I. Definition

Project Overview

This project is based in the field of Computer Vision. Computer vision involves using a computer to make classification, identification, and/or inference from visual data. Such data can include but it not limited to, images, videos, and footage. Currently, the best way to implement vision is to remap visual data into numerical data, such as color intensities and color values. This remap enables deployment of traditional learning algorithm onto the data. In my line of work, I often come across data such as scans of legal documents, drug labels, and handwritten data. Such data requires techniques such as OCR, classification, and image identification that all fall under the field of Computer Vision. There also many other applications of this such as facial recognition, barcode scanning, automatic check deposits at ATMs, etc.

Problem Statement

The goal of this project is to develop a ML model that can identify images of dogs and guess their breed. In addition to this, the model will also be able to identify images of humans, and guess a dog breed that they most resemble. As dogs generally fall into well defined breeds and are visually distinct from humans, there is a clear set of criteria for benchmarking the performance of this algorithm.

Current Approaches

Several prominent models have been developed for this task of image recognition and classification. The most well known is the VGG16. This is a convolution neural network trained on millions of free images found on the internet. The model has over 130 million parameters and works by passing the input through a stack of convolution layers. The model achieves over 90% accuracy over 1000 classification values on a dataset of 14 million images.

Solution Strategy

The planned pipeline for constructing a solution is:

- Basic testing of alternative algorithms (OpenCV, VGG16)
- Data Exploration and Preprocessing
- Designing and Training Convolution Neural Network (CNN) model from scratch
- Iterate on the model architecture until appropriate benchmarks are met
- Designing and Training a CNN model using Transfer Learning.
- Iterate until appropriate benchmarks are met
- Designing centralized to invoke relevant algorithms based on user input and produce desired prediction.

Datasets and Inputs

The data for these types of projects usually consists of a large number of medium to high quality images. Since images vary greatly in size, quality, and color, appropriate transformations are often necessary to obtain a uniform dataset (discussed in detail later).

Metrics

The main evaluation metrics here will be accuracy. In other words what proportion of supplied images were correctly identified and classified. There will be subcategories of accuracy such as identification of human vs dog and identification of dog breed for a dog image. There is no effective measure for guessing similarity between a human and a dog breed as there is no clearly defined correct answer. A visual review will be the only metric used for this prediction.

In addition to this false positive, false negative, recall, and precision rates can also be looked at to review the performance of the human vs dog identification.

The from-scratch model is expected to perform at 10% accuracy, while the Transfer Learning model is expected to reach 60%.

In particular, for the dog breed classification, the accuracy is defined as:

$$Accuracy^{*} = \frac{Images\ with\ correctly\ identified\ breeds}{Total\ input\ images}$$

*This is also known as the top-1 accuracy, the terminology will be explained in greater detail in the Benchmark section.

Similar formulas are used for identifying accuracy of dog image recognition and human face recognition.

II. Analysis

Data Exploration

The input dataset consists of about 13K images of humans and 8K images of dogs, separated into folders by breed. Most of these images are medium sized (4-500 kilobytes) and are of good enough quality for a human viewer to identify the content. The dog images were obtained from an S3 bucket hosted here. The human images were obtained from a public data store hosted by University of Massachusetts here. The datasets will be used in both the training and testing of the ML model to be used for identification. Additionally, the pre-trained models contain over a million images obtained from various URLs on the Internet. For the dog training set the images are distributed as follows:

Type	**Count**
Train	6,680
Test	836
Validation	835
Total	8,351


The above images represent the kind of noise and conflicting information we may receive from the training images. Is the first one a dog or a human? Where is the dog in the second one?

There is also a lot of variation in the data. The smallest image is just 4.25 KB:

It is pretty hard to recognize the dog breed from the above low-res image. On the other hand, the largest image is almost 7 MB in size (Alaskan_malamute_00366.jpg)! The resolution is approximately 3014x2386.

Exploratory Visualization

A sample image from the humans dataset is presented at the beginning of the notebook. The image has been rescaled and refitted to a particular size (250px). Similarly, a variety of dog images have been printed in Step 3 of the notebook. The images of the dog pictures illustrate the difficulty of the task. Many dogs in the example look very similar yet have a completely different breed (e.g. Curly Coated Retriever and American Water Spaniel). There is also considerable intra-breed variation. In many cases it presents a difficult task even for a human observer.

In addition to the variety of dog appearances, there is also a variety in the image properties. Some images are very high resolution (> 300 KiB) while others are relatively smaller (~50 KiB). Clearly, some transformation will be required to build a uniform training set. Further some images have some additional, non-relevant information such as dead game in American_water_spaniel_00621.jpg, and humans in American_eskimo_dog_00463.jpg and Welsh_springer_spaniel_08210.jpg. Clearly these have the potential to confuse the algorithms. In particular, for the last image it isn't even clear if it should be identified as a human with a breed likeness or a dog and a predicted breed (ideally both but this is beyond scope).

The images that follow illustrate the difficulties of classification:

The left-hand side image is of an American Water Spaniel while the right-hand image is of a Curly Coated Retriever. However, the two are quite difficult to distinguish.

Algorithms and Techniques

This project will use Convolution Neural Networks (CNNs) to address the problem of dog breed classification. Neural Networks are a form of Machine Learning technique that mimic the structure of the human brain. They consist of nodes or neurons that receive an input from the previous set of neurons and send a processed value to the subsequent set. Unlike a real human brain, neural networks are one-directional i.e., the neurons only receive inputs from the preceding set and can only feed into the subsequent set. In a real human brain, neurons both receive and fire towards all connected neurons. CNNs contain an additional refinement of "convoluting" the input. This involves using a matrix (a "kernel") to transform the input image (a 3-dimensional object representing height, width, and color depth) into an intermediate input stream. This step ameliorates the problem of overfitting to a great extent.

The pretrained VGG16 uses a large CNN with multiple convolution layers. It is an extremely bulky model that was took many weeks to train. To avoid using this complex a first pass will involve training a CNN model from scratch. The architecture and design of this model will be experimented on.

In addition to the custom model, the project will utilize an alternative training scheme known as transfer learning. This involves leveraging learned parameters from an existing, similar model for use in a different project. The breed classifier will add extra layer(s) on top of the existing VGG16 model and only train the added layers. This is expected to provide vastly better results than training from scratch.

For both implementations it will be necessary to clean and preprocess (more on that to follow) the image data to make it uniform. Various transformations and augmentations will be experimented with to eliminate biases from the training set (e.g. all golden Labradors may be looking to their left).

For human facial recognition, the project will reuse OpenCV's classifier built using the Haar feature based cascade classifier.

Benchmark

The most obvious benchmark to be used will be accuracy, with the formula as defined before. Variations of this benchmark are commonly used, the VGG16 uses the so-called "Top-5" accuracy

which is the accuracy of guessing correctly within the top-5 best guesses for a given images. Since project does not have pre-classified top-5 classes for all images in the validation it will use the simpler top-1 accuracy (defined before). This is simply the percentage of the samples that were correctly identified as proportion of the total samples.

The VGG16 model performed at an approximately 92.7% top-5 accuracy. This is impressive considering the sheer number and variation among classes.

In addition to accuracy, we can also look at training speed as a benchmark for the model. The latest benchmarks for most enterprise implementations of ML algorithms are available publicly on MLPerf[1]. Of particular interest to the model is the MLPerf benchmark for ImageNet ResNet-50v1.5. This is an image classification algorithm similar to that implemented in the project trained on a TPUv3.32. This contrasts with the project which is trained on an Nvidia GTX 1050. Nevertheless, this should provide a good ballpark comparison for the CNN model.

The accuracy benchmarks for Image Classification found on PapersWithCode.com[2] can also be looked at.

III. Methodology

Data Preprocessing

The first step to training/testing our algorithm is transforming the input images into a uniform dataset. For testing the base VGG16 model, it can be discovered from the model's documentation that it accepts a 224px square image, with RGB color intensities. RGB consists of a 3 hexadecimal numbers from 0 to FF (0 to 255 in decimal) representing the intensity for the primary colors Red, Green, and Blue respectively. In addition to this, the model normalizes these RGB using a set of means and standard deviations ([0.485, 0.456, 0.406] and [0.229, 0.224, 0.225]). One consequence of this normalization is that the model does not "see" the same image as a human eye but rather a color normalized, rescaled version. After implementing these transformations, the data is converted to a PyTorch Tensor representation.

For training the new CNN model from scratch, we use a slightly different set of data processing. In particular the input image is a 182px square. This 182px square is obtained through a random crop of the image. This serves to augment the data by moving the focus away from the background, and unrelated subjects. The image is also augmented using a random horizontal flip. This flips the image horizontally (mirror image) with a probability of 50%. The goal of this augmentation is to add variance to the images. As mentioned before, suppose all Labrador Retrievers faced rightwards, then the algorithm might believe that a rightward face is the most important property of being a Labrador, which is obviously untrue. Such a training dataset might lead to inflated confidence for right facing dogs being Labradors, reducing the accuracy.

In addition to the augmentations, the training set is also color normalized using the same rule as for the VGG16.

---

[1] https://mlperf.org/training-results-0-6/

[2] https://paperswithcode.com/sota/image-classification-on-imagenet

For training the transfer learning model, there is less choice regarding the inputs. The data has to be same specification as required by first layers of the VGG16 (as stated above). Thus, the projected reverted back to the 224px square for the transfer learning approach.

Implementation

After implementing the data preprocessing above, the project feeds a combination of human and dog images into the VGG16. It is important to note here that classes 151 to 268 (inclusive) correspond to dogs. The project contains a function that can take a path to an image file as an input, load the image, apply the transformation and augmentation, and then produce the appropriate VGG16 predicted class. The model performed quite well on a sample of 200 images (100 human, 100 dog), producing a 93% accuracy at identifying images with dogs (note, it was not predicting breeds) and never misidentified a human face as a dog.

Next, the project constructed a CNN model from scratch. The final model that was settled on consists of the following pieces:

1 Convolution layer with kernel size 3 that convolutes the input image tensor.

1 Max Pool layer than aggregates the result of the convolution (using max).

1 Convolution layer with kernel size 3 that convolutes the output of the preceding Max Pool.

5 Fully connected (FC) layers with the following input/output dimensions:

Layer	Input Dimension	Output Dimension
FC-1	72,900	364
FC-2	364	307
FC-3	307	250
FC-4	250	193
FC-5	193	133

The Input dimensions for the FC-1 layer are determined by the output of the 2 convolution and max pooling layers. Multiple FC layers are used to boost the algorithm's ability to approximate nonlinear functions accurately however it does come with the risk of overfitting.

The forward function of the CNN class defines the forward step of computing the predicted value of an input by passing it through the network.

For the transfer learning approach, a slightly different design was used. The first layers of the model were borrowed from the VGG16. On top of the VGG16, the following layers were added:

1 Linear Layer

1 RELU (takes the positive part of the input)

1 Dropout (.4 chance of dropping an observation)

1 Linear Layer

1 Log SoftMax

Layer	Input Dimension	Output Dimension
Linear-1	4096	256
Linear-2	256	133

The 4096 for the first input dimension is inherited directly from the VGG16 model. The Dropout layer has the effect of regularizing the training by avoiding the chance of overfitting. The Log layer has the effect of converting the output of the algorithm into a value between 0 and 1. This allows for the nice probabilistic interpretation of class prediction confidence that the project needs.

For the loss function, a cross entropy loss was used. This function has the effect of penalizing high confidence in false prediction very severely (in a non-linear method). The optimizer chosen is the Stochastic Gradient Descent (SGD) with a learning rate (alpha) of 0.001. This setting is commonly used in literature and is known to produce convergence at a good pace. It is stochastic (random) because instead of using the true gradient it uses an approximation obtained from a sample subset of the data.

The final step in the implementation is the construction of a function that combines all these parts and produces the desired workflow. This is done by the run_app(img_path) function. It takes an image path as its input. The image is then fed into the dog_detector and face_detector (VGG16 and OpenCV) algorithms. If a dog is detected, the algorithm runs the transfer learning classifier to predict its breed. If a human is detected, the algorithm greets the human as such, then runs the transfer learning classifier to predict a dog breed that the human's likeness most resembles. If neither human or dog are detected the algorithm returns an error message.

Refinement

The from-scratch model underwent several stages of testing and refinement before settling on its final form above. In particular it wasn't clear from the beginning how many fully connected layers or convolution layers. While more FC layers are generally recommended, they can have a two-fold disadvantage: 1) Overfitting and 2) Training and Prediction time. The original version of the model started with just 1 convolution layer and 1 FC layer. Unfortunately, this primitive model performed very poorly, at just 1%. To improve on this score, the number of FC layers were incrementally increased to 5. Increasing the FC layers resulted in a steady increase in the accuracy reaching about 7-10% for 5 FC layers. No more than 5 FC layers were added to prevent problems with overfitting. To improve accuracy even further, an extra convolution and max pooling layer with the same specifications as the original ones were added. This produced a significant boost in accuracy to 13%.

The Adam optimizer was initially chosen as the optimizer function. This optimizer is known for its speed and convergence. Unfortunately, it was consistently producing lower accuracy on validation sets. Consequently, the optimizer was switched to a more standard Stochastic Gradient Descent.

The transfer learning model was initially implemented with just one Linear layer. The algorithm still performed well producing a 66% accuracy on the validation set. To boost this value even further to make it comparable to the VGG16 values (92.7%), a second Linear layer and a dropout layer was added. This resulted in an accuracy of 86%.

IV. Results

Model Evaluation and Validation

The final from-scratch CNN model performs at a 13% accuracy. This is better than the 10% minimum although it is not particularly impressive. Unfortunately, attempts to further boost this accuracy are very expensive in terms of computational power and time required. The 13% accuracy number comes from the validation set. The validation set as described before has about 835 image samples.

One of the main issues encountered when training the from-scratch CNN model was variance in the output model based on the training dataset. As the dataset contains random horizontal flips and resized crops, there is some randomness to the dataset across different training attempts despite using the same underlying images. For the more primitive implementations (1 FC layer, 5 FC layer with one convolution layer), there was considerable variance (3-5%) in the accuracy on the validation set. This was concerning as it reduces the reproducibility of model and suggests the underlying structure is poorly designed and is predominantly just lucky with its correct predictions. This was the main motivation (in addition to the accuracy concerns) for modifying the model to add more convolution and pooling.

After these changes, the model performance is still variant across trainings but the validation accuracy never dropped below the required threshold of 10%. Given the poor accuracy value however, it is of limited value to the main app.

The transfer-learning based CNN on the other hand, performed at an 86% accuracy. This is quite impressive given the difficulty of classifying dog breeds. Not only that, but the training is fairly stable across multiple attempts, with the accuracy never dropping below 80%. This stability is impressive and suggests the underlying model is well defined. The model can also be trained in just 25 epochs as opposed to the 100 required for the from-scratch model.

Comparison to Benchmark

In terms of training, the from-scratch model took about 6 hours to complete 100 epochs of training. This is poor compared to MLPerf benchmark for ResNet which completed in a mere 42 minutes despite being a more sophisticated algorithm. However, the computers on which ResNet was trained were much faster. In terms of accuracy the from-scratch model does very badly, performing at just 13% when the top models on PapersWithCode.com perform close to 86.4% for Top-1.

The transfer-learning CNN model performs better. It completed its training in about 1 hour, which is still slower than the benchmark albeit better than the from-scratch model. It is important to note that this model was trained in just 25 epochs instead of the 100 for the from-scratch. In terms of accuracy the model performs much closer to the benchmark, at 86% accuracy against a benchmark

of 86.4% for Top-1. This is impressive but there are several caveats that will be discussed in the subsequent sections. It is also worth noting that the CNN algorithm only has 133 classes while the top algorithms work on the full 1000-class set of ImageNet.

Justification

The final form of the transfer-learning CNN performed above the required benchmark set at the beginning of the project (60%). It's stability over training sets is reassuring. Given these results, it is my conclusion that the model succeeds in solving the problem outlined in the introduction. However, there are several areas for improvement. One stress-test for the model involves using noisy images. This is common in a real-world setting as images received may be low quality or garbled due to bad scanning or transmission. For example, consider the following color distorted image of a Labrador Retriever:

The algorithm actually succeeds in predicting the breed in the corresponding original image. However, when fed the above distorted image, the algorithm predicts the breed as "Anatolian Shepherd Dog"

While the two breeds do look similar, this susceptibility to small disturbances (to a human eye) in the image is concerning and might cause the algorithm to behave in unexpected ways to real-world data. Unfortunately, this greatly reduces the amount of trust one can place in the algorithm. The problem is quite deep running and may require a rethink of the algorithm's fundamental architecture and design. Some way of reducing sensitivity to image disturbances needs to be found. Introducing color noise into the training set may achieve this, although it might come at the cost

of accuracy on the validation set. It might also be useful to calibrate the validation set to include examples of distorted images like the one above to catch such problems sooner.

V. Conclusion

Free Form Visualization

The final part of the project, consists of a function that takes an image path as an input and does one of the following:

1. If human is detected, greets the human, then predicts a breed with the most similar likeness.
2. If a dog is detected, greets the dog, then predicts the breed of the dog.
3. If neither is detected, presents an error message.

On a sample of 6 images (a coffee mug, a sports car, a Labrador, a Dogue de Bordeaux, Prince William, and Tom Holland), the algorithm produced the correct identification and prediction for each. Of course, the prediction of breed likeness for the human images are only for fun and there is no "correct" answer here.

Reflection

The most challenging part of the project was deciding on the exact architecture of the CNN model. There were numerous conflicting objectives to balance such as overfitting and accuracy, simplicity and capturing the underlying function. The range of options for layers for a CNN model also further complicates the task of choosing the appropriate design. While there is considerable research and literature on the subject, almost every task is unique in its own way and requires some amount of trial-and-error to get an acceptable result.

One of the most surprising and interesting part of the project is the very high accuracy obtained by the final model. This is despite having the high difficulty of classifying dog breeds, even for a human. I believe that not only does this model satisfy its goals, it can also be very easily tweaked to deployed in a much more professional or enterprise setting for use cases such as facial recognition, classification of large volume of unstructured data, and security software. The transfer learning approach in particular is of great interest because of the speed with which it can redeploy a highly sophisticated, existing model to a different but related task.

Improvement

There are several areas for improving the final and intermediate (from-scratch) model.

Speed of Prediction: Currently the model is a little sluggish when predicting (about a second per input). This is probably a consequence of the complexity of the model, especially considering the underlying VGG16 model has over 130 million parameters. One solution to this could be moving the trained model to a more performant Sagemaker endpoint. The model can then be queried via the API instead of hosting it on a personal computer.

Complexity of the Model: While the CNN model performs admirably, it is hard to understand why exactly it is so successful or which parts of its architecture contribute most to its success. In particular the final architecture was arrived largely through a process of trial and error. Perhaps

having a more theoretical understanding of the model's success will enable a cleaner and more understandable version that is just as accurate. This might also address concerns the aforementioned concerns about speed.

Less Dependence on Training Set: This is a particular problem for the from-scratch model. It is very sensitive to small changes in the training set. This is perhaps suggestive of the fact that the model is mis-specified and requires further refinement. This might also help increase the model's tolerance to noisy images as seen before.

Alternative base algorithms: While this model was trained on top of the VGG16 model, it might be interesting to see how its performance compares when transferred from other major image processing algorithms like ResNet or Xception. Alternatives to CNN models such as Support Vector Machines or Random Forest classification could also be attempted.

I believe that implementing the above fixes might enable the model to perform even better than the 86% benchmark set by this project.