Capstone Project Report Priyatham Susarla

Machine Learning Engineer Nanodegree September 24, 2020

Dog Breed Classifier using CNN

Project Overview

The dog classifier problem comprises identifying a dog breed using machine learning algorithms to train a model based on a database of images of dogs and humans. This project makes use of deep learning frameworks to create a model that can predict breeds of dogs based on user input.

The user inputs an image next to the model first identifies if the image contains a dog or not. If so, it predicts the breed of the dog, but if the image is of a human, then the model predicts a breed that closely resembles the person. In case the input is neither of those mentioned before, an error message displayed to the user.

Problem Statement

To predict the breed of dog based on user input images, we make use of Neural Networks, specifically Convolutional Neural Networks or CNN for short. We use CNN since they are better adapted to image classification problems when compared to RNN (Recurrent Neural Networks).

There are two parts to the problem as described above. First, we need to create an algorithm that detects dogs and their breed in each input image. Second, we need another one that is trained to associate with the dog's breed that resembles features of human faces.

For the detection of human faces, computer vision technology is required. OpenCV's Haar Cascade face classifier is used here for this purpose. Although few

other classifiers could produce better results, this will serve as a benchmark model that can be improved upon later. The algorithm should reproduce the most similar dog breed that recognizes the correct dog breed successfully in as many cases as possible. Besides, an accuracy of 60 percent or greater should be achieved.

Metrics

We split the data into train, test, and validate. The common practice is to use a 60/20/20 approach for that. Accuracy is what we do as suggested in the notebook. Accuracy is the ratio of correctly classified objects over the total number of classified objects.

Using the test and validation dataset allows us to gauge the performance over a faster runtime to allow for tuning the model parameter before deploying the model. Log loss is also a measure that we look at as a measure of uncertainty in model prediction.

Precision will use to see how precisely our Haar Cascade classifier can identify human faces and not human faces.
Precision = True Positive / (True Positive + False Positive)

Accuracy will a metric to measure the efficiency of our custom-created networks.
Accuracy = True Positive + True Negative / Size of all train images

Data Exploration and Analysis

The following datasets provided by Udacity will use:
Dog images dataset: The dog image dataset can download here:
https://s3-us-west1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
Human images dataset: The human dataset can download here:
https://s3-us-west1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The dataset contains images of humans as well as dogs. The dataset contains images of humans and dogs. With 13,233 images of humans and 8351 images of dogs, there seems to be a reasonable amount of variation present in the dataset that sort into a train (6,680 Images), test (836 Images), and valid (835 Images)

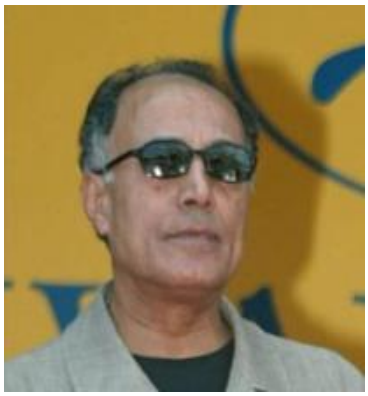
directories. Each of these directories (train, test, valid) has 133 classes corresponding to dog breeds.

The images are of different sizes and different backgrounds, and some pictures are not full-sized. The data imbalanced because the number of photos provided for each breed varies. Few have four images, and while some have eight. The human dataset is also imbalanced because we have one picture for some people and many pictures for some.

The human dataset will use to detect human faces in images using OpenCV's implementation of Haar feature-based cascade classifiers. The dog dataset will use to identify dogs in images using a pre-trained VGG-16 model.

For instance, the number of images associated with each breed varies for the dog's dataset, and sometimes there were several more image for the same person compared to others in the case of the human dataset.

Below are examples of an image from each dataset


Data Preprocessing

We had to conduct some data pre-processing before building and training our model. All images were resized to 224x224 and normalized to the train, test, and validation datasets. We also used image augmentation to minimize over-fitting. We then apply random data rotation and horizontal flip. The last step comprised converting the data into tensor before transferring to the model. After that we converted the image to tensor so that It can be passed to our PyTorch model. At the end and before passing it to the model, we will also apply standard normalization of values (mean= [0.485, 0.456, 0.406], std= [0.229, 0.224, 0.225])

Algorithms and Techniques

For the face-detector: Haar Cascade

```
# 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
```

Convolution is the process of adding each element of an image to its local neighbors, weighted by the kernel. Convolutional Neural Network or CNN is used, in this case, to solve a multiclass classification problem.

CNN is an algorithm that is frequently used in the context of deep learning. Its advantage relies on the fact that it can take an image as input, analyze features and objects that contained inside it, and use ones specific to each image to differentiate between images.

In machine learning, CNN's used for image classification and recognition because of its high accuracy. To train the CNN algorithm, we first used Open CV's cascade classifier to detect features in the human faces dataset. We then use a pre-trained Pytorch model (VGG16) to detect dog's images. We then use the trained CNN model to process and predict the dog's breed in the input image of a human or dog.

VGG-16 is a CNN architecture that was introduced in 2014 by Simonyan and Zisserman in the paper, Very Deep Convolutional Networks for Large Scale Image

Recognition. (this paper is also the influence for the choices made in this project) The network is pre-trained on over a million images from the database ImageNet. It's widely used in the field of computer vision and classification.

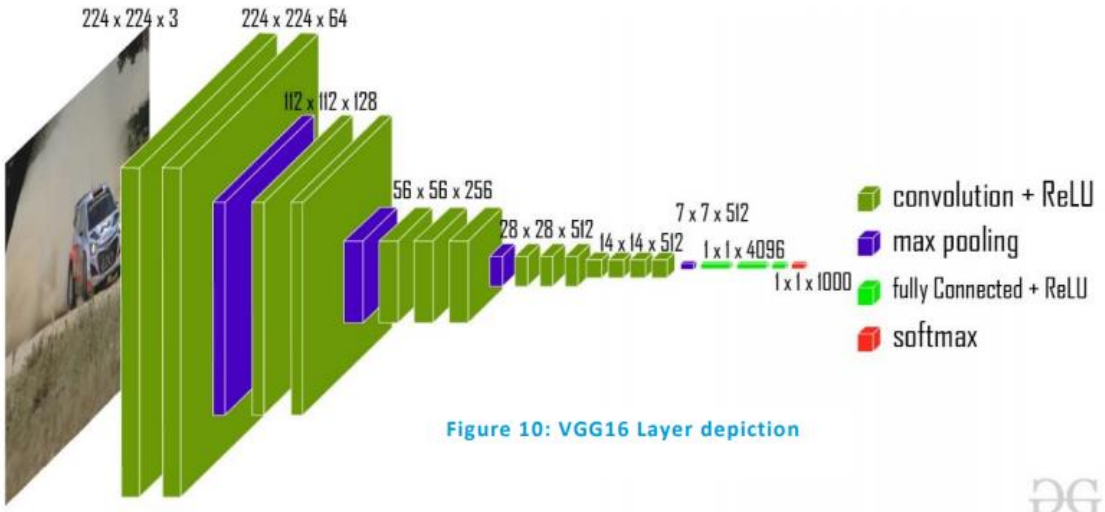


Figure 10: VGG16 Layer depiction

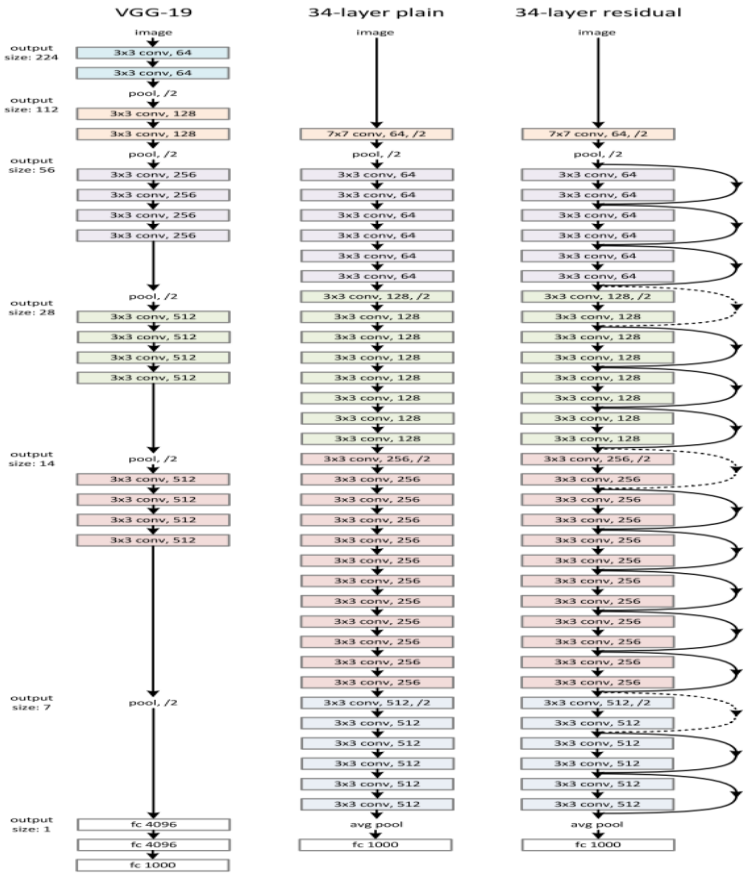
Although the network is capable of producing highly accurate results, owing to being trained on a large variety of datasets, there are newer and more capable models which, as well as producing better results, also provide much quicker results. This VGG model will also serve as a benchmark for the project. The major drawback to VGG networks is their relatively slow speed of training and implementation.

The predictions from the model returned an accuracy score as shown in the image below:

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    prediction = VGG16_predict(img_path)
    #print(prediction)
    if (prediction >= 151) and (prediction <= 268):
        return True
    return False   #true/false
```

ResNet101 (Residual Network) architecture has the capability of extended over 100 layers deep. This network also eliminates another drawback in traditional/sequential networks, which is the vanishing or explosion of gradients as the network gets deeper.

In these traditional networks, as the number of layers is increased beyond a certain level, the accuracy of the output diminishes owing to the problems stated above. Ideally, we would want a deeper network to produce much better results than shallow ones. ResNet uses Residual Blocks, which establish skip/identity connections between the input and output of the layers.



Benchmark Model

The CNN created from scratch serves as the benchmark for this project and test accuracy of at least 10%. That will confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.
The CNN model created using transfer learning must have an accuracy of 60% and above. This value can use to get accurate results, which can further be improved using similar hyper-parameter tuning methods.

Implementation

Here I used PyTorch to design, train and test a model from scratch. The architecture is composed from a feature extractor and a classifier. We use the VGG16 network that was pretrained on the ImageNet dataset. The ImageNet dataset contains similar images with our own dataset so we can keep the feature extractor part.

```
# 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, 32, 3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=1, padding=1)
        self.conv3 = nn.Conv2d( 64, 128, 3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 128, 3, stride=1, padding=1)

        # Pooling
        self.pool = nn.MaxPool2d(2, 2)

        # Define fully connected layers
        self.fc1 = nn.Linear(14*14*128, 4096)
        self.fc2 = nn.Linear(4096, 134)

        # drop-out
        self.dropout = nn.Dropout(0.25)
```

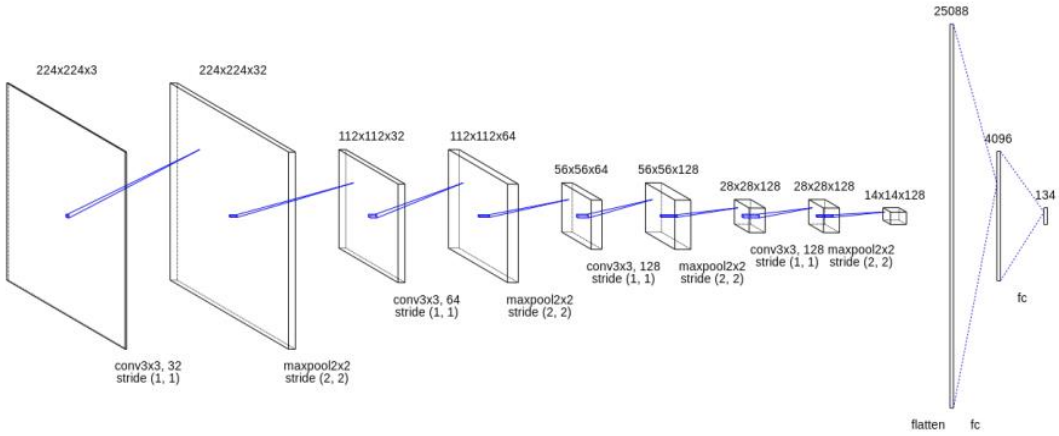
• Convolutional Layer with 3 input dimensions and 32 output dimensions - kernel size 3
• Relu Activation function
• Pooling layer - kernel size 2
• Convolutional Layer with 32 input dimensions and 64 output dimensions - kernel size 3
• Relu Activation function
• Pooling layer - kernel size 2

• Convolutional Layer with 64 input dimensions and 128 output dimensions - kernel size 3
• Relu Activation function 13
• Pooling layer - kernel size 2

• Convolutional Layer with 128 input dimensions and 128 output dimensions - kernel size 3
• Relu Activation function

• Pooling layer - kernel size 2
• Flatten layer to convert the pooled feature maps to a single vector with a length of 25088
• Dropout with a probability of 0.25
• Fully connected Linear Layer with an input size of 25088 and an output size of 4096
• Relu Activation function
• Dropout with a probability of 0.25
• Fully connected Linear Layer with an input size of 4096 and an output size of 134

Here you can see the architecture of the model from scratch



Forward function:

```
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)))

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

    x = self.dropout(x)

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

    x = self.fc2(x)
    return x
```

The ReLU function provides a simple and effective method for activation. For positive values, the function is a linear model and for negative values, it stays at zero. This results in much less complicated computations thereby reducing training time effectively. The linearity also ensures that there are no vanishing gradients. Cross-Entropy Loss, also known as Log Loss, provides an output between 0 and 1 thereby making it easy to predict the outcome directly.

As the prediction reaches the ground truth, the loss value decreases. This also means that incorrect predictions are penalized heavily, which improves the final accuracy score and the confidence/reliability in the model.

Methodology

The problem required the use of multiclass classification, and we used one of the CNN algorithms available in PyTorch to achieve this. We first built a human face detector using a cascade classifier available in OpenCV. We then used VGG16 available in PyTorch to train the dog face detector on the dog image input dataset. We finally passed on the processed image (dog or human) onto a CNN model which will identify the most dog's breed that most resembles the input image. Some model accuracy minimum requirements would have been at 10%.

This is to maintain a reasonable level of prediction accuracy. To achieve our aim, we use the conventional approach to split the data into train, test, and validate segments. Our model performance then tested using the test dataset portion, and we used accuracy as a metric to test our CNN model. Hyper-parameter tuning was achieved by comparing cross-validation and testing datasets to get a model with good performance.

Training the model:

The model was trained on 20 Epochs and the initial and final loss values stood at

```
Epoch: 1        Training Loss: 1.889123         Validation Loss: 2.345647
validation loss has decreased (inf --> 2.345647).  saving model ...
Epoch: 20       Training Loss: 0.109001         Validation Loss: 0.015963
validation loss has decreased (0.027489 --> 0.015963).  saving model ...
```

As stated in the Benchmark section, the testing accuracy of this network was 11%.

Results

The human face detector identified 98% of humans correctly in human input images, while 17% of humans were confused with dogs. The dog detector detected only 1% of humans as dogs while detecting dogs with 100% accuracy.
As far as results from our train and tested CNN model, an 11% accuracy was achieved (98/836) with Test Loss 7.719, therefore above the accuracy minimum set requirements (10%).

```
# call test function
test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 7.719106


Test Accuracy: 11% (98/836)
```

The results improved by using the transfer learning classifier approach, and the tested model performed better with an accuracy of 85% (716/836) and a test loss of 1.06.

```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.484345


Test Accuracy: 85% (716/836)
```

Examples from own image tested with algorithm:

Justification

The model performance is satisfying. BY using transfer learning, we improved the accuracy to 85% from the previous 11% obtained with the model created from scratch. In both situation the benchmark was outperformed.

Improvement

I think the model created using transfer learning performed well, so the predictions seem to be very good.

1. Improving the accuracy of the human detector, using a different approach than haar.
2. Hyper-parameter tuning will also help in improving the performance and accuracy of the human detector.

3. If this diversity were represented in the data set, the algorithm could provide an estimate for all these dog breeds.

Conclusion

In this project, we have built an algorithm to detect and classify dogs in images according totheir breed.
First, we tried a basic CNN architecture, which performed poorly with 11% classification accuracy.
Second, we tried a more complex CNN architecture based on a pretrained ResNet101 model with a custom final fully-connected layer. This model performed extremely well with a classification accuracy of 85% on our test set of 836 dog images.
The result of the transfer learning is much better than I had expected and probably much better than a human (without a trained eye) could classify dog breeds. Personally, I have not even heard of some of the dog breeds before, so having an algorithm tell me the type of dog is magical.

References

1. Original repo for Project- GitHub:
https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/dog_app.ipynb
2. Resnet101:
https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet101
3. PyTorch Documentation:
https://pytorch.org/docs/master/
4. Very deep convolutional networks for large-scale image recognition:
https://arxiv.org/pdf/1409.1556.pdf
5. ImageNet training in PyTorch:
https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f6173/imagenet/main.py#L197-L198
6. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53