Machine Learning Engineer Nanodegree

Capstone Project – Dog Breed Classification

Lucas Moura - May 22nd, 2020

Definition

Project Overview

Nowadays computer vision has gained a big boost due to the advances in Deep Learning architectures. It is a reality now for a computer to recognize and even classify objects in the wild using, normally, Convolutional Neural Networks [3] based architectures. And for that reason, it is becoming necessary for anyone who wants to work in the area to have a knowledge on these kinds of application and its development.

Due to the increasing popularity of this field and my curiosity of doing something related to it, it was decided that this project would try to accomplish a dog breed classifier using transfer learning technique within PyTorch deep learning framework inside a Cloud Based Machine Learning service such as the SageMaker with the Stanford Dogs Dataset[1].

Project Statement

The goal is to construct a deep learning model that can successfully classify dog breeds. This involves completing the following tasks:

- Download and explore the data from Stanford Dog Breed dataset
- Train a model using transfer learning out of a successful deep learning architecture
- Evaluate its metrics and discuss the results

The resulting model is expected to accomplish at least 60% accuracy and F1 score, and above 50% recall and precision on validation dataset.

Metrics

Accuracy is one of the most common metrics for classification problems. Its capacity is simply to show how much your model is making right predictions, and that is something good to compare models and to see how well your model is doing.

$$Accuracy = \frac{true\ positives + true\ negatives}{All\ samples}$$

But accuracy cannot tell the whole story by itself. By example: if I have 2 classes *0, 1* and for *class 0,* I have 990 samples and for *class 1* only 10 samples. My model then made a prediction and got right all of *class 0* right but none of *class 1*. By these terms, my model got 99% accuracy, but what that means is that it can't classify anything from *class 1* rightly.

For that problem we have precision and recall:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

Precision specifies how much false positives we got: that is a good metric to keep a look for false positives. The more false positives you have the lower the precision.

Recall is a metric that we can see as being the model sensitivity to detect items of each class. In the accuracy example a model would have 100% recall if also got all the items of *class 1* right. The more false negatives the lower the recall.

$$\text{F1 Score} = 2 * \frac{(\text{Recall} * \text{Precision})}{\text{Recall} + \text{Precision}}$$

And by suggestion of the capstone project reviewer: the F1-Score. It's said to be a better metric than accuracy due to the fact that is a weighted average of precision and recall. In a sense that means that both false positive rate and false negatives rate are taken in consideration.

All these metrics will be used to evaluate the success of the dog breed classification model.

Analysis

Data Exploration

The Stanford dog breed dataset is composed of over 120 classes with over 20000 images. These images are part of ImageNet Database and was created for the task of fine-grained image classification - due to some dogs having near identical colors, it can be quite trick for a network to work well. The images have colors and different sizes, illuminations, and dog location on the scene.

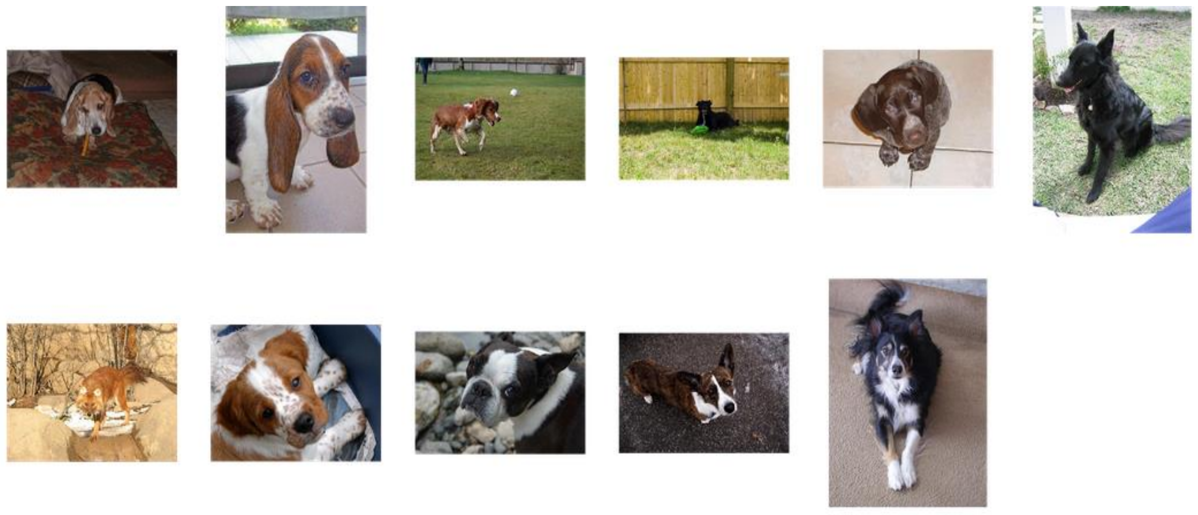


Figure 1 - *Images from Stanford Dog Breed Dataset*

The Dataset separates all the 120 different breeds per folder and each folder contains n distinct sample images of dogs.

Exploratory Visualization

One common worry about using data for modeling is how well is this data distributed. We already know that we have more than 20000 images separated between 120 classes. Although we do not know how well distributed these classes are by the quantity of samples inside them on this dataset. One thing that could occur if there is a big imbalance is that of too much bias to a certain class and, even with ways to contour it, should be avoided.

Below a plot of the frequency of classes per quantity of images in the dataset:

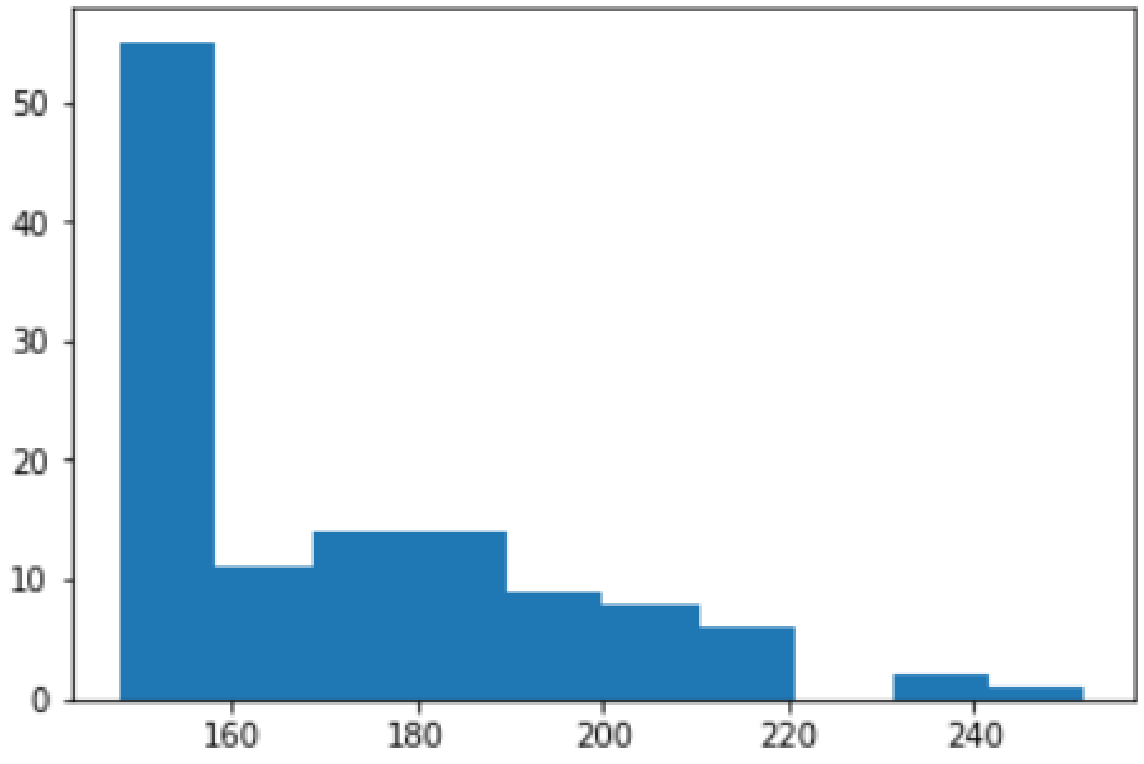


Figure 2 *- Distribution of classes per quantity of images in it. We can see that even though there is a certain spread, this distribution is mostly exponential and goes from 150 - 260 (min - max).*

Even though the classes are well distributed, something bad still could happen on the train/test split, the classes could be underrepresented. For that reason, it was made explicitly that the split and creation of these dataset would be stratified based on the original one. What that means is that an algorithm will try to distribute the data between the split the same way as it is distributed in the population (in our case the Stanford Dog Dataset).

Algorithms and Techniques

The classifier that will be used is a ResNet50 classifier, one that is based on Convolutional Neural Networks and had state-of-the-art performance on the ImageNet Dataset. It is already pretrained on this same dataset and in the output of this network it will be added custom layers to fit specifically to our problem!

Since the network is already pretrained this means that our training jobs will take much less time to complete due to not needing to update these weights no more, not only this, the already pretrained filter are able to extract good features and that will make our classifier will mostly likely perform better than if we had trained it all from scratch. The literature is extensive on transfer learning and nowadays is a state-of-the-art technique.

Besides that, the images that will be inputted in the classifier for training will be augmented. That means that we will add deliberate add noise on the image to make the life of the classifier harder, or in other words, prevent overfit and make it more likely to better generalize.

For the task of constructing and optimizing the model, the cross-entropy loss was chosen to be the loss function of the classifier and ADAM as the optimizer. Cross-Entropy Loss is the most common loss metric for this kind of problem, and it works well across many fields. Adam is a faster converging optimizer than Stochastic Gradient Descent, it may not be optimal for fine tuning a model, but in our case, since we will be initializing new layers from scratch, we may want them to converge a bit faster.

Benchmark

It was decided that the model would be benchmarked with the top performants' submissions of the Kaggle competition for this same dataset. The best result to this day [https://www.kaggle.com/phylake1337/0-18-loss-simple-feature-extractors] used transfer learning to achieve a really good performance on the model, using the weights from multiple models, it could achieve a training loss of 0.1705, training accuracy of 0.9461, and validation loss and accuracy of 0.1966 and 0.9404 respectively.

This is a ceiling of performance that the model will set as a goal. But it is going to be acceptable to have any accuracy and F1-Score above 60%. This exact metric means that our model was indeed able to learn, maybe not to an applicable manner, but it's definitely better than having a classifier that classifies randomly based on the class, especially as a multi-class problem, this kind of classification becomes even less accurate.

Methodology

The objective of this project is to successfully create and train a model in SageMaker using the Pytorch Framework. For that the following steps were followed:

Environment Setting

Some time were put to create the GitHub repository of the project and create the jupyter notebook instance inside of SageMaker. Besides that, it was solicited to amazon to raise the limit on the p2 instance of EC2. Now we have the project environment set and can go to de development.

The project was separated in two notebooks, each one dealing with a part of the project: Data Preparation and Analysis, Model training and evaluation. The model construction and training scripts where put as python modules inside the source folder.

Data Preparation

The first part of the project involves fetching the data from the web. A link to a repository with the dataset is kindly given by Stanford [http://vision.stanford.edu/aditya86/ImageNetDogs/images.tar] After that there is some assertions to check the fidelity and consistency of the downloaded data.

Next step is to visualize some of the data and run the analysis that were already discussed on this report. We want to look the data and visualize some of the images inside the dataset, besides that, it's important to look for some features of it and look for inconsistencies.

After that it is time to prepare the data for the test/train/validation split. The result will be all the images separated by its normal folder classes each inside a train/test/validation folder. And the ending folder architecture is the following.

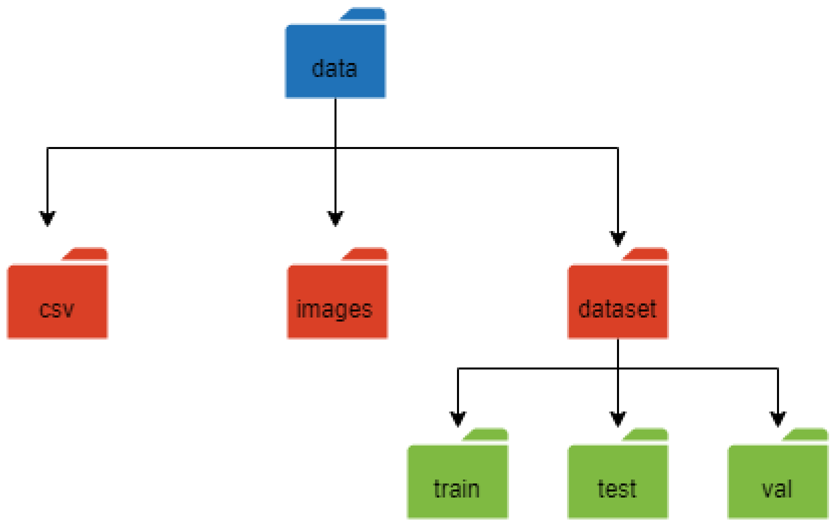


Figure 3- Images of data folder archtecture. The csv folder contains the csv with metadata about the images. The images folder contains the original dataset. The dataset contains 3 folders each one with its respectives dog breed classes and images inside.

Since this dataset is widely known, and as a scientific dataset, it was created for the community to benchmark its models, and to train them in tasks such as fine grain image classification or in general dog breed classification. All the actual preprocessing of the data referring to the lookout of outliers or corrupted images and such were already pre removed by the creators of this dataset. Even though that being true some data corruption could happen in the downloading or data preparation process and that is why we run some of these assertions.

Model definition

Uploading data to S3

After organizing the folders in the cited manner, we must upload the data to S3 so that the PyTorch estimator, that will be running the training job, on SageMaker, can get the training and test data. This is all simplified using SageMaker's python APIs.

Running Training

The estimator is rightly set with all specific hyperparameter to the model and training, and it is ready to start the training job. For that it is called "*estimator.fit*" with parameter "*{'train': os.path.join(input_data, 'train'), 'test': os.path.join(input_data, 'test')}*" wich have the S3 location of train and test data.

```
# instantiate a pytorch estimator
estimator = PyTorch(entry_point='train.py',
                    source_dir='source', # this should be just "source" for your code
                    role=role,
                    framework_version='1.5.0',
                    train_instance_count=1,
                    train_instance_type='ml.p2.xlarge',
                    output_path=output_path,
                    sagemaker_session=sagemaker_session,
                    hyperparameters={
                        'output_dim': 120,
                        'hidden_dim': 1000,
                        'epochs': 20, # could change to higher
                        'batch-size': 32,
                        'lr': 0.01
                    })
```

During training it is always good to look for the running metrics of your model training, if you see that the model is not conversing, then it is time to cancel the training job and restart with different configuration. During the construction of a model this process will happen lots of times.

Deploying model on SageMaker and Evaluating it

After the training is finished, it is time to deploy the model. For that it is simple: "*estimator.deploy*" and setting the wanted configuration of ec2 instance and instance count. This is done so that we can evaluate our model or simply serve it to a service. In our case it was used to test it with the validation data. What that means is to feed our model with data that it did not seen in train/test and to see how it behaves. After that simply remove the endpoint and everything is done.

Refinement

There was a problem with the speed of training in this setup, a problem that this incurs is the big pricing in each training. To resolve this, I had to take some steps that were based on notebooks made by the SageMaker team. Before I was using the train loaders with default worker, that means that only the central process would be responsible for the loading of images, and the images would be loaded to memory only when requested by the program. The machine was the model was trained contained 4 CPUs and setting the number of workers to that of the CPU reduced the training time from 30 minutes to around 20 minutes. Also, for further optimization, the model was set to run on Data Parallel mode.

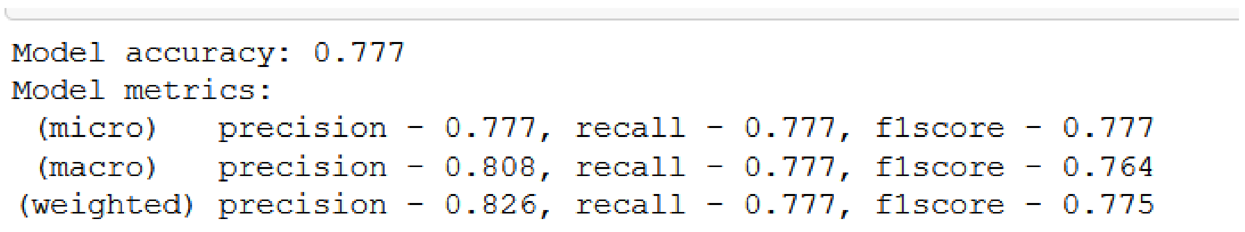
Results

Model Evaluation and Validation

The resnet50 model based that was constructed used feature extraction of the old model and a custom fully connected layer to make the final classification. This is a common approach when trying to make transfer learning. The *ADAM* optimizer was used to try to make it converge faster, even though *SGD* had worked well in some tests. It was trained in 20 epochs with a batch size of 32, and default learning rate of *0.01*.

The train input was transformed with a random crop on the image, a random rotation of up to 10º degrees, and resized to *244 x 244*, that is the input of the *ResNet* model. This made for a good set of images for training. Meaning that it would be hard to the classifier to overfit on this training data in such few epochs. The best model was saved based on the validation accuracy.

The model was then validated with unseen data separated from the dataset on the train/test split and got the following results:



```
Model accuracy: 0.777
Model metrics:
  (micro)    precision - 0.777, recall - 0.777, f1score - 0.777
  (macro)    precision - 0.808, recall - 0.777, f1score - 0.764
 (weighted) precision - 0.826, recall - 0.777, f1score - 0.775
```

These metrics are enough to say that the model has learned, although it seems that in this training it did not achieve the performance of other types of models based on transfer learning such as those that were already shown in this report.

One possible justification is that it did not train for long enough, or that the learning rate could be high. Although something unexpected happened during the final epochs of training: The train loss was still going down, meaning that the model was still learning, and accuracy was getting higher for both the test and train set in a expected manner, although the test loss was rising slightly up. One possible explanation found is that while the model was getting better at identifying certain classes of dog breed it could be getting way worse at classifying some others specific classes and missing them by a lot.

Justification

In the 3 types of looking up metrics, based on micro, macro and weighted averaging on the validation dataset we can see that the model has indeed surpassed the wanted 60% accuracy and F1-score, but it did not got anywhere as close to those really performatic models based on transfer learning trained on this dataset.

In the specific model to benchmark the author used various kinds of ImageNet based classifier, such as VGG, Resnet, AlexNet to transfer learn. This can occur in even more capacity of abstraction from the model since they have trained different filters on the same dataset. I tried a similar approach in PyTorch with 2 different models and even using an ensemble technique, but, that did not end up in success. All the losses got as low as that observed in the model made by Ahmed Saied, but with accuracies going around 50%.

Due to the incurring costs of training on SageMaker and the fact that I do not live in the United States made me drift away from trying to experiment with more powerful machines and models.

Conclusion

Free-Form Visualization

These problems of image visualization are quite important for humanity in general. Being able to detect objects of all kind and manners is useful to a lot of fields, going from logistics and medical application to security and document analysis. Although it can be difficult and labor intensive to create and execute tasks for these different tasks. It all starts on creating and labeling data and processing it to be used as input for our model. This ETL (Extract, Transform, Load) general process is another field by itself. Then we come to the task of modelling a Deep Learn model in some framework (Keras, Caffe, Pytorch, …), training it and evaluating its performance, to eventually deploy it in the real world. This last part can go through lots of experimentation, failing, and iterations. And for lots of times this training requires time, good hardware, and subsequently money.

The transfer learning gives us the possibility to cooperate between teams. Award winning architectures have proved their value and performance. Usually it takes years for a team of scientists and engineers to come up with such networks, then why not use them as some base initialization for our models and keep on cooperating on this field as much. This will cut down costs, it will tighten those in the community who works in research and those who works on applications. It’s a great form of making this field more productive in terms of good performance model deployments for real world applications!

Reflection

I have seen that this project is a good way to initialize yourself in this chosen stack: Python, Pytorch, SageMaker (AWS on general), and Deep Learning. During the making of this project I had to go through some problems that it would not have surged if I were only doing an already ready-made question or test. And in the end, I had to solve all those side problems to be able to focus on the real problem in the table.

The transfer Learning part of it all makes It an even nicer problem to go over and to try to understand. It is something that is entirely experimental and can give models great results such as that of the benchmarking model.

From going through data wrangling, fetching and visualization to creating a model to be trained on a cloud-based service such as SageMaker was a nice experience that sure has aggregated much to my experience as a developer. Even though the resulting model did not achieve prime results, and that it might not have that much options of real life application (*maybe a dog breed identification app*) it does not mean that it cannot be modified and become better. This is the cool thing of working on such way. I may leave this project hanging for some months, but maybe comeback to it and find a better solution for the task at hand. Everything is nicely setup up, all that it takes is to change the training scripts for the liking and most of all the model!

Improvement

Since the model did not reach the same level as the benchmark mode, some improvements can be pointed:

- A good improvement on this project would be the addition of other pretrained models on the model. To try and replicate the steps taken on the benchmark model on PyTorch.
- Preprocess the data a bit more to extract even more features from the images
- Serve this model to an API to consume images from a smartphone app. (*Dog breed identification app*)
- Leave the model to train for more epochs using SGD with small learning rate and back propagation enabled on the already pretrained layers
- Research more techniques and papers on fine grained image classification
- Make the final layer more complex with some normalization to try to compensate over that validation loss increase problem
- Use more than one pretrained network on the modeling

This problem leaves a lot of margin for improvements and for testing. It can be a good source of future information on certain implementation as well. Surely there is a whole world of possibilities waiting to be tried here!

Referencies

[1] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao and Li Fei-Fei. **Novel dataset for Fine-Grained Image Categorization.** *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. **Deep Residual Learning for Image Recognition.** arXiv:1512.03385

[3] O'Shea, Keiron & Nash, Ryan. (2015). **An Introduction to Convolutional Neural Networks**. ArXiv arXiv:1511.08458

[4] Metrics module. https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

[5] SageMaker Pytorch examples. https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/pytorch_cnn_cifar10