# Cat Dog Classifier

May 9, 2018

## 1 Cat Dog Clssification

from blog.keras.io. It uses data that can be downloaded at: https://www.kaggle.com/c/dogs-vs-cats/data In our setup, we: - created a data/ folder - created train/ and validation/ subfolders inside data/ - created cats/ and dogs/ subfolders inside train/ and validation/ - put the cat pictures index 0-999 in data/train/cats - put the cat pictures index 1000-1400 in data/validation/cats - put the dogs pictures index 11500-12499 in data/train/dogs - put the dog pictures index 13500-13900 in data/validation/dogs So that we have 1000 training examples for each class, and 401 validation examples for each class.

Fisrt we need to import required models,layers from keras. As we are dealing with image data , we need to import ImageDataGenerator as it Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).

```
In [1]: from keras.preprocessing.image import ImageDataGenerator
        from keras.models import Sequential
        from keras.layers import Conv2D, MaxPooling2D
        from keras.layers import Activation, Dropout, Flatten, Dense
        from keras import backend as K
```

```
/home/nikhil/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversio
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In keras, we need to give the shape of the input data to first layer of the network.

```
In [2]: img_width, img_height = 150, 150
```

Initialise the batch size, number the epoches you want to perform and input directories of train and validation data

```
In [3]: train_data_dir = 'data/train'
        validation_data_dir = 'data/validation'
        nb_train_samples = 2000
        nb_validation_samples = 800
        epochs = 5
        batch_size = 16
```

```
In [4]: if K.image_data_format() == 'channels_first':
            input_shape = (3, img_width, img_height)
        else:
            input_shape = (img_width, img_height, 3)
```

we are using the sequential model , which is the stack of multiple layers. construction of model with required layers. *In conv2D 32 is the number of hidden layers and (3, 3) is the kernal size.* we are relu activation *maxpolling2D is used to reduce the features of the data i.e only high influenctial features are extracted and pool_size is the required pool scale i.e pool_size = (horizontal,vertical).* Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. *Dense is a fully connected layer,in which all the neurons in this layer is connected to next layer.* Flatten layer is used to flatten the input and it does not effect the batch size. *This model is the 12 layer network.

```
In [5]: model = Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=input_shape))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

In [6]: model.add(Conv2D(32, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

In [7]: model.add(Flatten())
        model.add(Dense(64))
        model.add(Activation('relu'))
        model.add(Dropout(0.5))
        model.add(Dense(1))
        model.add(Activation('sigmoid'))
```

Now we are going to compile the model, while compiling the model we need to define the three parameters which are *As we are doing binary classification , we are using BINARY_CROSSENTROPY.* Here we are using rmspprop *As we are dealing with classification problem , we are using accuracy metric.

```
In [8]: model.compile(loss='binary_crossentropy',
                      optimizer='rmsprop',
        metrics=['accuracy'])
```

We are using ImagedataGenerator as it generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches). *rescale is the value that is used to multiply the give data.

```
In [9]: train_datagen = ImageDataGenerator(
            rescale=1. / 255,
            shear_range=0.2,
            zoom_range=0.2,
        horizontal_flip=True)

In [10]: test_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
In [11]: train_generator = train_datagen.flow_from_directory(
             train_data_dir,
             target_size=(img_width, img_height),
             batch_size=batch_size,
         class_mode='binary')

Found 2000 images belonging to 2 classes.


In [12]: validation_generator = test_datagen.flow_from_directory(
             validation_data_dir,
             target_size=(img_width, img_height),
             batch_size=batch_size,
             class_mode='binary')

Found 802 images belonging to 2 classes.
```

for training the model , FIT function is typically used.

```
In [13]: model.fit_generator(
             train_generator,
             steps_per_epoch=nb_train_samples // batch_size,
             epochs=epochs,
             validation_data=validation_generator,
         validation_steps=nb_validation_samples // batch_size)

Epoch 1/5
125/125 [==============================] - 57s 454ms/step - loss: 0.7213 - acc: 0.5610 - val_los
Epoch 2/5
125/125 [==============================] - 52s 419ms/step - loss: 0.6611 - acc: 0.6250 - val_los
Epoch 3/5
125/125 [==============================] - 53s 420ms/step - loss: 0.6496 - acc: 0.6480 - val_los
Epoch 4/5
125/125 [==============================] - 52s 420ms/step - loss: 0.6299 - acc: 0.6690 - val_los
Epoch 5/5
125/125 [==============================] - 53s 420ms/step - loss: 0.5968 - acc: 0.6965 - val_los


Out[13]: <keras.callbacks.History at 0x7fae30c67f60>
```

If we perform 50 epochs we will get accuracy of around 85, as its taking more time to train the model i just put only 5 epochs.