Machine Learning Engineer Nanodegree

Capstone Proposal

CNN Project: Dog Breed Classifier

Shikher Srivastava

I. Definition

Project Overview

The Dog breed classifier is a well-known problem in Machine Learning. The problem is to identify a breed of dog if dog image is given as input. So if dog owners are unsure about their dog's breed, a dog-breed classifier can help them identify their dog's breed.

Machine Learning/Deep Learning and Computer Vision helps you build machine learning models where you train a model to recognize the dog breed. CNN was a huge invention in deep learning, this gave a boost to lots of applications in visual recognition. I built a CNN from scratch using Pytorch and use some of the popular CNN architectures for our image classification task.

My motivation of working on this project comes from my interest in the field of computer vision and deep learning and solving problems by deploying solutions through AWS SageMaker using the knowledge learn from this nanodegree course.

Problem Statement

The objective of the project is to build a machine learning model that can be used within web app or any operating system app to process real-world, user-supplied images. I used CNNs and pretrained models in solving this problem.

The models used have to perform following tasks:

- **Dog face detector**
  Given an image of a dog, the model will identify an estimate of the canine's breed.

- **Human face detector:**
  If supplied an image of a human, the model will identify the resembling dog breed.

We used the OpenCV Haar feature-based cascade classifier which is explained detailed on solution statement section. We imported this classifier using and gave outputs on human images. We used VGG-16 pre-trained model to detect dog breeds. We just load the model to our workspace and using the model, test our images.

Metrics

The expected accuracy result for CNN from scratch should be at least %10.

The expected accuracy result for using transfer learning should be at least %60.

Accuracy is used to evaluate of CNN from scratch Model using transfer learning.
Accuracy is be calculated with the help of the confusion matrix. $Accuracy$ = $(TP+TN)/($ $TP+TN+FP+FN$)

TP: True Positive
TN: True Negative
FP: False Positive
FN: False Negative

The accuracy can also define as –
Accuracy = (number of correct predictions)/ (the total number of input samples)

The main reason to use accuracy as a metric of performance is-

Considering the accuracy paradox which states-

"Accuracy Paradox for Predictive Analytics states that Predictive Models with a given level of Accuracy **may** have **greater** Predictive Power than Models with **higher** Accuracy."

As in this case, as the training set of multiple classes contains slightly imbalanced classes, which means along with accuracy as a metric of performance we will need to use multi-class **log loss** which will punish the classifier if the predicted probability leads to a different label than the actual and cause higher accuracy.

In my implementation, I used **CrossEntropyLoss** as the loss function along with accuracy as the metric of performance.

```
torch.nn.CrossEntropyLoss()
```

II. Analysis

Data Exploration

For this project, the input format is of image type, we input an image and identify the breed of the dog.

The following datasets is provided used (by Udacity):

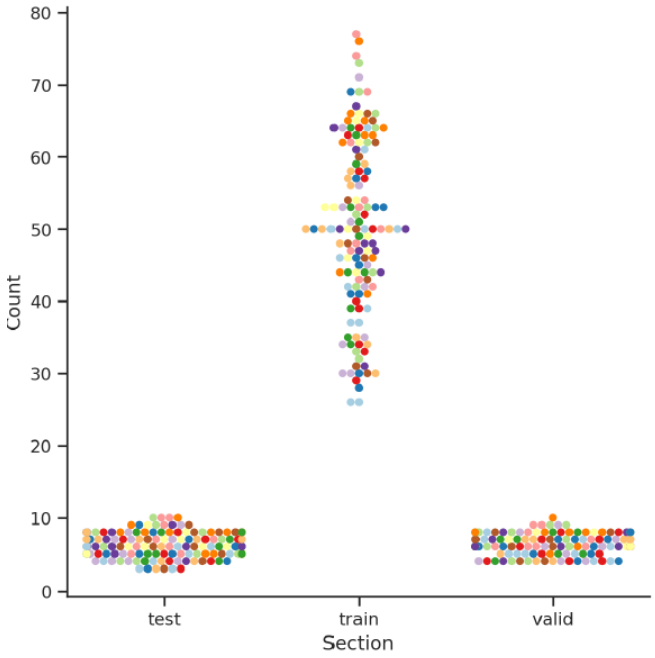
1. Dog Images: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
2. LFW: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

- **The dog dataset**
  with 8351 dog images which are sorted into train (6,680 Images), test (836 Images) and valid (835 Images) directories. Each of this directory (train, test, valid) have 133 folders corresponding to dog breeds. The images are of different sizes and different backgrounds, some images are not full-sized. The data is not balanced because the number of images provided for each breed varies. Few have 4 images while some have 8 images.

- **The human dataset**
  with 13233 human images which are structured in folders by celebrities' name (5750 folders). All images are of size 250x250. Images have different background and different angles. The data is not balanced because we have 1 image for some people and many images for some.

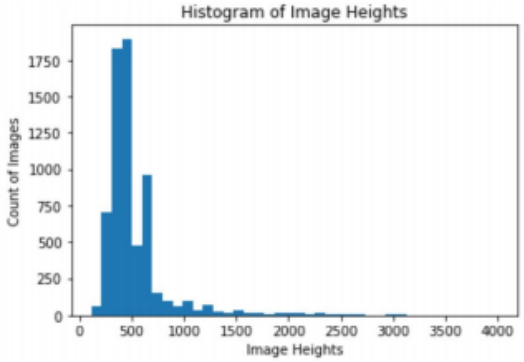
Exploratory Visualization

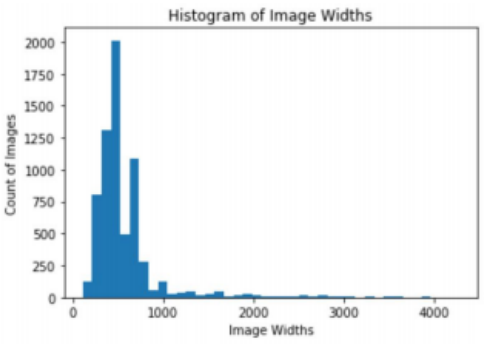
On the following plot you can see the count of images of each dog breed across the sections training:

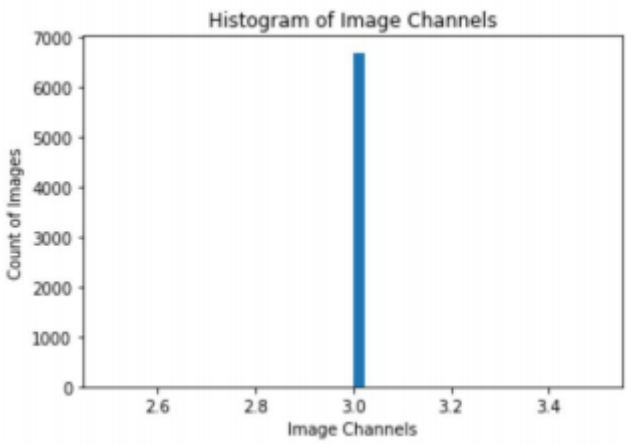


Here from the above graph/chart we can see that for the train dataset we have a slight imbalance. The most common dog breed has 77 images and dog breed and the least common dog breed has 77 images. Although this dataset imbalance could have been problem, but we have overcome it by using techniques like log loss function and data augmentation.

I visualised the heights, widths and channels of the images in the dataset to get an idea of the variations in their shapes.







From the above graphs we can see that all the images have three channels, but their heights and widths differ. We can also see that the mostly of the images have their heights and widths lesser than 1000 px and some outliers in the range 1000-4000 px.

Algorithms and Techniques

In this project, firstly, we detected the image was human or dog. We used openCV's Haar feature-based cascade classifiers for human detection and pre-trained VGG-16 model for dog breeds classifying.

CNNs are a kind of deep learning model that can learn to do things like image classification and object recognition. They keep track of spatial information and learn to extract features like the edges of objects in something called a convolutional layer. The convolutional layer is produced by applying a series of many different image filters called convolutional kernels to the input image.

Other than the CNNs, OpenCV's implementation of Haar feature cascade classifier was used to perform face detection of the human images.

On tasks involving computer vision CNNs outperforms other Neural Networks like MLP by a far. This project requires two CNNs (model from scratch and model from transfer learning) mainly for classifying the breed and one for detecting dogs

It is very efficient to use pre-trained networks to solve challenging problems in computer vision. Once trained, these models work very well as feature detectors for images they weren't trained on. Here we'll use transfer learning to train a network that can classify our dog photos.

Here, I chose the VGG16 model. I thought the VGG16 is suitable for the current problem. Because it already trained large dataset. And it performed really well (came

2nd in ImageNet classification competition) The fully connected layer was trained on the ImageNet dataset, so it won't work for the dog classification specific problem. That means we need to replace the classifier (133 classes), but the features will work perfectly on their own.

Hence, I selected VGG16 pre-trained model for transfer learning. I froze all feature parameters. I just changed the last fully connected layer output as 133 and trained classifier again. In this model, cross entropy loss criteria was selected because of making classification and Adam optimizer was selected.

For performing this multiclass classification, I used Convolutional Neural Network to solve the problem.

The solution involves steps:

1. Import the necessary dataset and libraries, Pre-process the data and create train, test and validation dataset. Perform Image augmentation on training data.
2. Detect human images, we can use existing algorithm like OpenCV's implementation of Haar feature based cascade classifiers.
3. Create dog-images detector where we used a pretrained VGG16 model. where we used a pretrained VGG16 model.
4. Create a CNN to classify dog breeds from scratch, train, validate and test the model. I will implement this using PyTorch.
5. Create a CNN to Classify Dog Breeds using Transfer Learning with resnet-101 architecture. Train and test the model. I will implement this too using PyTorch.
6. Write an algorithm to combine Dog detector and human detector.
    - If dog is detected in the image, return the predicted breed.
    - If human is detected in the image, return the resembling dog breed.
    - If neither is detected, provide output that indicates the error.

    After the image is identified as dog/human, we can pass this image to an CNN which will process the image and predict the breed that matches the best out of 133 breeds.

7. Test my algorithm.

The accuracy of the predicted breed of dogs' images will be measurable, because we have a labelled data that we can compare the predictions against.

The accuracy of the predicted breed for humans' images will not be measurable.

Benchmark Model

The benchmark was pre-stated as the model from scratch must achieve a test accuracy of at least 10% and the model from transfer learning must achieve a test accuracy of at least 60%.

Evaluation Metrics –

- VGG16 model used in transfer learning for dog detection should predict dog in images with high precision. This ensures that our dog-detector is well-trained.
- The custom CNN must have some accuracy to get an intuition of whether the model is working, if working it must be able to output only one dog breed among 133 total dog breeds. This will ensure that our custom model is woking and can be trained on full dataset
- I would use test accuracy as an evaluation metric to compare my models with the benchmark model. Additionally, the performance will be compared between the following convolutional neural networks: dog breed's classifier trained from scratch and dog breed's classifier trained with transfer learning.
- Another creiteria would be like having our OpenCV's Haar cascades classifier to work with high precision.

III. Methodology

Data Pre-processing

As we have already noticed, not all images have the same resolution. Also the distribution of the images is not the same in the various dog breeds. Here there is a slight data imbalance.
The images were resized and cropped to 224x224 pixels to achieve an input tensor of shape 224x224 which suits the pre-trained model and the model from scratch., because it is default input size for VGG16 models. Resizing image as 250*250 was the important pre-processing for my model. I think that is because images have

different size in dog dataset and when we resize as 250*250, we standardize width and height then crop so we did not lose any important subject in the image.
After the images are resized, a Random Resized Crop to 224x224 pixels is performed. Then a Random Horizontal Flip is performed and then a Random Rotation is performed.

At the final step, Data normalization is applied

Implementation

The implementation of this project involved 5 major parts:
1. Human detector
2. Dog detector
3. Create a CNN from scratch to classify dog breeds
4. Create a CNN Using transfer learning to classify dog breeds
5. Final algorithm to use these models.

Below is the code snippet for Human Detector.

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

Below is the code snippet for Dog Detector.

```
from PIL import Image
import torchvision.transforms as transforms

def VGG16_predict(img_path):
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to VGG-16 model's prediction
    '''

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image

    # from https://github.com/pytorch/examples/blob/42e5b996718797e45c46a25c55b031e6768f8440/imagenet/main.py#L89-L101
    transform_pipeline = transforms.Compose([transforms.RandomResizedCrop(224),
                                             transforms.RandomHorizontalFlip(),
                                             transforms.ToTensor(),
                                             transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])])
    img = Image.open(img_path)
    img = transform_pipeline(img).unsqueeze(0)

    # solution to cuda issue found here https://discuss.pytorch.org/t/runtimeerror-expected-object-of-type-torch-floattensor-but
    if torch.cuda.is_available(): img = img.cuda()
    pred = VGG16(img)
    # used technique to get predictions from here: https://discuss.pytorch.org/t/making-a-prediction-with-a-trained-model/2193
    if torch.cuda.is_available(): pred = pred.cpu().data.numpy().argmax()

    return pred  # predicted class index
```

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    preds = VGG16_predict(img_path)

    return (preds >= 151) & (preds <= 268) # true/false
```

Below is the code snippet for CNN from scratch to classify dog breeds.

```
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        self.conv1 = nn.Conv2d(3, 16, 3, padding = 1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding = 1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding = 1)

        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128*14*14, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 133)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        ## Define forward behavior

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))

        #print(x.shape)
        x = x.view(-1, 128*14*14)

        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x

#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

Below is the code snippet for CNN Using transfer learning to classify dog breeds.

```
for param in model_transfer.features.parameters():
    param.requires_grad=False


# I tried reconstruct the classifier but rather than do that, i decided to change just last layer to get higher accuracy.
new_layer = nn.Linear(4096, 133)
model_transfer.classifier[6] = new_layer

if use_cuda:
    model_transfer = model_transfer.cuda()
```

```
print(model_transfer.classifier)
```

```
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (6): Linear(in_features=4096, out_features=133, bias=True)
```

Below is the code snippet for Final algorithm to use these models.

```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    img = cv2.imread(img_path)
    cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    human_found = face_detector(img_path)
    dog_found = dog_detector(img_path)

    if human_found:
        plt.imshow(cv_rgb)
        plt.show()
        print('Hello Human! Your doggy-doppelganger is a: ' + predict_breed_transfer(img_path))
    elif dog_found:
        plt.imshow(cv_rgb)
        plt.show()
        print('Good Doggy! You look like a: ' +  predict_breed_transfer(img_path))
    else:
        plt.imshow(cv_rgb)
        plt.show()
        print('Error: No Dog or Human found in picture. Is there either in picture?')
```

I tried various options before arrive at this solution, My process trial and error method to reach a model with a accuracy greater than 10 percent on test set

Refinement

I tried using 2 convolution layer and 1 fully connected layer Tried more deep models , though was was concerned that having too many layers would be computationally expensive I tried 3 convolution layer and 2 fully connected layer, accuracy improved I tried using 4 convolution layer , output channel 14 and 3 fully connected layer Optimizer set to adam Learning rate set to 0.001 Batch size as 128

For transfer learning, I selected VGG16 pre-trained model. I froze all feature parameters. I just changed the last fully connected layer output as 133 and trained classifier again.

IV. Results

Model Evaluation and Validation

It was already recommended in the given content to use the model from transfer learning in the final app. It was an obvious choice as the model from transfer learning outperformed the model from scratch by far. The models were selected during training, epoch number constant as 20 for my every trial. In every epoch, validation accuracy was compared with previous epoch validation accuracy and if it is lower than previous epoch, model was saved automatically. Thus, we had the best model given the lowest accuracy in that training.

Both models passed the required accuracy requirement.

Justification

The expected accuracy result for CNN from scratch should be at least %10.

- My model reached 11% testing accuracy so requirement met successfully.

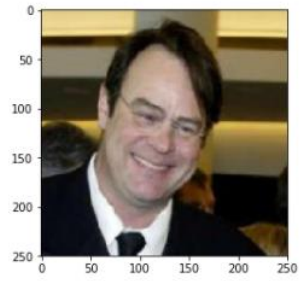
The expected accuracy result for using transfer learning should be at least %60.

- My model reached 73% testing accuracy so requirement met successfully.

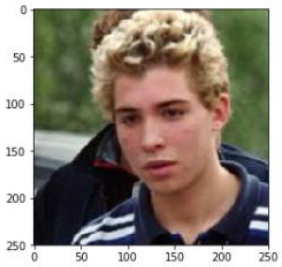
V. Conclusion

Free-Form Visualization

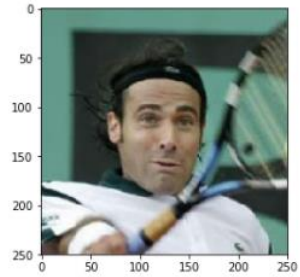
Some outputs of the developed Algorithm



Hello Human! Your doggy-doppelganger is a: American staffordshire terrier

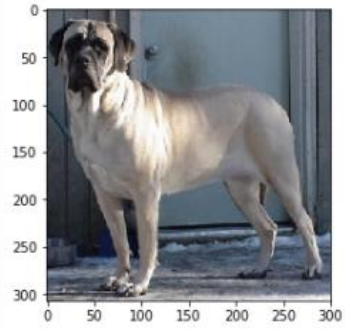


Hello Human! Your doggy-doppelganger is a: Airedale terrier

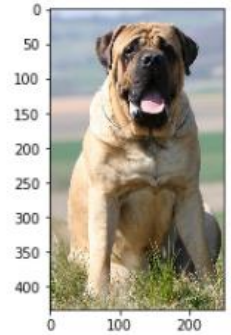


Hello Human! Your doggy-doppelganger is a: Chinese crested

Good Doggy! You look like a: Mastiff



Good Doggy! You look like a: Chinese shar-pei



Good Doggy! You look like a: Cane corso

Reflection

Firstly, I would like to thanks udacity course creators, editors and reviewers, It was really exciting and instructive course. I like working on this project as It was based on real problem based task and used machine learning to solve it. I had to many trial and error to reach a acceptable accuracy, which was tiring but entertaining journey.

I found the domain relevant reading prior to implementing the project particularly interesting. Working on a GPU and CUDA bases machine learning problem was a challenging but fun exercise the model took a lot of time to train even with a GPU and CUDA which highlighted the importance of making smart decisions over trial and error in terms of tuning architectures and parameters. Implementation of the feature extraction section was also an interesting exercise.

Improvement

For further making this solution as a usable product and improving user experience, I would like to integrate this within an website by using flask and aws, to host the model as endpoint and using it on the website. On the Webpage user will have the options to upload pic or maybe take on picture on camera based device and it will give result about the breed or dog, or similar looking breed in case of human.

I could try to improve my accuracy and precision , although my model reached the accuracy of 73% but in real usery will want a more accurate mode. I can try different epoch number and architectures but this would require more computing power especially more powerful GPUs. For the transfer learning model, I could experiment with other pre-trained models and see if they perform better.

References

1. https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html
2. https://pytorch.org/docs/master/torchvision/models.html
3. https://cs231n.github.io/convolutional-networks/
4. https://github.com/udacity/deep-learningv2pytorch/tree/master/project-dog-classification
5. https://arxiv.org/abs/1512.03385
6. https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e
7. LFW Dataset, Udacity https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip
8. Dog Image Dataset, Udacity https://s3-us-west-1.amazonaws.com/udacity-aind/dogproject/dogImages.zip