UDACITY MACHINE LEARNING NANO DEGREE

CAPSTONE PROJECT

CLASSIFICATION OF DOG BREED USING CONVOLUTION NEURAL NETWORKS

Gowtham Mallikarjuna

- Definition
  - Project Overview
  - Problem Statement
  - Metrics
- Analysis
  - Data Source
  - Data Exploration
  - Exploratory Visualization
  - Algorithms and Techniques
  - Benchmark
- Methodology
  - Data Preprocessing
  - Implementation
    - Detect Humans
    - Detect dogs
    - Detect the dog classification
  - Create a CNN to Classify Dog Breeds (from Scratch)
    - Specify Loss Function and Optimizer
    - Train and test the model
  - Use a CNN to Classify Dog Breeds (using Transfer Learning)
    - Specify Loss Function and Optimizer
    - Train and test the model
    - Algorithm to detect input image
  - Refinement
- Results
  - Model Evaluation and Validation
  - Justification
- Conclusion
  - Reflection
  - Improvement
  - Reference

Definition

Project Overview

This project is part of Udacity Machine Learning Nanodegree. Objective of the project is to build a deep learning model using convolution neural networks to classify the given image to one of the dog breeds.

Problem Statement

Given an input image, the algorithm will first detect if the input image is a dog image or human image, if the input image is of dog the algorithm will detect the breed. If the input image is human image, the algorithm will detect the reselembing dog breed. If the input device is neither dog of human image the algorithm will print a friendly message.

I follow these steps to active the project goal.

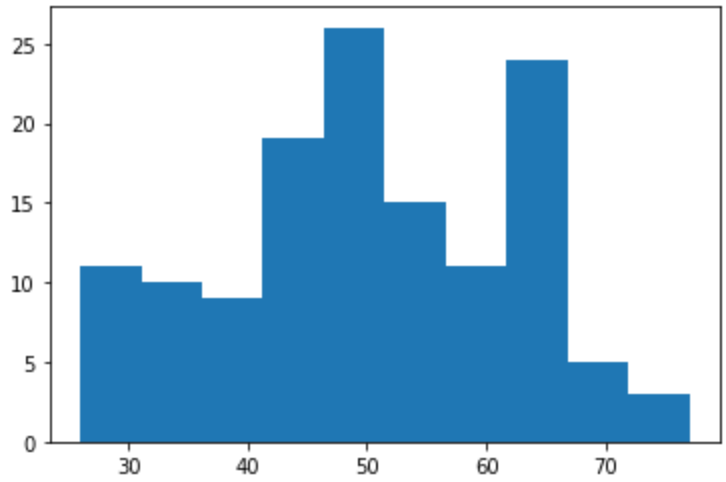
- We use opencv implementation for predicting if the given image is the image of a human.
- Use pretrained vgg16 classification model to predict if the given image the image of dog.
- After finding if the image is of a dog or human. We train our model using transfer learning to utilize the pre trained network to accurately predict the dog breed.

Further implementation details are in the following sections in the document.

Metrics

I will be using accuracy as metrics. We have 133 classes to predict each class have fairly evenly distributed number of samples in the training dataset. Although number of images are in the range of 30 to 90 in each of the categories which is less for an image training, we will be using transfer learning to learn the features without having to learn from scratch. That will give us a lot of leverage in terms of prediction accuracy.

Accuracy of prediction would be the right metric to measure the performance given the nature of problem with multi class image classification. We are concerned about the number of correct predictions out of all the test cases.
We would go for precision or recall if we are more concerned about getting better true positives or reduced false negatives. In this case it’s important to predict the correct prediction out of all given samples.



In the above plot we can see the distribution of number of images in each dog breed classification.
Distribution looks fairly normal. Since we see the count distribution doesn't looks largely skewed also we are using transfer learning to learn common image features and use the training images to only train the last layer. We should be fine using this reasonably unbalanced dataset and accuracy of prediction as a metric without the need of any balancing techniques for training.

Accuracy of prediction will be measured by total current predictions out of number of test cases scaled to 100.

Analysis

Data Source

Datasets are provided by Udacity. Downloaded from the below locations.
`dogimages_url = "https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip"`
`humanimages_url = "https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip"`

Data Exploration

Both dog and human datasets are downloaded and unzipped to project folder.
Human images consists of the 13233 number of images to train human detector.
Dog images are placed in 3 folders train, test and val.
Each of the folders further have 133 sub folders for each of the dog breeds.
Each of the 133 folders in turn have several dog images for training, validation and testing.

Storing the data in this folder format enable the pytorch/keras to read the data with folder name and classification labels

Here is the count of images in each of the dog breed classification in training set

Here is the distribution of count of number of images in each of the dog classification. 64 : ./001.Affenpinscher

58 : ./002.Afghan_hound

52 : ./003.Airedale_terrier

63 : ./004.Akita

77 : ./005.Alaskan_malamute

64 : ./006.American_eskimo_dog

50 : ./007.American_foxhound

66 : ./008.American_staffordshire_terrier 34 : ./009.American_water_spaniel

50 : ./010.Anatolian_shepherd_dog

66 : ./011.Australian_cattle_dog

66 : ./012.Australian_shepherd

46 : ./013.Australian_terrier

69 : ./014.Basenji

73 : ./015.Basset_hound

59 : ./016.Beagle

62 : ./017.Bearded_collie

50 : ./018.Beauceron

48 : ./019.Bedlington_terrier

62 : ./020.Belgian_malinois

64 : ./021.Belgian_sheepdog

47 : ./022.Belgian_tervuren

65 : ./023.Bernese_mountain_dog

62 : ./024.Bichon_frise

37 : ./025.Black_and_tan_coonhound 41 : ./026.Black_russian_terrier

64 : ./027.Bloodhound

35 : ./028.Bluetick_coonhound

74 : ./029.Border_collie

52 : ./030.Border_terrier

56 : ./031.Borzoi

65 : ./032.Boston_terrier

45 : ./033.Bouvier_des_flandres 64 : ./034.Boxer

53 : ./035.Boykin_spaniel

65 : ./036.Briard

50 : ./037.Brittany

57 : ./038.Brussels_griffon

69 : ./039.Bull_terrier

53 : ./040.Bulldog

69 : ./041.Bullmastiff

63 : ./042.Cairn_terrier

50 : ./043.Canaan_dog

64 : ./044.Cane_corso

53 : ./045.Cardigan_welsh_corgi

67 : ./046.Cavalier_king_charles_spaniel 54 : ./047.Chesapeake_bay_retriever

54 : ./048.Chihuahua

50 : ./049.Chinese_crested

50 : ./050.Chinese_shar-pei

62 : ./051.Chow_chow

49 : ./052.Clumber_spaniel

47 : ./053.Cocker_spaniel

57 : ./054.Collie

50 : ./055.Curly-coated_retriever

65 : ./056.Dachshund

71 : ./057.Dalmatian

50 : ./058.Dandie_dinmont_terrier

47 : ./059.Doberman_pinscher

60 : ./060.Dogue_de_bordeaux

61 : ./061.English_cocker_spaniel

53 : ./062.English_setter

53 : ./063.English_springer_spaniel

39 : ./064.English_toy_spaniel

42 : ./065.Entlebucher_mountain_dog 33 : ./066.Field_spaniel

34 : ./067.Finnish_spitz

63 : ./068.Flat-coated_retriever

51 : ./069.French_bulldog

47 : ./070.German_pinscher

62 : ./071.German_shepherd_dog

48 : ./072.German_shorthaired_pointer 42 : ./073.German_wirehaired_pointer 41 : ./074.Giant_schnauzer

44 : ./075.Glen_of_imaal_terrier

64 : ./076.Golden_retriever

43 : ./077.Gordon_setter

40 : ./078.Great_dane

59 : ./079.Great_pyrenees

46 : ./080.Greater_swiss_mountain_dog 56 : ./081.Greyhound

61 : ./082.Havanese

46 : ./083.Ibizan_hound

50 : ./084.Icelandic_sheepdog

37 : ./085.Irish_red_and_white_setter

53 : ./086.Irish_setter

66 : ./087.Irish_terrier

51 : ./088.Irish_water_spaniel

53 : ./089.Irish_wolfhound

58 : ./090.Italian_greyhound

57 : ./091.Japanese_chin

44 : ./092.Keeshond

35 : ./093.Kerry_blue_terrier

44 : ./094.Komondor

49 : ./095.Kuvasz

43 : ./096.Labrador_retriever

50 : ./097.Lakeland_terrier

46 : ./098.Leonberger

42 : ./099.Lhasa_apso

34 : ./100.Lowchen

48 : ./101.Maltese

29 : ./102.Manchester_terrier

58 : ./103.Mastiff

42 : ./104.Miniature_schnauzer

31 : ./105.Neapolitan_mastiff

50 : ./106.Newfoundland

46 : ./107.Norfolk_terrier

26 : ./108.Norwegian_buhund

45 : ./109.Norwegian_elkhound

33 : ./110.Norwegian_lundehund

44 : ./111.Norwich_terrier

54 : ./112.Nova_scotia_duck_tolling_retriever 39 : ./113.Old_english_sheepdog

35 : ./114.Otterhound

63 : ./115.Papillon

30 : ./116.Parson_russell_terrier

48 : ./117.Pekingese

53 : ./118.Pembroke_welsh_corgi

31 : ./119.Petit_basset_griffon_vendeen 39 : ./120.Pharaoh_hound

28 : ./121.Plott

32 : ./122.Pointer

44 : ./123.Pomeranian

50 : ./124.Poodle

34 : ./125.Portuguese_water_dog

30 : ./126.Saint_bernard

41 : ./127.Silky_terrier

30 : ./128.Smooth_fox_terrier

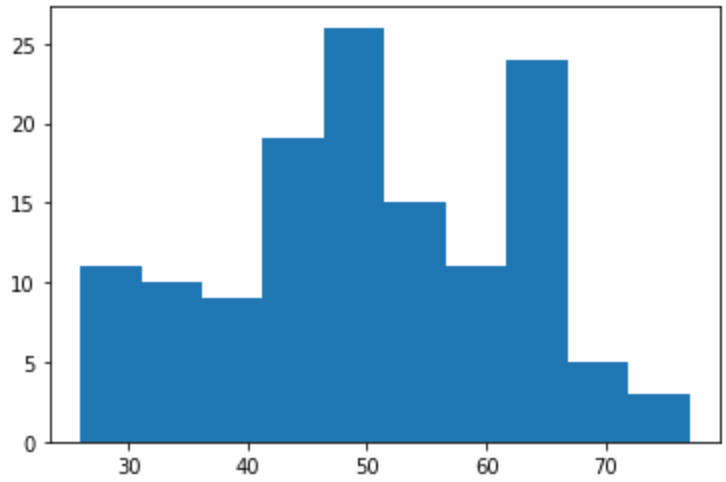
48 : ./129.Tibetan_mastiff

44 : ./130.Welsh_springer_spaniel

30 : ./131.Wirehaired_pointing_griffon 26 : ./132.Xoloitzcuintli

30 : ./133.Yorkshire_terrier

Distribution of count of images in each category looks as below

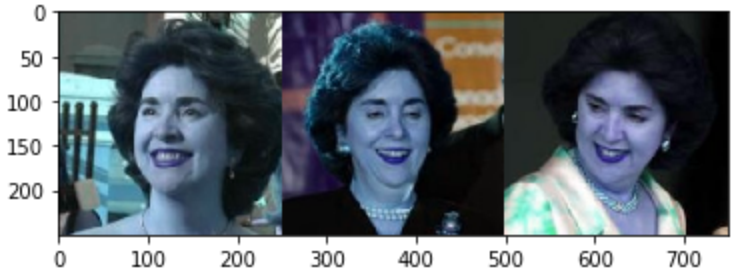


The distribution doesnt looks way too skewed.

Exploratory Visualization

Sample dog images

Sample human images



Algorithms and Techniques

There are multiple steps involved in achieving the results which will be discussed in detail later in this document. We will be mainly using these algorithms.

1. OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

   We use opencv implementation algorithm to predict the human since we don't have sufficient/required training data to classify the human face vs non human face in the image. Although we can use the dog image vs human image to train the classification, but with the initial analysis the pre trained models gave fairly acceptable prediction for our problem with 98 to 99 % accuracy with repeated tries.

   Approach here is that the algorithm will predict the number human faces in the given image, if the count is greater than 0 we say the image is of human image or else we continue further with the next steps.

2. Predict if the given image is an image of dog.

   Here again we are using pre trained model and predict if the given image is an image of a dog, since we don't have sufficient data to predict the image for dog vs non dog image.
   And also the VGG16 model is trained on large amount of crowd sourced dataset, with works with great accuracy for our problem in this step.

   VGG16 have 1000 image classification. After the network predicting the index of the classification and analyzing the classification value,we can see the index starting from 151 to 268 corresponds to different dog breed classifications.
   We use these values to predict if the image is an image of a dog or not.

3. Pytorch based CNN implementation from scratch to classify the dog images
   We are using CNN neural network algorithm to solve the image classification problem since this techniques are proven to work best with the image data. Several convolution layers are stacked with number of kernels and nodes in each layer are taken mostly with the trail and prior knowledge in the area.
   Each of the CNN layers will learn different features of the image like lenes, edges, curves, color gradations etc, we can already see the model is trying to mimic the human behaviour like we also try to see different features of the image to understand what the image is.

4. Transfer learning technique to leverage pre trained weights to classify the dog images

   Initial model implemented from scratch gave us only limited accuracy given the amount of data in the training data also the computation complexity involved.
   In the next steps we use transfer learning where we can use the pre trained network weights which are trained on large amount of images on a complex network using huge computation power probably for several days of training.

We can utilize this learning to give head start for out model.
We first use this pre trained network to predict all the initial features in the image and add an additional layer at the end to customize the model to predict the dog breeds as needed in our problem statement.

5. As a final step we combine all the above implementation in the simple python function to make the required prediction. More on this later..

Benchmark

We will use the model trained from scratch as our benchmark, that's about 10 to 14% accuracy. Although we could have done further training and fine tuning to improve the accuracy, learning is limited by the limited amount of data we have.
Will try to improve the performance by using the pre trained network weights using transfer learning technique.

Methodology

Data Preprocessing

Input images needs to be converted to numpy for any numerical computation.
Before feeding to the network. We perform several processing steps.

1. Random horizontal flip. Images will be flipped along vertical axes horizontally so as to make the model training more robust and generalize better. This is performed only on training data. Validation and testing data is fed to the network without this transformation.
2. Image data is normalized to avoid any bias due to different scale/magnitude of the numbers.
3. Perform center crop and resize transformation on all the train, val and test dataset.

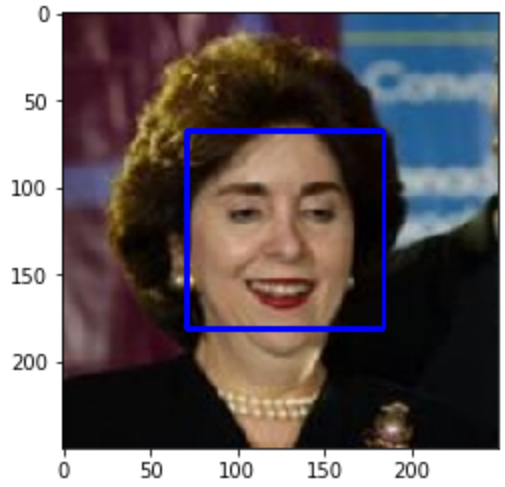
Implementation

Project is implemented in multiple steps

Detect Humans

we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

We use the pre trained face detector. The algorithm will detect the fact of a human in the given image.



The pre trained algorithm will detect the number of faces in the given images. If the number of faces greater than 0 we classify the image as a human image.

Next step we will test the function to detect the human face. By passing sample of human images and sample of dog images. Algorithm gives fairly good performance of about 98% of accuracy.

Detect dogs

Next step of the project is to detect if the input image is an image of a dog.
We use Vgg15 pre trained classifier trained on imagenet dataset.

Download the pre trained model weights using pytorch

```
import torch
import torchvision.models as models
# define VGG16 model
VGG16 = models.vgg16(pretrained=True)
```

Vgg16 will classify the image into one of the 1000 categories on which it is trained on. Categories of dogs appear in the index between 151-268 (inclusive). We can use this information to classify if the input image is dog or not. If the model predicts any index number between this range the input image is dog otherwise not.

Dog detection can be done using this simple function.

```
def dog_detector(img_path):
# in VGG16 index 151 to 268 are dog classifications
    return VGG16_predict(img_path)>= 151 and VGG16_predict(img_path)<=268
```

Since Vgg16 is trained on a large number of images it is fairly accurate for this purpose without further training.

Detect the dog classification

Create a CNN to Classify Dog Breeds (from Scratch)

In this step we will implement a deep learning classifier from scratch using pytorch convolution neural networks. Convolution neural networks does great job working with image dataset since each of the convolution layers tries to learn different features of the image like lines, edges and so on.

This involves several steps.

Preproces and specify data loaders.

We apply several transformation function to datasets as follows.

```
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

Random Horizontal flip is applied only to train dataset to make the model more generalized. Model should be able to predict the dog image even if the image is flipped on horizontal axis.

Image is converted to torch tensor to perform computation on the cuda gpu.

Image data is normalized to avoid any bias due to different scale/magnitude of the numbers.

Build network architecture

We use simple 3 convolution layers with 64 nodes, kernel size of 3 and 2d max pooling between each layer.
One fully connected node with 256 nodes.
A final fully connected layer with 133 nodes for each of the classification categories.

Specify Loss Function and Optimizer

We use categorical cross entropy loss function for multiple class classification.
SGD optimizer with learning reat or 0.001

Train and test the model

We train the model with pytorch library. Training the model for 8 epochs will get us a fair starting performance of 10% accuracy on the validation data. We will further improve the performance of the prediction using transfer learning.

Use a CNN to Classify Dog Breeds (using Transfer Learning)

Building image classification model from scratch requires lot of training images and the great amount computation power/time. After each training fine tuning the model by testing different hyper parameters will take a great amount of time.

We can use the pre trained network weights and tune the model for our custom images will get us great performance advantage. Pretrained network weights will have layers that can detect the basic shapes and image elements like lines, edges and so on.

Transfer learning involve the similar steps as our previous model implementation from scratch.

Preproces and specify data loaders.

We are going to use the same data loading /image processing techniques as our previous implementation.

Build network architecture

Instead of building model architecture from scratch. Let's use resnet18 pretrained network. Freeze the pre trained network weights to avoid training all the initial image features. Add a final fully connected layer with output nodes for each of the classification classes.

In this case we are not using any softmax function in the output layer because we are interested in the prediction of the image into one of the classes and not interested in prediction probabilities.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 133)
model_ft =model_ft.to(device)
```

We need to transfer both the dataset and the model to cuda device to train on available gpu.

Specify Loss Function and Optimizer

We use categorical cross entropy loss function for multiple class classification.
SGD optimizer with learning reat or 0.001

Train and test the model

Using transfer learning gives us the prediction accuracy of 84% with just 8 epochs. That's a great improvement from the model built from scratch.

Algorithm to detect input image

As a final step to achieve our project goal. We will build a simple function that will take an input image and perform these tasks.
Predict if the image is dog or human or neither of these.
Predict the dog classification and provide the respective output.

```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    if haar_face_detector(img_path):
        print('hello human')
        prediction = predict_breed_transfer(img_path)
        print(prediction)
    elif dog_detector(img_path):
        print('hello dog')
        prediction = predict_breed_transfer(img_path)
        print(prediction)
    else:
        print("couldn't detect dog or human image")
```

Refinement

Neural network is tried and tuned for different optimizers, learning rates.
Initially network is implemented from scratch and tuned for different network parameters like number of layers and number of nodes in each network.
After several trials the model performance didn't improve much. I could get the accuracy from 8% to around 11%. This is not a significant improvement. This is expected given the limited amount of training data.

Most of the image learning involves transfer learning. Because we can use the pre trained network parameters to greatly leverage the learning accuracy without having to train the large and complex network to learn all the initial image features.

The network parameters itself aren't changed. I only replaced the last layer of the network with a fully connected layer with 133 nodes corresponds to our number of dog breed classes.

We didn't have to use sigmoid layer in the pytorch since we are interested in the prediction if the dog image belongs to the class and not interested in the prediction probabilities.

For fine tuning i only had to play around with optimizer and learning rate.
I tried with RMSProp it gave a decent performance of approximately around 78%.

SGD gave a good performance improvement of around 84% accuracy with only 8 epochs of training.
For learning rate i have used learning decay to decrease the learning rate after 7 epochs.
Here is the code to update the learning rate during training. Although i did train the network only for 8 epochs that gave me acceptable performance.

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_scratch, step_size=7, gamma=0.1)
```

Results

Model Evaluation and Validation

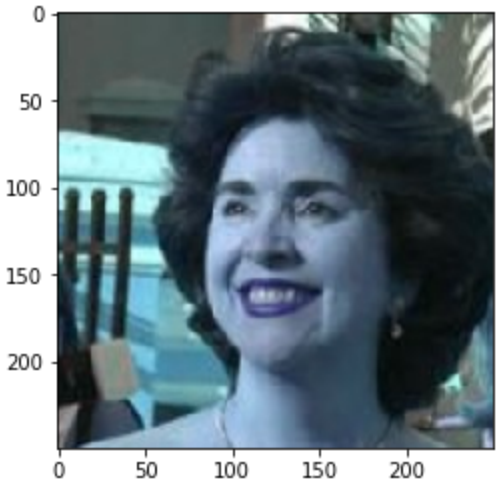
Model is first evaluated on validation data during training. We can see the acceptable performance of 84% accuracy with just 8 epocs.

And then I tested the model on unseen data set reserved for final test. I saw consistent performance on unseen data set with 84% accuracy.

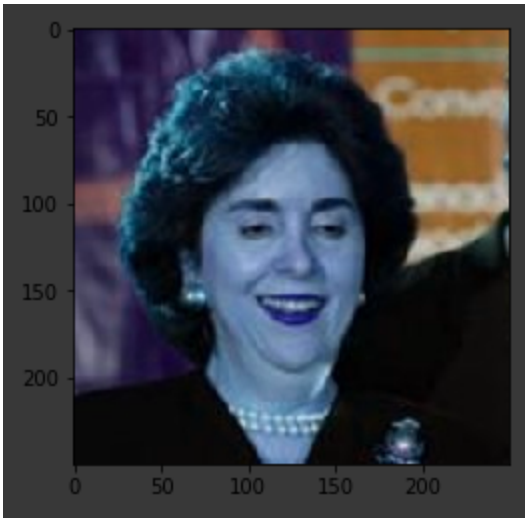
I can see model is very reliable within the accuracy limits since it consistent performance on both validation and unseen test data.

Aslo the model is tested on real world data set downloaded from internet. It was able to identify if the image is dog or human or neither accurately.
It didn't give decent performance of predicting dog breed with scope for improvement (discussed below)

Some example prediction from real world images.

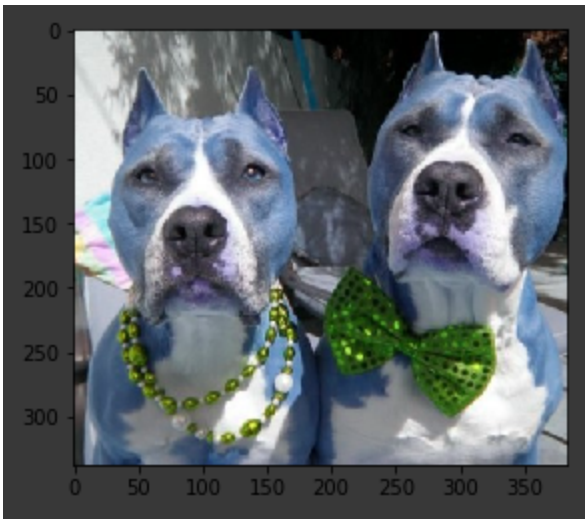



```
hello human
['123.Pomeranian']
```




```
hello human
['124.Poodle']
```


```
hello dog
['124.Poodle'
```




```
hello dog
['124.Poodle']
```

Justification

Neural network implemented from scratch give a decent performance of 10 to 14% with the limited amount of data which is trained on.
I have leveraged the pre trained network weights to improve the performance to usable level.

Conclusion

When tested on custom datasets from real world, network was able to give decent performance with lot of scope for improvement. It can further be improved as detailed in the document below.

Reflection

To summarize objective of the project to analyse the input image and perform a series of steps.

1. Predict if the input image is an image of dog
2. Predict if the input image is an image of human
3. If the input image is an image of the dog, algorithm will predict the class of dog
4. If the input image is an image of human image the algorithm will predict the closest resemblance of dog class

Improvement

Model performance can be further improved using following techniques.

1. Try different data augmentation methods to make model more robust
2. Try other optimizers like RMSProp with different learning rate and momentum
3. Try implementing more complex network architecture with,
    a. More nodes in each layer
    b. More CNN layers
    c. More linear layers in the end
    d. Batch normalization
    e. Different kernel size for convolution layers
4. Try other pre trained networks
5. Train the model for more epochs to get better performance. Pay attention to overfitting
6. Use different batch size

Reference

1. https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce
2. https://pytorch.org/docs/stable/nn.html
3. https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md