Machine Learning Engineer Nanodegree

Capstone Project

Carolina Abs da Cruz Udacity
Ocotber 5th, 2019

I. Definition

I. Overview

The main goal of this project is to develop an algorithm that could eventually be used as part of a mobile or web app. The code accepts any user-supplied image as input. If a dog is detected in the image, it provides an estimate of the dog's breed over 100 dog breed categories. In case a a human face is detected, it provides an estimate of the dog breed that is most resembling. I've utilized the Udacity training space with Pytorch which was GPU-enabled to allow for the models to run quickly.

II: Problem Statement

The main goal of the project is to use a convolutional neural network (CNN) to predict dog breeds. Initially, a CNN model from scratch was built but the results were poor (11% accuracy). Since image recognition requires more complex feature detection the results were acceptable.

A transfer learning approach, using a p**re-trained VGG-16 Model¶**, was a much more accurate approach with 85 % of accuracy in only 5 epochs!

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

The end goal of the project is to provide a predicted breed for dog images and for humans, the function will tell you which dog the human most resembles.

Metrics

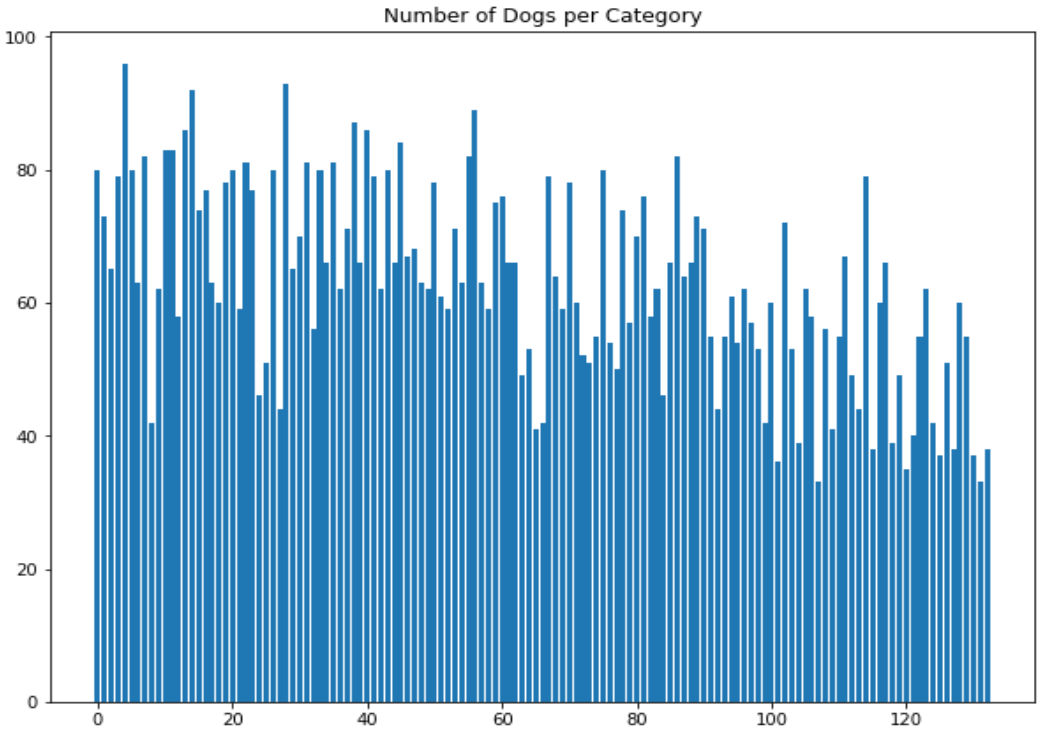
The models were evaluated using an accuracy metric utilizing a test set. The human and dog detectors were tested on 100 images of each. There are 6680 training dog images, 835 validation dog images and 836 test dog images.

OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images has been used. Thet percentage of the first 100 images in `human_files` having a detected human face is calculated.

Among the transfer learning methods, the VGG-16 model hasl been used, along with weights that have been trained on Imagenet, a very large, very popular dataset used for image classification and other vision tasks.

Here a generalized funtion count_detection_type has been created in order to be used with detect dogs and avoid code ducplication (both tasks are very similar).

Accuracy is chosen as the metric to evaluate the model performance. Each dog breed has around 50 images as you can see in the figure bellow, what makes the trainning dataset balanced, so accuracy should be a proper metric to select a good model.



II. Analysis

Data Exploration

The files containing the dog images and the list of dog breed categories were provided by Udacity. In addition, some human faces were provided to test the algorithm.

There are in all 8351 dog images with 6680 of them being training images and 13233 human faces.

Exploratory Visualization

The image sizes have different sizes from around 300–500 pixels height and width but many were much larger, but I didn't visualize or analyse more because I just reshaped all images to avoid any problem.

Each dog breed has around 50 images what makes the trainning dataset balanced.

I didn't think about more data exploratory visualization, I just transformated all the figures to avoid issues and actually it worked...

Algorithms and Techniques

Deep learning whit pre-trained models are used on computer vision and natural language processing tasks. Pre trainned models can be pratical when the dataset is not large for example. Models developed from scratch allows more flexibility and control of the code, but they are computing time more demanding which makes them more expensive.

Other learning algorithms or models can also be used for image classification. However CNN has been used as the model of choice for multiple reasons. The CNN architecture implicitly combines the benefits obtained by a standard neural network training with the convolution operation to efficiently classify images. The CNN (and its variants) are also scalable for large datasets, which is often the case when images need to be classified.

The reason why CNNs do so much better than classic neural networks on images and videos is that the since neural networks don't see any order in their inputs**.** If you shuffled all your images in the same way, the neural network would have the very same performance it has when trained on not shuffled images. In opposition CNNs take advantage of local spatial coherence of images and they reduce dramatically the number of operation needed to process the image. In a neural network each layer is connected to each pixel of the image what makes computational expensive.

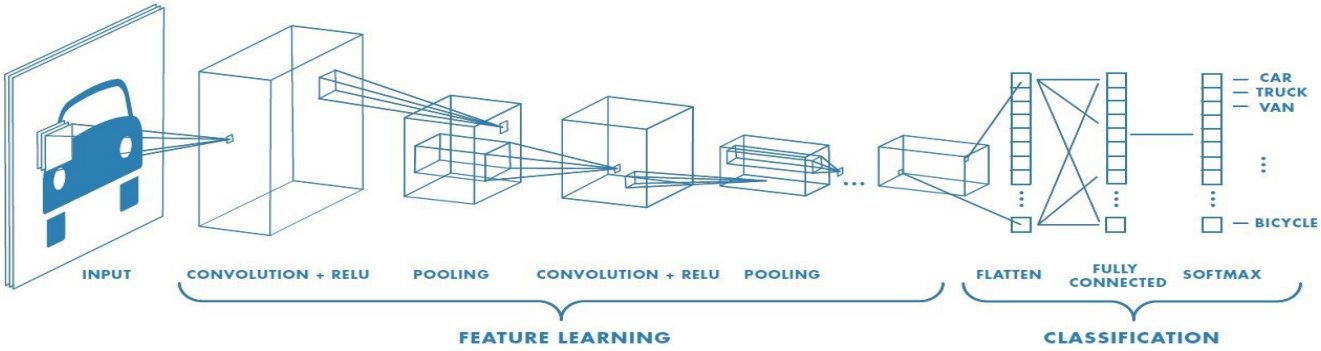
'Typically, a CNN architecture consists of convolutional layers, activation function, pooling layers, fully connected layers and normalization layers. A Convolution layer is able to successfully capture the spacial details in an image through the application of relevant filters. Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride and zero-padding. Pad of 32 has been used for the first layer, the stride of 2 was used which helped to decrease dimensionality and a padding of 1 was used in order to get details of the figure near to the bord.

The pooling layer, usually between the convolution layers, resizes the input spatially excpet in depth. The max pool layer has been used in this project.

The ReLu layer is used to effectively remove negative values from an activation map.

The Fully connected layer is applied after several convolutional and max pooling layers.

The advantage of this architecture performs a better fitting to the image dataset because there are a reduction in the number of parameters involved. So we can think that the network is trainned to understand better the sophistification of the image. Bellow there is a figure showing a typical CNN structure (figure obtained from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.



There are hyperparameters related to Network structure:

- Number of Hidden Layers and units:

Hidden layers are the layers between input layer and output layer.

Increasing the number of hidden layers can increasy accuracy. However, underfitting can be generated if the number of these layers is small.

- Dropout:

  Dropout is a technique to avoid overfitting (increase the validation accuracy).
- Network Weight Initialization:

  Mostly uniform distribution is used.
- Activation function:

  Activation functions are used to introduce nonlinearity to models.The **rectifier activation function** is the most popular and it has been used in this project.

Hyperparameters related to Training Algorithm:

- Learning Rate:

The learning rate defines how quickly a network updates its parameters.
Low learning rate makes the convergence being very slow. Larger learning rate speeds up the learning but and it can may not converge.

- Number of epochs:

  Number of epochs is the number of times the entire training data is shown to the network while training. A high number of epochs can lead to an overfitting.

- Batch size:

  Is the number of sub samples sent to the network after which parameter update happens.

Benchmark

The accuracy of the model developed from scratch was asked to be at least 10 % and the accuracy of the transfer learning model should reach 70 % at least. This values come from the notebook used to develop the models. These accuracies were determined by udacity. A low value for the model from scratch is reasonable since it would demand lot of GPU time and a big cluster to develop a model to caputure the detail of the images. A higher value for the pre trainned value is reasonable since we can have a much more complex network in this configuration and we can capture much more details of the image.

III. Methodology

Data Preprocessing

A major project development stage was the data pre processing. All the images were reshaped to 224 in order to be used in both pre trainned and from scratch models.

Moreover, the trainning dataset was augmented. Image data augmentation is a technique that is used to increase the size of a training dataset by creating modified versions of images in the dataset.

The augmentation techniques produce variations of the images that can affect the ability of the fit models to generalize what they have learned to new images.

All datasets were normalized. The pixel values in images must be scaled prior to providing the images as input to a deep learning neural network model during the training or evaluation of the model. The standard normalization of mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225] have been used.

And all images have been converted to tensor what improves performance computation ;-) I spent most of the time here since I got many errors trainning the model...

Implementation

Once the functions for detecting humans and dogs in images are ready, the dog breed from images has been be predicted. In this step, a CNN algorithm that classifies dog breeds has been created *from scratch* in order to get at least 10% in the test accuracy

This task was very interesting. The idea in this section is to develop a CNN model as simple as possible to avoid performance issues to get a minimal accuracy of 10 %. I did some searchs in internet about the subject and there are lots of differents solutions.

In my configuration I decided to have only 3 convolulation layers and 2 FC layers. I am obsiously using a Relu function and maxpool layers between the convolution layers.

I had some performance issues so fixed it was a **major project development stage.** I decided to use a stride of size 2 and it worked very nicely! It helped a lot. Moreover, I started with 32 filters in the first Convolution layer. In the figure bellow there is a print of the layers used and the parameters of the model.

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=6272, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
  (dropout): Dropout(p=0.3)
)
```

The accuracy was 11 %.

I used Cross entropy loss and Adam optimizer. Another **major project development stage was** the choice of the learning rate of the optmizer I chosed. I noticed the accuracy was more than 10 times better decreasing the learning rate!

In the next step step, transfer learning has been used to create a CNN that can identify dog breed from images in order to get at least 60% accuracy on the test set.

I just adapted the last layer of the pre trainned model vgg16 to have 133 outputs (number of dog breeds).

I used the same dataloader what was very pratical.

Refinement

Major contribution to improve results were data normalization, adjusting the stride value to the vaue 2 in order to reduce dimensionality and changing the learning rate parameters. I started with the standard value and I reduced to 0.0001 what improved a lot the validation loss (around 10 times for the first value).

Ps.: In the last report I wrote the accuracy changed but actually I made a mistake, what changed 10 times was the first values of the validation loss.

IV. Results

Model Evaluation and Validation

In this step, an algorithm has been developed that takes a path and determines if the image contains a human face, a dog or neither. Then:

•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** is detected in the image, provide output that indicates an error.

In this step I used the pre trainned model to get the dog breed for obvious reasons.

I Tested the algorithm with at least six images (from internet).

I download the images from internet and the images were stored locally so they could be used in the model since I needed a path.

I had interesting results:

```
Dog detected :-) and the breed is:  Chihuahua
```

```
Human detected :-) and the resembling dog breed is: Poodle
```

Justification

```
Epoch: 7        Training Loss: 3.849107         Validation Loss: 4.026291
Validation loss decreased (4.082339 --> 4.026291).  Saving model ...
Epoch: 8        Training Loss: 3.725503         Validation Loss: 3.996468
Validation loss decreased (4.026291 --> 3.996468).  Saving model ...
Epoch: 9        Training Loss: 3.611827         Validation Loss: 3.961194
Validation loss decreased (3.996468 --> 3.961194).  Saving model ...
Epoch: 10       Training Loss: 3.504261         Validation Loss: 3.941575
Validation loss decreased (3.961194 --> 3.941575).  Saving model ...
```

The accuracy of the model developed from Scracth was not significative, 11%, but it was around the asked value, 10 %. I only used 10 epochs and as you can see in the figure bellow both loss and validation accuracies were decreasing so it was completely possible to use more epochs to improve the results but since I didn't want to spend all my Gpu time so I stopped there. This low value is due to the

simplicity of the model. It would be necessary to increase much more the complexity to reach better accuracies.

The validation loss was much better than the ones obtaines from the previous model from scratch as you can see bellow and for the test value I got 84 % what it is a good result sinve I was asked to obtain at least 70 %.

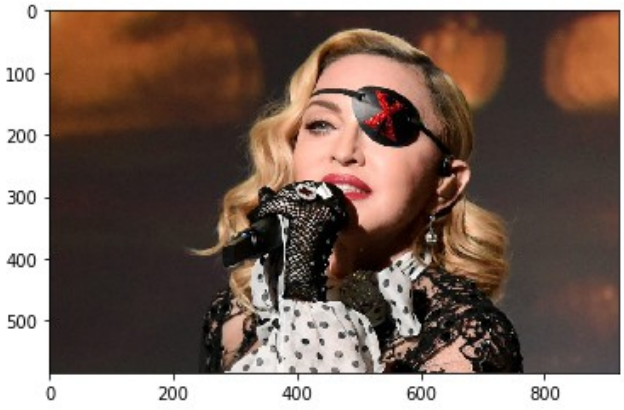
```
Epoch: 1        Training Loss: 0.593417         Validation Loss: 0.447194
Validation loss decreased (inf --> 0.447194).  Saving model ...
Epoch: 2        Training Loss: 0.534306         Validation Loss: 0.451095
Epoch: 3        Training Loss: 0.505782         Validation Loss: 0.424775
Validation loss decreased (0.447194 --> 0.424775).  Saving model ...
Epoch: 4        Training Loss: 0.477269         Validation Loss: 0.418957
Validation loss decreased (0.424775 --> 0.418957).  Saving model ...
Epoch: 5        Training Loss: 0.471790         Validation Loss: 0.402350
Validation loss decreased (0.418957 --> 0.402350).  Saving model ...
```

V. Conclusion

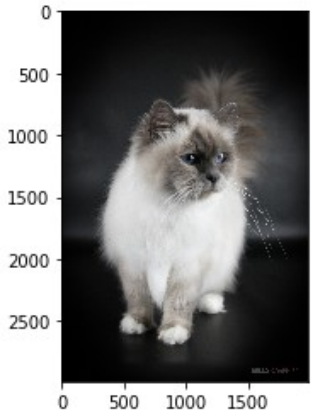
Free-Form Visualization

The detection of dogs seems quite good since the cat of the figure bellow was not detected as dog but the human face recognition is poor as expected from the performance obtained in the beginning of this project and the cat was detected as human!

```
Error, neither face or dog detected!
```




```
Human detected :-) and the resembling dog breed is: Japanese chin
```



Reflection

It was interesting to final part where I compared different images form internet with the model.

I faced difficulties during the data pre processing, I had many errors when I was running the CNN model from scratch so the data transformation was very important to get acceptable results.

The “difficult part” (and not interesting at all) during this project is to write a report of more than 9 pages...

I wouldn't use the CNN model from scratch because the accuray was very low but the pre trainned model gives good results. However the final results fit the expectations of the problem from udacity.

Improvement

The model from scratch could be massivaly improved adding more layers but it would demand much more GPU computing time.

For the pre trainned model the results are quite trustable

The results for the human detection were very poor. So I would probably use search and use another algorithm for it.