Cat and dog classification using CNN

Project Objective:

Build a CNN model that classifies the given pet images correctly into dog and cat images. The project scope document specifies the requirements for the project "Pet Classification Model Using CNN." Apart from specifying the functional and non-functional requirements for the project, it also serves as an input for project scoping.

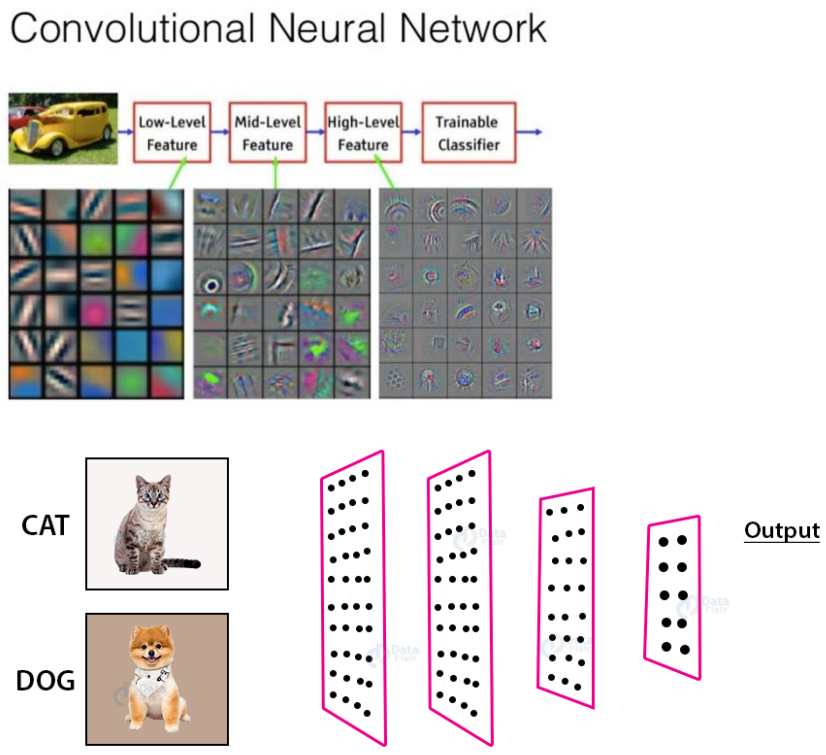
Project Description and Scope:

You are provided with a collection of images of pets, that is, cats and dogs. These images are of different sizes with varied lighting conditions and they should be used as inputs for your model.

You are expected to write the code for CNN image classification model using TensorFlow that trains on the data and calculates the accuracy score on the test data.

Starting the Projects:

Convolutional Neural Network (CNN) is an algorithm taking an image as input then assigning weights and biases to all the aspects of an image and thus differentiates one from the other. Neural networks can be trained by using batches of images, each of them having a label to identify the real nature of the image (cat or dog here). A batch can contain few tenths to hundreds of images. For each image, the network prediction is compared with the corresponding existing label, and the distance between network prediction and the truth is evaluated for the whole batch. Then, the network parameters are modified to minimize the distance and thus the prediction capability of the network is increased. The training process continues for every batch similarly.



Dogs vs. Cats Prediction Problem

The main goal is to develop a system that can identify images of cats and dogs. The input image will be analysed and then the output is predicted. The model that is implemented can be extended to a website or any mobile device as per the need. The dataset contains a set of images of cats and dogs.

Cat and dog classification using CNN

Our main aim here is for the model to learn various distinctive features of cat and dog. Once the training of the model is done it will be able to differentiate images of cat and dog.

Import Libraries

NumPy- For working with arrays, linear algebra.

2. Pandas – For reading/writing data

3. Matplotlib – to display images

4. TensorFlow Keras models – Need a model to predict right !!

5. TensorFlow Keras layers – Every NN needs layers and CNN needs well a couple of layers.

```
[259] import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      from keras.models import Sequential
      from keras.layers import Convolution2D
      from keras.layers import MaxPooling2D
      from keras.layers import Dense
      from keras.layers import Flatten
      from keras.layers import BatchNormalization
      from keras.preprocessing.image import ImageDataGenerator,load_img
      from tensorflow.keras.utils import to_categorical
      from sklearn.metrics import accuracy_score,classification_report,confusion_matri
      from sklearn.model_selection import train_test_split
      import random
      import os
```

CNN does the processing of Images with the help of matrixes of weights known as filters. They detect low-level features like vertical and horizontal edges etc. Through each layer, the filters recognize high-level features.

We first initialize the CNN

```
[30] Cnn_model=Sequential()
```

```
[31] Cnn_model.add(Conv2D(filters=32,kernel_size=(5,5),activation='relu',input_shape=(Image_Width,Image_Height,Image_Channels)))
     Cnn_model.add(BatchNormalization())
     Cnn_model.add(MaxPooling2D(pool_size=(2,2)))
     Cnn_model.add(Dropout(0.4))
```

```
[32] Cnn_model.add(Conv2D(filters=64,kernel_size=(5,5),activation='relu'))
     Cnn_model.add(BatchNormalization())
     Cnn_model.add(MaxPooling2D(pool_size=(2,2)))
     Cnn_model.add(Dropout(0.4))
```

```
[33] Cnn_model.add(Conv2D(filters=128,kernel_size=(5,5),activation='relu'))
     Cnn_model.add(BatchNormalization())
     Cnn_model.add(MaxPooling2D(pool_size=(2,2)))
     Cnn_model.add(Dropout(0.4))
```

Cat and dog classification using CNN

For compiling the CNN, we are using adam optimizer.

Adaptive Moment Estimation (Adam) is a method used for computing individual learning rates for each parameter. For loss function, we are using Binary cross-entropy to compare the class output to each of the predicted probabilities. Then it calculates the penalization score based on the total distance from the expected value.

Image augmentation is a method of applying different kinds of transformation to original images resulting in multiple transformed copies of the same image. The images are different from each other in certain aspects because of shifting, rotating, flipping techniques. So, we are using the Keras ImageDataGenerator class to augment our images.

```
[199] #part2-fitting the cnn to the images
      from keras.preprocessing.image import ImageDataGenerator,load_img
      image_datagen = ImageDataGenerator(rotation_range=30,# rotate the image 30 degrees
                                         rescale=1./255, # Rescale the image by normalzing it.
                                         shear_range=0.1,# Shear means cutting away part of the image (max 10%)
                                         zoom_range=0.2, # Zoom in by 20% max
                                         horizontal_flip=True,# Allow horizontal flipping
                                         width_shift_range=0.1,# Shift the pic width by a max of 10%
                                         height_shift_range=0.1, # Shift the pic height by a max of 10%
                                         fill_mode='nearest' # Fill in missing pixels with the nearest filled value
                                        )
```

```
[154] plt.imshow(image_datagen.random_transform(cat_image_read))
```

```
<matplotlib.image.AxesImage at 0x7fd7d5b3c4d0>
```

To check your images with ImageDataGenerotor Model

```
[153] plt.imshow(image_datagen.random_transform(dog_image_read))
```

```
<matplotlib.image.AxesImage at 0x7fd7d5bc7610>
```

We need a way to turn our images into batches of data arrays in memory so that they can be fed to the network during training. ImageDataGenerator can readily be used for this purpose. So, we import this class and create an instance of the generator. We are using Keras to retrieve images from the disk with the flow_from_directory method of the ImageDataGenerator class.

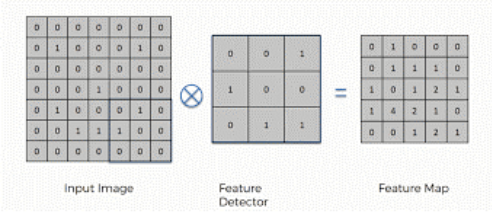
```
[136] test_image_gen = image_datagen.flow_from_directory(test_image_path,
                                                         target_size=image_shape[:2],
                                                         batch_size=batch_size,
                                                         class_mode='binary')
```

```
Found 20 images belonging to 2 classes.
```

Convolution

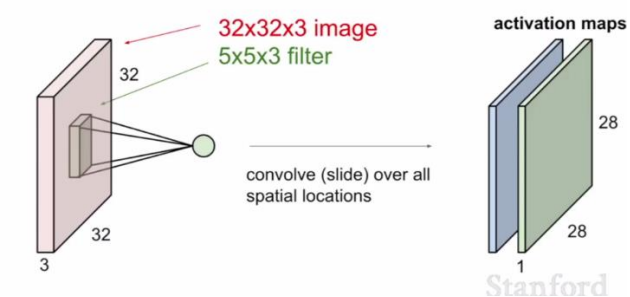
Cat and dog classification using CNN

Convolution is a linear operation involving the multiplication of weights with the input. The multiplication is performed between an array of input data and a 2D array of weights known as filter or kernel. The filter is always smaller than input data and the dot product is performed between input and filter array.



Activation

The activation function is added to help ANN learn complex patterns in the data. The main need for activation function is to add non-linearity into the neural network.

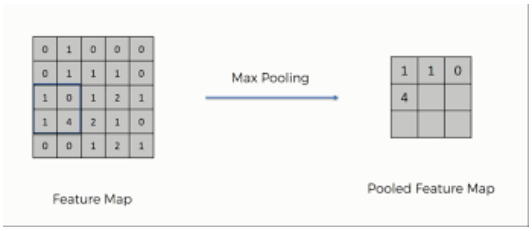


Pooling

The pooling operation provides spatial variance making the system capable of recognizing an object with some varied appearance. It involves adding a 2Dfilter over each channel of the feature map and thus summarise features lying in that region covered by the filter.

So, pooling basically helps reduce the number of parameters and computations present in the network. It progressively reduces the spatial size of the network and thus controls overfitting. There are two types of operations in this layer: Average pooling and Maximum pooling. Here, we are using max pooling which according to its name will only take out the maximum from a pool. This is possible with the help of filters sliding through the input and at each stride, the maximum parameter will be taken out and the rest will be dropped.

The pooling layer does not modify the depth of the network unlike in the convolution layer.



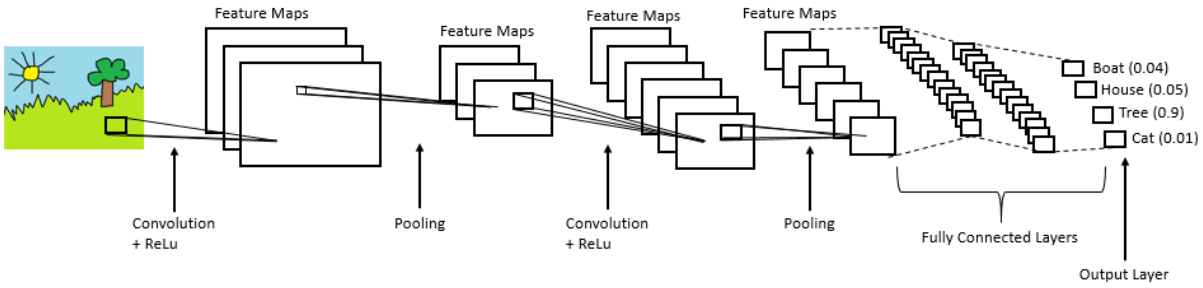
Fully Connected

Cat and dog classification using CNN

The output from the final Pooling layer which is flattened is the input of the fully connected layer.

The Full Connection process practically works as follows:

The neurons present in the fully connected layer detect a certain feature and preserves its value then communicates the value to both the dog and cat classes who then check out the feature and decide if the feature is relevant to them.




```
[34] Cnn_model.add(Flatten())
     Cnn_model.add(Dense(128))
     Cnn_model.add(Activation('relu'))

     # Dropouts help reduce overfitting by randomly turning neurons off during training.
     # Here we say randomly turn off 50% of neurons.
     Cnn_model.add(Dropout(0.5))

     # Last layer, remember its binary, 0=cat , 1=dog
     Cnn_model.add(Dense(1))
     Cnn_model.add(Activation('sigmoid'))

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

We are fitting our model to the training set. It will take some time for this to finish.

```
epochs=15
results = Cnn_model.fit_generator(train_image_gen,epochs=epochs,
                                  steps_per_epoch=3,
                                  validation_data=test_image_gen,
                                  validation_steps=12)
```

```
Epoch 1/15
3/3 [==============================] - ETA: 0s - loss: 0.5015 - accuracy: 0.9250WARNING:tensorflow:Your input ran out of data; interrupt
3/3 [==============================] - 5s 2s/step - loss: 0.5015 - accuracy: 0.9250 - val_loss: 4.6956 - val_accuracy: 0.4500
Epoch 2/15
3/3 [==============================] - 3s 1s/step - loss: 0.2473 - accuracy: 0.9000
Epoch 3/15
3/3 [==============================] - 3s 935ms/step - loss: 0.2669 - accuracy: 0.8750
Epoch 4/15
3/3 [==============================] - 3s 956ms/step - loss: 0.4884 - accuracy: 0.8250
Epoch 5/15
3/3 [==============================] - 3s 922ms/step - loss: 0.2323 - accuracy: 0.9000
Epoch 6/15
3/3 [==============================] - 3s 1s/step - loss: 0.2607 - accuracy: 0.9000
Epoch 7/15
3/3 [==============================] - 3s 948ms/step - loss: 0.4029 - accuracy: 0.8500
Epoch 8/15
3/3 [==============================] - 3s 1s/step - loss: 0.3503 - accuracy: 0.8750
Epoch 9/15
3/3 [==============================] - 3s 1s/step - loss: 0.1914 - accuracy: 0.9000
```

Cat and dog classification using CNN

```
Epoch 11/15
3/3 [==============================] - 3s 947ms/step - loss: 0.2668 - accuracy: 0.9000
Epoch 12/15
3/3 [==============================] - 3s 943ms/step - loss: 0.3809 - accuracy: 0.8000
Epoch 13/15
3/3 [==============================] - 3s 953ms/step - loss: 0.1445 - accuracy: 0.9500
Epoch 14/15
3/3 [==============================] - 3s 925ms/step - loss: 0.1555 - accuracy: 0.8750
Epoch 15/15
3/3 [==============================] - 3s 1s/step - loss: 0.2237 - accuracy: 0.9750
```

It is seen that we have 0.9750 accuracies on our training set.

We can predict new images with our model by predict_image function where we have to provide a path of new image as image path and using predict method. If the probability is more than 0.5 then the image will be of a dog else of cat.

```
#to predict new images
def predict_image(imagepath, classifier):
    predict = image.load_img(imagepath, target_size = Image_Size)
    predict_modified = image.img_to_array(predict)
    predict_modified = predict_modified / 255
    predict_modified = np.expand_dims(predict_modified, axis = 0)
    result = classifier.predict(predict_modified)
    if result[0][0] >= 0.5:
        prediction = 'dog'
        probability = result[0][0]
        print ("probability = " + str(probability))
    else:
        prediction = 'cat'
        probability = 1 - result[0][0]
        print ("probability = " + str(probability))
        print("Prediction = " + prediction)
```

```
[ ] dog_file = test_image_path+ '/dogs/110.jpg'
    dog_img = image.load_img(dog_file, target_size=(128, 128))
    dog_img = image.img_to_array(dog_img)
```

```
[ ] dog_img.shape
```

```
(128, 128, 3)
```

```
[ ] dog_img = np.expand_dims(dog_img, axis=0)
    dog_img = dog_img/255
```

```
Cnn_model.predict(dog_img)
```

```
array([[0.9882313]], dtype=float32)
```

```
[ ] print(f'Probability that image is a dog is: {Cnn_model.predict(dog_img)} ')
```

```
Probability that image is a dog is: [[0.9882313]]
```

```
[ ] cat_file = test_image_path+ '/cats/109.jpg'
    cat_img = image.load_img(cat_file, target_size=Image_Size)
    cat_img = image.img_to_array(cat_img)
```

Cat and dog classification using CNN

```
cat_img.shape
```

```
(128, 128, 3)
```

```
[ ] Cnn_model.predict(cat_img)
```

```
array([[0.98061913]], dtype=float32)
```

```
Cnn_model.predict(cat_img)
```

```
array([[0.98061913]], dtype=float32)
```

```
[ ] print(f'Probability that image is a cat is: {Cnn_model.predict(cat_img)} ')
```

```
Probability that image is a cat is: [[0.98061913]]
```

Features Provided:

We can test our own images and verify the accuracy of the model.

We can integrate the code directly into our other project.

We can extend the project to different entities by just finding the suitable dataset, change the dataset and train the model accordingly.