In [1]:
```
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
```

```
Using TensorFlow backend.
```

In [5]:
```
batch_size = 32
num_classes = 10
epochs = 10
data_augmentation = True
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model.h5'

# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

In [3]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

In [6]:
```
# initialize adam  optimizer
opt = keras.optimizers.adam(lr=0.0001, decay=1e-6)

# Let's train the model using Adam
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the da
taset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        zca_epsilon=1e-06,  # epsilon for ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 t
o 180)
        # randomly shift images horizontally (fraction of total width)
        width_shift_range=0.1,
        # randomly shift images vertically (fraction of total height)
        height_shift_range=0.1,
        shear_range=0.,  # set range for random shear
        zoom_range=0.,  # set range for random zoom
        channel_shift_range=0.,  # set range for random channel shifts
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        cval=0.,  # value used for fill_mode = "constant"
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False,  # randomly flip images
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and
 1)
        validation_split=0.0)

    # Compute quantities required for feature-wise normalization
```

```
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)

    # Fit the model on the batches generated by datagen.flow().
    model.fit_generator(datagen.flow(x_train, y_train,
                                     batch_size=batch_size),
                        epochs=epochs,
                        validation_data=(x_test, y_test),
                        workers=4)
```

```
Using real-time data augmentation.
Epoch 1/10
1563/1563 [==============================] - 1386s 887ms/step - loss: 1.8046
- acc: 0.3312 - val_loss: 1.5402 - val_acc: 0.4480
Epoch 2/10
1563/1563 [==============================] - 1308s 837ms/step - loss: 1.5268
- acc: 0.4421 - val_loss: 1.3277 - val_acc: 0.5256
Epoch 3/10
1563/1563 [==============================] - 1209s 773ms/step - loss: 1.4208
- acc: 0.4843 - val_loss: 1.3159 - val_acc: 0.5288
Epoch 4/10
1563/1563 [==============================] - 1261s 807ms/step - loss: 1.3435
- acc: 0.5181 - val_loss: 1.1970 - val_acc: 0.5703
Epoch 5/10
1563/1563 [==============================] - 1263s 808ms/step - loss: 1.2825
- acc: 0.5400 - val_loss: 1.1581 - val_acc: 0.5885
Epoch 6/10
1563/1563 [==============================] - 1295s 828ms/step - loss: 1.2272
- acc: 0.5651 - val_loss: 1.1002 - val_acc: 0.6056
Epoch 7/10
1563/1563 [==============================] - 1247s 798ms/step - loss: 1.1771
- acc: 0.5823 - val_loss: 1.0436 - val_acc: 0.6302
Epoch 8/10
1563/1563 [==============================] - 1449s 927ms/step - loss: 1.1322
- acc: 0.5985 - val_loss: 1.0214 - val_acc: 0.6376
Epoch 9/10
1563/1563 [==============================] - 1217s 779ms/step - loss: 1.1036
- acc: 0.6096 - val_loss: 1.0035 - val_acc: 0.6468
Epoch 10/10
1563/1563 [==============================] - 1198s 767ms/step - loss: 1.0701
- acc: 0.6200 - val_loss: 1.0250 - val_acc: 0.6365
```

In [7]:
```
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
Saved trained model at /Users/devikasubramanian/Dropbox/devika/Documents/Teac
hing/ml_short_course/code_2019/lab2/MLBootCamp/Python3/lab4/saved_models/kera
s_cifar10_trained_model.h5
10000/10000 [==============================] - 74s 7ms/step
Test loss: 1.0249755365371704
Test accuracy: 0.6365
```

In [ ]:
```

```