Machine Learning Engineer Nanodegree

Capstone Project

SparkiO
January 7th, 2020

I. Definition

Project Overview

Image Classification task has a very long history within Artificial Intelligence area. The main challenge that aimed to create a competition in which researchers from all around the world would test their algorithms on one extremely large dataset in order to push advancement of Computer Vision is called ImageNet Classification Challenge. Thousands of researchers joined the competition since launching – it shows how crucial for our future society is to teach computers how to recognise and classify things. This project aimed to use several different algorithms in order to make the machine learn to decide the dogs' breeds and tell given a human image – what dog bread it resembles the most.

Problem Statement

Deep Learning, which is a subset of Machine Learning focusing on Artificial Neural Networks has been very successfully applied to various fields like Speech Recognition, Bioinformatics and, the most importantly, Computer Vision. In this project, several Deep Learning models will be used to solve the following task: *given an input image create an algorithm which decides if a dog is detected in the image – if yes, provide its breed, or if a human is detected in the image – if it is a human, the algorithm should output a label of the dog breed that the human is most resembling*.

To achieve the goal, several steps must be made – first we need to download and load relevant datasets with images of dogs and humans (both are provided by Udacity: dogs dataset consists of 3 sets: training, testing and validation with 133 classes – breeds of dogs – in total; the human dataset consists of images of 5750 different humans where each human has 1-8 representing images). Then the model must be taught using appropriately pre-processed data to detect human faces on the picture, and then dogs. The project takes two approaches to get the final breed

prediction – one by implementing Convolutional Neural Network totally from scratch, while the second is by "Transfer Learning" where we get pre-trained model (which was pre-trained usually on ImageNet dataset from the first paragraph) and we change the pre-trained model to suit our task. Finally, the output must be shown in a neat, easy to understand way. Example of the program's output can be seen below.

Figure 1 Example output of the program

Metrics

Evaluation metrics are rather straightforward task in Convolutional Neural Networks. The dogs dataset, as mentioned before, has three subsets: train, test and validation. While the training is done on the train set and the validation set is used to help with hyperparameters tuning, the test set can be used to verify the performance of the constructed models in an objective and clear manner. During the training, we can get information about the loss function of the training set and validation set after each epoch using Categorical-Cross-Entropy which calculates the difference between two (or more) probability distributions for multi-label classification, and after the training we can see the accuracy of the model by using the model on the test set images. The loss combined with the accuracy are perfect metrics to quickly see how good, or bad, the developed model is.

II. Analysis

Data Exploration

In total there are 4 folders with data images needed for the project:

1. Main datasets:
   a) dog dataset ("dogImages" folder, weight 1.08GB) which contains 133 classes of different dog breeds. The classes have different number of samples ranging from ~20 to ~60 images per class in training dataset. There are 3 sets provided:
   - training: 6 680 images
   - test: 836 images
   - validation: 835 images
   b) human dataset ("lfw" folder, weight 0.2GB) which contains images of 5750 humans, each human has 1-8 representing image. The set is not split to train/test/validation sets.

   c) Two very small datasets – "images" and "step6images" folders which are needed for Step 5 and Step 6 of the project – showing the working algorithm on new example images not belonging to the main datasets used during creation of models.

The data size is not the same for all images – the input needs to be standardised (more in section 'Data Preprocessing').

Exploratory Visualization

Firstly, after loading the datasets we can count the number of images in each dataset to see how big they are and on what scale we will be working on:

```
There are 13233 total human images.
There are 8351 total dog images.
```

Figure 2 Output of the function to count images in each dataset.

After that, I used matplotlib and OpenCV to quickly lock at the data images. I visualised 5 images from both humans and dog set to check example data, part of the output is shown on the next page. We can clearly see from the X and Y axes which represents the size of the images that they are not the same size which may be

troubling later: i.e. the first image is 256x256, second image is 320x320 and the while image is again 256x256. We can also clearly determine that the data has 3 dimensions: RGB, none of the images are black and white only. This will be important for training the model appropriately.

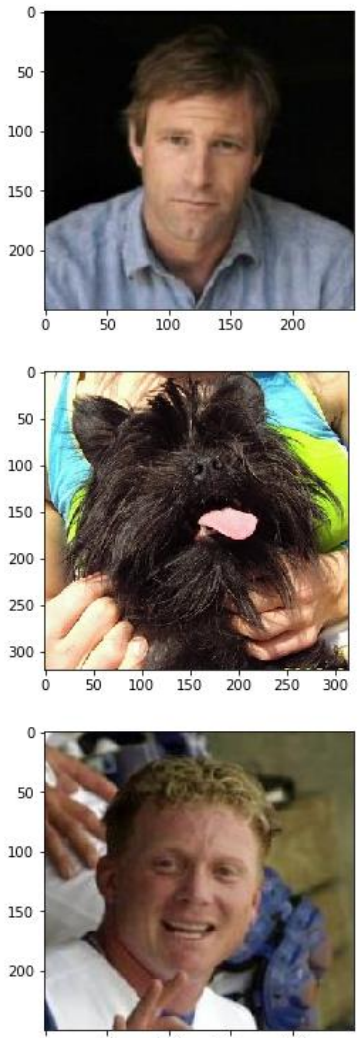


Figure 3 Example data visualisation.

Algorithms and Techniques

The main algorithms to accomplish the task will be Convolutional Neural Networks. They belong to Deep Learning area of Machine Learning. CNNs are known for high-accuracy and relatively quick training networks to accomplish Computer Vision tasks like the Image Classification. Since the project requires to write the algorithm from scratch and using transfer learning, I decided to implement VGG11 as my scratch-CNN, while for transfer learning I used ResNet50 (more about it in "Implementation" section). Convolutional Neural Networks take as an input an image (usually 224x224 in dimensions) and use appropriate layers to process the image to get features from them (in Feature Extraction part) and then classify the image to appropriate label using the extracted features (in Classification part), for example:

a) Convolutional Layers which apply Convolution filter matrix to appropriately change the image so we can extract things we need (like edges, pattern etc.)

b) Pooling Layers to reduce special dimension of the input

c) Fully Connected Layers to classify the image given the features from Feature Extraction part

d) Drop-Out Layers to reduce overfitting by randomly dropping neuron connections in Fully Connected Layers

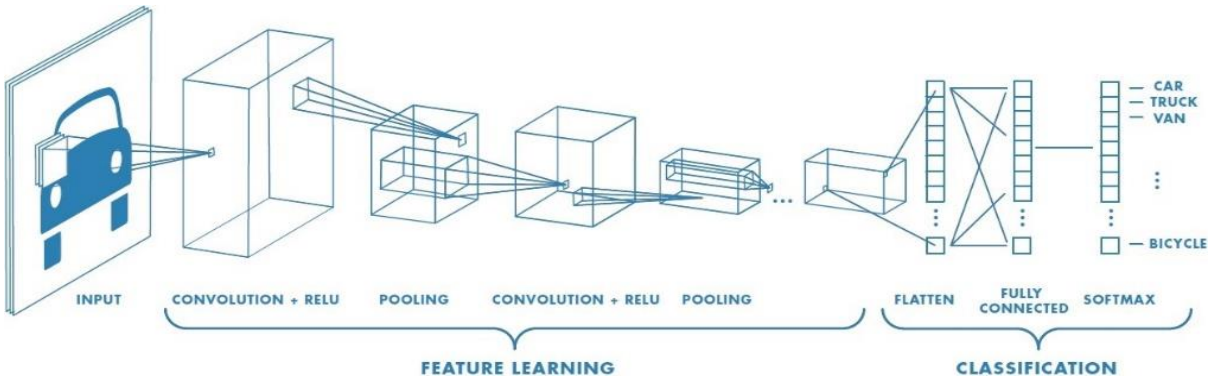


Figure 4 an example of a typical CNN architecture. Source: medium.com

Benchmark

State-of-the-art models for Image Classification like GoogLeNet, ResNet or NASNet achieve 95%+ of accuracy in the classification tasks. However, they were trained on a huge ImageNet dataset for thousands of GPU hours. In this project, the dataset is much smaller and of much lower quality, so the benchmark model accuracy that should be possible to achieve is around 60-70% by transfer learning and 20% by creating the model from scratch. The tasks given says that at minimum the model must achieve 10%+ from scratch and 60%+ from transfer learning which seems reasonable.

III. Methodology

Data Preprocessing

The huge part of the project was data pre-processing part. Convolutional Neural Networks are very strict when it comes to the input they want. All the images must be of the same size (in this case of 224x224 because it is the standard size used by all major models like VGG, AlexNet or GoogLeNet), the data must be provided in form of a tensor and other pre-processing has to be done to reduce typical Machine Learning problems like Overfitting in which the model represents the training samples instead of general population which reduced to lower accuracy on testing set. The transforms I performed can be shown on the code below:

```
transform = {'train': transforms.Compose([
    transforms.RandomResizedCrop(224), #random crop for train data
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
5])]),
           'val': transforms.Compose([
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
5])]),
           'test': transforms.Compose([
    transforms.Resize(size=(224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
5])]),
          }
```

Figure 5 Transforms of the data.

For training data I crop the 224x224 size from random place on the image to reduce overfitting by introducing randomness, I also for the same reason add Horizontal flip for some samples, I transform the data into a Tensor and finally Normalise it using by giving mean and std parameters which then

are used to do the following for each channel of the image: image = (image – mean) std. Some normalization schema usually helps in increase the accuracy of CNN models. For validation and test sets I do similarly but I do not give them any randomness: validation images are just cropped at the centre and test images are just resized to 224x224. I've written a function to visualise the images after preprocessing done:

Figure 6 Images after data preprocessing step.

Implementation

The main algorithms developed during the project were creating Convolutional Neural Network from Scratch and using Transfer Learning for changing pre-trained model to suit our task.

The CNN from Scratch is inspired by VGG11. I wanted to see how it performs compared to the pre-trained VGG16. The main difference between those two is the number of layers - 11 in VGG11, 16 in VGG16. I tried implementation of other famous networks (i.e. AlexNet and ZFNet) but they struggled to get wanted accuracy (10%+) in the test task.

The layers used in the model are:
a) Convolutional Layers – for feature extraction, they take as an arguments: input and output sizes, size of the convolutional filter applied to them (usually 3x3), stride (how big of a step does the filter make), padding (how the filter should behave if some part of the image does not fit properly to it).
b) Batch Normalisations - the Batch Normalisation was not present in the original VGG implementation (it came later with ResNet in 2015), however, I added it since it is a good way to standardise the input and possibly improve the accuracy of the model. It reduces the amount by what the hidden unit values shift around.
c) MaxPooling to down-sample an input representation, it takes the highest value from the area- in the model the Max-Pool is size of 2x2.
d) Fully Connected Layers – each neuron of one layer is connected to each neuron from the next layer – used in classification part of the architecture.
e) Flatten – they take multiple dimensional input (for example matrix) and create one dimensional output (a vector) to be processed by Fully Connected Layer.

f) Dropout – they drop random connections of neurons with given probability (in this case 0.3) to reduce overfitting.
The activation function used is ReLU – standard activation function used in CNNs, its output is max(0, n) – so if the input value is negative, the output will be 0.

The architecture of the CNN is as following:

a) Convolutional Blocks (Feature Extraction part):

There are 5 blocks, each of them has similar strategy: first there is a Convolution2d, followed by Batch Normalisation then ReLU activation function is applied, and at the end MaxPooling to reduce spacial dimension is added.

1. Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
2. Batch Normalisation layer 64
3. ReLU
4. MaxPool2d(2,2)
5. Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
6. Batch Normalisation layer 128
7. ReLU
8. MaxPool2d(2,2)
9. Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
10. Batch Normalisation layer 256
11. ReLU
12. MaxPool2d(2,2)
13. Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
14. Batch Normalisation layer 256
15. ReLU
16. MaxPool2d(2,2)
17. Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
18. Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
19. Batch Normalisation layer 512
20. ReLU
21. MaxPool2d(2,2)
22. Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
23. Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
24. Batch Normalisation layer 512
25. ReLU
26. MaxPool2d(2,2)

b) Classification Part:

1. Flatten
2. Fully Connected Layer (25088, 4096)
3. Dropout(0.3)
4. Fully Connected Layer (4096, 4096)
5. Dropout(0.3)
6. Fully Connected Layer (4096, 133)

For the second algorithm that uses Transfer Learning – I chose to use pre-trained ResNet50. ResNet is a good model developed by Microsoft Research that won 2015 ImageNet Classification Challenge. It is very well suited for the task of classifying dogs' breeds. ResNet was revolutionary in several things - the team behind it aimed to show how deep CNNs can go (before ResNet CNNs couldn't go very deep because of vanishing gradient problem where the gradient of learning function came close to 0 in deep models, so they started to losing accuracy after certain depth).

ResNet used identity shortcut connection blocks to skip layers in order to nivelate vanishing gradient problem. I loaded the pretrained model using torch.models module, then I frooze part of the model using "requires_grad = False" and then I changed the final classification Fully-Connected Layer to output 133 classes.

I should also mention here the train function and how it works: for each batch in the data we initialise the weight of optimiser to zero, then we make step forward in the network, we calculate the loss of the model, update the previous neurons using backpropagation, we make a gradient step and finally we update loss. The aim of the training is to find a global minimum of a loss function. The loss function tells us how bad or good the model learns to recognise the images and its labels. About other things connected with the training like optimiser, cross entropy and learning rate I talk in the next section.

Refinement

There were many refinements made during the work on the project. Starting from the CNN from Scratch:
Firstly, I tried to implement AlexNet – one of the first Convolutional Neural Networks presented in 2012 – but it turned out to be not enough to pass the 10% accuracy score required. Then, I tried to transform it into ZFNet which is basically AlexNet but with smaller filters (the researches behind ZFNet argues that the bigger filters lead to loss of information due to too big pixel-size convolution matrix and skipping important small patterns). It was still not enough. Then I totally started from the beginning and implemented VGG11 with some modifications. I added Batch Normalisation and Drop-Out layers to increase the model accuracy and reduce overfitting which was enough to pass the required threshold.

For the training: obvious loss function to pick was Cross Entropy which measures the relative entropy between two probability distributions. It is standard pick for a loss function in classification tasks. For the optimiser I used firstly Adam optimiser which was not performing very well, and the accuracy increased when I changed to SGD – Stochastic Gradient Descent, it is an iterative method for optimizing an objective function. The learning rate (which tells us how big of a “step” the optimiser should make) at the beginning I picked 0.05. It was too big, the model couldn’t converge appropriately, after lowering it to 0.0005 the accuracy steadily increased.

For Transfer Learning I tried first with GoogLeNet and then out-of-curiosity checked ResNet50, it turned out that ResNet, given the same number of epochs, performed better. All the training parameters were the same as for CNN from Scratch algorithm.

IV. Results

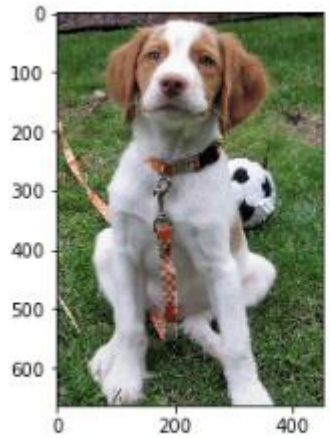
Model Evaluation and Validation

a) Model Evaluation and Validation of CNN from Scratch:
after tuning the hyperparameters and the architecture of the model, the results were good enough to pass the required threshold of the testing task (which was 10%) – my model achieved 16%. When we look at the training which consist of 25 epochs, we can see that the training loss and validation loss are steadily decreasing overall – the model started at 4.58677 of Training Loss and 4.617118 of Validation Loss and ended on 3.028922 of Training Loss and 3.883439 of Validation Loss.

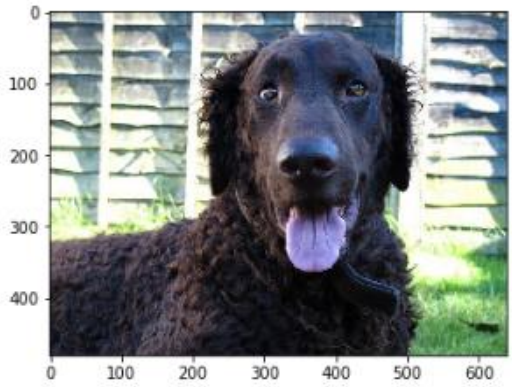
```
Epoch: 1        Training Loss: 4.586777         Validation Loss: 4.617118
Validation loss decreased (inf --> 4.6171).  Saving better model.
Epoch: 2        Training Loss: 4.454539         Validation Loss: 4.490163
Validation loss decreased (4.6171 --> 4.4902).  Saving better model.
Epoch: 3        Training Loss: 4.345674         Validation Loss: 4.398857
Validation loss decreased (4.4902 --> 4.3989).  Saving better model.
Epoch: 4        Training Loss: 4.240252         Validation Loss: 4.329185
Validation loss decreased (4.3989 --> 4.3292).  Saving better model.
Epoch: 5        Training Loss: 4.141191         Validation Loss: 4.321353
Validation loss decreased (4.3292 --> 4.3214).  Saving better model.
Epoch: 6        Training Loss: 4.054660         Validation Loss: 4.409719
Epoch: 7        Training Loss: 3.987124         Validation Loss: 4.312366
Validation loss decreased (4.3214 --> 4.3124).  Saving better model.
Epoch: 8        Training Loss: 3.899882         Validation Loss: 4.191396
Validation loss decreased (4.3124 --> 4.1914).  Saving better model.
Epoch: 9        Training Loss: 3.831546         Validation Loss: 4.337115
Epoch: 10       Training Loss: 3.757028         Validation Loss: 4.095991
Validation loss decreased (4.1914 --> 4.0960).  Saving better model.
Epoch: 11       Training Loss: 3.721058         Validation Loss: 4.331937
Epoch: 12       Training Loss: 3.646838         Validation Loss: 4.084986
Validation loss decreased (4.0960 --> 4.0850).  Saving better model.
Epoch: 13       Training Loss: 3.589387         Validation Loss: 4.117603
Epoch: 14       Training Loss: 3.520276         Validation Loss: 4.103459
Epoch: 15       Training Loss: 3.480394         Validation Loss: 4.731010
Epoch: 16       Training Loss: 3.408424         Validation Loss: 7.304500
Epoch: 17       Training Loss: 3.360398         Validation Loss: 3.805287
Validation loss decreased (4.0850 --> 3.8053).  Saving better model.
Epoch: 18       Training Loss: 3.310644         Validation Loss: 4.114785
Epoch: 19       Training Loss: 3.272128         Validation Loss: 3.948756
Epoch: 20       Training Loss: 3.205258         Validation Loss: 4.076798
Epoch: 21       Training Loss: 3.175328         Validation Loss: 4.475209
Epoch: 22       Training Loss: 3.142456         Validation Loss: 5.399040
Epoch: 23       Training Loss: 3.123379         Validation Loss: 3.531993
Validation loss decreased (3.8053 --> 3.5320).  Saving better model.
Epoch: 24       Training Loss: 3.051080         Validation Loss: 4.213202
Epoch: 25       Training Loss: 3.028922         Validation Loss: 3.883439
```

b) Model Evaluation and Validation of CNN from Transfer Learning:
using the same training function as for CNN from Scratch and giving 20 Epochs of training the final loss was 3.039938 for Training and 2.781188 for Validation. The test task was completed successfully with the result of 65%

accuracy, where the threshold wanted was 60%. Example output of the model can be seen below – the label (breed) predicted is good:




```
It is a dog! It looks like a Brittany
```




```
It is a dog! It looks like a Curly-coated retriever
```

Justification

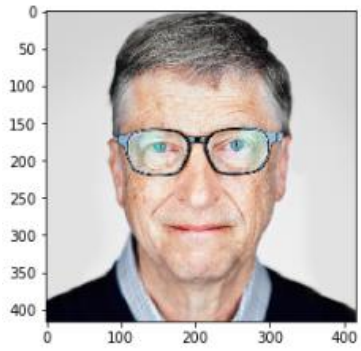
The final solution accuracy is higher than the benchmark threshold set by the creator of this project which means the models are good enough to pass the challenge. The output of the programs is exactly as I thought it will be when I started the project – I knew it is not possible to achieve very high accuracy in a such small, low-quality dataset while not having any access to powerful multi-GPU computers. The final solution is presented by me in this report and in the notebook very clearly and thoroughly. The problem of the project: classify the dog breeds, recognise humans and find the most ensembling dog breed was clearly met by the program I've created.

V. Conclusion

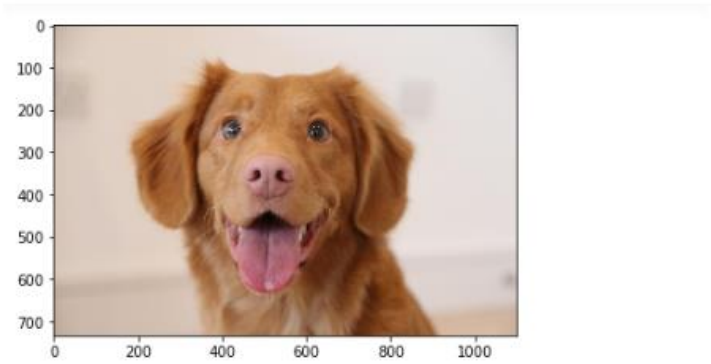
Free-Form Visualization

The problem to solve in this project was: *given an input image create an algorithm which decides if a dog is detected in the image – if yes, provide its breed, or if a human is detected in the image – if it is a human, the algorithm should output a label of the dog breed that the human is most resembling.*

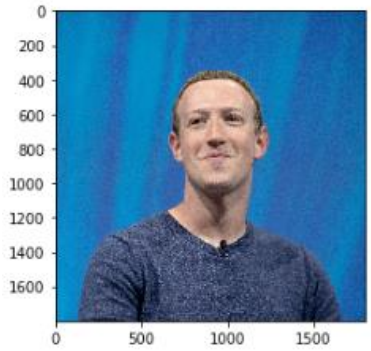
If we consider the task given, we can see that clearly the output that my project produces is exactly the answer we wanted to find. The output is easy to understand, looks good and has the functionality wanted:



Hello, Human!
You look like a... Havanese



It is a dog! It looks like a Nova scotia duck tolling retriever



Hello, Human!
You look like a... Italian greyhound

It is a dog! It looks like a Bulldog

Reflection

I believe that I presented the main algorithms of the project very thoroughly in this report. The models could be better which I talk about in the improvement section, but due to the computational power needed for Computer Vision task – it is very difficult to implement anything that scores really high accuracy. We have to remember that famous models like GoogLeNet or ResNet were constructed by entire

teams of world-level researchers with having almost unlimited computational power (i.e. ResNet was trained on 300GPUs simultaneously). For the scope of the project, I believe my implementation was good, and most importantly I learnt a lot of new stuff and improved my ability to work in Deep Learning libraries like Pytorch which will be very beneficial for the future.

Improvement

Possible Improvements to the models and the application which could lead to improvement to the accuracy are:

Better dataset with larger number of high-quality images should greatly increase the accuracy. Some classes of the dog have only 4-5 images, it is not enough to train properly recognising the class.

Longer training - due to computational power needed only 20 epochs could be done for transfer learning model, as we see with more epochs the accuracy increases and the training hasn't started to converge yet and it hasn't reached the possible global minimum of loss function.

There are ways like 'mixture of experts' or 'decision fusion' where we can take predictions from 2 or more models and weight them appropriately and then create one final prediction - though it is very expensive to do so in terms of computing power compared to just 1 model.