CS109A Introduction to Data Science:

Homework 9 AC 209 : Convolutional Neural Networks

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

In [1]:
```
# RUN THIS CELL FOR FORMAT
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-C
HTML(styles)
```

Out[1]:

In [2]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets

%matplotlib inline

# Imports
import keras
from keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten, Dropou
from keras.models import Model
from keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.datasets import cifar10
```

```
Using TensorFlow backend.
```

Question 3 [12 pts]

3.1 What is the motivation for convolutional layers in image analysis?.

3.2 Let C be a CNN with the following layers:

1. Input layer, 64x64x3 RGB image
2. Convolutional Layer, 16 3x3 filters, stride 1, padding = same, activation = relu
3. Convolutional layer, 32 5x5 filters, stride 1, padding = same, activation = relu

4. Maxpool layer, size 3x3, stride 2
5. Convolutional layer, 128 3x3 filters, stride 1, padding = same
6. Fully connected layer, 5 outputs

a) Without doing any calculations, which layer will have the most parameters?

b) How many parameters does this CNN have in total? Show the number of parameters of each layer.

3.1 What is the motivation for convolutional layers in image analysis?.

Semantic meaning in an image has a tree-like structure. The meaning of the entire image is a function of the large objects represented in the image, which in turn is a function of the smaller objects making up the larger objects, and so on until you reach the level of individual pixels. CNNs capture this structure by applying filters and compressions to the image that capture increasingly global characteristics. For example, beginning layers may have edge detectors, middle layers might have shape detectors, and end layers might have objects detectors.

Also, by applying the same filter to the entire image, they are more robust to translations of the image.

3.2 Let C be a CNN with the following layers:

1. Input layer, 64x64x3 RGB image
2. Convolutional Layer, 16 3x3 filters, stride 1, padding = same, activation = relu
3. Convolutional layer, 32 5x5 filters, stride 1, padding = same, activation = relu
4. Maxpool layer, size 3x3, stride 2
5. Convolutional layer, 128 3x3 filters, stride 1, padding = same
6. Fully connected layer, 5 outputs

a) Without doing any calculations, which layer will have the most parameters?

The fully connected layer will have the most parameters because it will assign 5 parameters to each element in the 128 x 22 x 22 tensor. This tensor has the largest volume out of all input tensors and the fully connected layer has the highest ratio of parameters to input tensor volume.

b) How many parameters does this CNN have in total? Show the number of parameters of each layer.

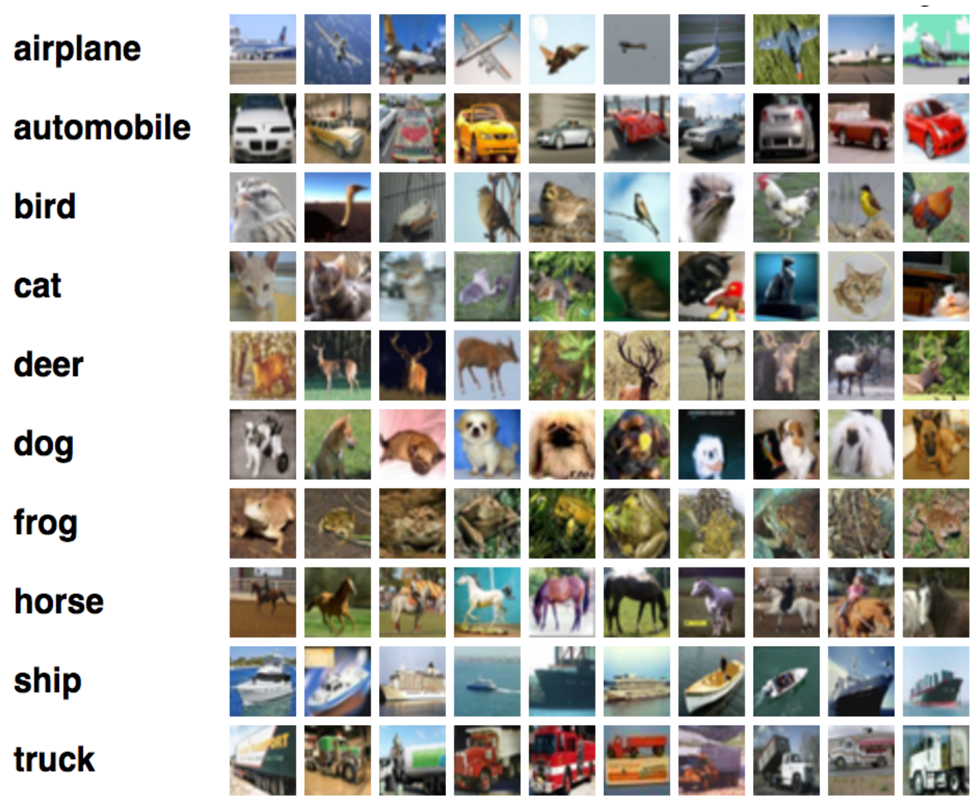
Input layer, 64x64x3 RGB image: 0 parameters
Convolutional Layer, 16 3x3 filters, stride 1, padding = same, activation = relu: 448 paramters
Convolutional layer, 32 5x5 filters, stride 1, padding = same, activation = relu: 4640 parameters
Maxpool layer, size 3x3, stride 2: 0
Convolutional layer, 128 3x3 filters, stride 1, padding = same: 36992
Fully connected layer, 5 outputs: 309765

Question 4 [13 pts]

We will now compare a Fully Connected Network (Multi-Layer Perceptron) and a simple CNN on the task of image classification. We'll use a well known open dataset: CIFAR10. We will be using Keras for our networks.

CIFAR10 is a classic dataset released by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton (Machine learning legends). It consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

Here are some examples of the images in the dataset:



Good news: Keras allows us to easily imports well-known datasets like CIFAR10. We'll import the CIFAR10 dataset with `keras.datasets.cifar10.load_data()` . This will return two tuples of numpy arrays: `(x_train, y_train), (x_test, y_test)` .

After that, we will implement an MLP and a CNN to classify the 10 classes of CIFAR10.

Keras can build models in two different ways: *Sequential* and *Functional*.

The Sequential API is a good starting point, as it allows you to easily create models layer by layer. You used it during the 109 part of the homework. Building a Sequential model just requires to instantiate a `Sequential()` object with `model = Sequential()` , and adding layers after that is easily done with `model.add(layer)` . This API has several limitations, as it does not allow you to easily create bypass connections, share layers between models or have multiple inputs or outputs.

Small example of the Sequential API:

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(10, input_dim=1))
model.add(Dense(1))
```

The functional API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers. You can connect layers to any other layer, add more inputs to your network (even in the middle of it) and concatenating outputs easily. Creating complex networks, such as ResNets, becomes feasible.

Small example of the Functional API:

```
# We define an Input layer
inp = Input(shape=(64,64,3))

# We instantiate a Dense layer and connect it to the previous layer
 with (inp)
x = Dense(10)(inp)

# We repeat the process, connecting here with (x)
x= Dense(10)(x)

out = Dense(1)(x)

# We build the full model
model = Model(inputs=in, outputs=out)
```

We will be using the Functional API for this problem, as this is the main mode that you will be using should you decide to build a serious network.

4.1 Import the CIFAR10 dataset, one-hot encode the labels and put your x_train and x_test into the [0,1] range. Plot some train and test images to get a better feel for the data. Hints:

1. Keras has a very convenient function for converting labels to categorical: `keras.utils.np_utils.to_categorical()` .
2. Your `x_train` and `x_test` will come as 8-bit images, so numpy array of integers with values from 0 to 255.
3. `plt.imshow` is your friend.

In [4]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.ta
r.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 62s 0us/step
```

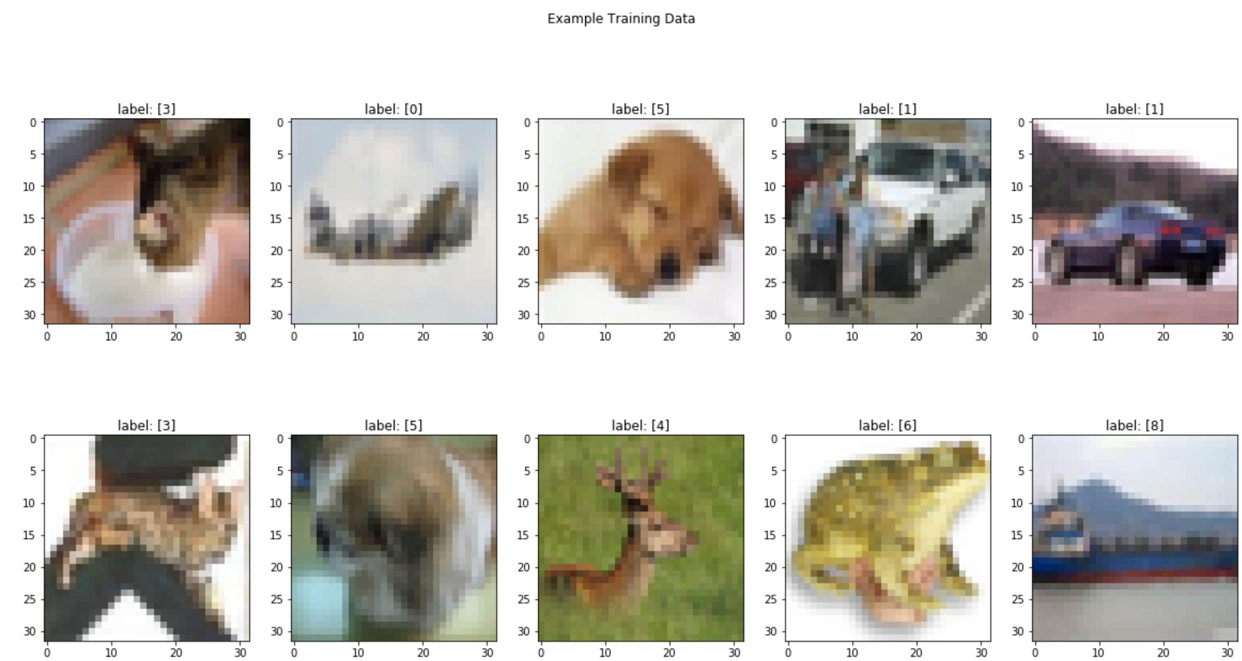
In [8]:
```
def preprocess(x,y):
    x_scaled = x/255
    y_one_hot = np_utils.to_categorical(y)
    return x_scaled, y_one_hot
```

In [9]:
```
x_train_preprocessed, y_train_preprocessed = preprocess(x_train, y_train)
x_test_preprocessed, y_test_preprocessed = preprocess(x_test, y_test)
```

In [19]:
```
def plt_random_images(x, y, title):
    fig, axes = plt.subplots(2,5,figsize = (20,10))
    for ax in axes.ravel():
        idx = np.random.randint(len(x))
        ax.imshow(x_train_preprocessed[idx])
        ax.set_title('label: {}'.format(y[idx]))
    fig.suptitle(title)
    fig.show()
```

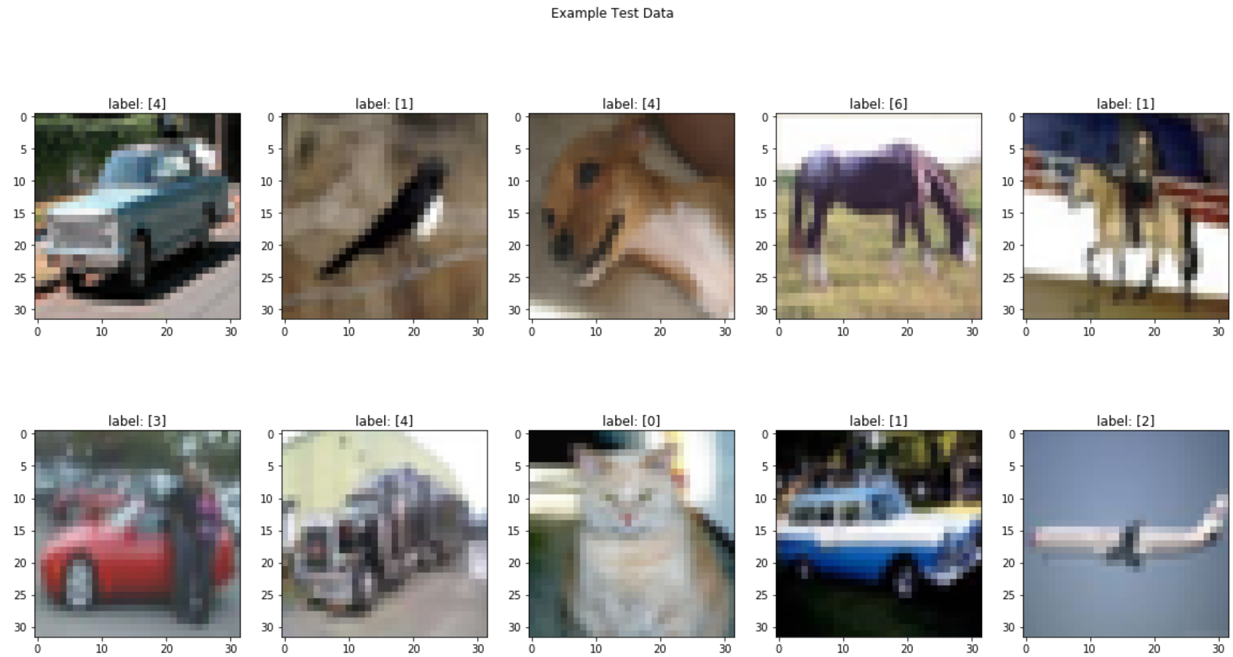
In [22]:
```
plt_random_images(x_train_preprocessed, y_train,'Example Training Data')
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/matplo
tlib/figure.py:457: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



In [23]:
```
plt_random_images(x_test_preprocessed, y_test,'Example Test Data')
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/matplo
tlib/figure.py:457: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



4.2 Using the functional API, build two networks:

(a) MLP with the following layers:

1. Input layer
2. Flatten layer (so that we can feed easily to Dense layers afterwards)
3. Dense layer, 128 nodes, relu activation
4. Dropout layer, 0.2 probability
5. Dense layer, 256 nodes, relu activation
6. Dropout layer, 0.2 probability
7. Dense layer, 512 nodes, relu activation
8. Dropout layer, 0.2 probability
9. Dense layer, 10 nodes, softmax activation

(b) CNN with the following layers:

1. Conv2D, 32 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
2. Conv2D, 32 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
3. Maxpool, strides (2,2), pool_size (2,2)
4. Conv2D, 64 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
5. Conv2D, 64 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
6. Maxpool, strides (2,2), pool_size (2,2)
7. Conv2D, 128 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
8. Conv2D, 128 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
9. Maxpool, strides (2,2), pool_size (2,2)
10. Flatten layer

11. Dropout layer, 0.2 probability
12. Dense layer, 512 nodes, relu activation
13. Dropout, 0.5 probability
14. Dense layer, 10 nodes, softmax activation

Some extra help: here are the definitions of generic Conv2D layers, MaxPooling2D layers, Dense, Flatten and Dropout layers:

```
x = Conv2D(32, (3,3), strides=(1, 1), padding='same', activation='re
lu', use_bias=True)(x)
x = MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same')(x)
x = Flatten()(x)
x = Dropout(0.2)(x)
x = Dense(1024, activation='relu')(x)
```

Use model.summary() to report the number of parameters on each model. Train both models on CIFAR10 with a batch size of 32 for 5 epochs and report the result.

Some helpful tips:

1. Once you've defined your layers and connected them to each other, remember to use `model = Model(inputs=inp, output=out)` to build the full model.
2. Once you have the model, you will need to define an optimizer (SGD, Adam, etc), define a loss (use categorical_crossentropy), indicate which metrics to use, and send those parameters to the compile function before fitting it. Your code should look like this:

```
optimizer = YourOptimizer(lr=yourLearningRate)
model.compile(optimizer, loss='categorical_crossentropy', metrics=
['accuracy'])
model.fit(x_train, y_train, batch_size=yourBatchSize, epochs = yourE
pochs, validation_split=0.2)
```

A note about `validation_split` : If set to 0.2, the fit function automatically sets apart 20% of your training data and doesn't train on it. It will use that 20% to give indications on how the model is doing by showing a val_acc value at the end of each epoch. You could also extract a validation dataset yourself and send it to the function with `validation_data` .

Answers

Input layer Flatten layer (so that we can feed easily to Dense layers afterwards) Dense layer, 128 nodes, relu activation Dropout layer, 0.2 probability Dense layer, 256 nodes, relu activation Dropout layer, 0.2 probability Dense layer, 512 nodes, relu activation Dropout layer, 0.2 probability Dense layer, 10 nodes, softmax activation

(a) MLP with the following layers:

1. Input layer
2. Flatten layer (so that we can feed easily to Dense layers afterwards)
3. Dense layer, 128 nodes, relu activation

4. Dropout layer, 0.2 probability
5. Dense layer, 256 nodes, relu activation
6. Dropout layer, 0.2 probability
7. Dense layer, 512 nodes, relu activation
8. Dropout layer, 0.2 probability
9. Dense layer, 10 nodes, softmax activation

In [32]:
```
## We define an Input layer
inp = Input(shape=(32,32,3))
x = Flatten()(inp)
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.2)(x)
out = Dense(10, activation='softmax')(x)

# We build the full model
model = Model(inputs=inp, outputs=out)
```

In [33]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         (None, 32, 32, 3)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 3072)              0         
_________________________________________________________________
dense_10 (Dense)             (None, 128)               393344    
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 256)               33024     
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 512)               131584    
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                5130      
=================================================================
Total params: 563,082
Trainable params: 563,082
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
optimizer = Adam(lr=0.001)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['accurac
model.fit(x_train_preprocessed, y_train_preprocessed, batch_size=32, epochs
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
40000/40000 [==============================] - 9s 219us/step - loss: 2.04
65 - acc: 0.2221 - val_loss: 1.9226 - val_acc: 0.2783
Epoch 2/5
40000/40000 [==============================] - 8s 203us/step - loss: 1.94
80 - acc: 0.2686 - val_loss: 1.8997 - val_acc: 0.2932
Epoch 3/5
40000/40000 [==============================] - 9s 225us/step - loss: 1.91
47 - acc: 0.2826 - val_loss: 1.8292 - val_acc: 0.3300
Epoch 4/5
40000/40000 [==============================] - 9s 216us/step - loss: 1.90
33 - acc: 0.2944 - val_loss: 1.8306 - val_acc: 0.3390
Epoch 5/5
40000/40000 [==============================] - 8s 209us/step - loss: 1.89
06 - acc: 0.2966 - val_loss: 1.8519 - val_acc: 0.3289
```

Out[34]: <keras.callbacks.History at 0x1915c37b8>

(b) CNN with the following layers:

1. Conv2D, 32 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
2. Conv2D, 32 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
3. Maxpool, strides (2,2), pool_size (2,2)
4. Conv2D, 64 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
5. Conv2D, 64 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
6. Maxpool, strides (2,2), pool_size (2,2)
7. Conv2D, 128 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
8. Conv2D, 128 3x3 filters, (1,1) strides, padding=same, activation=relu, use_bias=True
9. Maxpool, strides (2,2), pool_size (2,2)
10. Flatten layer
11. Dropout layer, 0.2 probability
12. Dense layer, 512 nodes, relu activation
13. Dropout, 0.5 probability
14. Dense layer, 10 nodes, softmax activation

In [36]:
```
## We define an Input layer
inp = Input(shape=(32,32,3))
x = Conv2D(32, (3,3), strides=(1, 1), padding='same', activation='relu', us
x = Conv2D(32, (3,3), strides=(1, 1), padding='same', activation='relu', us
x = MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same')(x)

x = Conv2D(64, (3,3), strides=(1, 1), padding='same', activation='relu', us
x = Conv2D(64, (3,3), strides=(1, 1), padding='same', activation='relu', us
x = MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same')(x)

x = Conv2D(128, (3,3), strides=(1, 1), padding='same', activation='relu', u
x = Conv2D(128, (3,3), strides=(1, 1), padding='same', activation='relu', u
x = MaxPooling2D(pool_size=(2, 2), strides=(2,2), padding='same')(x)

x = Flatten()(x)
x = Dropout(0.2)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.2)(x)
out = Dense(10, activation='softmax')(x)

# We build the full model
model = Model(inputs=inp, outputs=out)
```

In [38]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (InputLayer)         (None, 32, 32, 3)         0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         73856     
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 2048)              0         
_________________________________________________________________
dense_14 (Dense)             (None, 512)               1049088   
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 10)                5130      
=================================================================
Total params: 1,341,226
Trainable params: 1,341,226
Non-trainable params: 0
_________________________________________________________________
```

In [39]:
```
optimizer = Adam(lr=0.001)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['accurac
model.fit(x_train_preprocessed, y_train_preprocessed, batch_size=32, epochs
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/5
40000/40000 [==============================] - 242s 6ms/step - loss: 1.58
78 - acc: 0.4126 - val_loss: 1.2589 - val_acc: 0.5454
Epoch 2/5
40000/40000 [==============================] - 241s 6ms/step - loss: 1.11
53 - acc: 0.6022 - val_loss: 0.9528 - val_acc: 0.6602
Epoch 3/5
40000/40000 [==============================] - 238s 6ms/step - loss: 0.91
57 - acc: 0.6773 - val_loss: 0.8839 - val_acc: 0.6880
Epoch 4/5
40000/40000 [==============================] - 243s 6ms/step - loss: 0.78
61 - acc: 0.7207 - val_loss: 0.7835 - val_acc: 0.7266
Epoch 5/5
40000/40000 [==============================] - 257s 6ms/step - loss: 0.70
24 - acc: 0.7521 - val_loss: 0.7351 - val_acc: 0.7437
```

Out[39]:
```
<keras.callbacks.History at 0x191d194a8>
```

In [ ]: