PROJECT REPORT :
DOG BREED CLASSIFIER

PROJECT OVERVIEW :

The aim of this project is to identify breeds of dogs from the images provided.
Given an image of a dog, your algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.
Dog breed identification is representative of a set of fine-grained classification problems in which corresponding object parts exist across classes, but the geometry and appearance of these parts vary significantly. dogs are both the most morphologically and genetically diverse species on Earth.
The solution of this problem is also applicable to other fine grained classification problems as well. The methods used can also help in classifying breeds of cats, horses and other species like birds and plants or even models of cars. In real world an identifier like this could be used in biodiversity studies. These studies may also help veterinarians in various ways like breed specific treatments for stray, unidentified dogs that need medical care.
A lot of work has been done in the field of fine-grained classification [1] [2] [4]. One of the earlier works in fine-grained classification was an attempt at identifying plant species by Belhumeur et. al[3]. This approach involved segmenting a leaf and then using shape to determine the species.More relevantly, however, a 2012 paper by Liu et. al attempted dog breed identification.
This project seems interesting to experiment with as dogs have immense diversity and a very loving natures, also I want to expand my understanding of Image Classification, Neural networks and computer vision.

PROBLEM STATEMENT

To identify an estimate of dog's breed given the image of the dog as input. Ideally our algorithm aims at predicting the breed of given image of a dog.If supplied an image of a human, the algorithm will identify resembling dog breed.

EVALUATION METRICS

The evaluation metrics that can be used to quantify the performance of the model is accuracy score on the test dataset.

I decided to choose accuracy score as the metrics as the only limitation to this metric is when our input data is not balanced but in this case of Dog breed classification our data is balanced and so accuracy is the best choice to my knowledge.

BENCHMARK MODEL

The benchmark model for Dog Breed Classifier is using CNN.

The benchmark model includes feeding the input image to a convolutional network in form of 2D array for grayscale images and for colored images it also includes a depth value of 3 for RGB values.

A convolutional network have three main layers namely convolutional layer, pooling layer and relu activation layer.

The Convolutional layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

The function of pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

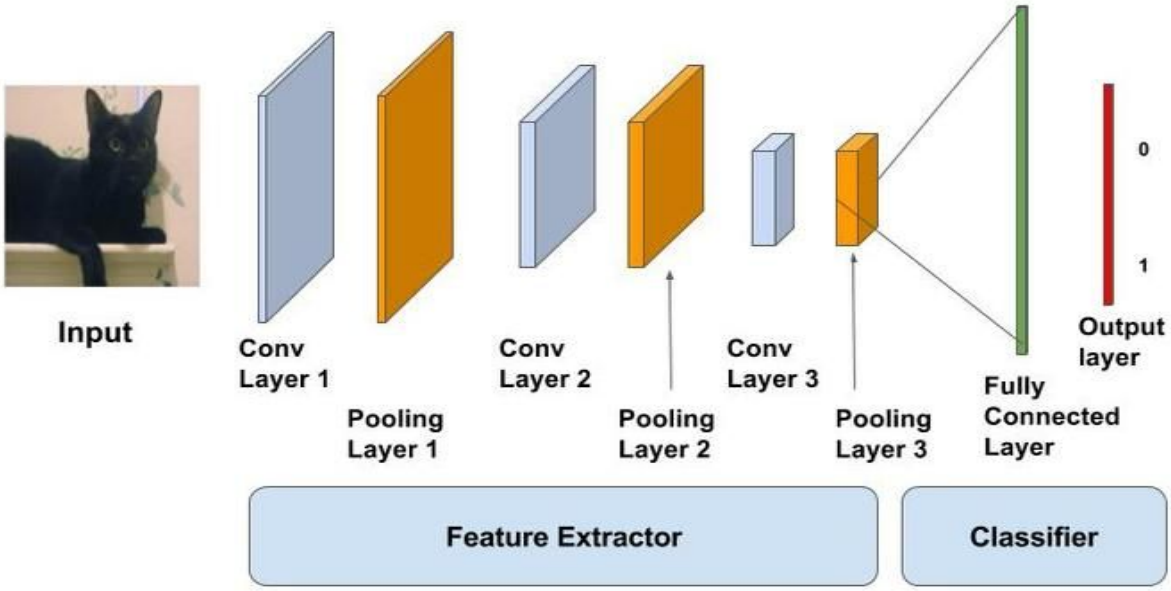
The relu activation layer standardize the input values by setting some minimum threshold.

The output of convolutional network is fed to the fully computational layer which takes as input the image representation in form of a vector and classifies the input into the specified class labels.

All the layers are defined in the __init__ method and how the data will be treated and forwarded among various layers is defined in the forward method of the model. This function is responsible for defining the feedforward behavior of the model.

This is the basic benchmark model of our project.

BASIC WORKFLOW :



The workflow will go as follows :

1. Data Preprocessing - In this step we do all the changes that are required for the image data to be fed as input to the CNN. This involves resizing the image to a specified square size, changing it into a tensor image and normalizing the image.
2. Data Visualization - This step includes visualizing the dataset for better understanding of the input data. Data is visualized in forms of bar graphs and pie charts using matplotlib library.
3. Training the model - After preprocessing and visualizing the data, we feed this data to the convolutional neural network for feature extraction. Convolutional neural network is designed using three layers i.e. Convolutional layer(extracts the main features of the image in different filters), pooling layer(responsible for minimizing the dimensions of the image by extraction the important pixels from the 2D image array) and relu activation layer(standardize the input using the minimum threshold). The output from the CNN which is in the form of 2d arrays is flattened to provide a vector representation of the image so that this data can be fed to the fully computational neural network. The fully computational neural network then classifies the image data as per the class labels provided.

MAJOR 5 PROJECT DEVELOPMENT STAGES ARE AS FOLLOWS :

STEP 1 : Obtain The Data

The first step is to get the data we want to work on and which will be used for training and testing our model.

The dataset that i used contains two files one with dog images and one with humans images. The link to download the dataset is provided in README file. The dog dataset contains total number of 8351 images with 133 dog classes. The humans dataset contains 13233 images of humans which are used to give as output the breed of dog which is most similar to the human in the image.

The dogs dataset is split into training and testing datasets in the ratio of 70:30 and then the training dataset is further split into training and validation dataset in the ratio of 80:20.

The images in the dataset are colored images and so the depth for the convolutional neural network will be 3.

STEP 2 : Data Preprocessing

This step involves preprocessing the data as the data that we obtain includes various forms of issues which are to be corrected before we use it for training our machine learning models.

As this is a project for image classification and our data is in the form of images we need to apply image processing steps to filter our data according to our needs.

The following steps are applied to training, testing and validation data, as these are the basic steps that are required for all the data that is to be fed to the CNN :

- The first step that I took for preprocessing the image data is resizing all the images to a value of 224*224. This helps is to have a standard image size which is necessary for training CNNS. I did this by calling a Resize function provided in the PyTorch library on all the images in the dataset.So after this resizing step all the images that we have in our dataset are of same size.
- Then comes the step of converting the image data into tensor data. This is a mandatory step or we can say a prerequisite for training PyTorch model.Tensor data is similar to a numpy array the only difference is that tensor data can be moved to the GPU for much faster calculations.
- Then comes the normalization step. As images are made up of pixel values from 0 to 255 representing various gray levels for each pixel, so for making out data more relatable we need to change this pixel values by normalizing them into a range of 0 to 1, where 0 represents black, 1 represents white and the values in between represent various colors between white and black.

  This normalization step is done by subtracting all the pixel values by their mean and dividing them by the standard deviation. In PyTorch we have a method named Normalize() to do this.

Data Augmentation Steps : These steps are applied only to the training data with the purpose of adding more variation to the input data to provide better model training. The Data Augmentation steps that I took in my project are as follows :

- RandomResizedCrop - This step randomly scale the images and crops them to a size of 224.This will extract a patch of size (224, 224) from our input image randomly. So, it might pick this path from top-left, bottom-right or anywhere in between.
- RandomHorizontalFlip - Once we have our image of size (224, 224), we can choose to flip it. This is another part of data augmentation and flips our data horizontally in either direction.
- RandomRotation - This step rotates our data randomly and thus provide more variations in data

So all these data augmentation steps can be used for the better learning of our model as it will be trained with images in all directions and positions. And say if we have an image of a dog lying on the ground, training with data which augmented provided more probability that it will be predicted correctly as which breed the dog belongs to.

STEP 3 : EXPLORING THE DATA :

To explore the data,we first check the number of humans and dogs detected in the humans dataset. This is done with the help of Haar feature-based cascade classifiers. Ideally it should return 100% detected faces of humans and 0% for dogs but it returns 98% detected as humans and 17% detected as humans.

Then we check the number of humans and dogs detected in the dogs dataset. This is done with the help of VGG16 model and ImageNet, a very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. Ideally it should return 100% dogs detected and 0% humans and it returns 100% dogs detected in dogs images and 2% humans.

STEP 4 : IMPLEMENTATION :

- BUILDING THE MODEL FROM SCRATCH :
  For the purpose of classifying breeds of dogs based on their image provided, I trained a Convolutional Neural Network from scratch.
  The first thing that we need to do for building a model from scratch is some research work to get an insight of the previous work that has been done in this field and what model configuration gave what model accuracy.So after doing my research I decided to use 5 convolutional layers in my model.
  We can define a convolutional layer using the following format:
  self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0)

Where the arguments explanation is as follows :

- in_channels - The number of inputs (in depth), 3 for an RGB image, for example.
- out_channels - The number of output channels, i.e. the number of filtered "images" a convolutional layer is made of or the number of unique, convolutional kernels that will be applied to an input.
- kernel_size - Number specifying both the height and width of the (square) convolutional kernel.
- stride - The stride of the convolution. If you don't specify anything, stride is set to 1.
- padding - The border of 0's around an input array. If you don't specify anything, padding is set to 0.(Stride and padding are optional arguments and have a default value of 1 and 0 respectively.)

The convolutional layers in my model are as follows :

```
self.conv1 = nn.Conv2d( 3, 16, 3)
self.conv2 = nn.Conv2d(16, 32, 3)
self.conv3 = nn.Conv2d(32, 64, 3)
self.conv4 = nn.Conv2d(64, 128, 3)
self.conv5 = nn.Conv2d(128, 256, 3)
```

Pooling Layers
Pooling layers take in a kernel_size and a stride typically the same value as is the d own-sampling factor.The pooling layer that I used in my code is the following.

```
self.pool = nn.MaxPool2d(2,2)
```

This layer down-sample the input's x-y dimensions, by a factor of 2.

Computational Layers :
These layers are responsible for classifying the data based on the class labels provided In the training data.The fully computational layers in my model are :

```
self.fc1 = nn.Linear( 256 * 5 * 5, 500)
self.fc2 = nn.Linear(500, 133)
```

These layers take in as input the input and output dimensions of the image.
The neural networks train themselves with the loss functions and optimization criteria Which is defined as follows :
Loss Functions : These functions are helpful to train a neural network. Given an input and a target, they calculate the loss, i.e difference between output and target variable.

The loss function that I used in my model is CrossEntropyLoss() as it is considered the best for multi class classification purposes.

Optimisation Algorithms : These Algorithms are used to update weights and biases i.e. the internal parameters of a model to reduce the error.The optimization algorithm that I used is Adams as it works well in practice, is faster, and outperforms other techniques. So this was everything that i used to train my model and using this model I got an accuracy of 29%.

- CREATE A CNN USING TRANSFER LEARNING :
  Then I used transfer learning to create a CNN that can identify dog breed from images. In transfer learning, we can leverage knowledge (features, weights etc) from previously trained models for training newer models.So the model that I chose for this purpose is ResNet architecture since it is known to perform the best in cases of image classification.Also the original work that ResNet was trained on matches a lot witch this task. I chose to use all the pretrained parameters and just added one last fully computational layer to provide an output from 133 class labels provided in the dataset.

  The accuracy that I got using this model to predict dogs breed is 82%.

STEP 5 - WRITING THE ALGO AND TESTING THE MODEL

After training my model and getting an accuracy of 82% on the testing data. The last step of my project is to write an algo as what would happen when a particular image is fed to the model to predict the output.The algorithm involves the following conditions :

- if a **dog** is detected in the image, return the predicted breed.
- if a **human** is detected in the image, return the resembling dog breed.
- if **neither** is detected in the image, provide output that indicates an error.

So when the path of an image is passed to the model it predicts the results with 82% accuracy. The following are the things that can possibly be done to improve the accuracy of the model :

1. If the training loss is decreasing till the end of the numbers of epochs we can increase the number of epochs to a greater value to provide more optimization to our model.
2. We can also work with data augmentation process to provide more variation to the data then there already is.
3. We can also add more data to the datasets to provide the model with a larger dataset than provided. This will help in a better training of the model as more data generally means more learning.
4. We can also add more layers to our convolutional network by starting it with number of 8 filters in place of 16.This will help us extract more information in the input image.

REFERENCES

[1] N. Z. et al. Deformable part descriptors for fine-grained recognition and attribute prediction. 2013 IEEE International Conference on Computer Vision, 2013.

[2] O. M. P. et al. Cats and dogs. 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012.

[3] P. N. B. et al. Searching the world's herbaria: A system for visual identification of plant species. 2008.

[4] E. Gavves. Fine-grained categorization by alignments, 2013.