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.

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.

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

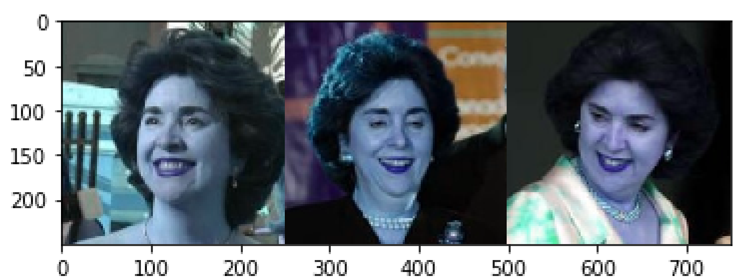
Both dog and human datasets are downloaded and unzipped to project folder.
Human images consists of the 13233 number of images to train human detector.

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

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.
2. Pytorch based CNN implementation from scratch to classify the dog images
3. Transfer learning technique to leverage pre trained weights to classify the dog images

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.

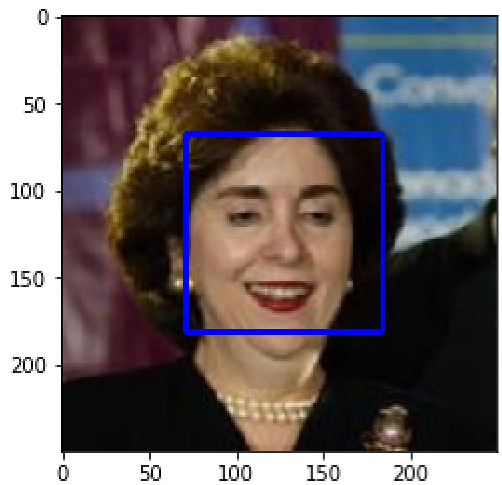
1. Random horizontal flip. Images will be flipped along vertical axes horizontally so as to make the model training more robust and generalize better. This is performed only on training data. Validation and testing data is fed to the network without this transformation.
2. Image data is normalized to 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.
For the transfer learning, last layer updated with fully connected layer with 133 nodes to classify the 133 classify the dog classes.

Results

Model Evaluation and Validation

Model is evaluated with the validation dataset. Transfer learning gave 84% of accuracy with 8 epocs.
Model gave consistent performance of 84% accuracy on test dataset as well.

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