Machine Learning Engineer Nanodegree

Capstone Project

Abhilasha Kumari

Project: Write an Algorithm for a Dog Identification App

I. Definition

Problem Statement

This project includes algorithm to detect breed of dog as per the sample input and if instead of dog human is present it identifies that and then tells that which breed of dog they are most resembling to. In this project, a series of models are used together to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed.

Strategy to solve the problem

1. Import Datasets
2. Detect Humans
3. Detect Dogs
4. Create a CNN to Classify Dog Breeds (from Scratch)
5. Create a CNN to Classify Dog Breeds (using Transfer Learning)
6. Write your Algorithm
7. Test Your Algorithm

Metrics
The CNN model to classify dog breed should have an accuracy of 10% minimum and the model with transfer learning should have a minimum accuracy of 60%.

II. Analysis

Data Exploration

- *Dog datset can be downloaded from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip . Unzip the folder and place it in this project's home directory, at the location /dog_images.*
- *Human dataset can be downloaded from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zipDetect . Unzip the folder and place it in the home directory, at location /lfw.*

1. Human Detection

Human detection is done using 'Harcascade Frontal Face Algorithm'. The file for this ie. 'haarcascade_frontalface_alt.xml' needs to be downloaded before using it. Haarcascade is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. It can also be done using 'LBP Cascade Algorithm'

2. Dog Detection

It is done on a pre-trained model of VGG-16 where dog range lies in the dictionary key 151-268.

3. Dog Breed Classification

This is done in two ways, one from scratch and another using the pre-trained model. The two ways are described below:-

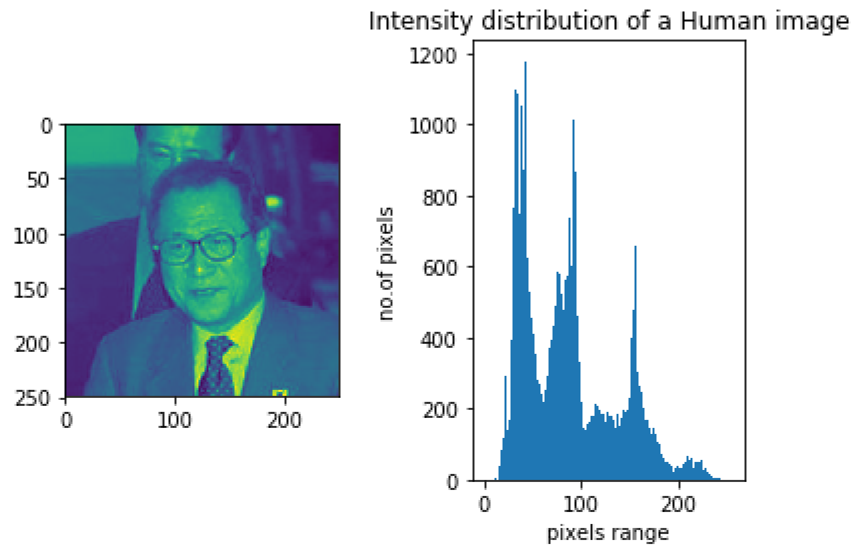
i. At first train and test data are classified, then CNN classifier architecture is

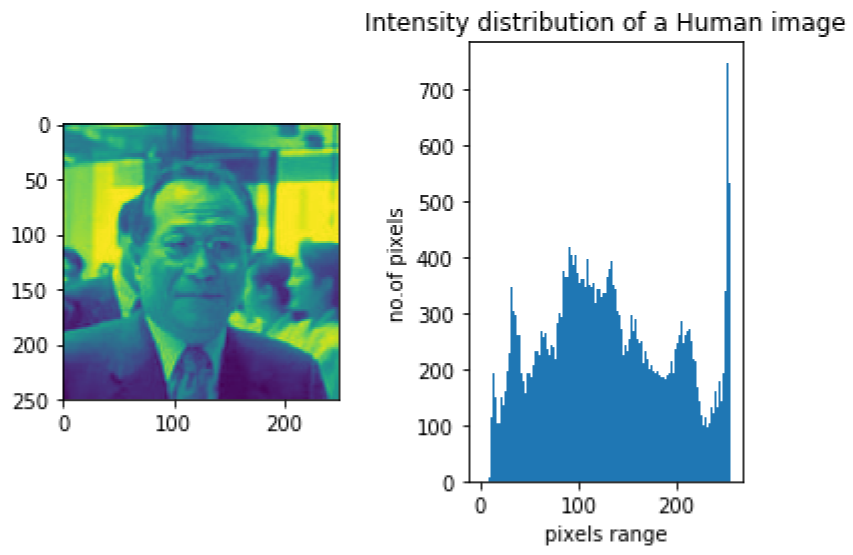
written and loss function are specified.

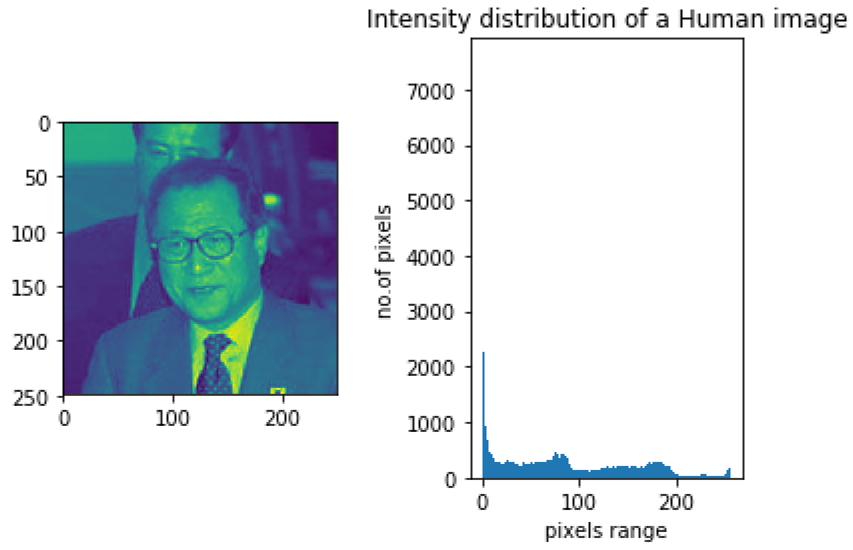
ii. Another model is written as CNN classifier using transfer learning.

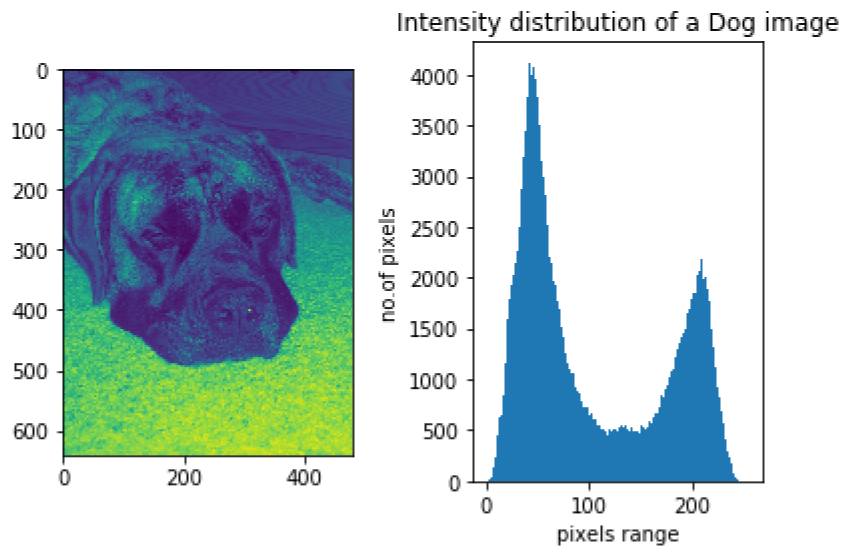
Exploratory Visualization

The model is based on CNN model and to build the model we need to identify intensity of pixel of image of dogs and humans. Few of the examples of intensity distribution is mentioned below in histogram form.

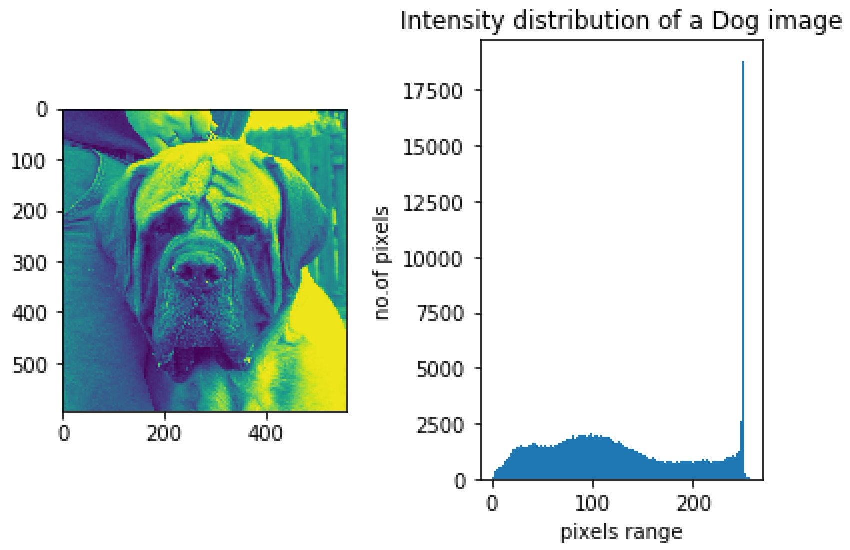



Intensity distribution of a Human image
0
50
100
150
200
250
0
100
200
no.of pixels
700
600
500
400
300
200
100
0
0
100
200
pixels range

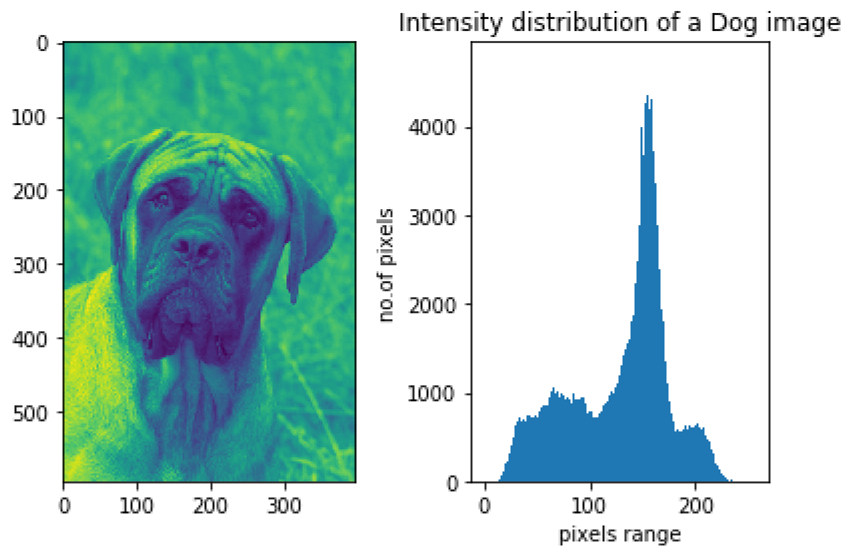

Intensity distribution of a Human image
0
50
100
150
200
250
0
100
200
no.of pixels
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
pixels range



Intensity distribution of a Dog image
no.of pixels
pixels range



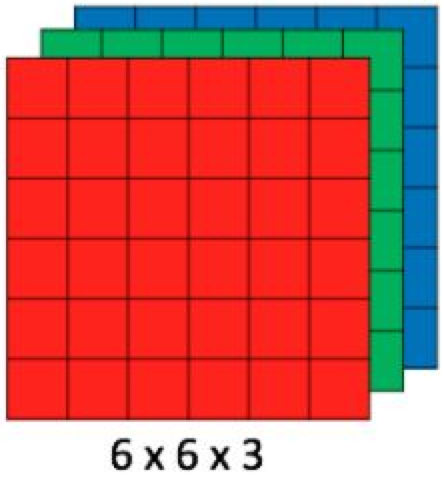
Intensity distribution of a Dog image
no.of pixels
pixels range



Algorithms and Techniques

CNN Architecture

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of



grayscale image.

Figure 1 : Array of RGB Matrix

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

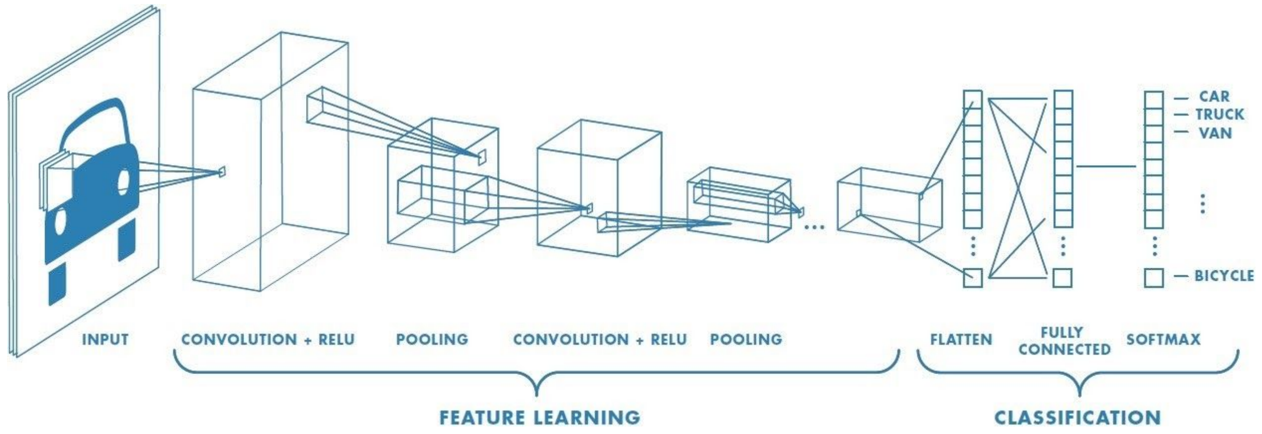


Figure 2 : Neural network with many convolutional layers

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension **$(h \times w \times d)$**
- A filter **$(f_h \times f_w \times d)$**
- Outputs a volume dimension **$(h - f_h + 1) \times (w - f_w + 1) \times 1$**

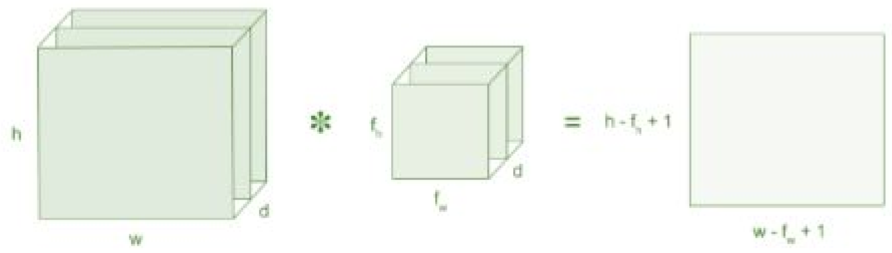


Figure 3: Image matrix multiplies kernel or filter matrix

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below.

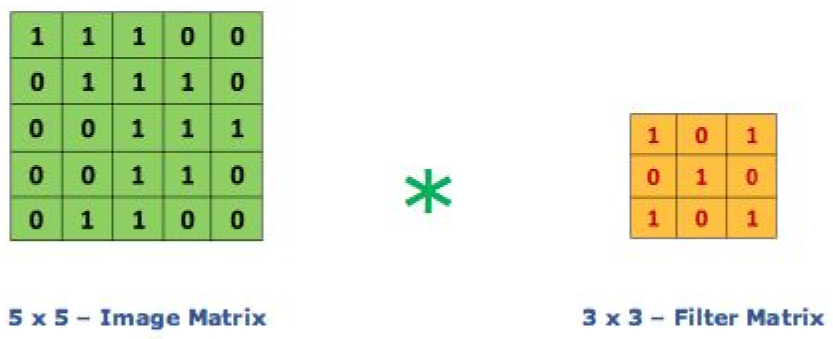


Figure 4: Image matrix multiplies kernel or filter matrix

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “Feature Map” as output shown in below.

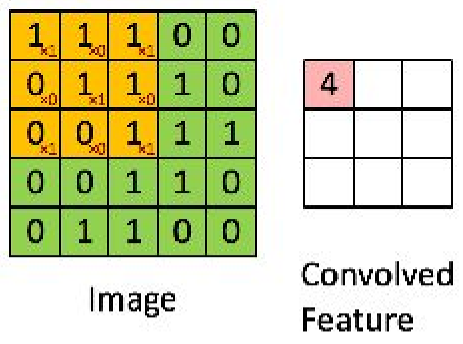
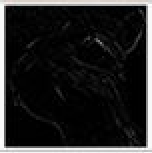
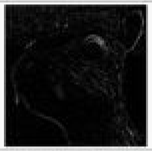
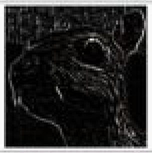
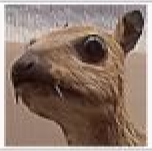
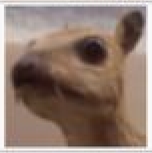
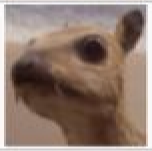


Figure 5: 3 x 3 Output matrix

Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

(Kernels).
Figure 7 : Some common filters

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

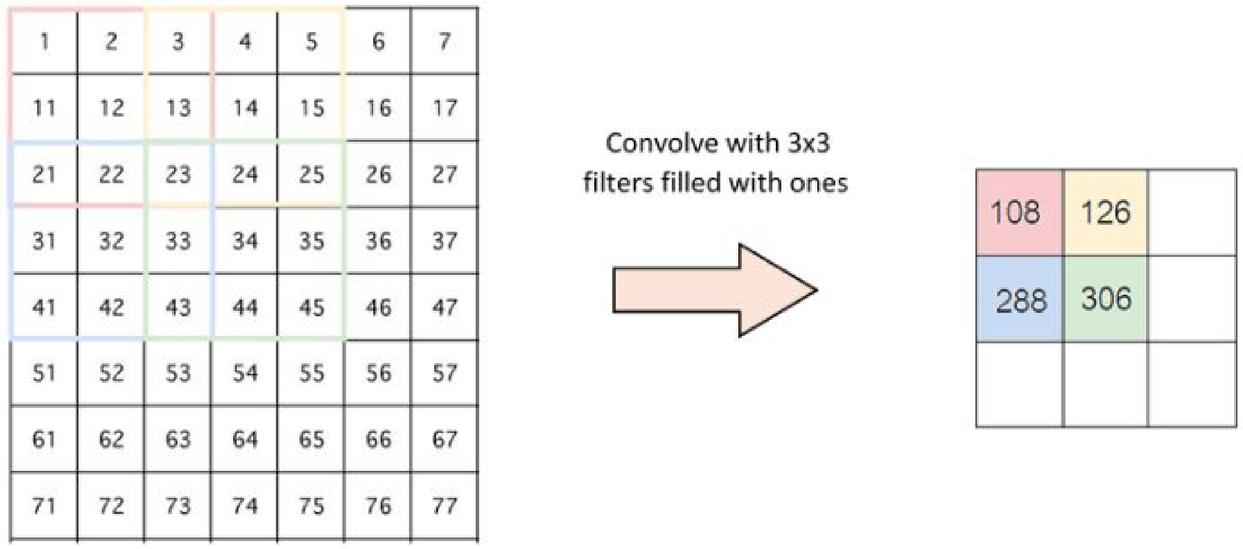


Figure 6 : Stride of 2 pixels

Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is *ƒ(x) = max(0,x).*

Why ReLU is important : ReLU’s purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values

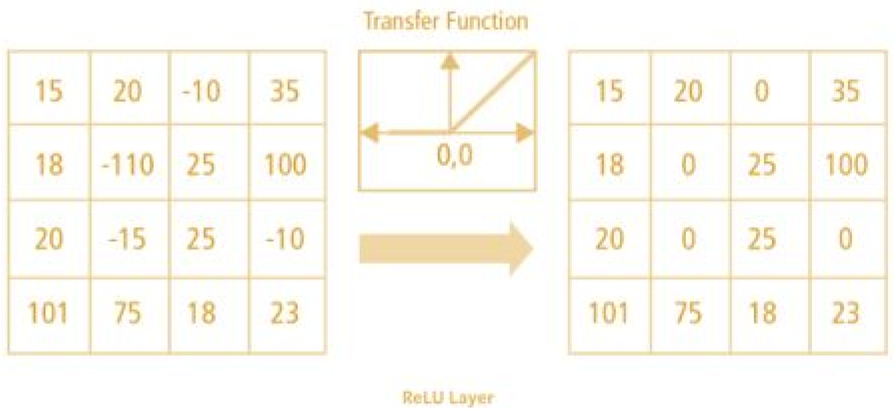


Figure 7 : ReLU operation

There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

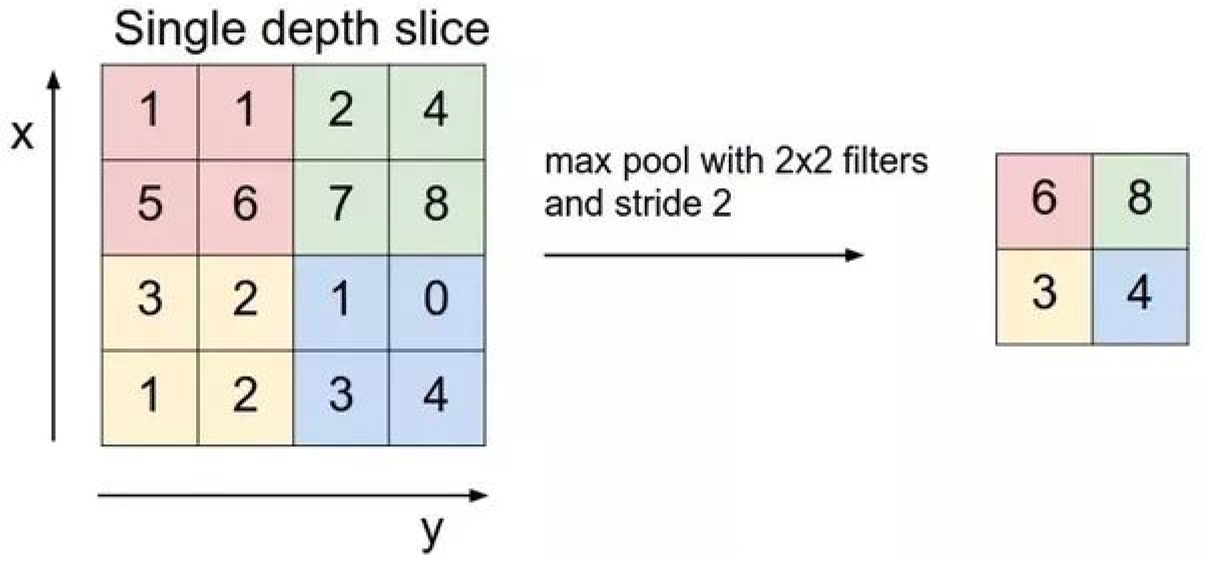


Figure 8 : Max Pooling

Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

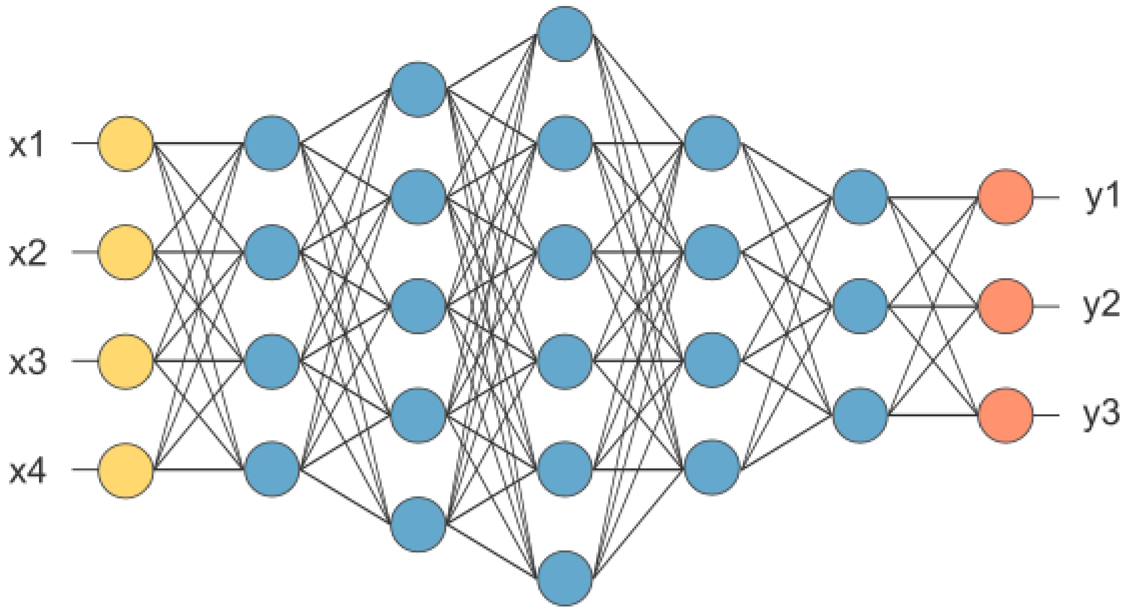


Figure 9 : After pooling layer, flattened as FC layer

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,

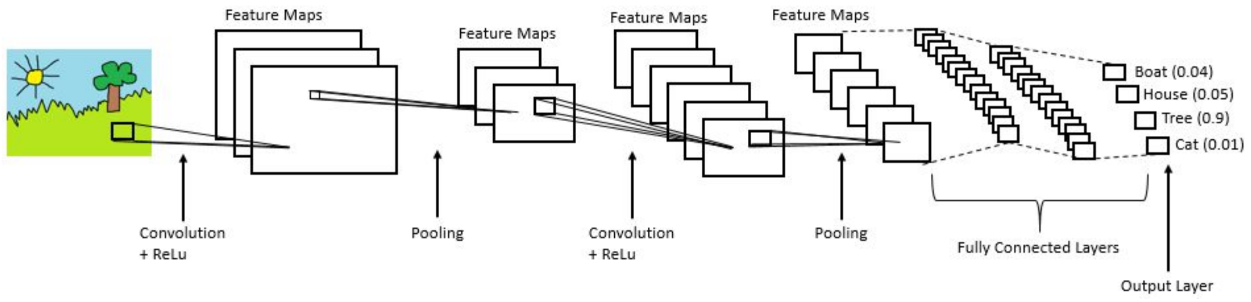


Figure 10 : Complete CNN architecture

Transfer Learning

Transfer learning (TL) is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

Stochastic Gradient Descent Algorithm

The word ‘*stochastic*‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get really huge.

Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

Benchmark

The benchmark classifier is a simple CNN classifier trained on different training dataset.

The primitive model architecture has an input layer with a shape of 224*224*3. The inputs are then fed to two convolutional layers, followed by a max-pooling layer (for downsampling). The outputs are then fed to a global average pooling layer to minimize overfitting by reducing the number of parameters. The output layer returns 133 probabilities for our breeds.

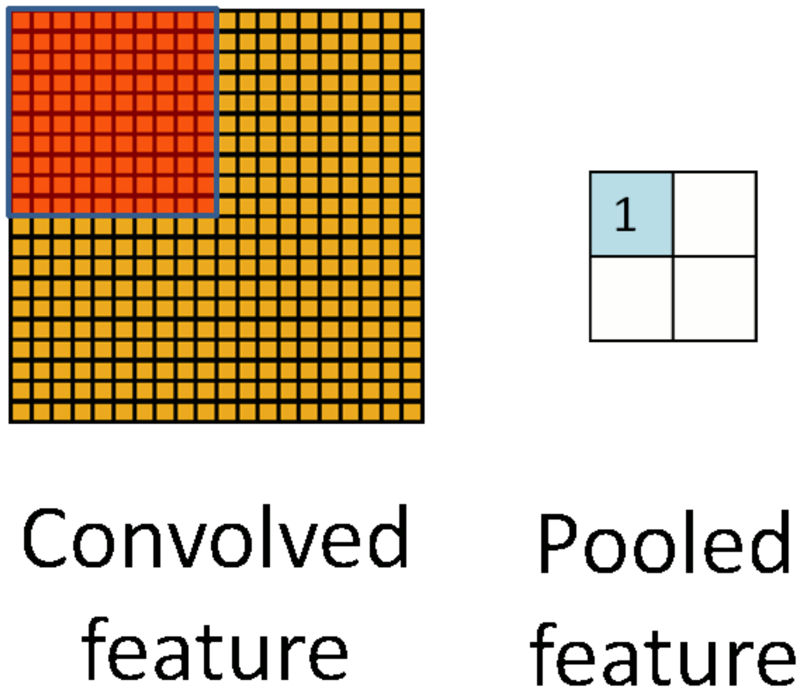
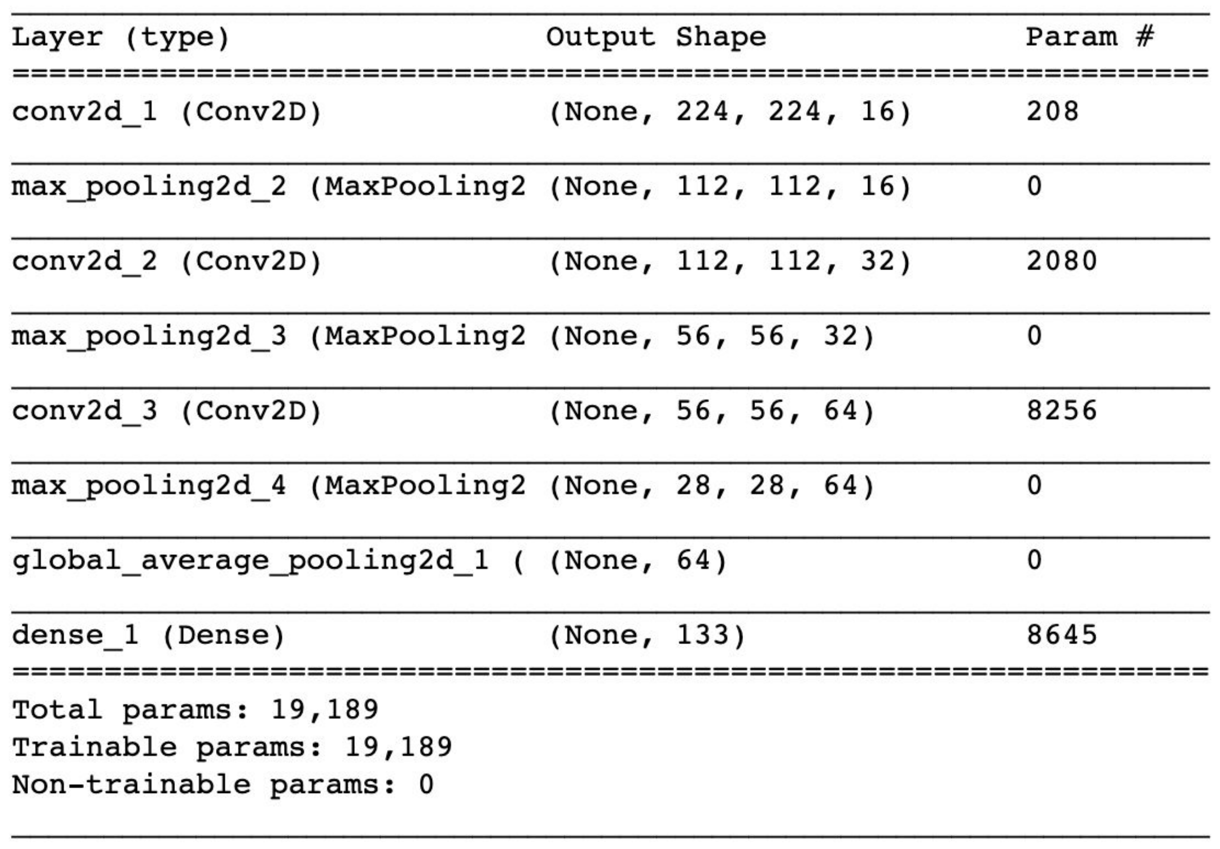


Image credit: Visualization of pooling layers such as MaxPooling and AveragePooling



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               8645
=================================================================
Total params: 19,189
Trainable params: 19,189
Non-trainable params: 0
_________________________________________________________________
```

III. Methodology

Data Preprocessing

RandomResizedCrop & RandomHorizontalFlip is used to train data and resized it to 256 and then center crop to make 224 X 224. Image augmentation gives randomness to the dataset so, it prevents overfitting and I can expect better performance of model when it's predicting toward test_data.

The algorithm for this is written below:-

```
import os
from torchvision import datasets
import torchvision.transforms as transforms
```

```
num_workers = 0
batch_size = 20

data_dir = '/data/dog_images'


data_transforms = {
   'train' : transforms.Compose([
   transforms.Resize(256),
   transforms.RandomResizedCrop(224),
   transforms.RandomHorizontalFlip(), # randomly flip and rotate
   transforms.RandomRotation(10),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
   ]),

   'valid' : transforms.Compose([
   transforms.Resize(256),
   transforms.CenterCrop(224),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
   ]),

   'test' : transforms.Compose([
   transforms.Resize(256),
   transforms.CenterCrop(224),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
   ]),
}


train_dir = data_dir + '/train'
valid_dir = data_dir + '/valid'
test_dir = data_dir + '/test'

image_datasets = {
   'train' : datasets.ImageFolder(root=train_dir,transform=data_transforms['train']),
   'valid' : datasets.ImageFolder(root=valid_dir,transform=data_transforms['valid']),
   'test' : datasets.ImageFolder(root=test_dir,transform=data_transforms['test'])
}

# Loading Dataset
loaders_scratch = {
   'train' : torch.utils.data.DataLoader(image_datasets['train'],batch_size =
batch_size,shuffle=True),
   'valid' : torch.utils.data.DataLoader(image_datasets['valid'],batch_size = batch_size),
   'test' : torch.utils.data.DataLoader(image_datasets['test'],batch_size = batch_size)
}
```

Implementation

1. Human Detection

   Human detection is done using 'Harcascade Frontal Face Algorithm'. The file for this ie. 'haarcascade_frontalface_alt.xml' needs to be downloaded before using it. Haarcascade is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. It can also be done using 'LBP Cascade Algorithm'

2. Dog Detection

   It is done on a pre-trained model of VGG-16 where dog range lies in the dictionary key 151-268.

3. Dog Breed Classification

   This is done in two ways, one from scratch and another using the pre-trained model. The two ways are described below:-

   i. At first train and test data are classified, then CNN classifier architecture is written and loss function are specified.

   ii. Another model is written as CNN classifier using transfer learning.

4. Final app algorithm

   Final algorithm is written where if a human is detected his/her likable dog breed and if a dog is detected his breed is shown.

IV. Results

Model Evaluation and Validation

The OpenCV model for human face detector is able to identify 98% of human face as humans and 17% of dog faces as human.

The dog detector detected 0% of human faces as dog and 98% of dog faces as dog.

The CNN architecture developed from scratch detects 1% of dog breed correctly and the model with transfer learning detects 83% of dog breed correctly.

Justification

The model developed from scratch performed poorly because of less number of convolution layers and fully connected layers. It could have worked better with more number of layers as VGG16 which has 16 convoluted layers.

V. Conclusion

Improvement

1. The algorithm can be made better for identifying multiple dogs and humans in a single image.
2. The model could be more fine tuned to give better accuracy.
3. The top N predicted classes and their probabilities should be returned rather than just one class