Brandon Amos

Machine Learning Capstone Project
Dog Identification Convolutional Neural Network

Project Overview:

Image classification is an important field of study with many different applications in numerous different industries. Due to the problems complexity deep learning has made it to forefront of techniques used to solve these image classification problems. This project focuses on using a convolutional neural network to classify images as humans or dogs and their respective closest dog breed. Dog breed classification is a common deep learning benchmark test due to its complexity, ease of human understanding, and large existing image datasets. This project uses two large data sets of dog and human images and uses a transfer learning convolutional neural network algorithm to classify these images correctly. These algorithms have progressed greatly over the past few years. Some transfer learning models offer super human performance in image classification these days, meaning, a deep learning CNN can classify an image with greater accuracy than a human.

Problem Statement:

Create an image classifier to identify dogs and humans and their respective closest breed from a supplied input image. The image classifier used in this project is a Convolutional Neural Network (CNN) using transfer learning from ResNet-18 to classify the dog breed from an input image. The algorithm will accurately classify if an image is a human or a dog, and accurately determine what breed that dog is, or what dog breed the human most looks like.

Metrics:

There are multiple metrics used to measure the performance of the model. The final and most important metric is accuracy. The model accuracy is simply the count of correctly classified images divided by the count of all images. The second metric used measures the models performance during its training and testing, this is the log loss evaluation metric. A training vs validation loss and a model vs test loss evaluation metric will be used to measure the models performance. This loss metric will measure the CNNs performance by accounting for the uncertainty of the predicted value based on how much it varies from the actual label. This will be used to access how the model is performing after each training epoch by comparing the training loss to the validation loss. If validation loss is decreasing then model will save, if the training and validation diverge, it will not save, this will prevent overfitting. The model will then go through the similar steps with the model and test set, finally giving a test loss and test accuracy percentage.

Analysis:

Data Exploration:

This project uses dog and human image data sets.

Dog data set:

- The dog data set has classified dog images in individual folders per dog breed (133 different dog breeds) and comes pre-split into training (6680 images), validation (835 images), and test (836 images) sets. The dog images are randomly sized with dogs in various environments and settings and positions. Dogs will be found at different angles and positions with different lighting and picture quality. Images may be or only the face or the full body or a portion of both.

Human data set:

- The human dataset is comprised of 13233 250x250 named human images in 5750 folders. Each image has humans in various environments and settings and positions. Classified human images can have as few as one image per named human.

Inputs:

- Pre trained CNNS (ResNet-18 and VGG16):
- ResNet-18
    - A pre-trained convolutional neural network with 18 layers. ResNet-18 pre-trained on the ImageNet database of over 14 million images. ResNet-18 can classify images into 1000 object categories including animals.
- VGG-16
    - A pre-trained convolutional neural network with 16 layers. VGG-16 pre-trained on the ImageNet database of over 14 million images. VGG-16 can classify images into 1000 object categories including animals.
- OpenCV pre-trained face detector.
    - A pre-trained classifier for faces, eyes, smiles, foreheads, chins, and other face features. The OpenCV face detector uses Harr Cascades feature based cascade classifiers to identify face features.

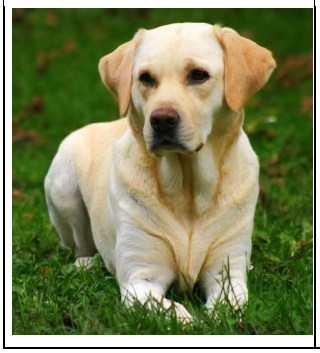
Exploratory Visualization:

Dog breed Image data set:
Many dog breeds are difficult for humans to tell apart, see the Brittany vs the Welsh Springer Spaniel below.

Brittany	**Welsh Springer Spaniel**
	

Curly-Coated Retriever	**American Water Spaniel**

Some dog breeds look similar but have different colors.

Yellow Labrador	Chocolate Labrador	Black Labrador
		

After preprocessing the images with data augmentation (see Data Preprocessing) it is possible to see the algorithm will extract these obvious or minute features from the images in order to classify the different dog breeds. What color is the dog, is the hair curly, etc.

Algorithms and Techniques:

A Convolutional Neural Network (CNN) is used to solve this image classification problem. CNNs are a class of deep neural networks ideal for extracting features from images. CNNs differ from other neural networks because they do not incorporate fully connected layers until the final stage. This allows them to learn at a faster rate and extract minute features from different areas of the supplied images rather than looking at the image as a whole. This technique is ideal to classify an animal such as a dog. Having numerous features that represent the dog will yield higher performance in image classification. For example, is the dog hair curly, is the dog brown, does the dog have solid colored fur or spots, what is the shape of its tail, its eyes, its nose. By extracting these bits of information it will be possible for the CNN to determine which breed this dog might be.

The algorithms used to create the final dog/human breed classifier used CNNs and transfer learning. Transfer Learning is an ideal technique for this problem as we can transfer what a larger, longer running model has learned (feature extracted), and use those features to identify the features in the smaller input data set. Since the ResNet-18 model was run on over 14 million images, animals being one of the 1000 categories, it can be assumed that many of the relevant dog features will already exist. By using these features against the supplied dog images the transfer learning algorithm can correctly classify these dog breeds.

Finally the algorithm must also classify whether the image is a human or a dog, in order to do this a pre-trained transfer learning algorithm for VGG-16 and openCV face detector is used to determine if the

image is a human or a dog. By going through the same feature extraction process explained above the face detector can correctly identify if an image is a human or an animal.

The overall process of the algorithms is as follows:

1. Input an image
    a. Classify is this image is a human or a dog
        i. If the image is a human, find its closest resembling dog breed and print the image and breed.
        ii. If the image is a dog, find its closest resembling dog breed and print the image and breed.
        iii. If it is neither, print and error comment.

Benchmark:

There are two benchmark models, one for the CNN from scratch and one for the transfer learning CNN. The CNN from scratch must perform with an accuracy equal to or greater than 10%. This will prove that the model is working because it exceeds the probability of a random guess which would be around 1% accuracy. The transfer learning model must have an accuracy of equal to or greater than 60%.

Methodology:

Data Preprocessing:

The images are split into three sets, training, validation, and testing.

Training Images:

- The training set data is augmented with randomization, normalization, and resizing functions that can boost the performance of the convolutional neural network to by allowing it to generalize. The training images are first randomly resized and cropped to 224x224. Then they are randomly rotated 15 degrees and randomly flipped horizontally. Since these are color images the RGB color channels are then normalized to be in a range from 0 to 1.

Validation and Testing Images:

- The validation and test sets are first resized to 256, then center cropped to 224, then normalized. None of the randomization transformations are performed on these sets because they should be kept as close to their original image as possible. After many iterations I found that resizing and center cropping brought about the best performance.

In both sets the tensor size of 224X224 was chosen because this is the same size used by the VGG16 neural network. Having a 1 to 1 relationship on image size and format is important for the pre trained VGG16 network to train against the new data set.

Implementation:

The model was implemented in two ways, a CNN from scratch was built as well as the transfer learning CNN. The transfer learning CNN was the used in the final algorithm due to its superior performance.

CNN from scratch:

- The CNN architecture was determined by the size of the images and the final linear layer size I wanted to achieve.
- There are five convolutional layers, the first beginning with a depth of 3 because of the 3 RGB color channels in the image. This layer then outputs a convolutional layer with a depth of 16, so this layer outputs 16 filtered images with filters of size 3 x 3. A padding of 1 pixel is included to ensure all edges of the image are used.
- After this layer is processed it is passed into the Relu activation function, then its X and Y dimensions are down sampled by a factor of 2 by a max pooling layer with a kernel and stride size of 2.
- The output is then fed into a batch normalization function which helps to increase the training speed.
- This process is repeated 5 times through 5 convolutional layers. Each layer increases the feature image depths by a factor of 2, ending at 256. The 5 layers are added because I wanted to reach a final tensor of 7x7.
- Each convolutional layer decreases the image tensor size by a factor of 2, so the starting image tensor of 224x224x3 eventually ends as 7x7x256.
- This final convolutional output is then flattened and a dropout layer of 0.25 is applied before it is fed into the 1st hidden fully connected layer with a relu activation function. Then a second dropout is performed before the final 2nd hidden fully connected layer.
- After which cross entropy loss and a SGD optimizer is applied.
- The final result was a 17% accuracy at 10 epochs. The epochs could have continued, increasing the accuracy, as the validation loss was never greater than the training loss.

CNN from Scratch:

```
import torch.nn as nn
import torch.nn.functional as F

#number of dog breeds
dog_breeds = 133

# defining the CNN architecture
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # convolutional layer (sees 224x224x3 image tensor)
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        # convolutional layer (sees 112x112x16 tensor)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        # convolutional layer (sees 8x8x32 tensor)
        # convolutional layer (sees 56x56x32 tensor)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        # convolutional layer (sees 28x28x64 tensor)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
        # convolutional layer (sees 14x14x128 tensor)
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)

        self.bn1 = nn.BatchNorm2d(16)
        self.bn2 = nn.BatchNorm2d(32)
        self.bn3 = nn.BatchNorm2d(64)
        self.bn4 = nn.BatchNorm2d(128)
        self.bn5 = nn.BatchNorm2d(256)


        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)

        # linear layer (256 * 7 * 7 -> 512)
        self.fc1 = nn.Linear(256 * 7 * 7, 512)

        # linear layer (1024 -> dog breed count 133)
        self.fc2 = nn.Linear(512, dog_breeds)

        # dropout layer (p=0.25)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):

        # add sequence of convolutional and max pooling layers
        x = self.pool(F.relu(self.conv1(x)))
        x = self.bn1(x)
        x = self.pool(F.relu(self.conv2(x)))
        x = self.bn2(x)
        x = self.pool(F.relu(self.conv3(x)))
        x = self.bn3(x)
        x = self.pool(F.relu(self.conv4(x)))
        x = self.bn4(x)
        x = self.pool(F.relu(self.conv5(x)))
        x = self.bn5(x)

        # flatten image input
        x = x.view(-1, 256 * 7 * 7)

        # add dropout layer
        x = self.dropout(x)
        # add 1st hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        # add dropout layer
        x = self.dropout(x)
        # add 2nd hidden layer, with relu activation function
        x = self.fc2(x)

        return x
# instantiate the CNN
model_scratch = Net()
# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

ResNet-18 Transfer Learning Model:

The steps in the transfer learning process are as follows:

- Gather the data, I kept the same data augmentation parameters that were used in the CNN from scratch.
- Import the pre-trained resnet18 model and its features.
- Keep all the pre-trained model layers frozen and reset the final fully connected linear layer with the 133 dog breed nodes.
- Next set the optimizer and cross entropy loss.
- The model had an accuracy score of 85% on 15 epochs and could have continued.
- The reason this architecture is suitable for the current problem is because we have a relatively large training set, so we can afford to simply replace the final fully connected layer and use all the features generated from the pre trained ResNet-18 model. Dogs are a common image in the image net database, so with a large dataset of related images in ResNet-18 pre-trained model the replacement of the final fully connected layer is a reasonable choice.

Refinement:

Developing both these algorithms was an iterative process. The CNN from scratch had many iterations and refinement steps. A majority of the refinement came in the data preprocessing steps, it was not until the validation and test set were set to resize to 256 then center crop down to 224 that the algorithms began to perform to their benchmark expectations. The center crop is a key element to data preprocessing as it will center the image during the resizing, otherwise the image may have valuable information cropped out. The CNN from scratch also began to perform at a much higher level after the batch normalization steps were added. While the CNN probably did not need to be 5 layers, a final size of 7X7 was desired. This could have been achieved in a different way.

The refinement of the transfer learning ResNet-18 model was less iterative. By choosing to only reset the final fully connected layer with the 133 dog breed nodes there was not much work to be done. Model tweaking was greatly reduced because all but the final ResNet-18 pre-trained model layers were frozen. After its first run at 10 epochs it yielded a high accuracy. The only refinement was to run it with more epochs.

Results:

Model Evaluation and Validation:

The human vs dog classifier yielded an accuracy of 99% on 100 sample dog images. The classifier miss identified 4 humans as dogs out of the 100 sample human images. While these results are not perfect the human dog classifier met expectations to be used in the final algorithm.

The CNN from scratch reached an accuracy of 17% at 10 epochs and had the potential to keep training with greater accuracy. Using the log loss function it was seen that the validation loss was less than the training loss at every epoch, to increase the accuracy of the model it could have run until the two

diverged. That being said, the transfer learning model yielded superior results as expected so further work on the CNN from scratch was not required.

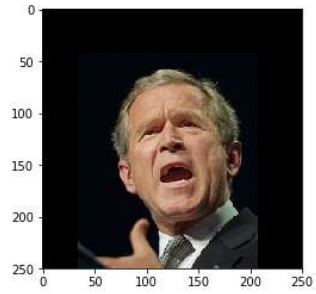
The ResNet-18 transfer learning model had an accuracy of 85% at 15 epochs and also had the potential to keep training according to the log loss function. 85% accuracy was above the benchmark 60% and further training was not performed. The ResNet-18 model was used in the final algorithm and on numerous tests was found to correctly classify dogs and humans and their breed counter parts.

Justification:

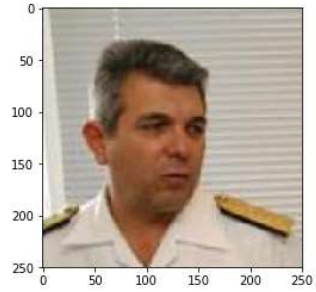
The output is what I expected from a model with 85% accuracy. It correctly identifies humans and dogs, and from my limited experience identifying dogs it looks to classify them correctly. Although it is subjective, I think that the humans are matched with the dog they resemble the most as well.

Algorithm Output:

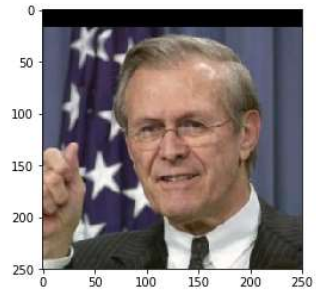
```
This image looks like a human
 This image resembles a Akita
```



```
This image looks like a human
 This image resembles a American staffordshire terrier
```



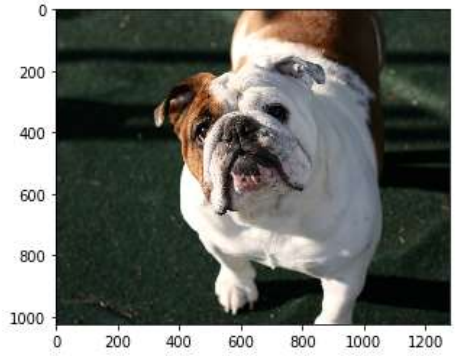
```
This image looks like a human
 This image resembles a Akita
```



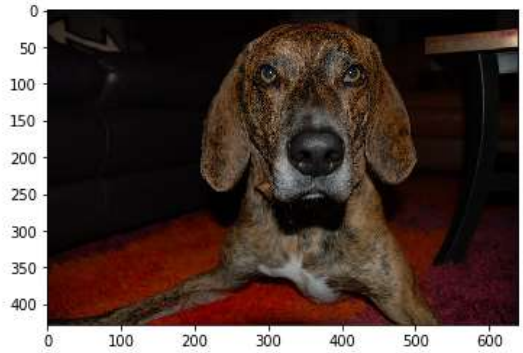
```
This image looks like a human
 This image resembles a Ibizan hound
```

```
This image looks like a dog
 This image resembles a German shepherd dog
```


```
This image looks like a dog
 This image resembles a Bulldog
```




```
This image looks like a dog
 This image resembles a Plott
```




```
This image looks like a dog
 This image resembles a Maltese
```

Three possible points of improvement in the algorithm are:

1. Greater data augmentation could be performed on our data set, we could increase the size of the data set by augmenting the data and adding the augmentations as new images. At this time we only augment existing images. With a larger data set we can ensure greater generalization.
2. The number of epochs could be increased. This model only had 15 epochs, but each epoch had a validation loss less than the training loss. This means that more epochs would most likely increase the accuracy of the model. Training could continue until the validation is greater than the training loss. Then we will know the greatest accuracy.
3. The optimization function could include a learning rate step wise decay. This could help it to converge faster with higher accuracy.

References:

1. PyTorch Documentation https://pytorch.org/
2. PyTorch Torchvision https://pytorch.org/docs/stable/torchvision/models.html
3. Project repo: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
4. Udacity: https://www.udacity.com/
5. FastAI Practical Deep Learning for Coders: https://course.fast.ai/
6. Log Loss Description: http://wiki.fast.ai/index.php/Log_Loss
7. VGG-16: https://neurohive.io/en/popular-networks/vgg16/
8. ResNet-18: https://www.kaggle.com/pytorch/resnet18
9. OpenCV: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
10. Convolutional Neural Network: https://en.wikipedia.org/wiki/Convolutional_neural_network