CNN Code 101

Notebook:	2. Building a CNN		
Created:	12/8/2018 11:47 PM	**Updated:**	5/5/2020 1:39 AM
Author:	yousef.kafif@outlook.com		

```
# Part 1 - Building the CNN

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (32, 64, 64, 3), activation =
'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
```

```
                                            batch_size = 32,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = 8000,
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = 2000)
```

 # Part 3 - Making new predictions

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg',
target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
```

DATASET STRUCTURE & Pre-Processing:-

- Not a .csv data file.
- Use keras.
- We split the data ourselves by using folders.
    - Training:-
        - dataset\training_set\cats\cats1.jpg #up to 4000 so 4000 images
        - dataset\training_set\dogs\dogs1.jpg #up to 4000 so 4000 images
    - Test:-
        - dataset\test_set\cats\cats4000.jpg #up to 5000 so 1000 images
        - dataset\test_set\dogs\dogs4000.jpg #up to 5000 so 1000 images
- We scale it using:-

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

PART 1 -> BUILDING THE CNN

Importing the Keras libraries and packages & Initialization:-

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Initialising the CNN

```
classifier = Sequential()
```

- Declares 'classifier' as a sequence of layers.

Step 1 - CONVOLUTION LAYER :-

```
classifier.add(Conv2D(32, (3, 3), input_shape = (32, 64, 64, 3), activation =
'relu'))
```

- First argument is filters -> 32 ***You double the filters in every consecutive layer**
  - The # of filters. -> Also our # of feature maps obviously.
- Second argument is the (Height, Width) -> (3, 3)
  - Our (height, width) of our matrix. Here it is a 3x3 matrix.
- input_shape argument is the (Batch Size, Rows/Height, Columns/Width, Channels) -> (32, 64, 64, 3)
  - *Our input specifications of our images*. i.e. what our conv layer should expect to receive.
    - NOTE: We have to resize the images during our importation of the image folders in the fitting stage.
  - (batch size, height, width, channels)
- activation argument -> 'relu'
  - Our activation function.
  - Using the rectifier to eliminate negative value to reduce linearity
  - This is done to make up for the linearity we might have imposed through the process of creating a FM.
- Padding argument. #to be researched
  - Padding is when you add '0's surrounding the image so that the matrix can fully scan the image.
  - $\begin{bmatrix} x & x & x & o & o & ? \\ x & x & x & o & o & ? \\ x & x & x & o & o & ? \\ x & x & x & x & x & \\ x & x & x & x & x & \end{bmatrix}$ -------> W/padding $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & o & o & o & 0 \\ 0 & 0 & x & x & x & o & o & o & 0 \\ 0 & 0 & x & x & x & o & o & o & 0 \\ 0 & 0 & x & x & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

NOTE: It is better to have multiple (Conv + Pooling) layers stacked on top of each other. DEEP learning broski.

- ***You typically double the amount of filter detectors in every consecutive layer.***

NOTE: The bigger the image the larger the stride, otherwise it would take too long.

Step 2 - POOLING LAYER:-

```
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

- pool_size -> Our (Width, Height) of pooling matrix

Step 3 - FLATTENING:-

```
classifier.add(Flatten())
```

- Self explanatory

Step 4 - ADDING A FULLY CONNECTED ANN on top:-

```
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
```

- There is no set method of finding the right amount of nodes. Just remember that we are having a ton of inputs (the maps pixels/values) flattened.

Step 5 - COMPILING THE CNN:-

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
```

- We normally use crossentropy for CNNs classification problems.
    - binary_crossentropy for a two class problem. e.g. cat or dog.

PART 2 -> FITTING THE CNN TO THE IMAGES
 # All taken from keras.io\preprocessing
Import:-

```
from keras.preprocessing.image import ImageDataGenerator
```

Step 1 - Setting up ImageDataGenerator object (for Feature Scaling + Data Augmentation)

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2, # to manipulate images in
batches - prevents over fitting
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

- Rescale argument
    - the feature scaling - scaling all our pixel values between 0-1. e.g. 255* 1/255 would be a 1 <- thats the max
- Shear_range + zoom_range + horizontal_flip and optional others:-
    - # data augmentation settings to manipulate images in batches.
    - Prevents over-fitting
    - Allows us to re-use the same images but in different variations. So we can use 'small' data sets.
- NOTE:
    - Test dataset does not get augmented. We are just validating the model after all.

Step 2 - Importing dataset using ImageDataGenerator object:-

```
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

- First argument is our directory.
  - our directory, make sure it is in your working directory.
- target_size argument:-
  - converting our images to the same size that is EXPECTED IN our CNN model.
- batch_size argument:-
  - size of batch in which the random samples of our images will be included.
- Class_mode argument:-
  - One of "categorical", "binary", "sparse".

Step 3 - Fitting & Training our model! :-

```
classifier.fit_generator(training_set, # training set argument
                         steps_per_epoch = 250, # total number of batches to be
done before declaring an epoch to finish, i.e. batches per epoch - should be
typically -> samples in dataset/batch size
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = 63) # batches per epoch
```

- First argument
  - Is our training set object.
- steps_per_epoch argument :-
  - The amount of batches per epoch.
  - Should typically be (samples in our set)/(batch size)
- epochs argument :-
  - nigga plz
- validation_data argument :-
  - Our test set object
- validation_steps argument :-
  - Just batches per epoch.

Step 3 - Using model to make new predictions on select images!

- Check my other note.