Tom Orth

Machine Learning Engineer Nanodegree

Table of Contents

Definition

Problem Overview

The project I have chosen to work on is creating a classification model for different dog breeds. The model is to be run on dogs or humans only. If it is a human, the model will predict the dog breed the person resembles.

Project Domain

The domain background for this project is image classification. This has been in the forefront of research and the world. There are a few different applications of this domain outside of the problem I am working on, such as:

- Medical (https://arxiv.org/pdf/1704.06825.pdf)
- Urban Planning (https://link.springer.com/chapter/10.1007/978-3-642-60105-7_11)
- Facial Recognition (https://arxiv.org/pdf/1804.06655.pdf)

Related Datasets

There are two main datasets for this project. The first is the actual dog dataset. This has different dog breeds that we will be interested in classifying. The second is a set of human faces. These are used for the face detector to help ensure that the model is only run on dogs or human images. The datasets are located at the following URLs:

- Dog dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
- Human Dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

Problem Statement

In order to accomplish the project overview described previously, a machine learning model will need to be implemented. For this project, a neural network model should be used. This is due to neural networks, specifically convolutional neural networks (CNN), being great at handling image data (https://medium.com/datadriveninvestor/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8#:~:text=CNNs%20are%20fully%20connected%20feed,above%20described%20abilities%20of%20CNNs). There are a few different ways to create a CNN which will be detailed in the Analysis section.

Metrics

The solution will use two metrics to quantify the results. The first is Accuracy (# of correct / total number of samples). This is a classic metric for model performance. A second metric that was chosen as well is F-Measure. This is because it is a more robust error metric and can help to glean insights into how models perform on data.

Analysis

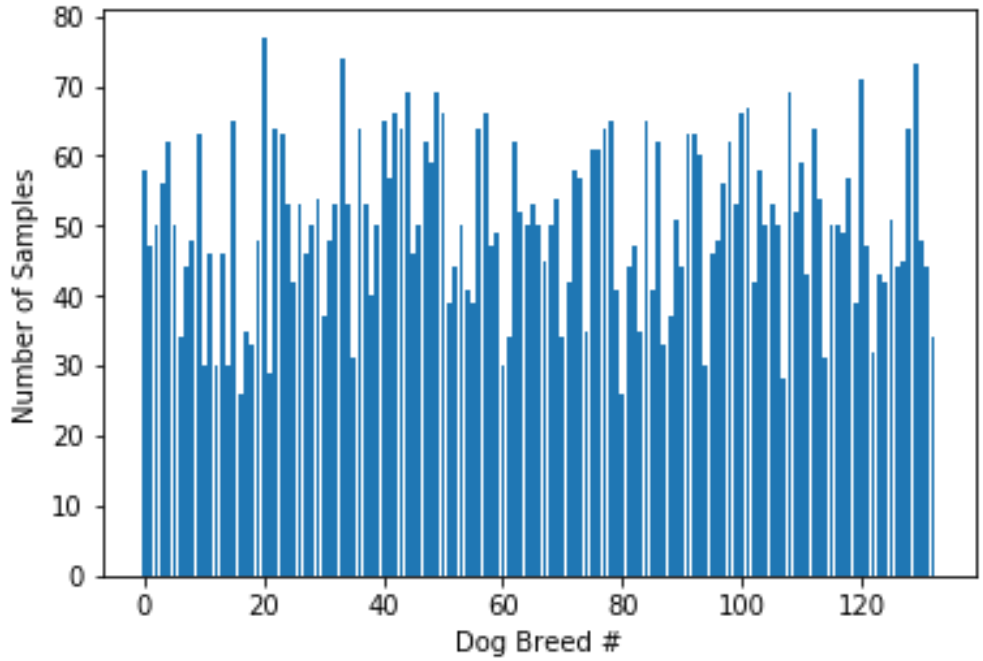
Data Exploration

For the human dataset, it consists of 13233 images. It is not important to have many samples for each person as the closest dog breed is predicted if a human image is given. Due to this, the distribution per person is not a relevant factor.

For the dog dataset, it consists of . There are 133 dog breeds total. That means there are 133 labels for our model to learn. The labels are listed here:

['Mastiff', 'Doberman pinscher', 'Curly-coated retriever', 'Borzoi', 'Bichon frise', 'Chinese crested', 'Finnish spitz', 'Welsh springer spaniel', 'Bedlington terrier', 'Papillon', 'Saint bernard', 'Australian terrier', 'Parson russell terrier', 'Norfolk terrier', 'Yorkshire terrier', 'Boston terrier', 'Norwegian buhund', 'Bluetick coonhound', 'Field spaniel', 'Tibetan mastiff', 'Alaskan malamute', 'Manchester terrier', 'Boxer', 'Flat-coated retriever', 'Irish wolfhound', 'Miniature schnauzer', 'Boykin spaniel', 'Greater swiss mountain dog', 'American foxhound', 'Nova scotia duck tolling retriever', 'Black and tan coonhound', 'German shorthaired pointer', 'English setter', 'Border collie', 'Cardigan welsh corgi', 'Neapolitan mastiff', 'Golden retriever', 'English springer spaniel', 'Great dane', 'Icelandic sheepdog', 'Bernese mountain dog', 'Japanese chin', 'Australian cattle dog', 'Belgian sheepdog', 'Bullmastiff', 'Leonberger', 'Beauceron', 'Belgian malinois', 'Beagle', 'Bull terrier', 'Irish terrier', 'English toy spaniel', 'Pomeranian', 'Lakeland terrier', 'Silky terrier', 'Pharaoh hound', 'American eskimo dog', 'Australian shepherd', 'German pinscher', 'Kuvasz', 'Wirehaired pointing griffon', 'Portuguese water dog', 'German shepherd dog', 'Airedale terrier', 'Canaan dog', 'Pembroke welsh corgi', 'Anatolian shepherd dog', 'Bouvier des flandres', 'Newfoundland', 'Chesapeake bay retriever', 'American water spaniel', 'Entlebucher mountain dog', 'Afghan hound', 'Collie', 'Kerry blue terrier', 'English cocker spaniel', 'Havanese', 'Cane corso', 'Dachshund', 'Black russian terrier', 'Xoloitzcuintli', 'Komondor', 'Belgian tervuren', 'Otterhound', 'Briard', 'Giant schnauzer', 'Bearded collie', 'Norwegian lundehund', 'Irish red and white setter', 'French bulldog', 'Glen of imaal terrier', 'Cairn terrier', 'Akita', 'Dogue de bordeaux', 'Smooth fox terrier', 'Ibizan hound', 'Pekingese', 'Greyhound', 'Chow chow', 'Bulldog', 'American staffordshire terrier', 'Cavalier king charles spaniel', 'Lhasa apso', 'Italian greyhound', 'Chinese shar-pei', 'Irish setter', 'Brittany', 'Plott', 'Basenji', 'Border terrier', 'Great pyrenees', 'Labrador retriever', 'Bloodhound', 'Chihuahua', 'Petit basset griffon vendeen', 'Poodle', 'Dandie dinmont terrier', 'Clumber spaniel', 'Brussels griffon', 'Old english sheepdog', 'Dalmatian', 'Cocker spaniel', 'Pointer', 'Gordon setter', 'German wirehaired pointer', 'Irish water spaniel', 'Norwich terrier', 'Norwegian elkhound', 'Affenpinscher', 'Basset hound', 'Maltese', 'Keeshond', 'Lowchen']

The distribution of each label is shown in this graph:



As we can see, each dog breed has a different number of samples. Some more, some less. This can affect our model performance later on.

Exploratory Visualization

Below are two example dog images:

Next, we will see two example human images:

It is important to note that while all the data are images, they are not the same size as can be seen. The data will need to be adjusted when loaded to make sure all images are the same size.

Algorithms and Techniques

There are a few algorithms that will be used in this project. There are two algorithms for checking if an image is a human or dog. In addition there are two main approaches for the machine learning model.

Image Detection Algorithms

There are two image detection algorithms. The first is a face detector algorithm to see if an image has a human face. The algorithm uses a CascadeClassifier to find a human face in an image, if one is there. The second is a dog detector algorithm that uses VGG16 (https://neurohive.io/en/popular-networks/vgg16/) to detect if a dog is in the image. These will be used to ensure our model only runs on the appropriate images.

Convolutional Neural Network

There are two methods to creating a CNN. One is from scratch with a custom model architecture. This model would consist of layers like a Conv2D, Pooling, Fully Connected, and Dropout. The second way would be to use a pretrained model and then retrain it on this new data. This is called transfer learning (https://machinelearningmastery.com/transfer-learning-for-deep-learning/). Both techniques will be explored. The projected best solution would be the transfer learning model due to the fact that transfer learning has resulted in improved performance in different models throughout the

machine learning community (https://arxiv.org/abs/1902.07208, https://openreview.net/pdf?id=ryxyCeHtPB).

Both of these models will be built using Pytorch (https://pytorch.org/), a popular machine learning framework. The framework has a subpackage called torchvision (https://pytorch.org/docs/stable/torchvision/index.html), that provides a powerful means to work with image data.

Benchmark

To benchmark the results, the custom CNN from scratch will be the benchmark model. The results of the model with transfer learning will be compared to it. If the transfer learning model surpasses it, then the transfer learning model is well suited for the problem. However, if it performs worse, then the model is not good enough. I will use Accuracy and F-Measure (computed using the macro average approach) to compare the two models.

It is important to note that in my original proposal that a pretrained model, not trained on the data, would also serve as a benchmark to compare to the transfer learning model. However, upon inspecting the classes in the pretrained models, it seems that they do not match up as I previously thought so it is infeasible to use that as a benchmark.

Methodology

Data Preprocessing

For both the human images and the dog images, some preprocessing was needed to be done. For the human images and dog images fed to our final model after training and evaluation, the image is resized to 224 by 224. The reason for this is that this is the size requirement for all images fed to the pretrained models from pytorch.

For the training, validation, and test sets of the dog images, the preprocessing is a bit more extensive. We need to set up different preprocessing for the train set than for the validation and test set. This is because in order to create a robust model, the training data needs to be augmented. However, we do not want to augment the validation and test set. The transformation for the training set is as follows:

1. Resize to 224x224
2. Center crop to 224
3. Apply a Random Horizontal Flip operations (randomly flip training samples horizontally)
4. Convert to a Tensor
5. Apply a recommended normalization for images fed to a pretrained model

Step #3 is the augmentation part I mentioned before. So that would mean our operations for the validation and testing sets:

1. Resize to 224x224
2. Center crop to 224
3. Convert to a Tensor
4. Apply a recommended normalization for images fed to a pretrained model

With all these operations, we can have a robust dataset where all samples are uniform.

Implementation

There are a few different parts to the implementation. The first part is the benchmark model with the CNN from scratch, the second is the transfer learning model, and the third part is putting together the final model and detection algorithms to solve the problem.

Benchmark Model

The CNN from scratch involves the code listed below for the architecture:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)

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

        self.fc1 = nn.Linear(7*7*128, 500)
        self.fc2 = nn.Linear(500, num_classes)

        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)
```

```
# flatten
x = x.view(-1, 7*7*128)

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

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

What this model consists of is 3 convolutional layers where a pooling layer is applied to each convolutional layer. Each convolutional layer increases in channel size, starting at 3 and then going on to powers of 2. The convolutional layers consist of a 3x3 kernel with a stride of 2 and padding of 1. The way a CNN works is the kernel is applied to the image and turns it into a smaller input for the next layer. The padding is applied to help make sure pixels are not lost during the kernel application. The particulars of these CNN aspects are outlined here: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html#stride. The stride allows the kernel to move at a faster rate. Additionally, there are 2 fully connected layers. These take the results of the convolutional layers and turn it into the probabilities for the different classes. In between them, a dropout layer is applied so that our model can handle new data better. With all these layers, an activation function called ReLU is applied to introduce some non-linearity to the model.

In addition to the model architecture, we need a loss function to describe the quality of our model during training and an optimizer to improve these layers after each iteration of training. I chose Cross Entropy for the loss function and Stochastic Gradient Descent (SGD) for the optimizer with a learning rate of 0.001 and momentum of 0.9. These parameters affect the rate at which the model architecture changes during training.

The main issue I encountered was the model performance. In order to have a good benchmark, the CNN forms scratch should have a test accuracy of at least 10%. However, my initial model architecture did not work well. This iterative process will be described in the “Refinement” section.

Transfer Learning Model

The transfer learning model I constructed used the pretrained model called resnet18. The reason I chose this model was that during my internship this past summer, a data scientist identified this model as highly effective and used it for a machine learning solution to a business problem. The details of the architecture is shown in the image below (https://www.researchgate.net/figure/ResNet-18-Architecture_tbl1_322476121):

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64$, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

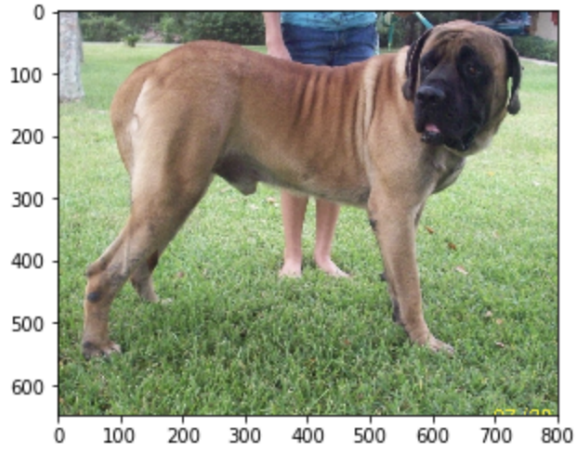
As can be seen, it has similar layer types as described in the “Benchmark Model” section. The main difference is the exact layout. This model has 5 convolutional layers with a max pooling layer applied to the second convolutional layer, a single pooling layer, and a fully connected layer. Additionally, a softmax activation function is applied at the end for the probabilities for this model’s classes. The kernel, padding and stride values are also different.

I used the same optimizer and loss function as the benchmark model when I trained it on the dog dataset.

All Together

I put together the detector algorithms and the transfer learning model in order to solve this problem. First, I applied the dog detector algorithm to see if the image was a dog, since it had a 100% accuracy of not marking humans as dogs. If it returned true, the system would output this:

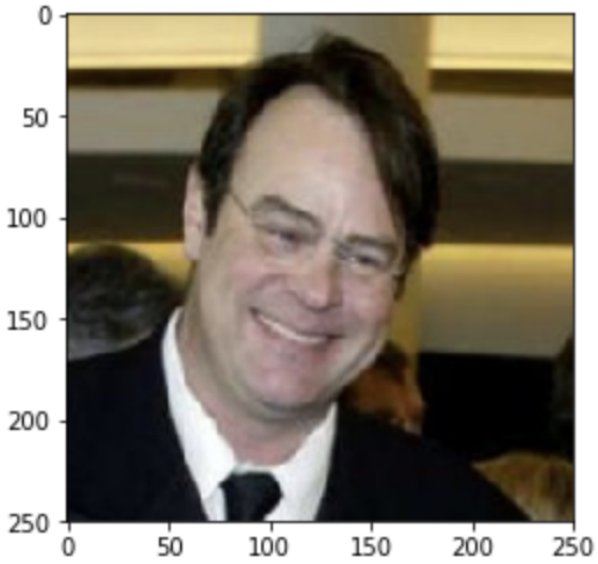
```
Hello Doggo!
```




```
You look like a...
Bullmastiff
```

If that detector fails, the face detector is applied. If that returns true, then the system outputs this:

```
Hello Human!
```




```
You look like a...
Bull terrier
```

If both fail, the system will not attempt to classify and print an error message.

Refinement

The only refinement I really needed to perform was with the benchmark model. I was familiar with the CNN architecture: I needed convolutional layers, pooling layers, and fully connected layers. I first looked at the example provided by pytorch: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. I adapted this architecture as a starting point. This initial solution yielded a 7% accuracy. This did not meet the standard needed for the benchmark. I looked at the cifar implementation provided by Udacity: https://github.com/udacity/deep-learning-v2-pytorch/blob/master/convolutional-neural-networks/cifar-cnn/cifar10_cnn_solution.ipynb. I realized I had forgotten to add a dropout layer to make sure my model was more robust and did not overfit. So I updated my model architecture. However, it still was shy of the accuracy requirement. As I was playing around with the parameters, I realized there were two key issues: I had not incorporated stride into my model architecture and I had chosen an Adam optimizer. I normally use this optimizer for past neural networks I have done for NLP tasks. Once I added in stride and changed the optimizer, I reached the necessary accuracy level.

Results

Model Evaluation and Validation

The final model architecture and parameters are described in the “Implementation” section, specifically in the “Transfer Learning Model” subsection. The model is robust as it uses resnet 18 as the starting point. It was trained on over 1 million images and a large number of classes (1000). Since it was trained on so many classes and images, it already has well tuned parameters. This allows for it to be easily tuned to a new set of data and classes.

This approach worked rather well. It achieved 85% accuracy and 83.35% F-Measure (macro averaged). This is a rather high score for both metrics.

Justification

The benchmark model had an accuracy of 12% and an F-Measure of 10.52%. This is significantly lower than the scores for the transfer learning model (accuracy of 85% and F-Measure of 83.35%). This is a difference of 73% and 72.83%, respectively. This, along with the output of the system as shown in the “Implementation” section, shows that this model adequately solves the problem stated at the beginning of the report.