CIFAR-10 Dataset

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

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

In [4]:
```
from keras.utils import np_utils
from keras.models import Sequential
```

In [5]:
```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.callbacks import ModelCheckpoint
```

In [6]:
```
from keras.datasets import cifar10
```

In [7]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data() # Need Time
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 [==============================] - 21s 0us/step
```

In [8]:
```
print(x_train.shape, y_train.shape,x_test.shape, y_test.shape)
```

```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

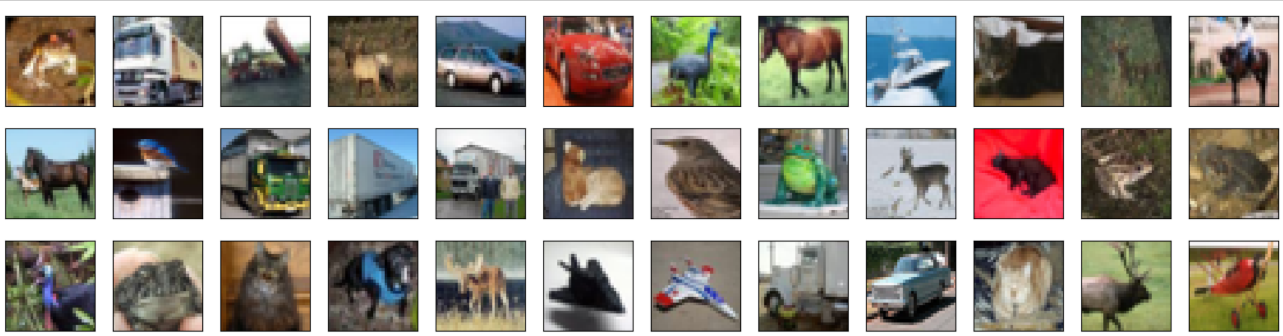
In [9]:
```
cifar10_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                  'dog', 'frog', 'horse', 'ship', 'truck']
```

In [10]:
```
y_train[:5]
```

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

2. Show first 24 Training Images

In [11]:
```
fig = plt.figure(figsize=(20,5))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
```



In [12]:
```
# rescale [0,255] --> [0,1]
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

4. Split Dataset into Training, Testing, and Validation Sets

In [13]:
```
# one-hot encode the labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

In [14]:
```
y_train[:5]
```

Out[14]:
```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

In [15]:
```
# break training set into training and validation sets
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]
```

In [16]:
```
print(x_train.shape, y_train.shape,x_valid.shape, y_valid.shape)
```

```
(45000, 32, 32, 3) (45000, 10) (5000, 32, 32, 3) (5000, 10)
```

```
### 5. Define the Model Architecture

![feature scaling](cnn-schema1.jpg)
```

In [17]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

In [18]:
```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2,
                 padding='same', activation='relu',
                 input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2,
                 padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2,
                 padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

#model.summary()
```

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 16)        208

 max_pooling2d (MaxPooling2D  (None, 16, 16, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 32)        2080

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          8256

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout (Dropout)           (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 500)               512500

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 528,054
Trainable params: 528,054
Non-trainable params: 0
_________________________________________________________________
```

6. Compile the Model

In [20]:
```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

7. Train the Model

In [21]:
```
from keras.callbacks import ModelCheckpoint
```

In [22]:
```
# train the model
checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1, save_best_only=True)

hist = model.fit(x_train, y_train, batch_size=32, epochs=5,
          validation_data=(x_valid, y_valid), callbacks=[checkpointer],
          verbose=2, shuffle=True)
```

```
Epoch 1/5

Epoch 1: val_loss improved from inf to 1.34982, saving model to model.weights.best.hdf5
1407/1407 - 20s - loss: 1.6136 - accuracy: 0.4120 - val_loss: 1.3498 - val_accuracy: 0.5258 - 20s/epoch - 14ms/step
Epoch 2/5

Epoch 2: val_loss improved from 1.34982 to 1.11654, saving model to model.weights.best.hdf5
1407/1407 - 27s - loss: 1.2820 - accuracy: 0.5410 - val_loss: 1.1165 - val_accuracy: 0.6040 - 27s/epoch - 19ms/step
Epoch 3/5

Epoch 3: val_loss improved from 1.11654 to 1.04901, saving model to model.weights.best.hdf5
1407/1407 - 27s - loss: 1.1585 - accuracy: 0.5897 - val_loss: 1.0490 - val_accuracy: 0.6218 - 27s/epoch - 19ms/step
Epoch 4/5

Epoch 4: val_loss did not improve from 1.04901
1407/1407 - 26s - loss: 1.0756 - accuracy: 0.6181 - val_loss: 1.1728 - val_accuracy: 0.5936 - 26s/epoch - 18ms/step
Epoch 5/5

Epoch 5: val_loss improved from 1.04901 to 0.94507, saving model to model.weights.best.hdf5
1407/1407 - 26s - loss: 1.0311 - accuracy: 0.6369 - val_loss: 0.9451 - val_accuracy: 0.6732 - 26s/epoch - 18ms/step
```

8. Load the Model with the Best Validation Accuracy

In [23]:

```
# load the weights that yielded the best validation accuracy
model.load_weights('model.weights.best.hdf5')
```

9. Test Accuracy rate

In [24]:

```
# evaluate and print test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

```
Test accuracy: 0.6646000146865845
```