Kim S. Harrington
June 2020

Capstone Proposal

Domain Background

Image recognition is a field in computer science that applies specific technologies such as machine learning algorithms and strategies to recognizing and/or identifying certain objects of interest such as animal or numerous other vision based subjects. This field has benefited from significant university and industry research stemming back to the 1950s and '60s from research on the brain of mammals by D.H Hubel and T.N Wiesel and; as of now the field applies deep learning neural network architectures to solving image recognition challenges.

My interest is computer based vision as applied to self-driving vehicles, but I first need to gain a thorough and rudimentary understanding of image recognition in order to make advancement toward my objective. To that end, I have selected the Dog Breed Classification project. This project's subject matter has had a lot of attention; a similar project was a Kaggle competition. The Kaggle Dogs vs. Cats Prediction Problem of 2013 used a data set comprised of photos of dogs and cats extracted from a much larger dataset of 3 million annotated photos. The competition applied convolutional deep learning Neural Networks (CNN) to maximize accuracy. Pierre Sermanet won the competition with accuracy of ~98+%. There are almost a half-dozen research papers sited on the Kaggle site pertaining to this topic. The site also expressed that it is possible to obtain an accuracy of 80% using manually designed CNNs and when transfer learning is applied 90%+ is achievable.

For this project I intend to apply the lesson learnt from the Kaggle competition following the success of P. Sermanet as expressed in his 21 Dec 2013 "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks" paper as well as the research publication dated June 2018, "A Study on CNN Transfer Learning for Image Classification", Aston Birmingham U.K. by Mahbub Hussain, Jordan Bid, and Diego Faria. I shall use the CNN architecture with the appropriate number of hidden layers to maximize accuracy. Noteworthy is the 2016 Stanford University paper on "Dog Breed Identification" by W. LaRow, B. Mittl, and V. Singh; were an accuracy of 50%+ was realized using non-CNN architectures such as SVM, Logistic Regression, and K-Nearest Neighbor. This paper in contrasted to the papers above, coupled with the Kaggle competition results reinforces that pursuing a CNN solution is the optimal means for solving this problem.

Problem Statement

This problem is one of three that is provided by Udacity for the Machine Learning Engineer Capstone project. The objective is to build a pipeline to process real-world, user-supplied images. Given an image of a dog, design and implement an algorithm that will identify an estimate of the canine's breed. If supplied an image of a human, develop code that will predict the resemblance to a particular dog breed specifying the probability of a match. Explore state-of-the-art CNN models for classification, and make important design decisions about the user experience for this application. The goal is to piece together a series of models designed to perform various tasks in a data processing pipeline while engineer a real-world application.

Datasets and Inputs

The data to be used is sourced by UDACITY. The UDACITY data has 13,233 human images and 8,351 images of dogs. Since I am choosing to use the jupyter notebook, these files are located in the '/data/lfw/' and '/data/dog_images/' respectively. Filename of the images will be transformed into labels for use during model training. Since part of this project requires the use of the pre-trained VGG-16 model for dog detection, all input data submitted to the model for this project will need preprocessing. Input image data will have to undergo some processing to align with the input data characteristics of the VGG-16 classifier. The dataset will be read into the notebook and set up as testing data for the dog detector, i.e., resized. All the input data will be normalized. Proposed transformation parameters for the images will be:

- Resize image to 256,
- Set the center crop to 224,
- Transform the image to a python tensor,
- Normalize the mean to 0.485, 0.456, 0.406,
- Normalize the standard deviation to 0.229, 0.224, and 0.225.

The transformation values are all float. This information is gathered from the pytorch.org, i.e., https://pytorch.org/docs/master/torchvision/models.html. The dataset for the custom built CNN will be read into the notebook and then split into three parts, training, validation, and testing. Training data will be processed through transformations like scaling, cropping, and then flipping. The input data will be resized to 224x224 pixels similar to what is required by the VGG-16 pre-trained network. As for the testing data reuse of the work described above and add a validation dataset which will also be resized.
Creating labels for the image data will be accomplished by first determining the number of dog breed categories in the data set then apply one-hot-encoding to the categories.

Solution Statement

I will follow the outline of the Dog Project notebook, steps 1-6 applying pytorch to all aspects of the problem. I'll use OpenCV's implementation of the Harr feature-based cascade classifier and the face detector to identify human faces in the input-images. The VGG-16 pre-trained classifier will be applied for the dog detector and I'll build a predictor that returns the index to the class index of the dictionary on ImageNet.
The solution approach for the creation of the CNN classifier from scratch is to build a feed-forward, fully connected network classifier and initially start with four hidden layers where the result/output layer will represent the total number of dog breeds. The solution will use ReLU activations and dropouts. The output of this classification solution will be the top number of class matches for the input image based on probability. Loss will be calculated using the negative log likelihood loss function (optimizer) paired with the Softmax function that will be used to calculate the output is a probability distribution. Hyper-parameters planned are as follows:

- Learning rate;
- Dropout, random node skipping;
- Number of epochs;
- Number of hidden layers;
- Batch – size.

Training of the CNN model will be realized using gradient decent using the Adam Stochastic Optimization method. Pytorch offers the Optim class for this purpose where the hyper-parameters to the Adam method are learning rate and the CNN model parameters. GPU will be enabled in the notebook during the training of the model.
The goal is to achieve training and test loss of approximately 1%. Lastly, I'll use transfer learning to create a CNN that can identify dog breed from images and attain at least 60% accuracy on the test set.

Benchmark Model

The bench mark model that I used for this project was the Kaggle Dog vs. Cat classifier. This problem is very similar to the dog breed classification problem. There are numerous examples on the internet using pytorch or Keras-tensorflow. This link https://medium.com/predict/using-pytorch-for-kaggles-famous-dogs-vs-cats-challenge-part-1-preprocessing-and-training-407017e1a10c provides the steps used to solve this problem using pytorch. A second benchmark that I used is the image classifier that I built in the Udacity Nano degree, Introduction to Machine Learning.
The Intro to ML Nano degree project classifier for flower category provided the following results:

- Accuracy of 83%;
- Training and testing loss of 0.7 and 0.69 respectively.

The Kaggle Dog vs. Cat project classifier provided the following results:

- Accuracy of 80%; without transfer;
- https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8 has a training and testing loss of 0.2 and 0.2 respectively for a total of ~17 epochs.

This project needs to secure at least 60% accuracy.

Evaluation Metrics

The metrics to be implemented for this project are Accuracy and Loss, training and test.
Track the ***loss*** and ***accuracy*** on the validation dataset. Accuracy is to be calculated by taking the output of the trained model looking at the predictions data and comparing these to the test data labels summating the results. Training and testing loss is calculated using Loss will be calculated using the negative log likelihood loss function (optimizer).

Project Design

The project notebook has the workflow for the solution seven steps zero through six. Below each step is described.

Step 0. Import Datasets:

Since the data is already in the notebook; I determine the number of human and dog images available, calculate the number of dog breeds for transforming categorical data into one-hot-encoded labels Numpy arrays. Determine if the data set is imbalanced.

Step 1. Detect Human:

Use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images and test the performance of the face detector. Calculate the percentages of the first 100 images in the human_file and dog_files have detected faces.

Step 2. Detect Dogs:

Use a pre-trained VGG_16 model and preprocess the data by transforming it using torchvision's datasets, transforms classes. Focus only on test data. Write a function that accepts a path to an image as input and returns the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive. Calculate the percentages of the first 100 images in the human_file and dog_files have detected faces.

Step 3. Create a CNN to Classify Dog Breeds (from Scratch):

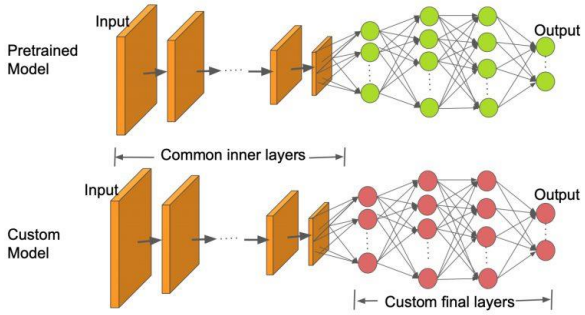
Create a CNN that classifies dog breeds *from scratch,* no transfer learning is to be applied. Accuracy must be at least 10%. I will use the preprocessed test data from above, but add validation and training datasets. A model with four hidden layers will be built and trained. I will use ReLU activations and dropouts. Loss will be calculated using the negative log likelihood loss function and the Softmax function to calculate the output. The metrics section of this proposal will be applied for accuracy and loss, both test and training.

Step 4.Create a CNN to Classify Dog Breeds (using Transfer Learning)

I'll create a CNN that uses transfer learning of the ResNet50 model classifier to improve the performance of the identify dog breed project. The goal is to attain at least 60% accuracy on the test set. The graphic below illustrates the process and where I will replace the final ResNet50 layer with custom final layers.

(1) Load the resnet_model = models.resent50(pretrained=true)
(2) Set the params in the resnet_model to param.requires_grad = False
(3) Use my_classifer, that is the model from scratch built above in step 3.

(4) Resnet_model.classifier = my_classifier



I'll write three separate data loaders for the training, validation, and test datasets of dog images (located at dogImages/train, dogImages/valid, and dogImages/test, respectively). For example:

```
Data        = { 'train': datasets.ImageFolder(root= dogImages/train, transform=image_transforms['train'])}
train_data = DataLoader(data['train'],batch_size=32,shuffle=True)
```

Test and Validation will be added to Data above as well. This is basically the same data loader from the previous steps. I'll plan on using the same loss and optimizer functions mentioned above. Next I'll train and validate the model and save the model parameters at file path 'model_transfer.pt'
The goal is 60% or greater accuracy.

Step 5. Write Your Algorithm:

I'll use the human face and dog detectors along with the transfer learner above to predict if a dog or human is detected or neither. If a human is detected the match to the closest dog breed will be presented.

Step 6. Test Your Algorithm:

This is the testing phase with various image inputs.