Prediction between the images of cats and dogs

Description of the Data:

The data was taken from a Kaggle competition available at : https://www.kaggle.com/tongpython/cat-and-dog

My Kaggle notebook

1. 10,000 images of cats and dogs is available
2. Dimensions of the images are variable, along with the position of the animals in the images
3. Few images were broken
4. Images were available in coloured format, and each image can be represented by a 3-dimensional array, where the first two dimension specifies the intensity of the pixel, and the third dimension specifies the colour, that the intensity belongs to.
5. These RGB intensities add up to give all the possible colours.

Data Pre-processing

Selecting an image from the data set

```
training_files = os.listdir("training_set/training_set/dogs")
training_files[1]
datadir = "training_set/training_set/dogs"
img_path = os.path.join(datadir, training_files[1])
import cv2
img_array = cv2.imread(img_path)
```

Plotting the image using Matplotlib

```
import matplotlib.pyplot as plt
print(img_array.shape)
plt.imshow(img_array, cmap = "gray")
```

```
(325, 359, 3)


<matplotlib.image.AxesImage at 0x7f46f0499d68>
```


Tackling the problem of different locations of animals within the images

This is done by data augmentation, which applies various transformations to the data, some of them includes

- Horizontal flip
- Smearing
- Zooming

Along with it, the data was rescaled to a target size of (256,256) using the ImageDataGenerator,

```
# importing the data
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('training_set/training_set',
                                                 target_size = (256, 256),
                                                 batch_size = 128,
                                                class_mode = 'binary')
```

```
Using TensorFlow backend.

Found 8005 images belonging to 2 classes.
```

Designing a deep learning model with convolutional neural networks

1. Convoluting the images:

a. These layers identifies different features from the images, by convoluting the images with the kernels. Kernel of size (3 X 3) were chosen.

b. These kernels acts as features detectors, for example, this kernel [-1, 0, 1 ; -1, 0, 1; -1, 0, 1] when convoluted with the image, identifies all the vertical edges, similarly the kernel [1,1,1; 1,0,1;1,1,1] identifies dark spots in the images.

c. ReLU was used as activation function, which essentially guides the lighting up of neurons

d. 32 feature detectors were used, this means a total of 32 weights were associated with this layer, which are adjusted through the epochs by backpropagation

2. Pooling layer:

a. This is applied to the feature maps obtained from convoluted layers.
b. Our model shouldn't about the relative distances withing features, and the noises due to these must be reduced, for that, Pooling layer is used
c. Maxpooling was used with a window of 2X2 pixel, this takes only the maximum value in the window, and hence prevents overfitting of the model on the noises
d. Other pooling methods are, average sampling, mean pooling etc, but it was found that maxpooling works the best.

3. Flatten:

This is more of a transformation layer which converts the entire image into a vector, for the artificial neural networks to learn on them

4. Artificial Neural Networks:

A deep learning architecture was applied with 2 convolutional layers, 2 max pooling layers, a layer for flattening, and one dense layer consisting of 32 neurons

Evaluating the model

```
model = Sequential()
model.add(Conv2D(32, kernel_size =(3,3), input_shape = (256,256,3), activation = "r
elu"))
model.add(MaxPool2D(2))
model.add(Conv2D(64, kernel_size =(3,3), input_shape = (256,256,3), activation = "r
elu"))
model.add(MaxPool2D(2))
model.add(Flatten())
model.add(Dense(32, activation = "relu"))
model.add(Dense(1, activation = "sigmoid"))
model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accurac
y"])
#model.summary()
```

```
model.fit(training_set,epochs = 50, validation_data = test_set)
```

Training log:

```
Epoch 45/50
63/63 [==============================] - 183s 3s/step - loss: 0.2739 - accuracy:
0.8904 - val_loss: 0.4898 - val_accuracy: 0.8131
Epoch 46/50
63/63 [==============================] - 184s 3s/step - loss: 0.2665 - accuracy:
0.8887 - val_loss: 0.4033 - val_accuracy: 0.8240
Epoch 47/50
63/63 [==============================] - 183s 3s/step - loss: 0.2568 - accuracy:
0.8961 - val_loss: 0.4694 - val_accuracy: 0.8181
Epoch 48/50
63/63 [==============================] - 183s 3s/step - loss: 0.2520 - accuracy:
0.8942 - val_loss: 0.4724 - val_accuracy: 0.8127
Epoch 49/50
63/63 [==============================] - 184s 3s/step - loss: 0.2573 - accuracy:
0.8922 - val_loss: 0.5569 - val_accuracy: 0.8151
Epoch 50/50
63/63 [==============================] - 186s 3s/step - loss: 0.2458 - accuracy:
0.8964 - val_loss: 0.4340 - val_accuracy: 0.8161
```

Accuracy on validation data was found to be 81%

Improving the model:

Standard accuracy for this data set on Kaggle was observed to be around 70%. The initial accuracy obtained was 60% which was improved to 81% by iterating and tuning over the architecture. After several experiments on the architecture and hyper parameters, best accuracy was found to be 81% from this architecture

Things that can be done to improve accuracy with additional computing power

1. Wrapping the model using keras model and using grid search cross validation for hyper parameter tuning
2. Plotting the training log and using early stopping
3. Adding dropout layers and iterating over it in case of overfitting, however we can already see from the log that overfitting is not an issue here as the validation data accuracy is stable over the range of epochs.