MLP Coursework 2: Investigating the Optimization of Convolutional Networks

Federico Arenas López

Abstract

During this study we will identify what causes the vanishing/exploding gradient problem present in deep Convolutional Neural Networks by comparing the learning performance of two VGG architectures trained on the CIFAR-100 datset, one shallow (8 convolutional layers), and one deep (38 convolutional layer). We then move on to explore 3 possible solutions present in the literature. From these solutions we choose to implement Deep Residual Networks on the VGG architecture, and test the solution under multiple hyperparameter scenarios. We finally train the best VGG ResNet architecture on increasing depths, and find that a 68 layered network provides the best performance yielding a test accuracy of 65.83 %.

1. Introduction

Convolutional Neural Networks are one of the most effective Machine Learning methods for visual image recognition. Indeed, multiple CNN architectures have proved to be highly performing when implemented on benchmark datasets (Lecun et al., 1998; Krizhevsky et al., 2017; Simonyan & Zisserman, 2015; Szegedy et al., 2014). In theory, as CNN models get deeper, one would expect the model to continue improving its performance, or to at least overfit to the data. This, however, is far from true. In reality, when a CNN model reaches a certain depth, it's gradient flow becomes too unstable, and produces the problem of vanishing or exploding gradient when its non-linearities saturate, and the CNN stops learning.

The first part of this paper focuses on identifying and understanding how this problem appears on a deep CNN Network with a VGG architecture. The VGG architecture is composed of S_k stages that contain n_{cb} convolutional blocks. Each block is composed of two "same" convolutional operations, each followed by a ReLu activation function (Simonyan & Zisserman, 2015). After each stage, a pooling operation is performed on its output feature maps.

To identify the problem, we compare a VGG Network of 8 layers with a VGG Network of 38 layers. And we train both models on the benchmark dataset CIFAR-100. By looking into the gradient flow on each of the network's layers, we find that while the shallow network presents a healthy gradient flow, and is able to converge, the deep network's gradient flow immediately drops to zero and thus is never able to converge. A clear sign of the vanishing gradient problem.

Consequently, the second part of the paper focuses on studying the multiple alternatives that there is in the literature to address this problem and to successfully build deep CNNs that improve their performance as the model gets deeper. We look into Batch Normalization, Deep Residual Networks, and Densely Connected Networks introduced by (Ioffe & Szegedy, 2015), (He et al., 2016), and (Huang et al., 2017), respectively.

Out of the 3 solutions explored above, we choose to implement Deep Residual Networks since they fit naturally with the VGG Network architecture, as shown in (He et al., 2016). Indeed, implementing this solution only requires us to create a connection between the input of each convolutional block and its output. We also apply Batch Normalization to each convolutional block, as suggested by (He et al., 2016). Therefore, we train a VGG ResNet with 38 layers, and verify that the gradient flow is healthy, and the error function converges accordingly.

This allows us to move on into the last stage of the paper. Here, we hypothesize that (1) a Deep CNN with Residual connections and Batch Normalization will never allow the non-linearities to saturate, and (2) the learning of a CNN with Residual mappings and Batch Normalization will always converge as the Network gets deeper and deeper. We test these hypothesis by running multiple experiments that use a sigmoid non-linearity, different Learning Rates, batch-sizes and finally, more layers.

These experiments allow us to prove both hypothesis and prove that the the batch normalization step robustly stabilizes the gradient flow in the network, allowing its loss function to converge even in the most extreme scenarios. We also prove that Residual mappings highly contribute in improving a deep CNNs accuracy.

In this paper we provide a theoretical and experimental analysis of the vanishing gradient problem in Convolutional Neural Networks, as well as a theoretical and experimental analysis of the multiple solutions to this problem. Our main contribution being a systematical set of experimental results that prove the solution's efficacy under multiple hyperparametrical extremes.

2. Identifying training problems of a deep CNN

In order to successfully design a deep convolutional neural network whose learning continues to improve as it gets deeper, we must first identify what causes a deep convolutional neural network's learning to get stuck. To do this, we decide to train a shallow VGG network of 8 layers, and a deep VGG network of 38 layers both for 100 epochs, with mini-batch size of 100. [1].

These two configurations will allow us to compare the learning behaviors of both networks. Thus, we will be able to inspect and identify what problems arise when training the deep VGG configuration, and what are the causes for these problems. In this line of thought, we plot the training and validation loss shown in *Annex A* and found in Figure 1 of (Practical, 2020).

Consequently, from this initial plotting we identify that while the VGG08 network is able to converge during optimization and bring its training loss down to 1.60 and its validaton loss to 1.97 , both the training and validation loss for the VGG38 network get stuck at 4.61. What's more important, the VGG38 netowrk is *never able to converge*. This means that the network was *never able to learn*, and is reflected in its training and validation accuracy of 0.01. It's optimization got stuck somewhere. To visualize this, we plot how the gradient is behaving as it passes through both networks for all epochs during training in *Figure 1*.

From both plots, the shallow network exhibits a healthy learning process, where the gradients at each epoch are maintained at a stable value above zero as it backpropagates through each layer. Through layer 2 to 7, the gradients are maintained at a level above 0, and bellow 0.025. Before the weight update, most gradients at the input layer oscillate between 0.05 and 0.15.

Moreover, the fact that they start at a value above zero and are maintained above zero is important since a gradient that starts at 0 would make all gradients disappear. This problem can be observed in the gradient flow of the VGG38 network, where the gradients start at a very low value and are backpropagated to 0 through all convolutional layers. This is a clear sign of the *vanishing gradient problem*, specially inherent to deep networks. Due to the exponential nature of the forward propagation algorithm, activation values that start at a value below 1 in the first layers will tend to exponentially decrease as these activations pass through more and more layers. If the model is deep enough, this will make the activations of the last layer to become zero, and when the backpropagation algorithm calculates the gradient, these will become zero as well, explaining the phenomenom we see in *Figure 1* for the deep model. Finally, it is important to note that this phenomenom is more prone to happen in networks that have sigmoid activation functions. However, if the model is deep enough (such as ours) activations such a ReLu, Tanh or LeakyReLu (Fred Agarap; Namin et al., 2009; Xu et al., 2015), will also saturate in very low values and cause a vanishing or exploding gradient. This is the phenomenon we wish to address during the rest of this study. In the following section we will thoroughly explain 3 possible solutions that already exist in the literature.

[1]The full list of hyperparameters used to train the initial VGG08 and VGG38 can be found in Section 5

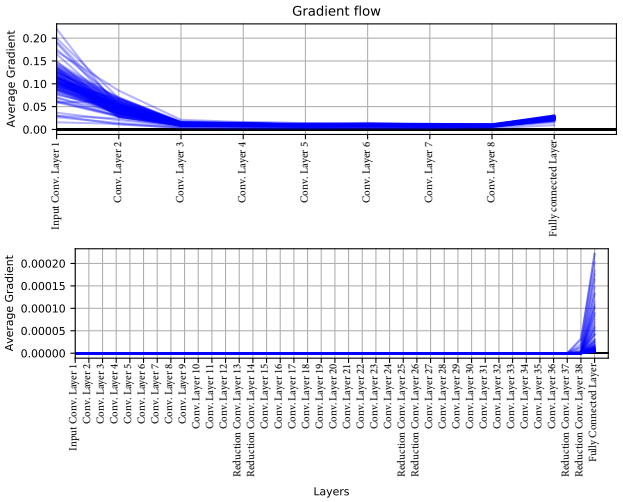


Figure 1. top: Gradient flow through all the layers in a VGG convolutional neural network of 8 layers. *bottom:*Gradient flow through all the layers in a VGG convolutional neural network of 38 layers. For both plots, each line represents one epoch of training through the complete training set

3. Background Literature

Firstly, (Ioffe & Szegedy, 2015) seeks to directly reduce the phenomenon that causes the problem mentioned in the previous section, and formally defined as internal covariate shift which, considering a deep neural network, is the continuous change in a specific layer's input distribution, to which this layer needs to continuously adapt. And so, with deeper models, those layers will need to adapt to the distributions of all previous layers. This "cumulative adaptation" leads those input distributions to saturate the non-linearity of the current layer into very high regimes (exploding gradient), or very low regimes (vanishing gradient), and this will completely stifle the learning of the deep network.

Consequently, (Ioffe & Szegedy, 2015) introduces the solution of Batch Normalization, which addresses the source of the problem by fixing the distribution of the input to each layer (so the activations from the previous one) to have mean 0 and standard deviation 1. In the case of a Convolutional Neural Network, this is done by applying the following transformation is applied to the direct output of the convolutional layer, over all its feature maps $x^{(k)}$:

$$y^{(k)} = \gamma^{(k)}\left(\frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}\right) + \beta^{(k)}$$

Where $E[x^{(k)}]$ and $Var[x^{(k)}]$, are the expectation and variance computed over the feature maps and β and γ are parameters to be learned by the model o ensure that the model can represent the identity transform. The result is then passed through the non-linearity.

Subsequently, the paper tests this method on multiple architectures and concludes that a deep model that uses Batch Normalization is capable of having a stable distribution of activation values throughout training, which allows deeper networks to actually learn. Not only that, the paper also finds that since the gradient flow is stabilized at each layer, higher learning rates can also be used to speed up learning.

Secondly, and building upon the last paper, (He et al., 2016) seeks to improve the performance of very deep CNNs that even with Batch Normalization, exhibit the problem defined as degradation. Indeed, as the network gets deeper, the accuracy of the model gets saturated and starts decreasing, a problem that is not attributed to overfitting, according to the paper.

Thus, the paper proposes the solution of adding a skip connection between the layer l and layer $l + 2$ to the original network, which adds the outputs of the layer l to the output of layer $l + 2$. This can be easily formalized as $F(x) + x$ where $F(x)$ is the mapping learned by layer $l + 2$, and x is the residual mapping from the layer l. Hypothetically, by adding this "residual mapping", layer $l + 2$ will have an easier time learning it's parameters, since a part of them were already learned in the previous layers. This skip connection will then be added to all of the layers in the network, and this will allow the learning of the network to never get stuck, and keep learning as the model gets deeper and deeper. These networks are then defined as ResNets.

Next, the paper moves on to implement deeper ResNets specifically with the VGG architecture in order to verify that the error decreases and the accuracy increases as the model gets deeper and deeper. Indeed, the authors verify that very deep ResNets with 50, 101, 152 layers continue to improve performance, with no signs of learning degradation.

The authors conclude the paper by exploring an extremely deep model of 1202 layers with ResNets, and verify that there are no optimization problems on such deep models, but conclude on the fact that aggressive regularization methods are needed for such deep models to avoid overfitting.

Thirdly, (Huang et al., 2017) builds upon the intuition introduced by (He et al., 2016) of adding connections between layers to improve information flow in the network and goes further by seeking to maximize it by connecting all layers with each other. This means that a given layer in a deep convolutional network will have as input all outputs from all previous layers. Conversely, the feature maps from that given layer will be fed into all following layers. This leaves a network with L(L+1)/2 connections, instead of L connections in traditional CNNs. Due to this dense setting, this type of networks are defined as DenseNets.

Apart from its dense characteristic, DenseNets differ from ResNets on the way the outputs from the previous layers are inputted to the current layer as $x_l = H_l([x_0, x_1, ..., x_{l-1}])$. Here the inputs from the previous layers $[x_0, x_1, ..., x_{l-1}]$ are concatenated (instead of summed as in ResNets) and then fed through H_l, which is a composite function composed of a series of consecutive convolution-ReLu operations on the inputs.

The authors move on to test this new architecture on multiple datasets and conclude that although ResNets and DenseNets have fairly similar performances, the concatenation operation and the fully-connected architecture allow non-optimal DenseNets to achieve this with much fewer parameters, and less computational complexity.

4. Solution Overview

From the solutions studied above, we choose to implement Deep Residual Networks. Proposed by (He et al., 2016), the method allows enough modularity to easily couple it with the already implementented VGG Architecture. This archtecture, first introduced by (Simonyan & Zisserman, 2015), is composed of k stages, each composed by b convolutional blocks. Each convolutional block is composed of two convolution "same" operations each followed by a non-linearity. Each stage S_k is interfaced by one convolutional block with a pooling layer that halves the size of the inputs from S_k, and feeds them to the following stage S_{k+1}. There is one convolutional layer for the input sample, and one fully connected layer for the output from this sample.

Therefore, we decide to implement Residual Networks because it allows us to easily create the skip connection mentioned in the literature between the input of the convolutional block and its output, by only summing them together. Additionally, (He et al., 2016) normalizes the distribution of the output of the convolution operations inside the convolutional block, following the method proposed by (Ioffe & Szegedy, 2015). Since we are using a VGG architecture, this becomes fairly straight forward to implement, and the training algorithm for a given VGG Architecture and input training data is given in *Algorithm 1*.

Finally, apart from the implemententional simplicity, Residual Networks that use Batch Normalization will firstly normalize each feature map outputted by each convolutional layer, and will feed this normalized distribution to the nonlinearity, which will prevent it from saturating, and eventually making the gradient disappear. Secondly, in case a given layer has problems learning the new inputs, the residual mapping will will facilitate the learning since it feeds it parameters that were already learned in the previous layers. This will improve the overall performance of the network, as shown in (He et al., 2016).

5. Experiments

We now focus on extensively testing the efficacy of our solution for multiple hyperparameter scenarios. To do this, we will train, validate and test the ability of our VGG archi-

Algorithm 1 Training Algorithm for a VGG CNN with ResNets and Batch Normalization

Input: x of shape (*# of training samples*, image height N_H, image width N_W, # channels N_C), *VGGConvArchitecture*, # of epochs, mini-batch size
$\gamma, \beta \leftarrow$ InitBNParams()
$w, b \leftarrow$ InitWeights&Biases()
repeat
  repeat
    $x_b \leftarrow GetMinibatch(x, minibatchsize)$
    $a_{l-1} \leftarrow InputConvolution(x_b, w, b)$
    for all *ConvBlocks* **in** *VGGConvArchitecture* **do**
      $Identity \leftarrow a_{l-1}$
      $a_l \leftarrow Convolution(a_{l-1})$
      $a_l \leftarrow BatchNormalize(a_l, \gamma, \beta)$
      $a_l \leftarrow LeakyReLu(a_l)$
      if *ConvBlock* **is** *ReductionBlock* **then**
        $a_l \leftarrow ReductionLayer(a_l)$
      end if
      $a_l \leftarrow Convolution(a_l)$
      $a_l \leftarrow BatchNormalize(a_l, \gamma, \beta)$
      if *ConvBlock* **is** *ReductionBlock* **then**
        $a_l \leftarrow a_l + ReductionLayer(Identity)$
      else
        $a_l \leftarrow a_l + Identity$
      end if
      $a_l \leftarrow LeakyReLu(a_l)$
      $a_{l-1} \leftarrow a_l$
    end for
    $a_{l-1} \leftarrow FullyConnectedLayer(a_{l-1})$
    $w, b, \gamma, \beta \leftarrow BackPropagate(a_{l-1})$
  until There are no more mini-batches
until There are no more epochs

tecture to learn to recognize images from the benchmark dataset CIFAR-100 (Krizhevsky, 2009). In total, the dataset contains 100 classes to recognize, and there are 600 32x32 RGB images per class, amounting to 60000 images. In our case, we use 47500 images for training, 2500 images for validation, and 10000 images for testing.

For all of our experiments, except where explicitly stated otherwise, we will use the following learning parameters: For optimization we will use the *Adam optimizer*. Proposed by (Kingma & Ba, 2015), it works efficiently with large datasets and large parameter settings such as the CIFAR100. For the learning rate annealing we will use the *cosine learning rate scheduler*. Proposed by (Loshchilov & Hutter), cosine annealing allows us to achieve good results with 2 times to 4 time less epochs. We will use the default parameters of the PyTorch implementation for Adam and Cosine Annealing. For the loss function we used the *corss-entropy loss function*. We will change all non-linearities to Leaky ReLu (Xu et al., 2015), since these fix the dying ReLu problem (Lu et al., 2019).

Finally, we use 100 epochs, a mini-batch size of 100 and no weight decay, we want to direcly compare our results with the inital tests.

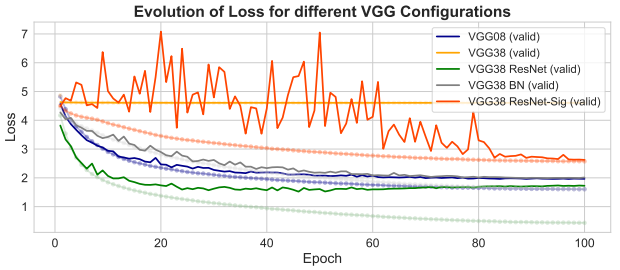


Figure 2. Loss function evolution during training for multiple VGG configurations. The light dotted lines correspond to the train loss function for each configuration.This will be the same for all accuracy and loss evolution plots shown in this paper

5.1. Testing the soluton's learning stability on the initial deep network and its ability to stabilize the sigmoid non-linearity

For this set of experiments we focus on comparing the optimization convergence performance of our VGG38 ResNet solution compared to the initial VGG08 and VGG38 layers, and only applying Batch Normalization. Finally, we add one more ResNet model with the sigmoid non-linearity to test how well the model is able to force the loss function to converge, even with a non-linearity that is very prone to saturate. These models can be seen in the following table.

Model	# of layers	Activation	# of Parameters
VGG08	8	LeakyReLu	59716
VGG38	38	LeakyReLu	336196
VGG38 BN	38	LeakyReLu	338500
VGG38 ResNet	38	LeakyReLu	338500
VGG38 ResNet-Sig	38	Sigmoid	338500

The learning performance of these models during training can be visualized in *Figure 2*. Since the VGG38 ResNet model achieves the lowest training loss and validation loss at 0.44 and 1.72, we will use this model in the following set of experiments.

5.2. Improving the model's generalization performance through L2 regularization

For this set of experiments we decide to improve the VGG38 ResNet model's generalization performance before moving on to continue testing its ability to stabilize the gradient during learning. To do this, we apply L2 Regularization (Cortes et al., 2012) since its implementation is integrated with the Adam optimizer PyTorch implementation, and so it's very straightforward to include it.

From (Hoon Park et al., 2019), we can see that a baseline VGG16 architecture reaches optimal performance on CIFAR-100 at L2 weight penalty (WP) λ =1e-3. We then decide to test this optimal value on our VGG38 ResNet architecture, and a stronger and weaker value of λ =1e-2 and λ =1e-4, respectively. These results can be seen in *Figure 3*. The final training and validation accuracy and

loss can be seen in the following table.

λ	Train Acc.	Val. Acc.	Train Loss	Val. Loss
1e-2	58.09%	53.52%	1.53	1.69
1e-3	**87.83%**	**65.32%**	**0.42**	**1.35**
1e-4	88.23%	62.64%	0.38	1.70
VGG38 ResNet	86.26%	62.32%	0.44	1.72

From the results, we prove that λ =1e-3 provides the best generalization performance out of the three weight penalties, and the original solution VGG38 ResNet. We then use this value for the rest of our experiments and we will call the improved model VGG38 RN+WP for brevity.

5.3. Testing the solution's learning stability with higher learning rates and bigger mini-batches

We now move on to testing if our solution is still able to bring the training loss down under a more unstable learning configuration. Firstly, we multiply the starting and ending learning rate of our cosine annealing scheduler by 10, and by 30. From this learning rate speedup, we expect the network to have a more difficult time making the loss function converge, since the optimizer will take larger steps going down the loss function, and thus will have a harder time finding any local optima. With the same objective, we increase the mini-batch size fromm 100 to 256 and 512. We plot the loss optimization results of this experiment, and further compare these with the VGG38 ResNet with best L2 weight decay. These results can be seen in *Figure 4*. The final loss values can be seen in the following table.

Mini-batch size	LR Speed-up	Train Loss	Val. Loss
100	x5	1.41	1.63
100	x30	3.16	3.17
256	None	0.39	1.51
512	None	0.48	1.55
VGG38 RN+WP	None	**0.42**	**1.35**

It's important to add that we also tried a x100 learning rate, but this made the learning too unstable and the loss function never converged, and stayed at a training and validation loss value of approximately 3.5, with peaks that would go as high as 8. Since these experiments didn't improve performance, we keep the VGG38 RN+WP model for the following experiments.

5.4. Testing the solution's ability to improve learning with deeper model's

For this set of experiments we focus on verifying that our solution, VGG38 RN+WP is able to continue to converge for multiple depths and VGG configurations. To do this, we define the following models as shown in the table below.

Model	Stages	Blocks/Stage	Layers	Params
VGG68	3	10	68	616900
VGG72	5	6	72	654148
VGG106	5	12	106	1210948
VGG38 RN+WP	3	5	38	338500

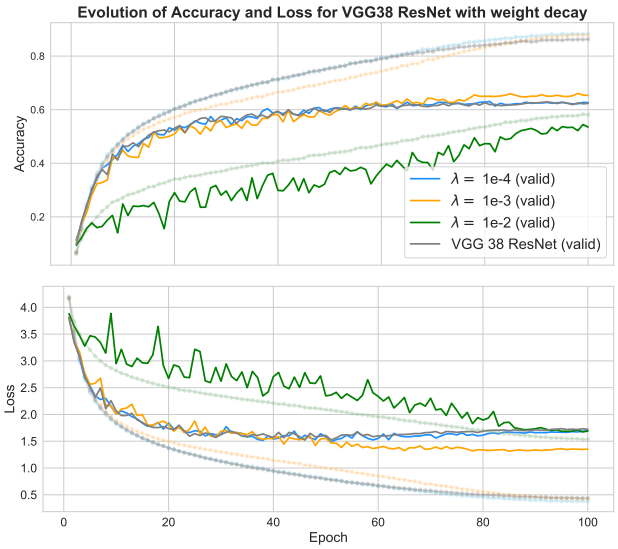


Figure 3. Accuracy and Loss evolution during training for multiple L2 weight decay coefficients.

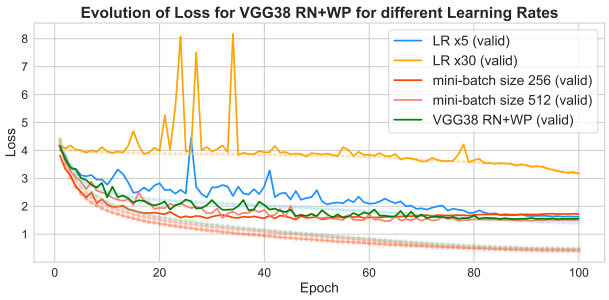


Figure 4. Loss function evolution during training for different learning rate speedups and mini-batch sizes.

Their performance during training can be seen in *Figure 5*, and their final results can be seen in the following table.

Model	Train Acc.	Val. Acc.	Train Loss	Val. Loss
VGG68	**86.85%**	**66.76%**	**0.44**	**1.30**
VGG72	90.02%	67.08%	0.33	1.32
VGG106	90.03%	66.56%	0.32	1.34
VGG38 RN+WP	87.83%	65.32%	0.42	1.35

Respectively, the four models train at 8.6, 2.34, 1.2, and 6.45 it/s. This means that the deep VGG68 Model was able to achieve a higher validation accuracy, and lower validation loss, and reduced training time by approximately 30%. We thus report the test results on this model as follows.

Final Model	Test Accuracy	Test Loss
VGG68	**65.83%**	**1.33**

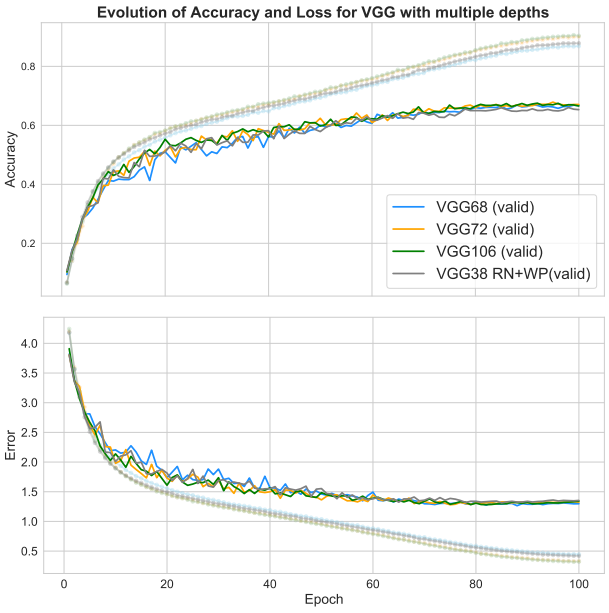


Figure 5. Evolution of Accuracy and Loss during training for deeper VGG models.

6. Discussion

From the comparison done in § 5.1 we found that Batch Normalization alone is indeed able to allow deeper models such as VGG38 to optimize their loss function without any vanishing/exploding gradients, this coroborates what was said by (Ioffe & Szegedy, 2015). From the same set of experiments, we see that a model that uses Residual Networks and Batch Normalization is not only able to solve the vanishing gradient problem, but is also able to improve performance. This is expected since, as explained by (He et al., 2016) and mentioned before, a residual mapping will allow each layer in the network to better learn its own parameters. Lastly, for this set of experiments we see that VGG38 ResNet-sig is able to start converging after around 60 epochs of training. This goes in line with our intuiton, this means that even if we use an activation function that saturates easily, the Batch Normalization step will still allow the network to learn, even if it is very slowly.

From the experiments in § 5.2 we can see that, as stated by (Hoon Park et al., 2019) the best L2 weight decay coefficient is λ = 1e-3. This weight decay coefficient improves the validation accuracy significantly by 3% compared to the VGG38 ResNet. From *Figure 3*, λ = 1e-2 hurts the learning too much and doesn't allow it to converge smoothly as for the rest of the models in this experiment. On the contrary, λ = 1e-4 does not penalize large weights enough to make a significant improvement, improving the validation accuracy by only 0.32%.

Moving on, with the experiments in § 5.3 we sought to strain the loss function by increasing the size of the update step down the optimization slope. For the learning rate experiments, the x5 speedup made the learning sloppier, but it was still able to converge to a validation loss of 1.35, 0.28 points higher than for our VGG38 RN+WP model. The x30 speedup made it much more difficult for the loss function to converge, and it only converged after 40 epochs of training. However, it was still able to converge, which proves the capacity of the Residual Networks and Batch Normalization to allow deep networks to learn, even under very unstable optimization conditions.

Finally, with the experiments done in § 5.4 we are able to prove that a deep convolutional network with residual networks and batch normalization is able to continue improving its performance as the model gets deeper. We also find that using more reduction stages increases the training time significantly, while just increasing the number of convolutional blocks can reduce training time. Lastly, we determine that the best model that comes out of this study is the VGG38 network, achieving a 65.83% test accuracy. This result is achieved thanks to the ability of the Residual Network to improve the learning of each layer. Additionally, since the number of convolutional blocks is increased, this improves the overall learning of the network.

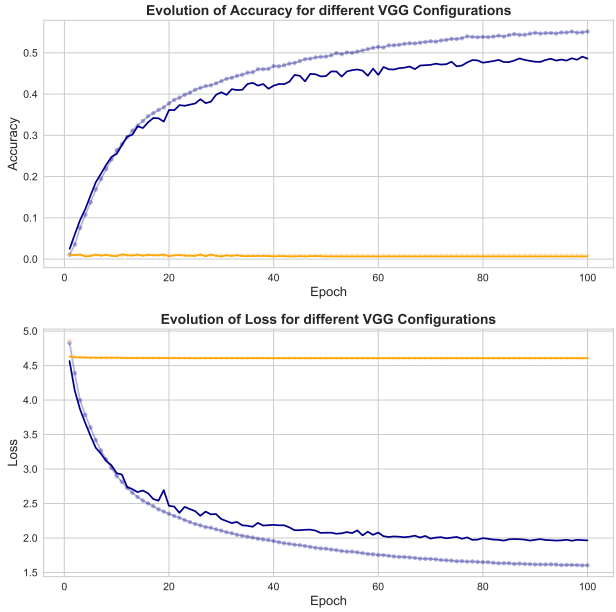
7. Conclusions

In this study we explored one of the optimization problems found in training very deep Convolutional Neural Netowrks. We identified that this problems are caused by the cumulative saturation of the activation functions during forward propagation when the model is too deep, and are formally defined as vanishing/exploding gradients. We then visualize this problem in the gradient flow of a 38 layered VGG Network, and look into the literature for its possible solutions. Consequently, we decide to implement Residual Networks with Batch Normalization, since it's is very straightforward to assemble with the VGG architecture, and provides further gains in accuracy due to its residual mappings.

We move on to experiment to what extent this solution is effective in helping the loss function of the model keep decreasing by training it under multiple hyperparameter scenarios on the CIFAR-100 dataset. We conclude that this solution is robust enough to force the loss function to continue decreasing under all of these scenarios. Lastly, we find that a 68 layered VGG model yields the best performance of 65.83% during test time.

To further improve this study, we would like to better understand why only increasing the number of convolutional blocks in a VGG architecture improves the training time as well as the validation accuracy. Studies on thhat compare this architecture to other notable CNN architectures (Rawat et al., 2020; Khan et al., 2020; Ahmed & a. A. Karim, 2020), suggest that it may be because a large number of small sized filters such as the VGG convolutional blocks reproduce the representational capacity of larger filters with less parameters. However, further experimenting needs to be done on this matter in order to better discriminate the benefits of using small filter sizes on the representational performance and complexity of a CNN.

Annex A



Comparison of the training loss and accuracy between a shallow convolutional neural network with a deep convolutional neural network. This figure can also be found in Figure 1 of the coursework handout.

References

Ahmed, W. S. and a. A. Karim, A. The impact of filter size and number of filters on classification accuracy in cnn. In *2020 International Conference on Computer Science and Software Engineering (CSASE)*, pp. 88–93, 2020. doi: 10.1109/CSASE48920.2020.9142089.

Cortes, Corinna, Mohri, Mehryar, and Rostamizadeh, Afshin. L2 Regularization for Learning Kernels. *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009*, pp. 109–116, may 2012. URL http://arxiv.org/abs/1205.2653.

Fred Agarap, Abien M. Deep Learning using Rectified Linear Units (ReLU). Technical report. URL https://github.com/AFAgarap/relu-classifier.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December:770–778, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.90.

Hoon Park, Dae, Man Ho, Chiu, Chang, Yi, and Zhang, Huaqing. Gradient-Coherent Strong Regularization for Deep Neural Networks with Stochastic Gradient Descent. Technical report, 2019.

Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:2261–2269, 2017. doi: 10.1109/CVPR.2017.243.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, Francis and Blei, David (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

Khan, Asifullah, Sohail, Anabia, Zahoora, Umme, and Qureshi, Aqsa Saeed. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020. ISSN 15737462. doi: 10.1007/s10462-020-09825-6.

Kingma, Diederik P. and Ba, Jimmy Lei. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015. URL https://arxiv.org/abs/1412.6980v9.

Krizhevsky, Alex. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. URL https://doi.org/10.1145/3065386.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Loshchilov, Ilya and Hutter, Frank. SGDR: STOCHASTIC GRADIENT DESCENT WITH WARM RESTARTS. Technical report. URL https://github.com/loshchil/SGDR.

Lu, Lu, Shin, Yeonjong, Su, Yanhui, and Karniadakis, George Em. Dying ReLU and Initialization: Theory and Numerical Examples. *arXiv*, mar 2019. URL http://arxiv.org/abs/1903.06733.

Namin, Ashkan, Leboeuf, Karl, Muscedere, Roberto, Wu, Huapeng, and Ahmadi, Majid. Efficient hardware implementation of the hyperbolic tangent sigmoid function. pp. 2117 – 2120, 06 2009. doi: 10.1109/ISCAS.2009.5118213.

Practical, Machine Learning. Machine Learning Practical 2020 / 21 : Coursework 2 Task 1 : Identifying training problems of a deep CNN. 2020(November):1–10, 2020.

Rawat, Jyoti, Logofătu, Doina, and Chiramel, Sruthi. Factors affecting accuracy of convolutional neural network

using vgg-16. In Iliadis, Lazaros, Angelov, Plamen Parvanov, Jayne, Chrisina, and Pimenidis, Elias (eds.), *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference*, pp. 251–260, Cham, 2020. Springer International Publishing. ISBN 978-3-030-48791-1.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, sep 2015. URL http://www.robots.ox.ac.uk/.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott E., Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical Evaluation of Rectified Activations in Convolutional Network. may 2015. URL http://arxiv.org/abs/1505.00853.