Multi-class Image Classifier using different Convolutional Neural Networks

Lorenzo Clementi
Sapienza University of Rome

Abstract

This project is about achieving a first milestone towards one of the toughest problem in Computer Vision, Object Detection. I have done some research when I first wanted to make that kind of Neural Network and it turned out that it is way more complex than I first thought. For this reason, and after having understood the complexity of the task, I decided to create an Image Classifier from scratch (using PyTorch) that will be used in the future to classify the bounding boxes generated by my future model.

Keywords: Deep Learning, CNNs, Image Classification

1. Introduction

As I said earlier, I wanted to make an Object Classifier but as soon as I started searching for data, architectures and other stuff... I had to stop because of the lack of data with bounding boxes, it was also requiring a very large amount of computational power and the accuracy was so low that for now it was not worth it. Therefore, I am now going to show how I designed this Image Classifier that achieves an accuracy between 75-80 percent on the Test Set.

2. Dataset

For the Dataset, I have used a Dataset containing around 12000 images of Sea Animals spread across 19 categories. Here are some samples of the images I have been working with:

I then preprocessed the images to make them of the same shape ready to be used by the DataLoader, reshaping the images to 150x150 pixels (quite small but not too small) and applying a data augmentation technique by random flipping some of the original images since the dataset was a bit small. I have also normalized the data so that the convergence of the model would have been faster.

3. Architectures

For this purpose I have decided to use a simple Convolutional Neural Network. In the following subsections, different models and their performances will be shown.

3.1 First Model

In order to achieve this amount of accuracy while not requiring the net to get too big (and so too expensive to train), I first tried to use the small Convolutional Neural Network that we have seen in Class with the Professor (in that case was still Image Classification with MNIST, just one channel as input). Hence, I started using that model, making some adjustments in the convolution phases and in the dense layer to make it fit the data/task. But as soon as I started training it, I had to make some adjustments because it was not learning the task. In that case the architecture had 3 convolutional layers (relatively small shapes) and 2 fully connected layers in the dense layer.

3.2 Second Model

The latter model was not performing very well in the test set, probably it was underfitting a bit, and in general the architecture we have seen in class was meant to be use with a slightly different kind of data. I have now added some convolutional layers in the first phase and added also quite a lot more of neurons in the classifier (dense layer) to make it more fine grained (since this time we have 19 classes as opposed to the 10 of MNIST). I have also used the Dropout techinque to reduce overfitting since the number of neurons was getting too big at times. After a relatively short amount of time using the Kaggle GPU, the model was able to achieve 78 percent of accuracy in the test set after just around 20 epochs. This is due to the (small) size of the Network, compared to the model we are going to analyze in the next subsection.

The net is composed by Four convolutions (more on this on the code) and two hidden layers for the dense layer. I have also made use of the maxpooling technique since the dimensionality of our data was getting way too smaller for my hardware. As the activation function I have used ReLU all the time, and for the optimizer the Adam optimizer. I have slightly rearranged the learning rate every 10 epochs. Starting from a learning rate of 0.003 down to 0.0001. The Loss function being used is, as one could have expected for this task, the Cross Entropy Loss.

3.3 ResNet50

Once I got my model, I wanted to compare it and gain some insight by re-training a pre-trained model, or maybe one of the best performing models when it comes to image classification, ResNet50. Hence, I downloaded the weights and the architecture of the feature extractor from PyTorch and then added a dense layer at the end as the classifier. The result is that the model performs quite well since the beginning of the training but it is quite slow in improving, this is because of the size of the net itself and the computational power that I had at my disposal.

3.4 VGG16

I then tried to get inspired by another CNN, the VGG16. This time, as soon as I started instantiating the model, I could not get any training since my device simply was not enough. It required tons of memory, also because it takes 224x224 input images, so for the time required to re-train it it would have required too many resources (time and power). But in the end it has been inspiring to look at the layers of the architecture and get some insights.

4. Conclusions

After this project, I can say that the Computer Vision problem and especially the Image Classification (not to mention the Object detection one) is a very hard problem. Hence, I tried to build up my own model for achieving that task and I compared the results with more expensive models that we can find out there. Turns out that to achieve high accuracy, you are required to have high computational power and quite a lot of data. For this image classification task I have been dealing with roughly 500 images per class, which is not that much (without using any transfer learning technique). One last thing that could be said about the results is that despite the accuracy it achieved, it still does not perform well in classifying between whales, dolphins and penguins. But I believe it is just a matter of training and sharpening the model a lot more.

5. Future Work and improvement

Realizing this project I finally and unfortunately got to know how hard is to find good-quality data and a model that performs really well in every condition. It was very fun doing it, now I think I understand when they say that making someone to learn it's difficult! For what concerns the future work and improvement, my goal is to create an Object Detector from scratch and to refine this model so that it can be easily be trained without spending a lot in training while increasing the accuracy.

Acknowledgments

I would like to thank for the support for this project from the Kaggle community, both for the power of the GPU and for the dataset provided free of charge.