Convolutional Neural Network (CNN)

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

For a compilation of published performance results on CIFAR 10, see: 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 convolutional neural networks. Lets go through the different layers and how they are configured.

At some point we will have to use that dense connected layer, similar to that fully connected layer.

We can also use dropout in order to regularize and ensure that it doesn't over fit.

We'll have our different activation layers that we can use as well. Whether it's ReLu or sigmoid or hyperbolic, whatever it is.

Then we also have this flattened layer. This flattened layer will be important as we move from our convolutional layers to our dense layers. Eventually, in order to make some type of prediction, we're going to need to flatten it out and then have that dense layer connected to that final prediction.

We're also going to import this Conv2D and this MaxPooling2D, which will allow us to build out our convolutional layers as well as our pooling layers

In [1]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
```

In [3]:
```
# 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 [4]:
```
## Each image is a 32 x 32 x 3 numpy array
x_train[444].shape
```

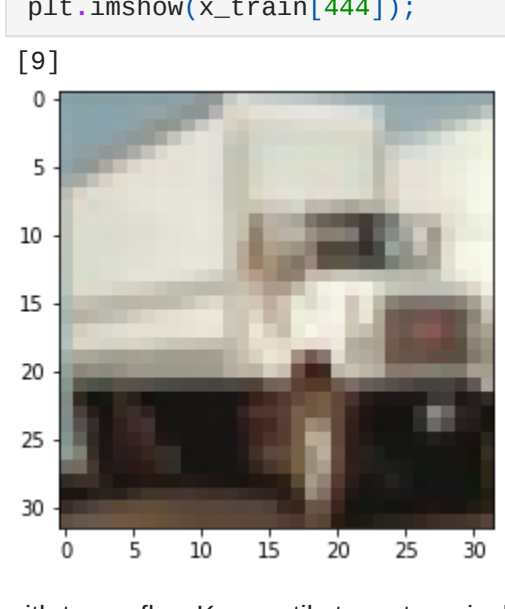
Out[4]:
```
(32, 32, 3)
```

Those numbers are from 0-255 for each one of these different colors, red, green, and blue.

In [5]:
```
## Let's look at one of the images

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

```
[9]
```



with tensorflow.Keras.utils.to_categorical we say what the set is that we want to change to categorical and the number of classes in that set, which is going to be equal to 10.

In [6]:
```
from tensorflow.keras import utils as np_utils

num_classes = 10
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

In [8]:
```
# now instead of classes described by an integer between 0-9 we have a vector 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)
```

Now if we look at that y_train that we had above, which we see was 9, the new value is going to be that one-hot encoded version with 0's everywhere except in the 9th place.

In [9]:
```
# As before, let's make all values  float and scaleed between 0 amd 1
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

all of our different pixels will be values between 0 and 255. If we divide by 255, we ensure that all of our values are going to be between 0 and 1.

In [10]:
```
x_train
```

Out[10]:
```
array([[[[0.23137255, 0.24313726, 0.24705882],
         [0.16862746, 0.18039216, 0.1764706 ],
         [0.19607843, 0.1882353 , 0.16862746],
         ...,
         [0.61960787, 0.5176471 , 0.42352942],
         [0.59607846, 0.49019608, 0.4       ],
         [0.5803922 , 0.4862745 , 0.40392157]],

        [[0.0627451 , 0.07843138, 0.07843138],
         [0.        , 0.        , 0.        ],
         [0.07058824, 0.03137255, 0.        ],
         ...,
         [0.48235294, 0.34509805, 0.21568628],
         [0.46666667, 0.3254902 , 0.19607843],
         [0.47843137, 0.34117648, 0.22352941]],

        [[0.09803922, 0.09411765, 0.08235294],
         [0.0627451 , 0.02745098, 0.        ],
         [0.19215687, 0.10588235, 0.03137255],
         ...,
         [0.4627451 , 0.32941177, 0.19607843],
         [0.47058824, 0.32941177, 0.19607843],
         [0.42745098, 0.28627452, 0.16470589]],

        ...,

        [[0.8156863 , 0.6666667 , 0.3764706 ],
         [0.7882353 , 0.6       , 0.13333334],
         [0.7764706 , 0.6313726 , 0.10196079],
         ...,
         [0.627451  , 0.52156866, 0.27450982],
         [0.21960784, 0.12156863, 0.02745098],
         [0.20784314, 0.13333334, 0.07843138]],

        [[0.7058824 , 0.54509807, 0.3764706 ],
         [0.6784314 , 0.48235294, 0.16470589],
         [0.7294118 , 0.5647059 , 0.11764706],
         ...,
         [0.72156864, 0.5803922 , 0.36862746],
         [0.38039216, 0.24313726, 0.13333334],
         [0.3254902 , 0.20784314, 0.13333334]],

        [[0.69411767, 0.5647059 , 0.45490196],
         [0.65882355, 0.5058824 , 0.36862746],
         [0.7019608 , 0.5568628 , 0.34117648],
         ...,
         [0.84705883, 0.72156864, 0.54901963],
         [0.5921569 , 0.4627451 , 0.32941177],
         [0.48235294, 0.36078432, 0.28235295]]],


       [[[0.6039216 , 0.69411767, 0.73333335],
         [0.49411765, 0.5372549 , 0.53333336],
         [0.4117647 , 0.40784314, 0.37254903],
         ...,
         [0.35686275, 0.37254903, 0.2784314 ],
         [0.34117648, 0.3529412 , 0.2784314 ],
         [0.30980393, 0.31764707, 0.27450982]],

        [[0.54901963, 0.627451  , 0.6627451 ],
         [0.5686275 , 0.6       , 0.6039216 ],
         [0.49019608, 0.49019608, 0.4627451 ],
         ...,
         [0.3764706 , 0.3882353 , 0.30588236],
         [0.3019608 , 0.3137255 , 0.24313726],
         [0.2784314 , 0.28627452, 0.23921569]],

        [[0.54901963, 0.60784316, 0.6431373 ],
         [0.54509807, 0.57254905, 0.58431375],
         [0.4509804 , 0.4509804 , 0.4392157 ],
         ...,
         [0.30980393, 0.32156864, 0.2509804 ],
         [0.26666668, 0.27450982, 0.21568628],
         [0.2627451 , 0.27058825, 0.21568628]],

        ...,

        [[0.6862745 , 0.654902  , 0.6509804 ],
         [0.6117647 , 0.6039216 , 0.627451  ],
         [0.6039216 , 0.627451  , 0.6666667 ],
         ...,
         [0.16470589, 0.13333334, 0.14117648],
         [0.23921569, 0.20784314, 0.22352941],
         [0.3647059 , 0.3254902 , 0.35686275]],

        [[0.64705884, 0.6039216 , 0.5019608 ],
         [0.6117647 , 0.59607846, 0.50980395],
         [0.62352943, 0.6313726 , 0.5568628 ],
         ...,
         [0.40392157, 0.3647059 , 0.3764706 ],
         [0.48235294, 0.44705883, 0.47058824],
         [0.5137255 , 0.4745098 , 0.5137255 ]],

        [[0.6392157 , 0.5803922 , 0.47058824],
         [0.61960787, 0.5803922 , 0.47843137],
         [0.6392157 , 0.6117647 , 0.52156866],
         ...,
         [0.56078434, 0.52156866, 0.54509807],
         [0.56078434, 0.5254902 , 0.5568628 ],
         [0.56078434, 0.52156866, 0.5647059 ]]],


       [[[1.        , 1.        , 1.        ],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         ...,
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686],
         [0.99215686, 0.99215686, 0.99215686]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [0.99607843, 0.99607843, 0.99607843],
         [0.99607843, 0.99607843, 0.99607843],
         ...,
         [0.99607843, 0.99607843, 0.99607843],
         [0.99607843, 0.99607843, 0.99607843],
         [0.99607843, 0.99607843, 0.99607843]],

        ...,

        [[0.44313726, 0.47058824, 0.4392157 ],
         [0.43529412, 0.4627451 , 0.43529412],
         [0.4117647 , 0.4392157 , 0.41568628],
         ...,
         [0.28235295, 0.31764707, 0.3137255 ],
         [0.28235295, 0.3137255 , 0.30980393],
         [0.28235295, 0.3137255 , 0.30980393]],

        [[0.43529412, 0.4627451 , 0.43137255],
         [0.40784314, 0.43529412, 0.40784314],
         [0.3882353 , 0.41568628, 0.38431373],
         ...,
         [0.26666668, 0.29411766, 0.28627452],
         [0.27450982, 0.29803923, 0.29411766],
         [0.30588236, 0.32941177, 0.32156864]],

        [[0.41568628, 0.44313726, 0.4117647 ],
         [0.3882353 , 0.41568628, 0.38431373],
         [0.37254903, 0.4       , 0.36862746],
         ...,
         [0.30588236, 0.33333334, 0.3254902 ],
         [0.30980393, 0.33333334, 0.3254902 ],
         [0.3137255 , 0.3372549 , 0.32941177]]],


       ...,


       [[[0.13725491, 0.69803923, 0.92156863],
         [0.15686275, 0.6901961 , 0.9372549 ],
         [0.16470589, 0.6901961 , 0.94509804],
         ...,
         [0.3882353 , 0.69411767, 0.85882354],
         [0.30980393, 0.5764706 , 0.77254903],
         [0.34901962, 0.5803922 , 0.7411765 ]],

        [[0.22352941, 0.7137255 , 0.91764706],
         [0.17254902, 0.72156864, 0.98039216],
         [0.19607843, 0.7176471 , 0.9411765 ],
         ...,
         [0.6117647 , 0.7137255 , 0.78431374],
         [0.5529412 , 0.69411767, 0.80784315],
         [0.45490196, 0.58431375, 0.6862745 ]],

        [[0.38431373, 0.77254903, 0.92941177],
         [0.2509804 , 0.7411765 , 0.9882353 ],
         [0.27058825, 0.7529412 , 0.9607843 ],
         ...,
         [0.7372549 , 0.7647059 , 0.80784315],
         [0.46666667, 0.5294118 , 0.5764706 ],
         [0.23921569, 0.30980393, 0.3529412 ]],

        ...,

        [[0.28627452, 0.30980393, 0.3019608 ],
         [0.20784314, 0.24705882, 0.26666668],
         [0.21176471, 0.26666668, 0.3137255 ],
         ...,
         [0.06666667, 0.15686275, 0.2509804 ],
         [0.08235294, 0.14117648, 0.2       ],
         [0.12941177, 0.1882353 , 0.19215687]],

        [[0.23921569, 0.26666668, 0.29411766],
         [0.21568628, 0.27450982, 0.3372549 ],
         [0.22352941, 0.30980393, 0.40392157],
         ...,
         [0.09411765, 0.1882353 , 0.28235295],
         [0.06666667, 0.13725491, 0.20784314],
         [0.02745098, 0.09019608, 0.1254902 ]],

        [[0.17254902, 0.21960784, 0.28627452],
         [0.18039216, 0.25882354, 0.34509805],
         [0.19215687, 0.3019608 , 0.4117647 ],
         ...,
         [0.10588235, 0.20392157, 0.3019608 ],
         [0.08235294, 0.16862746, 0.25882354],
         [0.04705882, 0.12156863, 0.19607843]]],


       [[[0.7411765 , 0.827451  , 0.9411765 ],
         [0.7294118 , 0.8156863 , 0.9254902 ],
         [0.7254902 , 0.8117647 , 0.92156863],
         ...,
         [0.6862745 , 0.7647059 , 0.8784314 ],
         [0.6745098 , 0.7607843 , 0.87058824],
         [0.6627451 , 0.7607843 , 0.8627451 ]],

        [[0.7607843 , 0.8235294 , 0.9372549 ],
         [0.7490196 , 0.8117647 , 0.9254902 ],
         [0.74509805, 0.80784315, 0.92156863],
         ...,
         [0.6784314 , 0.7529412 , 0.8627451 ],
         [0.67058825, 0.7490196 , 0.85490197],
         [0.654902  , 0.74509805, 0.84705883]],

        [[0.8156863 , 0.85882354, 0.95686275],
         [0.8039216 , 0.84705883, 0.9411765 ],
         [0.8       , 0.84313726, 0.9372549 ],
         ...,
         [0.6862745 , 0.7490196 , 0.8509804 ],
         [0.6745098 , 0.74509805, 0.84705883],
         [0.6627451 , 0.7490196 , 0.84313726]],

        ...,

        [[0.8117647 , 0.78039217, 0.70980394],
         [0.79607844, 0.7647059 , 0.6862745 ],
         [0.79607844, 0.76862746, 0.6784314 ],
         ...,
         [0.5294118 , 0.5176471 , 0.49803922],
         [0.63529414, 0.61960787, 0.5882353 ],
         [0.65882355, 0.6392157 , 0.5921569 ]],

        [[0.7764706 , 0.74509805, 0.6666667 ],
         [0.7411765 , 0.70980394, 0.62352943],
         [0.7058824 , 0.6745098 , 0.5764706 ],
         ...,
         [0.69803923, 0.67058825, 0.627451  ],
         [0.6862745 , 0.6627451 , 0.6117647 ],
         [0.6862745 , 0.6627451 , 0.6039216 ]],

        [[0.7764706 , 0.7411765 , 0.6784314 ],
         [0.7411765 , 0.70980394, 0.63529414],
         [0.69803923, 0.6666667 , 0.58431375],
         ...,
         [0.7647059 , 0.72156864, 0.6627451 ],
         [0.76862746, 0.7411765 , 0.67058825],
         [0.7647059 , 0.74509805, 0.67058825]]],


       [[[0.8980392 , 0.8980392 , 0.9372549 ],
         [0.9254902 , 0.92941177, 0.96862745],
         [0.91764706, 0.9254902 , 0.96862745],
         ...,
         [0.8509804 , 0.85882354, 0.9137255 ],
         [0.8666667 , 0.8745098 , 0.91764706],
         [0.87058824, 0.8745098 , 0.9137255 ]],

        [[0.87058824, 0.8666667 , 0.8980392 ],
         [0.9372549 , 0.9372549 , 0.9764706 ],
         [0.9137255 , 0.91764706, 0.9647059 ],
         ...,
         [0.8745098 , 0.8745098 , 0.9254902 ],
         [0.8901961 , 0.89411765, 0.93333334],
         [0.8235294 , 0.827451  , 0.8627451 ]],

        [[0.8352941 , 0.80784315, 0.827451  ],
         [0.91764706, 0.9098039 , 0.9372549 ],
         [0.90588236, 0.9137255 , 0.95686275],
         ...,
         [0.8627451 , 0.8627451 , 0.9098039 ],
         [0.8627451 , 0.85882354, 0.9098039 ],
         [0.7921569 , 0.79607844, 0.84313726]],

        ...,

        [[0.5882353 , 0.56078434, 0.5294118 ],
         [0.54901963, 0.5294118 , 0.49803922],
         [0.5176471 , 0.49803922, 0.47058824],
         ...,
         [0.8784314 , 0.87058824, 0.85490197],
         [0.9019608 , 0.89411765, 0.88235295],
         [0.94509804, 0.94509804, 0.93333334]],

        [[0.5372549 , 0.5176471 , 0.49411765],
         [0.50980395, 0.49803922, 0.47058824],
         [0.49019608, 0.4745098 , 0.4509804 ],
         ...,
         [0.70980394, 0.7058824 , 0.69803923],
         [0.7921569 , 0.7882353 , 0.7764706 ],
         [0.83137256, 0.827451  , 0.8117647 ]],

        [[0.47843137, 0.46666667, 0.44705883],
         [0.4627451 , 0.45490196, 0.43137255],
         [0.47058824, 0.45490196, 0.43529412],
         ...,
         [0.7019608 , 0.69411767, 0.6784314 ],
         [0.6431373 , 0.6431373 , 0.63529414],
         [0.6392157 , 0.6392157 , 0.6313726 ]]]], dtype=float32)
```

In [11]:
```
y_train
```

Out[11]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

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`: and (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 [13]:
```
# 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)) # Dropout for regularization

## Flatten turns 3x3x32 into 288x1
model_1.add(Flatten()) #flatten that out so that now we're working with just a 1D object rather than that 3D object
model_1.add(Dense(512)) # Dense layer so that we have one fully connected layer
model_1.add(Activation('relu')) # relu to have a non-linearity,
model_1.add(Dropout(0.5)) # Dropout for some extra regularization.
model_1.add(Dense(num_classes))
model_1.add(Activation('softmax'))

model_1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 16, 16, 32)        2432

 activation_4 (Activation)   (None, 16, 16, 32)        0

 conv2d_3 (Conv2D)           (None, 6, 6, 32)          25632

 activation_5 (Activation)   (None, 6, 6, 32)          0

 max_pooling2d_1 (MaxPooling  (None, 3, 3, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 3, 3, 32)          0

 flatten_1 (Flatten)         (None, 288)               0

 dense_2 (Dense)             (None, 512)               147968

 activation_6 (Activation)   (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

 activation_7 (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 [14]:
```
from tensorflow.keras import optimizers
batch_size = 32

# initiate RMSprop optimizer
opt = tf.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)
```

```
C:\Users\rsnen\anaconda3\lib\site-packages\keras\optimizer_v2\rmsprop.py:130: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
Epoch 1/15
1563/1563 [==============================] - 41s 25ms/step - loss: 1.7379 - accuracy: 0.3668 - val_loss: 1.5120 - val_accuracy: 0.4531
Epoch 2/15
1563/1563 [==============================] - 39s 25ms/step - loss: 1.4546 - accuracy: 0.4748 - val_loss: 1.2856 - val_accuracy: 0.5399
Epoch 3/15
1563/1563 [==============================] - 39s 25ms/step - loss: 1.3474 - accuracy: 0.5171 - val_loss: 1.2275 - val_accuracy: 0.5699
Epoch 4/15
1563/1563 [==============================] - 39s 25ms/step - loss: 1.2834 - accuracy: 0.5424 - val_loss: 1.2196 - val_accuracy: 0.5627
Epoch 5/15
1563/1563 [==============================] - 39s 25ms/step - loss: 1.2366 - accuracy: 0.5596 - val_loss: 1.2192 - val_accuracy: 0.5643
Epoch 6/15
1563/1563 [==============================] - 36s 23ms/step - loss: 1.2014 - accuracy: 0.5763 - val_loss: 1.1562 - val_accuracy: 0.5883
Epoch 7/15
1563/1563 [==============================] - 36s 23ms/step - loss: 1.1764 - accuracy: 0.5857 - val_loss: 1.0980 - val_accuracy: 0.6141
Epoch 8/15
1563/1563 [==============================] - 38s 24ms/step - loss: 1.1629 - accuracy: 0.5910 - val_loss: 1.1510 - val_accuracy: 0.6013
Epoch 9/15
1563/1563 [==============================] - 38s 24ms/step - loss: 1.1413 - accuracy: 0.5985 - val_loss: 1.0826 - val_accuracy: 0.6183
Epoch 10/15
1563/1563 [==============================] - 39s 25ms/step - loss: 1.1265 - accuracy: 0.6070 - val_loss: 1.1013 - val_accuracy: 0.6075
Epoch 11/15
1563/1563 [==============================] - 38s 24ms/step - loss: 1.1155 - accuracy: 0.6117 - val_loss: 1.1162 - val_accuracy: 0.6099
Epoch 12/15
1563/1563 [==============================] - 38s 24ms/step - loss: 1.1126 - accuracy: 0.6169 - val_loss: 1.0520 - val_accuracy: 0.6341
Epoch 13/15
1563/1563 [==============================] - 38s 25ms/step - loss: 1.1051 - accuracy: 0.6199 - val_loss: 1.1768 - val_accuracy: 0.5971
Epoch 14/15
1563/1563 [==============================] - 40s 25ms/step - loss: 1.0910 - accuracy: 0.6206 - val_loss: 1.0808 - val_accuracy: 0.6297
Epoch 15/15
1563/1563 [==============================] - 38s 24ms/step - loss: 1.0943 - accuracy: 0.6234 - val_loss: 1.0550 - val_accuracy: 0.6338
```

Out[14]:
```
<keras.callbacks.History at 0x18a83db5ca0>
```

In [15]:
```
model_1.predict(x_test)
```

Out[15]:
```
array([[7.5810449e-03, 5.2895662e-03, 3.2852419e-02, ..., 1.0692628e-02,
        7.8084935e-03, 5.6998804e-03],
       [1.1408026e-02, 4.6022582e-01, 4.1846583e-06, ..., 1.8463833e-08,
        5.1794839e-01, 1.0413391e-02],
       [9.0228975e-02, 4.8100509e-02, 2.4456149e-02, ..., 3.2928448e-02,
        5.7330596e-01, 7.3267162e-02],
       ...,
       [1.9124896e-05, 8.7166364e-07, 1.9910648e-02, ..., 2.1665717e-02,
        2.7145765e-07, 1.5294592e-05],
       [3.0577293e-01, 5.7187412e-02, 1.2445054e-01, ..., 8.7521486e-02,
        3.0119848e-02, 2.4947308e-02],
       [1.4049045e-05, 2.6132105e-07, 1.0127943e-03, ..., 9.8453426e-01,
        4.7254613e-08, 4.5032530e-06]], dtype=float32)
```

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?
2. Train it for 5 epochs. What do you notice about the training time, loss and accuracy numbers (on both the training and validation sets)?
3. Try different structures and run times, and see how accurate your model can be.

In [17]:
```
# Let's build a CNN using Keras' Sequential capabilities
from tensorflow.keras.models import Sequential
model_2 = Sequential()

model_2.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model_2.add(Activation('relu'))
model_2.add(Conv2D(32, (3, 3)))
model_2.add(Activation('relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.25))

model_2.add(Conv2D(64, (3, 3), padding='same'))
model_2.add(Activation('relu'))
model_2.add(Conv2D(64, (3, 3)))
model_2.add(Activation('relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.25))

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'))
```

In [20]:
```
## Check number of parameters

model_2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 activation_14 (Activation)  (None, 32, 32, 32)        0

 conv2d_9 (Conv2D)           (None, 30, 30, 32)        9248

 activation_15 (Activation)  (None, 30, 30, 32)        0

 max_pooling2d_4 (MaxPooling  (None, 15, 15, 32)       0
 2D)

 dropout_7 (Dropout)         (None, 15, 15, 32)        0

 conv2d_10 (Conv2D)          (None, 15, 15, 64)        18496

 activation_16 (Activation)  (None, 15, 15, 64)        0

 conv2d_11 (Conv2D)          (None, 13, 13, 64)        36928

 activation_17 (Activation)  (None, 13, 13, 64)        0

 max_pooling2d_5 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 dropout_8 (Dropout)         (None, 6, 6, 64)          0

 flatten_3 (Flatten)         (None, 2304)              0

 dense_6 (Dense)             (None, 512)               1180160

 activation_18 (Activation)  (None, 512)               0

 dropout_9 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 10)                5130

 activation_19 (Activation)  (None, 10)                0

=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
# initiate RMSprop optimizer
opt_2 = tf.keras.optimizers.RMSprop(lr=0.0005)

# Let's train the model using RMSprop
model_2.compile(loss='categorical_crossentropy',
              optimizer=opt_2,
              metrics=['accuracy'])
```

In [23]:
```
model_2.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=5,
              validation_data=(x_test, y_test),
              shuffle=True)
```

```
Epoch 1/5
1563/1563 [==============================] - 245s 156ms/step - loss: 1.5634 - accuracy: 0.4321 - val_loss: 1.2776 - val_accuracy: 0.5478
Epoch 2/5
1563/1563 [==============================] - 235s 151ms/step - loss: 1.1697 - accuracy: 0.5878 - val_loss: 1.0136 - val_accuracy: 0.6402
Epoch 3/5
1563/1563 [==============================] - 245s 157ms/step - loss: 0.9941 - accuracy: 0.6525 - val_loss: 0.9743 - val_accuracy: 0.6639
Epoch 4/5
1563/1563 [==============================] - 239s 153ms/step - loss: 0.9046 - accuracy: 0.6859 - val_loss: 0.8511 - val_accuracy: 0.7012
Epoch 5/5
1563/1563 [==============================] - 243s 155ms/step - loss: 0.8551 - accuracy: 0.7033 - val_loss: 1.0907 - val_accuracy: 0.6420
```

Out[23]:
```
<keras.callbacks.History at 0x18a8161d520>
```