False)

    # plot the image
    axs[i].imshow(img)

# show the plot
plt.show()
```



In [ ]:

In [ ]: