Final project report UF Europe

Michael Baumgärtner
Munich, Germany
mbaumgartner@ufl.edu

Ignacio Gómez
Madrid, Spain
igomezfernandez@ufl.edu

Hugo Chapado
Madrid, Spain
hugochapado@ufl.edu

Gonzalo Prats
Madrid, Spain
g.prats@ufl.edu

***Abstract*—In this paper we show two ways on how to approach an image classification problem. Firstly, what to consider when designing your own image classifier from scratch and secondly what to do when you have to little data to improve the accuracy of the Network to a satisfactory value. The solution to this is to utilize a pretrained Neural Network and to adjust it to your own needs and to retrain it.**

Keywords—Image Classification, Neural Networks, Machine Learning, TensorFlow

I. Introduction

Image recognition is a core task in computer vision, as it enables various downstream applications and has become a standard task on which to benchmark machine learning algorithms. Nowadays, the uses of image recognition are manifold, including for example drones, manufacturing. And Autonomous vehicles [1]

Building a good image classifier is not an easy task, you can find a lot of problems such as "noisy" images or a small data set. In this case it was difficult to build a good model from scratch with comparatively very little data, but there is a way to solve this problem. After discussing what we can do to solve this issue we ended up with the idea of using a pre-trained model, which is basically a network that has been previously trained on a large dataset. With retraining and fine tuning the pre-trained model, in this case the ResNet50, we managed to improve the accuracy in the validation set.

II. Implementation

We started the implementation of the project by obtaining the data from ('data_train.npy') and the labels from ('labels_train.npy'), and after converting the data to the proper format we needed to use TensorFlow. On the data we did some preprocessing, where we divided the data into 80% train and 20% validation. Here we used the in TensorFlow/Keras implemented functions "keras.preprocessing.image_dataset_from_directory" to preprocess the data to the specified batch size and image size and to randomly distributed the images in the test and validation dataset.

As we mentioned before, the development was based on trial and error, so we tried different batch sizes to see which one works better, which is basically the number of samples we processed before we update the model [2], and we ended up choosing the value 32 for both the model built from scratch and the retrained model.

One of our goals on the Neural Network was to generalize it, so we tried using random rotations and zooms, to avoid the problem of overfitting on the Neural Network. Additionally, we also implemented a dropout, so, every iteration the Neural Network selects random neurons that are ignored during training.

For our custom-built Neural Network, we used a sequential model with the ReLU (Rectified Linear Unit) [3] activation function. This activation function has the advantage that it does not activate all the neurons at the same time, which means that the neurons will only be deactivated if the output of the linear transformation is less than 0, what made it the activation function with the best results. The Neural Network has 2 convoluted layers, after each of the layers we first performed a MaxPool which helped to reduce overfitting and spatial size of the convolved features [4], then we did the dropout we mentioned before, and finally we implemented the option to perform a Batch Normalization for stabilizing the learning process and to reduce the number of training epochs required to train deep networks. We also implemented an additional layer that flattens and condenses the Neural Network. Flattening a tensor means to remove all the dimensions except for one, so we can input that 1-dimensional array to the next layer. With the use of flattening, we created a fully connected layer because it is connected to the final classification model.

We know that the choice of optimization algorithm can mean the difference between good and bad results, and after experimenting with different ones, we chose "ADAM" optimizer with a Sparse Categorical Crossentropy for the loss function. The "ADAM" optimization algorithm has a lot of benefits, being computationally efficient with little memory request [5]. Then, SparseCategoricalCrossentropy is very good when doing multi-class classification because it compares the predicted label and the true label and calculates the loss.

For our retrained model we chose to use the ResNet50 as our base model after experimenting with some other models (more in the next section). For the first 10 Epochs the ResNet50 is used as it is without the top layer. We then flattened and condensed it to 1024 neurons. After that we utilized the dropout again and then condensed it, so that the output was a neuron for each class.

As an optimizer we used the RMSprop with a learning rate of 0.0001 and the Sparse Categorical Cross entropy as our loss function. After the 10 epochs of training, we decreased our learning rate by 1/10 to get finer results and to avoid missing the best results.

III. Experiments

A. *First thoughts*

During the development of the project, we had to consider what would be the best approach to our problem. This being the ability to correctly recognize a high percentage of the validation set of pictures. To do that, we would have to choose among different algorithms the one that fit our data the best. The nature and variety of the set of images forced us to use Deep Learning as our approach. Any other non-deep learning or linear models were extremely biased by how the images were ordered or rotated and, hence, the success rate was very low. We also considered how much support there is for deep learning algorithms in python (in terms of libraries and functions), so that we could achieve the desired outputs without having to build the entire models from scratch. Apart from that, we should also distinguish between the two approaches we took. First, as described earlier, we used a framework to build a neural network ourselves but then, in the search for more accurate outputs, we adapted a pretrained convolutional neural network, ResNet50, which offered a higher percentage of success. These two will be discussed in the following sections.

B. *Frameworks discussion*

Before going into the framework discussion, it should also be mentioned that one fundamental part of our experimenting was the scaling and rotation of the images, looking for an unbiased model. We noticed that the images were not always representative of the class they belonged to so we needed the extracted features to be as general as possible so that outliers would not affect the result. The model selected would need the dataset to be as generalized as possible, and in some of the samples, their class was not as clear even for us to decide. By rotating and scaling the images randomly we were able to fix that problem.

For example, if the image was a stadium but the actual building was on the right side of the image (being the left just grass or other buildings), then the NN (neural network) would not be able to decide upon just one class (it could just randomly select between the "Building" or "Stadium" classes). However, thanks to this technique, the algorithm was able to learn what a stadium is in a more abstract way, getting rid of biases and outliers.

After looking into possible models for our neural network, our two final proposals were PyTorch and Tensorflow. Both these frameworks, however, cannot directly process images and start working on the model, for that we would have to make the most out of preprocessing. To do that we first had to investigate tensors and their properties, as these are the basic structure to store the data we want to work with. For the reader to follow with our reasoning, a tensor is a data structure used to encode the data we want to process or use [6]. Their structure is like arrays and matrices, but it could be more accurately defined as a multidimensional array, which is what we were looking for in terms of features. They are considered a generalization of most of the other data structures, but they have an interesting property which is: when a tensor modifies all the others in a certain structure, we can keep track of those changes the same way we would modify a vector and then save the changes in a separate matrix for later use. This whole structure is what is called a tensor.

Now we can move on to our first choice, PyTorch. It is a very fast and intuitive framework that is also supported by Python, which makes it the most suitable for experimenting. Pytorch offers a complete machine learning package which includes deep learning primitives such as a neural network layer or node structures, activation and loss functions and optimization algorithms to help the user build a deep learning model with basic python programming. Another advantage for this framework is that the graphs are dynamically built along with coding, which again, makes it faster to draw conclusions. For all these reasons, Pytorch was one of the top contenders to use in our final model [7].

While trying to build our model with this framework, we ran into two problems. First, the dimensionality of the problem was something to consider and, working with several layers of neural networks makes features and their weights more confusing than usual. To maintain coherence between the layers of the NN we should have been able to keep track of how the dimension of each layer would change. Second, to properly manage the whole dataset, we would need an iterator or at least a data structure that worked as such and traverse the whole list of items.

For that, Pytorch offers DataLoader, which turned out to be not as easy to manage as we would have expected. Then, after several failed attempts of compilation we decided to abandon this path and try with another framework to see if we could manage it better.

TensorFlow was the other reasonable alternative. Compared to Pytorch, this framework is a lower level of programming which has its advantages and disadvantages [8]. On the one hand, it provides a very flexible toolset for the development of models as lower-level programming languages tend to do. Because of that, the spectrum of experimentation widens and allows the user to get more detail on how to tune the parameters and the possible outcomes that they might have. Another advantage was it supports Keras, a deep learning API that helps with the implementation of all the NN functions in a very simple way. On the other hand, learning how to use this model gets more complicated than Pytorch as there are more variables and expressions to learn. This is directly connected to debugging, which was a key stage of our project. This framework is also known for being computationally more expensive than Pytorch but thanks to UF's HiPerGator, time management was not really a problem.

To conclude this section, some of the reasons we chose TensorFlow over Pytorch were the following.

- TensorFlow is a much more flexible framework and offers more direct manipulation of the code
- Although Pytorch offers dynamic graphs (with TensorFlow we are not able to modify graphs dynamically), that was not really a necessity as we did not have to deal with variable length features/inputs
- The learning curve is much faster with TensorFlow with the right parameters. So, if we learned how to use it, it became an advantage
- Following the last point, TensorFlow, was developed and supported by Google, which means that it has a big community behind that can solve doubts almost instantly. Pytorch, on the other hand, is newer and we noticed that when looking for answers to our problems online
- For the same reason, TensorFlow has many resources as to how to use its features, which eased our way in a major way

C. *Implementation discussion*

Then for the implementation itself, we had to choose how to design the neural network. In it, through trial and error we concluded that the best number of convoluted layers was two. When using less layers, we observed that the outputs were rather random and was highly biased, so it was not a reliable model if we wanted to extrapolate the results. This translates to the model recognizing different objects as the same such as buildings and stadiums or flowers and forests. Then, when using more layers (more neurons), we ended up with an overfitted model that was very sensitive to the outliers and small variations in the pictures we talked about earlier. Again, the accuracy of the model was quite low as the outcomes of the models varied.

Finally, for the batch sizes we tried different ones and ended up using 32. Batch sizes smaller than 20 leads to the inability to train the data properly. Batch sizes larger than 32 lead to a tendence of overfitting. The effect on having less or more samples per training cycle (batch size) was like the one explained earlier with the different number of layers, with the bias-variance trade-off being the main issue [9].

We also tried to utilize Batch Normalization which gave us not the not results we wanted as it increased the variance of the accuracy and the loss to sometimes over 40%.

D. *ResNet50*

The implementation just explained gave us an accuracy of 75-80%, which was far from optimal. That made us think about what other algorithms or approaches we could investigate. Pre-trained models were the best option. These are models that were developed by a third party on a much larger scale (in terms of the dataset and the size of the neural network) that solve a similar problem to the one we are working on. The idea behind our work was to adapt the model that already existed to the parameters and specific guidelines of the project. This technique is called transfer learning and is widely used in Deep Learning [10] and more specifically, in natural language or tasks processing. With it, we were able to benefit from the enormous datasets that exist nowadays and the algorithms that the community was able to train thanks to them.

One of the most famous pre-trained models in the world of image classification is ResNet50. This is a convolutional neural network made of 50 layers and is part of a larger group of NN called residual neural networks (its name comes from there).

The main difference with the rest of the models is that this one stacks the residuals of learning as it goes through the layers as seen in "Fig. 1", making it more accurate. Going back to the NN in question, ResNet50 was created by Kaiming He and three other computer scientists to participate in the 2015 ImageNet competition. The model won the contest, and they were able to show the world that deeper NN could be trained with accuracy as well. Their work showed a 3.57% error, which is rather small considering that the dataset offered by ImageNet has over 20,000 classes with more than 14 million images in total. Their paper in network architecture was very important for the development of these kind of deep learning algorithms [11].

The experimenting phase here was not as intense as when we built the model ourselves as most of the parameters are set for us. We had to change some of them for the model to work properly. Batch size and random rotations and zooms values stayed the same, we did not feel the need to change them for better results. However, the image size was changed from 300x300 to 224x224 (the algorithm required that size) and we changed the dropout factor from 0.3 to 0.24. To achieve generalization, the dropout factor should go as high as possible [12], but we found that for higher values, the results were not as accurate. As a result of this we had to perform a generalization-accuracy tradeoff.

Finally, the number of epochs was significantly reduced as this new algorithm learned much faster (pre-trained) and we would need less iterations to achieve an optimal result. For the epochs, as it can be seen in the figures displayed in Figure 3, even if we increase the number of them, we will not get much better results than that (or even worse), since the learning curve flattens and there is no room for improvement there. After some tries, we determined that the optimal number of epochs for this algorithm was between 25 and 40.

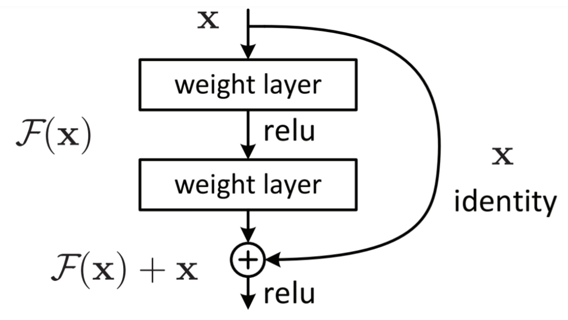
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1 - Residual Learning

E. *Experimenting conclusions*

Neural Networks are a very complicated structure so trying to understand how they work was the most difficult part. Luckily, nowadays we can find frameworks such as TensorFlow or Pytorch and pre-trained algorithms that allow us to test different parameters, layers, and models with very

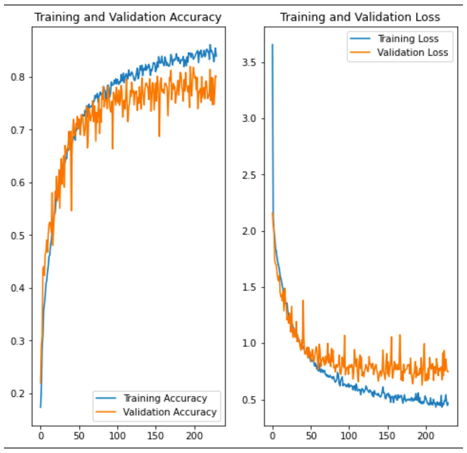

Figure 2 – Custom Neural Network

little programming. That was the philosophy we followed, to test a set of parameters and, when observing unwanted results (low accuracy), we would analyze the output, figure out why it behaved like that and change the parameters accordingly.

IV. Conclusions

Through the completion of this project, we were able to understand how difficult it is to create a fully functional neural network. The complexity that goes into every aspect of it, from the visible and hidden layers to each neuron, requires an exercise of deep comprehension of the matter. That is why we had to look for information in research papers, lectures, or books. And even then, we would have to experiment with the code to understand how certain changes would affect the outcome.

Our first approach, however, was the most time consuming. We tried to create the model from scratch, and, for that, we would need to tune the parameters one by one, looking at the outcome each time and figuring out why it did or did not work. However, no matter how we changed the parameters or the architecture of the network, we could only achieve a success rate of slightly above 75% (and with a very high number of epochs). Our second attempt was more straightforward as the ResNet50 library had most of its parameters already set. Nevertheless, debugging was still a central part of the development of the function as we had to adapt the NN to the problem given. In this case, the accuracy of the model was much better from the beginning. In fact, with only 10 epochs we were able to achieve the same result as the first model, and with just 40 epochs we achieved an accuracy of over 98.8%, which is a more appropriate value.

To end this section on a more personal note, this project has taught us how to work in a professional environment, learning how to use advanced libraries and elaborating a report with a professional style, worthy of showcasing it in an executive environment. As a group we were able to understand each other and each one of us contributed significantly to the project, making the development much faster.

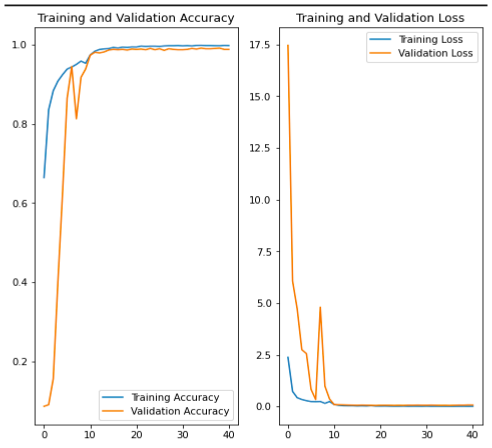

Figure 3 - Retrained ResNet50 Model

References

[1] K. Amer, M. Samy, M. Shaker and M. ElHelw, (2019). *Deep Convolutional Neural Network-Based Autonomous Drone Navigation*, Center for Informatics Science Nile University Giza, Egypt.

[2] Silva, C. (2021). *Lecture 37: Best learning practices & Introduction to Deep Learning.* Personal Collection of C. Silva's, University of Florida, Gainesville FL.

[3] Silva, C. (2021). *Lecture 34: Artificial Neural Network; Multi-Layer Perceptron.* Personal Collection of C. Silva's, University of Florida, Gainesville FL.

[4] Silva, C. (2021). *Lecture 38: Introduction to Deep Learning; Convolutional Neural Networks (CNNs).* Personal Collection of C. Silva's, University of Florida, Gainesville FL

[5] Silva, C. (2021). *Lecture 37: Best learning practices & Introduction to Deep Learning.* Personal Collection of C. Silva's, University of Florida, Gainesville FL

[6] Yuwang Ji, Qiang Wang, Xuan Li, Jie Liu, (2019). *A survey on Tensor techniques and applications in machine learning*. Beijing Natural Science Foundation, Beijing, China

[7] Adam Paszke et al. (2019) PyTorch: *An Imperative Style, High-Performance Deep Learning Library, Various Universities and Research Centers around the world*

[8] Martín Abadi et al. (2016) TensorFlow: *A system for large-scale machine learning. The advanced computing systems association*, Savannah, GA

[9] Silva, C. (2021). *Lecture 9: Avoiding Overfitting & The Bias-Variance Trade-Off.* Personal Collection of C. Silva's, University of Florida, Gainesville FL

[10] Fuzhen Zhuang et al. (2019) *A comprehensive Survey on Transfer Learning*. National Key Research and Development Program of China, China

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, (2015) *Deep Residual Learning for Image Recognition*. Property of Microsoft Corp., Redmond, WA

[12] Silva, C. (2021). *Lecture 37: Best Learning Practices in ANNs.* Personal Collection of C. Silva's, University of Florida, Gainesville FL