Capstone Project

Dog Breed Classifier

March 31, 2020

Machine learning Engineer nanodegree

AbdElrhman Mohamed

Udacity

Definition

Overview

The breed classifier is one of the most famous problems in machine learning. The problem we want to solve is making a model who can identify the dog and it's breed in a given image or human, this is a multi-class classification where we can use supervised machine learning to solve this problem as we have well labeled data, I would love to use this problem in web application.

Problem statement

The project goal is to build a machine learning model that can be used within web application to process real world images uploaded by the user to identify either:

- *if a dog is detected in the image, return the predicted breed.*
- *if a human is detected in the image, return the resembling dog breed.*
- *if neither human nor dog is detected in the image, provide output that indicates an error.*

We'll use Convolutional Neural Network (CNN) or we'll use CNN transfer learning technique to solve this problem.

Metrics

I used the sample accuracy rule to measure the model accuracy which is:

$$\text{Accuracy} = \frac{\textit{true predicated images}}{\textit{number of images in the data}}$$

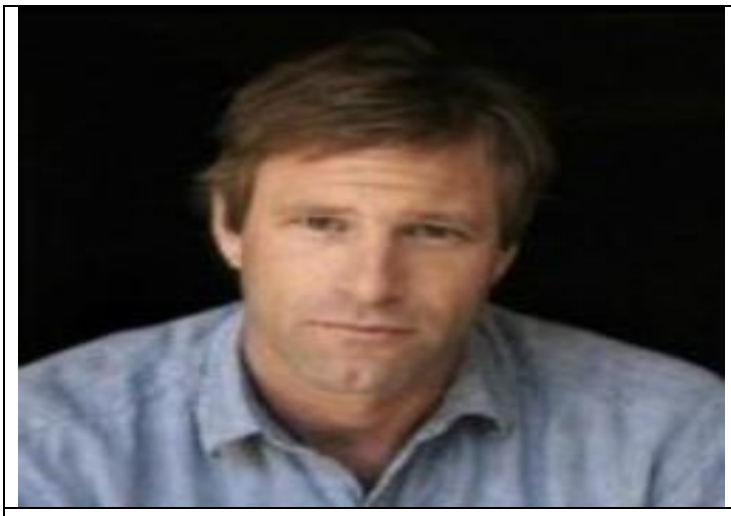
Analysis

Data Exploration

The data used in the project is provided by Udacity. The dataset has pictures of dogs and humans. Dog images dataset:

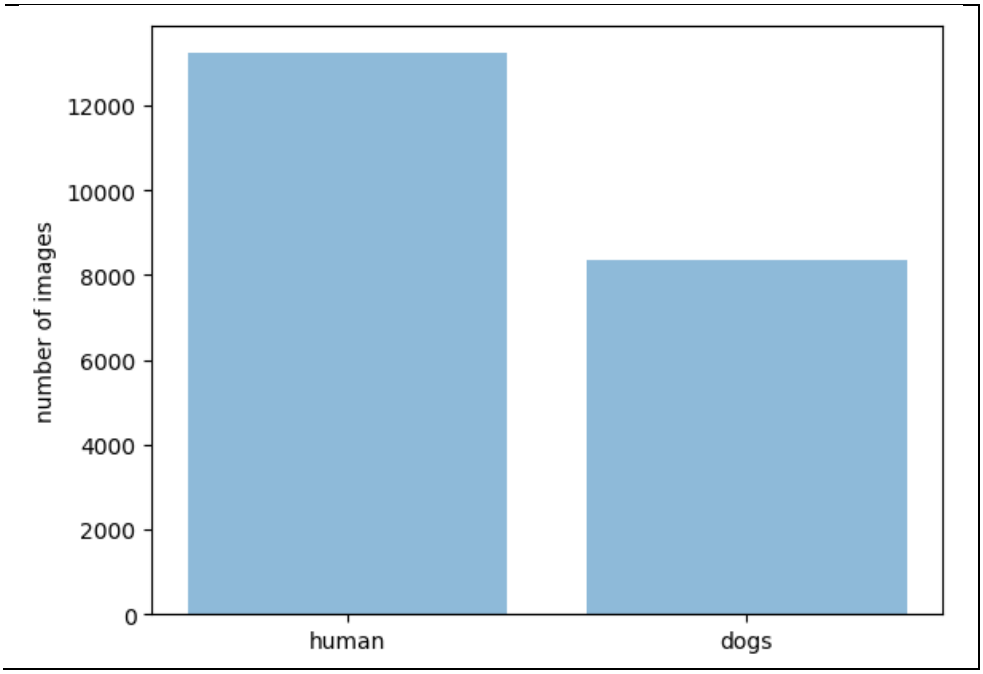
- *There are 13233 total human images.*
  - *With 5750 sorted by human names*
  - *All the images are 250X250*
  - *Have different angels and different backgrounds.*
- *There are 8351 total dog images with 133 classes*
  - *Train images 6,680 (each class have between 27 to77).*
  - *Test images 836 (each class have between 4 to 9).*
  - *Valid images are 835 (each class have between 3 to 10).*

Data visualization



Sample of the images from the used dataset for training.

There are 13233 total human images, as well as are 8351 tot al dog images.



Algorithms and Techniques

The classifier is a Convolutional Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches; fortunately, the COCO and COCO-Text datasets are big enough. The algorithm outputs an assigned probability for each class; this can be used to reduce the number of false positives using a threshold. (The tradeoff is that this increases the number of false negatives.).
The following parameters can be turned to optimize the classifier:

- *Classification classes*
- *Training parameters*
  - *Training length (number of epochs).*
  - *Batch size (how many images to look at once during a single training step).*
  - *Solver type (what algorithm to use for learning).*
  - *Learning rate (how fast to learn; this can be dynamic).*
  - *Weight decay (prevents the model being dominated by a few "neurons").*
  - *Momentum (takes the previous learning step into account when calculating the next one).*

- *Neural network architecture*
  - *Number of layers*
  - *Layer types (convolutional, fully connected, or pooling)*
  - *Layer parameters.*
- *Preprocessing parameters.*

During training, both the training and the validation sets are loaded into the RAM. After that, random batches are selected to be loaded into the GPU memory for processing. The training is done using the ResNet50 pretrained model. (Which has The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.)

Benchmark

To create an initial benchmark for the classifier, I used Pytorch deep learning library, to try multiple architectures. The "standard" ResNet architecture achieved the best accuracy, around 83%.
But the CNN model created from scratch has 12% accuracy.

Methodology

Data Preprocessing

The preprocessing done in the "Prepare data" notebook consists of the following steps:

1. *I've applied RandomResizedCrop & RandomHorizontalFlip to just train_data. This for doing both image augmentations and resizing jobs.*
2. *Image augmentation randomize the dataset to prevent overfitting.*
3. *I expect better performance of model when it's predicting toward test_data.*
4. *On the other hand, I've Resized the images to be 256px*
5. *then center crop for images 224 X 224.*
6. *because the valid_data will be used in validation process, I will not do image augmentations.*

This was chosen to stay in line with the standard input for pre trained pytoch models such as resnet50 I have chosen it over the VGG16 (because ResNet have much depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity). Seeing as those projects have success with those inputs as well as they have worked well for me in prior projects. The resize before the crop enlarges the subject in the image and makes them a more dominant part, likely removing a layer of not essential data from the edges.

As to extra edits to the test set I added in a random horizontal flip random rotation of 30. This was done in an effort to avoid overfitting. I chose these specific changes due to then working well for me in a different image classification model I made in the intro to AI with python nanodegree.

There are also some preprocessing steps which are done as the images get loaded into memory before training:

1- *The images are converted to grayscale.*
2- *The pixel values get transformed to 16-bit floats.*

Implementation

the classifier was trained on the preprocessed training data. This was done in a Jupyter notebook (titled "Create and freeze graph"), and can be further divided into the following steps:

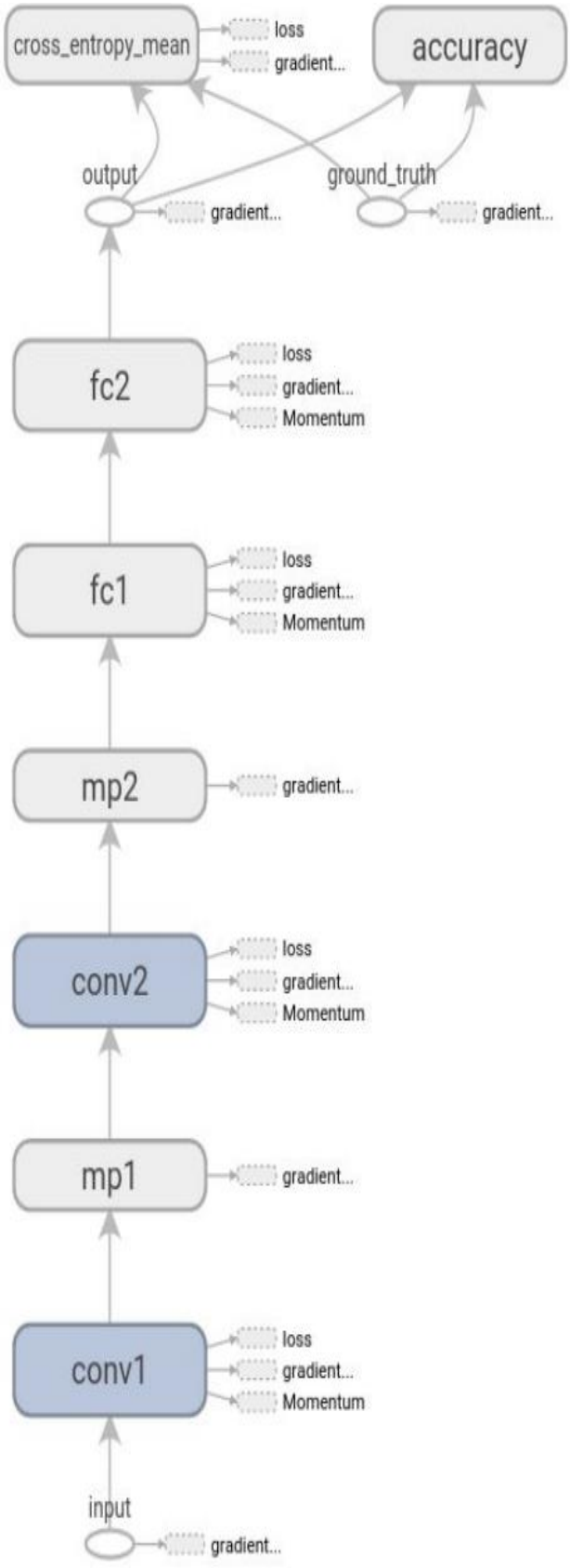
1- *Load both the training and validation images into memory, preprocessing them as described in the previous section.*
2- *the previous section:*
    a. *get_batch(...): Draws a random sample from the training/validation data.*
    b. *fill_feed_dict(...): Creates a feed_dict, which is a Python dictionary that contains all of the data required for a single training step (a batch of images, their labels, and the learning rate)*
3- *Define the network architecture and training parameters.*
4- *Define the loss function, accuracy.*
5- *Train the network, logging the validation/training loss and the validation accuracy.*
6- *Plot the logged values.*
7- *If the accuracy is not high enough, return to step 3.*
8- *Save and freeze the trained network.*

The illustration on the right shows the computational graph, which includes the network architecture and also the variables that are only present during training. The acronyms can be read as follows:

- *FC: Fully connected layer*
- *Conv: Convolutional layer*
- *MP: Max pooling layer*

The following can be seen by looking at the graph:

- *The loss function is mainly composed of the mean cross entropy error.*
- *Large weights and biases of all four layers are penalized by using weight decay. (The weights are added to the loss function after they are multiplied by their specific weight decay rates.)*



Refinement

The CNN have created from scratch have accuracy of 12%, Though it meets the requirement, the model can be significantly improved by using transfer learning. To create CNN with transfer learning, I have selected the ResNet50 model, which is pre-trained on ImageNet dataset, the architecture is 50 layers deep. The last convolutional output of ResNet50 is feed as input to our model. We must add a fully connected layer to produce 133 output (one for each dog breed). The model performed extremely well when compared to CNN from scratch. After 50 epochs of training the model accuracy is 83%.

Results

Model Evaluation and Validation

1. ***Human Face detector***: *Percentage of the first 100 images in human_files have a detected human face:98% but 17% of dog faces.*
2. ***Dog Face detector***:
    a. ***Using VGG16*** *: Percentage of the first 100 images in human_files have a detected human face:0% but 100% of dog faces.*
    b. ***Using ResNet50:*** *Percentage of the first 100 images in human_files have a detected human face:1% but 100% of dog faces.*
3. *Create a CNN to Classify Dog Breeds (from Scratch):*
    a. *You must create your CNN from scratch (so, you can't use transfer learning yet!), and you must attain a test accuracy of at 12%*
4. *Create a CNN to Classify Dog Breeds (using Transfer Learning)*
    a. *I have picked ResNet50 to make this model after 50 epochs of training, the model reached 83% of accuracy .*

Justification

By comparing the model benchmark result wich is 60% with the final model 83% accuracy.
The final model preforming good enough for having adequately solved the problem but there's also way for improving the performance by choosing model with more layer, more complex layers or increase the number of epochs.

Contact me at:
LinkedIn : @AbdElrhman-m
GitHub: @ AbdElrhman-m
AbdElrhman.m056@gmail.com

Thank you.