CLOTHES RECOGNITION AND DETECTION WITH CONVOLUTIONAL NEURAL NETWORKS

PROJECT REPORT - OCTOBER 26, 2019

Enrico Zerilli
ezerilli@gatech.edu

1 Introduction

Object Detection and Recognition in real-world images is a still a hard task to achieve, but it is especially critical for different applications. As example, faces recognition and cars detection are crucial tasks in the context of security applications and self-driving cars. In this context, modern approaches based on *Deep Learning* have been shown to be greatly beneficial, especially for real-time applications. Indeed, recent *Deep Learning* breakthroughs in images classification problems have produced an astonishing jump in object detection accuracies over different famous benchmarks, while allowing objects to be detected from an image in fractions of second, which is essential for running such algorithms in real-time applications.

2 State of The Art

Convolutional Neural Networks (CNNs) have aroused in the last decade as the new breakthrough in AI with thousands of relevant applications in natural language processing, vision, robotics and many other fields. These networks, which takes inspirations from the human brain even though they are far to be similar, are so powerful and effective in so many different applications to have become nowadays ubiquitous. Although they have been around since 1998 [1], they became popular and also really powerful 10 years after with the advances in the computational power of modern GPUs. Thus, only recently, the field has exploded and researchers have started experimenting in different challenges, such as ImageNet, with all kinds of different architectures.

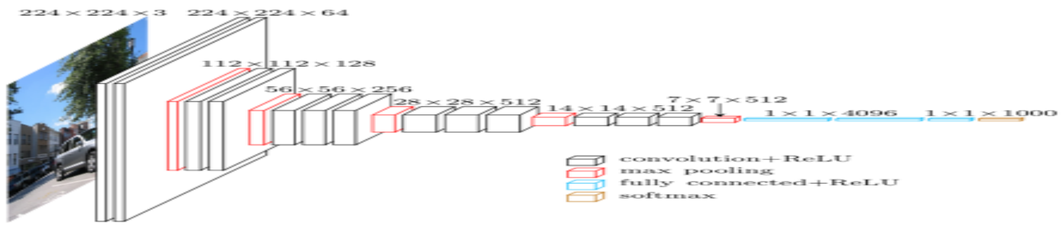

Figure 1: The VGG16 Architecture [2]

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [3] aims to make machine learning and vision researchers confront and advance the state of the art on ImageNet, a dataset containing millions of images of different objects, people and animals associated with 1000 different labels. In these challenges, always new and more advanced architectures have outperformed the state of the art one year after the other. Starting from AlexNet [4] in 2012 ($top\ 5\ error\ rate = 16.4\%$), which has truly represented a huge jump in terms of performances from classical machine learning techniques ($> 25\%$) to Deep Neural Networks, there's been a rush to invent new architectures to outperform the previous ones. In this race, excellent results have been achieved by going deeper and deeper in the network topology, as VGGNet [5] (Figure 1) and GoogleNet [6] have demonstrated in 2014, when for the first time the recognition error on ImageNet dropped below 10% for both architectures (7.3% and 6.7% respectively). Nonetheless, building deeper and deeper networks have induced information about the input and the gradients to vanish by the time its reaches one end of the network. Thus, a simple idea to make deep architectures efficient was that a deeper network should be at least as powerful and efficient as a shallower one of the same topology. In 2015, ResNet [7] solved the problem by introducing direct paths from one layer to the other by adding, for each layer, the input activation map with the results of the nonlinear transformation. These paths allow indeed to create direct ways for the gradients to flow easily upstream during training without vanishing. ResNet, for the first time, has overcome the 5% barrier on ImageNet with a top 5 error rate of 3.57%. From there, deeper and wider architectures, but also different topologies, have been explored all the way through. Two examples surely are Stochastic Depth in 2016 [8], which shortens ResNet by randomly dropping layers during training to allow better gradient flow, and DenseNet in 2017 [9], which build an interesting densely connected architecture which proved to outperform ResNet.

3 MiniVGG

VGG [5] has shown, for the first time, that deeper convolutional architecture do a better job in feature extraction and is considered one of the benchmark architecture in *Deep Learning*. As shown in Figure 1, the distinctive trait of VGG is its deep architecture, composed by a series of Convolutions + ReLu activations and MaxPooling layers. Convolutions are performed with an increasing number of small 3x3 filters that preserve spatial dimension by padding the input layer. ReLu activations are also used after each convolution to introduce nonlinearities, while avoiding the problem of saturation. Then, dimensionality reduction is performed by 2x2 MaxPoolings with stride 1, by dividing the input dimension by 2 at each block. Therefore, we can get a very deep and efficient feature extractor by chaining a certain number of these blocks and then perform classification with 2 Fully Connected (FC) layers followed by a Softmax output layer for outputting each of the classes prediction probabilities. In this project, we are going to use a small version of VGG, that we are going to call for simplicity MiniVGG, including either the first 2 or 4 convolutional layers of VGG's feature extractor, followed by a FC and a Softmax layer.

4 The Fashion-MNIST Dataset

Fashion-MNIST is "a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes" [10]. It is intended to be a drop-in replacement for MNIST for benchmarking *Machine Learning* algorithms. Indeed, MNIST is by now overused and not really representative of the difficulty of modern *Computer Vision* tasks, because too simple to highlight strengths and drawbacks of different ML algorithms. It has even been shown that two digits can be easily discriminated by just one single pixel. On the contrary, Fashion-MNIST, despite being intentionally simpler than a real-world images datasets, is better representative of the difficulties encountered when working with modern image classification tasks.

5 Recognition

For the recognition task, 3 different models have been trained on the dataset with Keras [11] using a GPU Nvidia Tesla P100 on the Google Cloud Platform [12] and compared in terms of performance. First, a MiniVGG4 architecture has been investigated. MiniVGG4 is composed by 2 Conv+ReLu followed by a MaxPooling, a FC and a Softmax layer. The 2 Conv layers use 32 3x3 filters each and preserve input-shape by padding, while the 2x2 MaxPooling with stride 1 is used to reduce input dimensionality from 28x28x32 to 14x14x32. Finally, a FC layer with 512 neurons and ReLu activations and a Softmax output layer with 10 nodes are used to output each of the 10 classes prediction probabilities. Also, Batch Normalization (BN) is used to smooth the objective function and help with training speed and efficiency, while DropOut (DO) is used to help mitigating overfitting. Specifically, a BN layer is inserted right after each of the Conv layers and after the FC layer, while DO is used both after the MaxPooling and the FC layer. As second model, a MiniVGG6 architecture has been investigated. MiniVGG6 is identical to MiniVGG4, with the only difference that another 2xConv+Relu+BN + MaxPooling + DO block is inserted after the first one, this time using 64 filters, thus inducing a deeper feature extractor whose output is a 7x7x64 feature map. Third and last model, a MiniVGG6 identical to the second model, but using Data Augmentation by random horizontal flip flops of the input images, has been investigated. Training was performed with the help of ImageDataGenerator [11] which automatically handles re-scaling in the interval $[0, 1]$, validation set splitting from the training set, data shuffling and augmentation. In our case, the batch size, which is used to approximate the loss function computation and speed up training, has been set to 64 in order to be at the same time small and representative, while a typical 90/10% split has been chosen for training and validation set respectively. Moreover, as Optimizer of the learning process, Adam has been preferred to Stochastic Gradient Descent (SGD) for its robustness and reliability. Indeed, even though SGD+momentum may perform better in some situations, Adam is usually the best starting choice. As loss function, the categorical crossentropy is used for multi-class classification to train the CNN to output a probability function over the 10 classes. As for the learning rate lr, which basically represents the step we take in the direction of the negative gradient during Gradient Descent, $lr = 0.01$ has been proved to be a good choice. The learning rate is critical in CNNs, since a too high lr may cause us to take too big steps and miss the minimum, while a too low lr may slow down too much training. Finally, Model Checkpoint, Reduce LR on Plateau and Early Stopping have been used to monitor the validation loss and respectively save the best model, reduce lr by a tenth after a period of plateau and stop the training procedure in case of an extended stall.

6 Detection and Localization

Work in progress.

7 Results

The results from training for the 3 models are shown in Figure 2 and 3. We can see the MiniVGG4 model performs very well with a validation accuracy over 93%, but presents a little bit of overfitting, which may be mitigated further by increasing regularization through Dropout or Data Augmentation. The MiniVGG4, on the other hand, also presents a little bit of overfitting, but achieves a higher validation accuracy (over 94%) by getting a better feature description out of the feature extractor thanks to its deeper architecture. Finally, the MiniVGG6 with data augmentation slightly improves results from simple MiniVGG6 and has thus little overfitting. From these Figures we can thus see how, by incrementally adding some useful components to our CNN, we can ameliorate our model and achieve a validation accuracy which is very close to the test accuracy of 95%, reported as benchmark for this dataset. However, the validation set is a little bit "optimistic", since we have made use of it to reduce our learning rate and stop training. So, we should expect the test accuracy to be slightly lower. Also, even if apparently MiniVGG4 seems to be less complex than MiniVGG6, MiniVGG4 has 3 millions weights to train, which is twice those of MiniVGG6. Indeed, MiniVGG4 produces a 14x14x32 feature map at the output of the convolutional layers, which means 6272 features to be classified by the following FC and Softmax layers. On the opposite, MiniVGG6, thanks to the supplemental convolutional block followed by Max Pooling, produces a 7x7x64 feature map, which means half of the features are produced and then classified by the FC and softmax layers. However, MiniVGG6 is longer to train because there are more layers to backpropagate the gradients through.

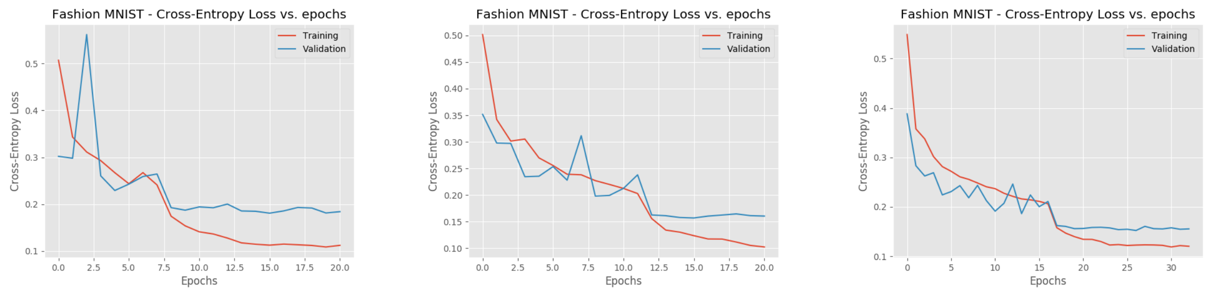

Figure 2: Loss for MiniVGG4 (left), MiniVGG6 (center) and MiniVGG6 with Data Augmentation (right) both for training and validation set

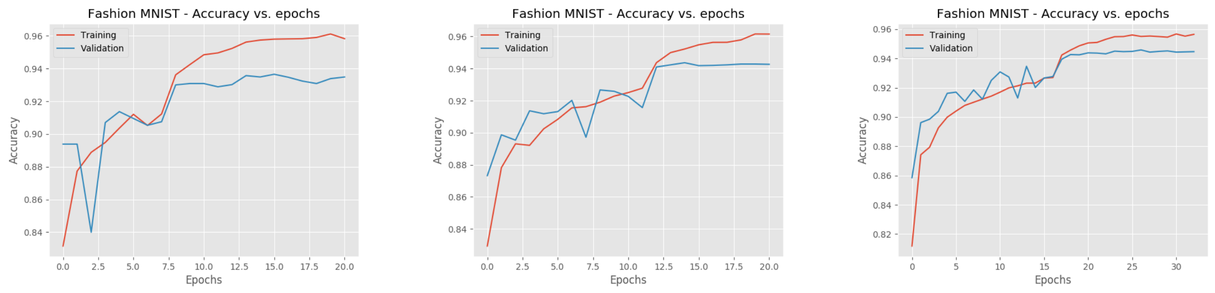

Figure 3: Accuracy for MiniVGG4 (left), MiniVGG6 (center) and MiniVGG6 with Data Augmentation (right) both for training and validation set

Table 1: Performances of the three models over the training, validation and test set

Model	MiniVGG4	MiniVGG6	MiniVGG6 with Data Augmentation
Training Accuracy (%)	95.9	95.2	95.5
Validation Accuracy (%)	93.8	94.2	94.6
Test Accuracy (%)	93.2	93.7	94.2
Avg. Training Time per epoch (s)	28	36	36
Avg. Training Time (min)	8.87	10.2	16.2
Avg. Inference Time (ms)	0.43	0.80	0.80
Trainable Weights (millions)	3.23	1.68	1.68

Performances for the 3 models over the training, validation and test set are reported in Table 1. As shown by the test accuracy, MiniVGG6 with Data Augmentation is the model the performs best, with a test accuracy of 94.2 % which is really close to the benchmark of 95% for Fashion-MNIST. We can see how overfitting is progressively reduced from the MiniVGG4 model by firstly decreasing model complexity and getting better features (MiniVGG6) and secondly introducing Data Augmentation as a form of regularization. However, introducing more layers to the feature extractor in MiniVGG6 increases a little bit the training time per epoch w.r.t. MiniVGG4, from 28s to 36s on average on the Nvidia Tesla P100. This trend is replicated in the average total training time and inference time as well, even though the scale of time are completely different. As a note, the inference time reported has not been measured on the Nvidia Tesla P100, which was used exclusively for training, but on a 3,1 GHz Intel Core i7 CPU. This is a more reasonable choice, since, while we accept GPU to be used to train *Deep Learning* models to speed up training time, inference usually has to be performed on commercial CPUs. Finally, an example of predictions with MiniVGG6 + Data Augmentation, the model that performed best in our experiments, are reported in Figure 4. We can see how the model is almost always right and also confident about its predictions, but it happens to get sometimes a wrong prediction. In particular, the T-shirt and Shirt classes are the only two classes reporting an accuracy below 90%, since these are the two classes that are most easily confused when working with images at such low resolution.

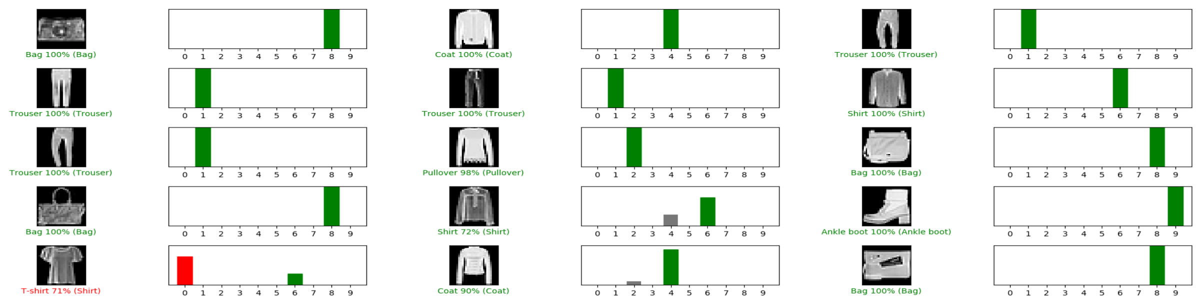

Figure 4: Predictions with MiniVGG6 + Data Augmentation on some random test images

8 Conclusions

In conclusions, we have trained 3 different CNNs and verified how deeper feature extractor and data augmentation help achieving better generalization in the context of Deep Neural Networks. Indeed, looking at the results discussed above, we can see ho the MiniVGG6 with Data Augmentation outperforms the other two in accuracy and space complexity at the cost of an increased, but still very manageable training and prediction time. As further development for the future, we are working on an object detector which takes advantages of the results achieved in this project and beneficially apply the MiniVGG6 with Data Augmentation model to detect and recognize clothing in real-world images and videos.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. IEEE, 1998.

[2] VGG in TensorFlow : model and pre-trained parameters for VGG16 in TensorFlow. `https://www.cs.toronto.edu/~frossard/post/vgg16/`. Last accessed: 2019-10-24.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

[5] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1312.6082v4*, 2014.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[8] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep Networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

[9] Gao Huang, Zhuang Liu, Laurens Van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[10] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[11] Keras: The Python Deep Learning library. `https://keras.io`. Last accessed: 2019-10-24.

[12] Google Cloud Platform for fast Deep Learning development. `https://cloud.google.com/deep-learning-vm/`. Last accessed: 2019-10-24.