Building a CNN to classify images in the CIFAR-10 Dataset

We will work with the CIFAR-10 Dataset. This is a well-known dataset for image classification, which consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The 10 classes are:

0. airplane
1. automobile
2. bird
3. cat
4. deer
5. dog
6. frog
7. horse
8. ship
9. truck

For details about CIFAR-10 see: https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html)

For a compilation of published performance results on CIFAR 10, see: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html (http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)

Building Convolutional Neural Nets

In this exercise we will build and train our first convolutional neural networks. In the first part, we walk through the different layers and how they are configured. In the second part, you will build your own model, train it, and compare the performance.

In [3]:
```
from __future__ import print_function
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 matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [4]:
```
# The data, shuffled and 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')
```

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

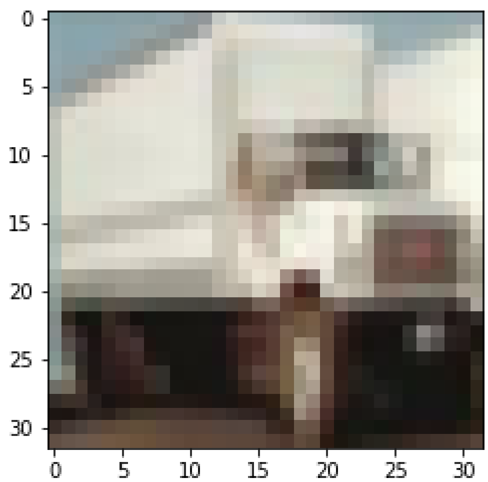
In [5]:
```
## Each image is a 32 x 32 x 3 numpy array
x_train[444].shape
```

Out[5]:
```
(32, 32, 3)
```

In [6]:
```
## Let's look at one of the images

print(y_train[444])
plt.imshow(x_train[444]);
```

```
[9]
```



In [7]:
```
num_classes = 10

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

In [8]:
```
# now instead of classes described by an integer between 0-9 we have a v
ector with a 1 in the (Pythonic) 9th position
y_train[444]
```

Out[8]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

In [9]:
```
# As before, let's make everything float and scale
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Keras Layers for CNNs

- Previously we built Neural Networks using primarily the Dense, Activation and Dropout Layers.
- Here we will describe how to use some of the CNN-specific layers provided by Keras

Conv2D

```
keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)
```

A few parameters explained:

- `filters` : the number of filter used per location. In other words, the depth of the output.
- `kernel_size` : an (x,y) tuple giving the height and width of the kernel to be used
- `strides` : an (x,y) tuple giving the stride in each dimension. Default is `(1,1)`
- `input_shape` : required only for the first layer

Note, the size of the output will be determined by the kernel_size, strides

MaxPooling2D

`keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)`

- `pool_size` : the (x,y) size of the grid to be pooled.
- `strides` : Assumed to be the `pool_size` unless otherwise specified

Flatten

Turns its input into a one-dimensional vector (per instance). Usually used when transitioning between convolutional layers and fully connected layers.

First CNN

Below we will build our first CNN. For demonstration purposes (so that it will train quickly) it is not very deep and has relatively few parameters. We use strides of 2 in the first two convolutional layers which quickly reduces the dimensions of the output. After a MaxPooling layer, we flatten, and then have a single fully connected layer before our final classification layer.

In [10]:
```
# Let's build a CNN using Keras' Sequential capabilities

model_1 = Sequential()


## 5x5 convolution with 2x2 stride and 32 filters
model_1.add(Conv2D(32, (5, 5), strides = (2,2), padding='same',
                 input_shape=x_train.shape[1:]))
model_1.add(Activation('relu'))

## Another 5x5 convolution with 2x2 stride and 32 filters
model_1.add(Conv2D(32, (5, 5), strides = (2,2)))
model_1.add(Activation('relu'))

## 2x2 max pooling reduces to 3 x 3 x 32
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Dropout(0.25))

## Flatten turns 3x3x32 into 288x1
model_1.add(Flatten())
model_1.add(Dense(512))
model_1.add(Activation('relu'))
model_1.add(Dropout(0.5))
model_1.add(Dense(num_classes))
model_1.add(Activation('softmax'))

model_1.summary()
```

```
WARNING:tensorflow:From /anaconda3/envs/cecs551/lib/python3.6/site-pack
ages/tensorflow/python/framework/op_def_library.py:263: colocate_with
(from tensorflow.python.framework.ops) is deprecated and will be remove
d in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/envs/cecs551/lib/python3.6/site-pack
ages/keras/backend/tensorflow_backend.py:3445: calling dropout (from te
nsorflow.python.ops.nn_ops) with keep_prob is deprecated and will be re
moved in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 16, 16, 32)        2432
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 32)          25632
_________________________________________________________________
activation_2 (Activation)    (None, 6, 6, 32)          0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 3, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               147968
_________________________________________________________________
activation_3 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 181,162
Trainable params: 181,162
Non-trainable params: 0
_________________________________________________________________
```

We still have 181K parameters, even though this is a "small" model.

In [11]:
```
batch_size = 32

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0005, decay=1e-6)

# Let's train the model using RMSprop
model_1.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

model_1.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=15,
              validation_data=(x_test, y_test),
              shuffle=True)
```

```
WARNING:tensorflow:From /anaconda3/envs/cecs551/lib/python3.6/site-pack
ages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.
python.ops.math_ops) is deprecated and will be removed in a future vers
ion.
Instructions for updating:
Use tf.cast instead.
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 22s 440us/step - loss:
1.7198 - acc: 0.3726 - val_loss: 1.4148 - val_acc: 0.4861
Epoch 2/15
50000/50000 [==============================] - 21s 429us/step - loss:
1.4487 - acc: 0.4801 - val_loss: 1.2905 - val_acc: 0.5304
Epoch 3/15
50000/50000 [==============================] - 21s 417us/step - loss:
1.3352 - acc: 0.5206 - val_loss: 1.2465 - val_acc: 0.5463
Epoch 4/15
50000/50000 [==============================] - 21s 427us/step - loss:
1.2671 - acc: 0.5476 - val_loss: 1.1455 - val_acc: 0.5948
Epoch 5/15
50000/50000 [==============================] - 23s 454us/step - loss:
1.2177 - acc: 0.5674 - val_loss: 1.1780 - val_acc: 0.5817
Epoch 6/15
50000/50000 [==============================] - 21s 423us/step - loss:
1.1856 - acc: 0.5804 - val_loss: 1.4959 - val_acc: 0.5037
Epoch 7/15
50000/50000 [==============================] - 21s 411us/step - loss:
1.1474 - acc: 0.5967 - val_loss: 1.0747 - val_acc: 0.6270
Epoch 8/15
50000/50000 [==============================] - 23s 460us/step - loss:
1.1278 - acc: 0.6024 - val_loss: 1.0719 - val_acc: 0.6252
Epoch 9/15
50000/50000 [==============================] - 21s 428us/step - loss:
1.1095 - acc: 0.6123 - val_loss: 1.1115 - val_acc: 0.6179
Epoch 10/15
50000/50000 [==============================] - 22s 436us/step - loss:
1.0993 - acc: 0.6163 - val_loss: 1.0177 - val_acc: 0.6425
Epoch 11/15
50000/50000 [==============================] - 22s 433us/step - loss:
1.0831 - acc: 0.6243 - val_loss: 1.1835 - val_acc: 0.5964
Epoch 12/15
50000/50000 [==============================] - 21s 411us/step - loss:
1.0734 - acc: 0.6291 - val_loss: 1.0251 - val_acc: 0.6429
Epoch 13/15
50000/50000 [==============================] - 20s 406us/step - loss:
1.0679 - acc: 0.6290 - val_loss: 1.0583 - val_acc: 0.6290
Epoch 14/15
50000/50000 [==============================] - 19s 387us/step - loss:
1.0672 - acc: 0.6315 - val_loss: 1.0903 - val_acc: 0.6276
Epoch 15/15
50000/50000 [==============================] - 19s 383us/step - loss:
1.0672 - acc: 0.6344 - val_loss: 1.0234 - val_acc: 0.6450
```

Out[11]:
```
<keras.callbacks.History at 0xb35e2de48>
```

Exercise

Our previous model had the structure:

Conv -> Conv -> MaxPool -> (Flatten) -> Dense -> Final Classification

(with appropriate activation functions and dropouts)

1. Build a more complicated model with the following pattern: - Conv -> Conv -> MaxPool -> Conv -> Conv -> MaxPool -> (Flatten) -> Dense -> Final Classification (Use strides of 1 for all convolutional layers.)

1. How many parameters does your model have? How does that compare to the previous model?

1. Train it for 5 epochs. What do you notice about the training time, loss and accuracy numbers (on both the training and validation sets)?

In [16]:
```
# Please provide your solution here
# Create model_2 as mentioned in the exercise
model_2 = Sequential()

## Conv -> Conv -> MaxPool -> Conv -> Conv -> MaxPool -> (Flatten) -> De
nse -> Final Classification
model_2.add(Conv2D(32, (5,5), strides = (1,1), padding='same', input_sha
pe=x_train.shape[1:]))
model_2.add(Activation('relu'))
model_2.add(Conv2D(32, (5,5), strides = (1,1)))
model_2.add(Activation('relu'))

model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.5))

model_2.add(Conv2D(32, (3,3), strides = (1,1)))
model_2.add(Activation('relu'))
model_2.add(Conv2D(16, (3,3), strides = (1,1)))
model_2.add(Activation('relu'))

model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.5))

## Flatten turns 3x3x32 into 288x1
model_2.add(Flatten())
model_2.add(Dense(512))
model_2.add(Activation('relu'))
model_2.add(Dropout(0.5))
model_2.add(Dense(num_classes))
model_2.add(Activation('softmax'))

model_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_11 (Conv2D)           (None, 32, 32, 32)        2432      
_________________________________________________________________
activation_17 (Activation)   (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 28, 28, 32)        25632     
_________________________________________________________________
activation_18 (Activation)   (None, 28, 28, 32)        0         
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 12, 12, 32)        9248      
_________________________________________________________________
activation_19 (Activation)   (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 10, 10, 16)        4624      
_________________________________________________________________
activation_20 (Activation)   (None, 10, 10, 16)        0         
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 16)          0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 512)               205312    
_________________________________________________________________
activation_21 (Activation)   (None, 512)               0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 10)                5130      
_________________________________________________________________
activation_22 (Activation)   (None, 10)                0         
=================================================================
Total params: 252,378
Trainable params: 252,378
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
batch_size = 32

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Training the model using RMSProp and for 5 epochs
model_2.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

model_2.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=5,
              validation_data=(x_test, y_test),
              shuffle=True)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/5
50000/50000 [==============================] - 199s 4ms/step - loss: 2.
0253 - acc: 0.2327 - val_loss: 1.7483 - val_acc: 0.3580
Epoch 2/5
50000/50000 [==============================] - 192s 4ms/step - loss: 1.
7265 - acc: 0.3528 - val_loss: 1.6286 - val_acc: 0.4019
Epoch 3/5
50000/50000 [==============================] - 207s 4ms/step - loss: 1.
6110 - acc: 0.4030 - val_loss: 1.5099 - val_acc: 0.4497
Epoch 4/5
50000/50000 [==============================] - 193s 4ms/step - loss: 1.
5437 - acc: 0.4320 - val_loss: 1.4623 - val_acc: 0.4670
Epoch 5/5
50000/50000 [==============================] - 198s 4ms/step - loss: 1.
4848 - acc: 0.4530 - val_loss: 1.3440 - val_acc: 0.5064
```

Out[17]:
```
<keras.callbacks.History at 0xb323b8e10>
```

My answers:

The new model has **252,378** parameters, as compared to the **181,162** parameters in the previous model.

Also, after trying several combinations of the filters and convolutional layers, model_2 yields an accuracy of **50.64%**. Also, the training as well as testing loss and accuracy show that both the models learn but simply adding more layers does not result in an increase in the accuracy, since with a deeper network, we would need more traing time to coverge the model.