Investigations on a Modern ConvNet

Group 60
Robert Andersson, Jan Kauper and Joel Kärn
KTH Royal Institute of Technology
`rgand@kth.se, kauper@kth.se and jkarn@kth.se`

Abstract

In deep learning, a Convolutional Neural Network (*CNN*) is a type of Artificial Neural Network that specializes in processing data that has a grid-like topology, such as images. The efficiency in which *CNN* exploits correlation in data is one of its most appealing features and has made *CNNs* the network architecture of choice when it comes to image classification. In this project, we implemented a Residual Network architecture *(ResNet)*, which is a type of (*CNN*) build on the concept of "skip-connections", to perform image classification using the *CIFAR-10* data-set. We studied various techniques such as *Dropout, Weight Decay, Batch Normalization, Early Stopping, Noisy Labelling, Data Augmentation, Scheduled Learning Rate* and the effect they had on our network. Our best model achieved an accuracy of 93.78% after 100 training epochs. The open source neural network library *Keras* in *Python* was used to build the network.

Repository with code can be found at:
`https://github.com/rgand1/DD2424_Final_Project`

1 Introduction

Classification between objects is a natural and self-evident task among us humans, on the contrary, this has proven to be an intricate problem for the machines. In computer vision the benchmark task is image classification. Image Classification, often referred to as Image Recognition, is the task of associating one,single-label classification, or more, multi-label classification, labels to a given image. It has revolutionized and launched technological advancements in the most eminent fields, among others the automobile industry, healthcare and manufacturing. Convolutional neural networks, *CNNs* are widely used in computer vision and have become the state of the art in various applications when it comes to image classification. *CNNs* are very efficient at picking up on patterns in the input image, such as gradients, circles and lines. It is this property that makes *CNNs* so dominant in computer vision. The power of a *CNN* comes from a certain kind of layer called the convolutional layer. A *CNN* contains many convolutional layers stacked on top of each other, each one capable of recognizing more complicated shapes. Interestingly enough the usage of convolutional layers in a *CNN* mirrors the structure of the human visual cortex, where a series of layers process an incoming image and identify successively more complex features [6]. But *CNN* is not perfect, it has been found that there is a maximum threshold for depth which can be achieved due to vanishing gradients problem. This problem of training very deep networks has been diminished with the introduction of Residual Netoworks,*ResNets*, where a *ResNet* is made up from Residual Blocks. The main difference now is that there is a direct connection which skips some layers. This connection is called "skip-connection" and is the core of residual blocks. The "skip-connections" in *ResNet* solve the problem of vanishing gradients in deep neural networks by allowing this alternate shortcut path for the gradient to flow through [2].

In this assignment, we implemented a custom *ResNet* architecture to perform image classification using the *CIFAR-10* data-set. In our best model we used *Dropout, Weight Decay, Batch Normalization,* *Data Augmentation, Scheduled Learning Rate* to achieve an accuracy of 93.78% after 100 epochs of training.

2 Related Work

For a long period of time, the architecture of *CNNs* has been the same, stacked convolutional layers often followed by max-pooling that is followed by one or more fully-connected layers [4].There have been various modifications of this basic architecture in the image classification literature and research. These models have yielded some great results on the *MNIST, CIFAR*, and especially on the *ImageNet* dataset [6]. Over the years, researchers tend to make deeper neural networks to solve more complex tasks and improve classification accuracy. But, it has been seen that as we go adding on more layers to the neural network, it becomes difficult to train them and the accuracy starts saturating and then degrades. Residual Network was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their paper "Deep Residual Learning for Image Recognition" and suggested a solution to the problem [2]. *ResNets* solved the problem by applying identity mapping. What this means is that the input to some layer is passed directly or as a shortcut to some other layer. *ResNet* models were extremely successful which can be seen in the following

- Won 1st place in the *ILSVRC* 2015 classification competition with a top-5 error rate of 3.57%
- Won the 1st place in *ILSVRC* and *COCO* 2015 competition in ImageNet Detection, ImageNet localization, *Coco* detection and *Coco* segmentation
- Replacing *VGG-16* layers in Faster *R-CNN* with *ResNet-101*. They observed relative improvements of 28%
- Efficiently trained networks with 100 layers and 1000 layers also.

3 Data

In our project we used the *Cifar10* to train and test our network structure. As we achieved accuracies around 93%, we eventually tested the same *ResNet* architecture on the *Cifar100* dataset. Without any changes, we still achieved 71.39 % accuracy.
The state-of-the-art models for *Cifar10* are able to achieve accuracies above 99% [7]. The best performing network at the time being is the *CaiT-M-36* 224 model, described in "Going deeper with Image Transformation" [10]. The paper introduces the Class-Attention in Image Transformers *(CaiT)* structure as an improvement of *ViT*, which uses Vision Transformers to split pictures into a sequence of patches along with a classification token [1].
However, the best performing *ResNet* so far is the *BiT-L* network with 99.37% accuracy, introduced in [5]. *BiT* stands for Big Transfer as they pre-train models on large image datasets and transfer the model to smaller datasets during fine-tuning. During upstream pre-training, they use a vanilla *ResNet-v2* architecture but replace batch normalization with group normalization and weight standardization. They further propose using an *SGD* optimizer with momentum 0.9 and learning rate scheduling as well as several augmentation techniques. During downstream fine tuning, where the model is transferred to smaller data sets like *Cifar10*, the pre-trained model allows to decrease the initial learning rate from 0.03 to 0.003 and stepwise decrease it during training. The model ultimately performed well on both *Cifar10* and *Cifar100*, with accuracies of 99.37% and 93.51% respectively.

4 Methods

To achieve 90% or better accuracy on validation data, it is critical to avoid overfitting the model on training data and instead train the model as generic as possible. To enhance the performance of our network on the unknown test batch in *Cifar10*, we explored two main fields of improvement.

Firstly, due to the rapid overfitting of deep Convolutional Neural Networks *(CNN)*, we developed and used a Residual Network *(ResNet)* instead. The big advantage of *ResNet's* is their good performance on complex and deep neural networks. Since we aimed for accuracy above 90%, the required complexity of a *CNN* exceeds a critical point where the error increases due to exploding and vanishing gradient descent. This also leads to an over-fitting of the model to the training data and subsequently to a decrease in accuracy in the validation data. The *ResNet* addresses this problem by using skip-connections to sustain the learning parameters of a network through many subsequent layers. It therefore allows us to build much deeper and more complex networks which are capable of reducing the validation error below 10%. [3]

Secondly, we testes several regularizations and optimization techniques to either improve the quality of the dataset or directly influence the learning algorithm.

- *Data Augmentation*
- *SGD/ Adam Optimizer*
- *Learning Rate Schedule*
- *Batch Normalization*
- *Early Stopping*
- *Model Checkpoint*

Data Augmentation can help improve the performance of a network by increasing the number of samples used during the training phase. *Tensorflow's ImageGenerator* uses a variety of operations to augment and create additional data batches. Some of the basic augmentation techniques are flipping, rotating, scaling, or cropping pictures.
A very important factor in training neuronal networks is choosing the right ***Optimizer***. The most common optimizers are *Adam* and *SGD*. While *Adam* is an adaptive optimizer itself and thus does not need a particular learning rate schedule, the *SGD* optimizer requires manually tuning the learning rate to efficiently obtain convergence. According to [8] it is recommendable to either use an adaptive optimizer or use *SGD* + momentum together with additional improvement strategies like *batch normalization* and *early stopping*.
Adjusting the *SGD* + momentum to the current learning epoch requires manual ***Learning Rate Scheduling***. It ensures both rapid and continuous learning progress. At the beginning of the training process, a high learning rate enables a fast but divergent learning progress. However, as the epoch progresses, the learning rate should decrease to ensure constant and convergent learning progress.
As recommended in [8], ***Batch Normalization*** can have a major impact on successful network training. *Batch normalization* complements initial data normalization by retaining the benefits of normalized data throughout the training process. Otherwise, benefits such as accelerated training, additional error regularisation and the ability to use higher learning rates would be lost as training progresses. Additional to *Batch Normalization* there are several similar techniques like *Layer Normalization*, *Group Normalization*, and *Instance Normalization*.
Another handy method to improve training is the callback-function ***EarlyStopping***. It monitors and evaluates key quantities like loss or accuracy after each epoch. The model stops training as soon as the chosen value does not show further improvement. This measure can safe valuable training resources like time and storage while allowing to configure models with hundreds of epochs.
Another callback-method with a similar approach is ***ModelCheckpoint***. Depending on individual settings it evaluates the model after each epoch regarding loss or accuracy and safes either the entire model or just the wights to an *HDF5* file. After training completed one could simply load and use the best performing model instead of relying on the last training epoch. Both callback-functions, the early stopping as well as model checkpoint tackle the problem of overfitting model in long training runs.
To further test the performance and robustness of our network we used *noisy labels* to corrupt training data. By systematically flipping a certain percentage of the labels and thus falsifying the training data, the network generally looses accuracy. However, by applying certain regularization techniques it is possible to achieve a sufficient generalization capability and overcome noisy labels. Popular, but not always sufficient techniques are *data augmentation*, *weight decay*, *dropout* and *batch normalization*. [9]

5 Experiments

5.1 CNN

With the goal to build a model that would reach 90% test accuracy on the *CIFAR-10* data set, we started off by following [4] to set up a baseline *CNN*. The data was normalized and converted from integers to floats and then the model itself was built from scratch, using *Keras*. The *SGD* optimizer was used with momentum and no regularization to begin with. The performance of that network can be seen in Fig.1. It achieved a test accuracy of 68%. From Fig.1 we can obviously tell the the model overfits to the training data and hence some different generalization methods were tried. The boost in performance of two of these can be seen in Fig.3, showing how the model performed with *data augmentation* Fig.2a and *dropout* Fig.2b. Both of these methods could reach over 80% in test accuracy. By combining them, together with *batch normalization* and *weight decay* with *L2-norm*, a test accuracy of 84% was achieved, and the performance can be seen in Fig.2c. To ensure *batch normalization* performs best, we tested it against *layer*, *group*, and *instance normalization*. In average, *batch normalization* achieved the highest accuracies. By tweaking the model and running it for more epochs, we achieved 88.62% with the *CNN*, as seen in Fig.2d. In order to achieve a higher test accuracy without training the model for an unreasonable time, we started testing optimizing a *ResNet* instead.

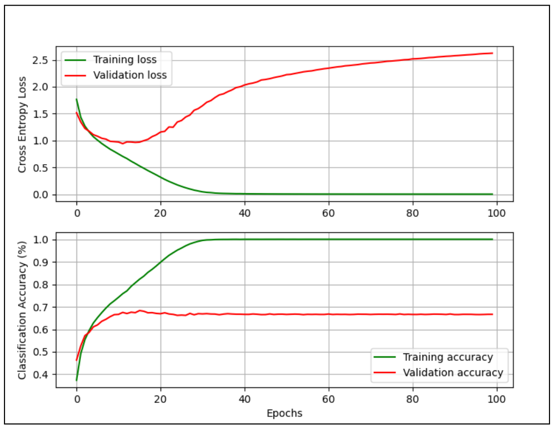
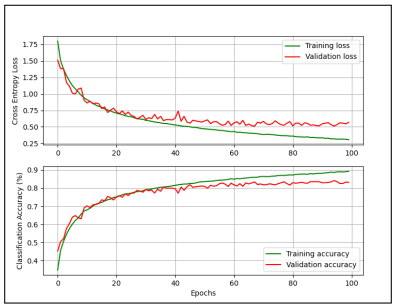
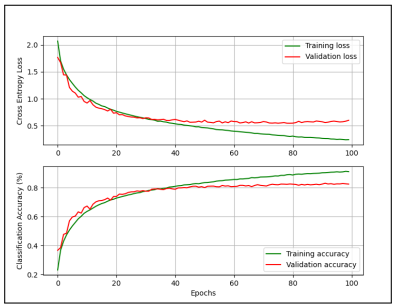


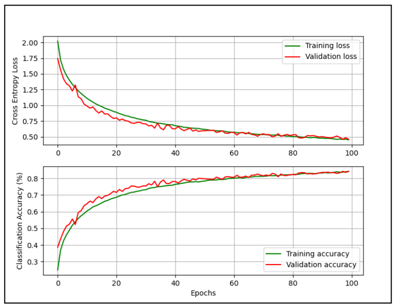
Figure 1: Baseline CNN.



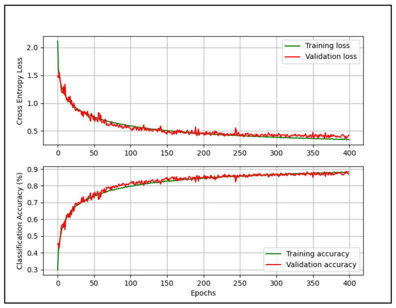
(a) CNN with data augmentation.



(b) CNN with dropout.

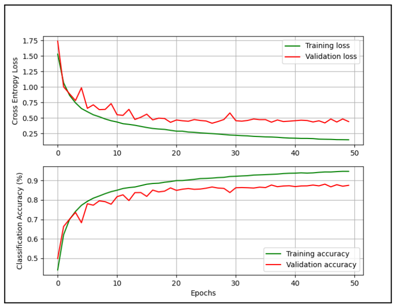


(c) CNN with various optimization techniques.

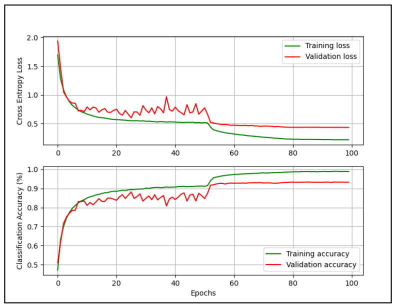


(d) Best performing CNN, training for 400 epochs.

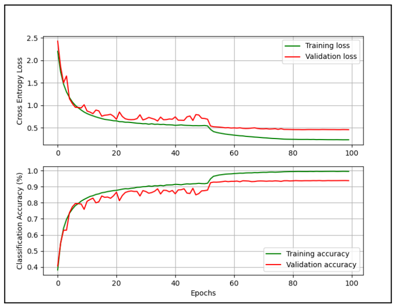
Figure 2: CNN with generalization methods.



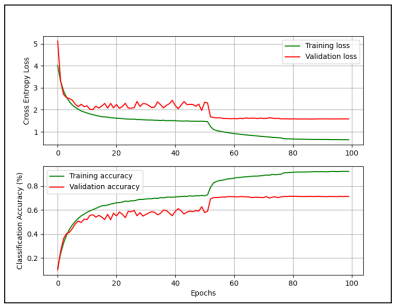
(a) ResNet with same optimization as CNN.



(b) ResNet with learning rate schedule.



(c) Deeper ResNet.



(d) ResNet on CIFAR-100 data set.

Figure 3: ResNet's with different optimization methods.

5.2 ResNet

In order to achieve the 90% test accuracy, we turned our *CNN* into a *ResNet*. Without too much effort the *ResNet* could achieve quite high test accuracy in much fewer epochs than the *CNN* could. It however peaked at around the same test accuracy, so despite the fast training, it didn't improve further despite training it for more epochs, as seen in Fig.3a. For this network, the same generalization methods were implemented, where *data augmentation, dropout, batch normalization, weight decay*, all proved useful but not enough to reach the 90%. The major breakthrough was a *learning rate schedule*.

5.2.1 Learning Rate Schedule

Different optimizers, such as *Adam* and *SGD* with different learning rates, with both linear and exponential decays, had been investigated but did not improve the final accuracy. *SGD* with a learning rate schedule was what the *ResNet* needed to improve beyond 89%. The one that was used started at a learning rate of 0.1 for the first half of the epochs of training, then it changed to 0.01, and then the last quarter of the epochs it was 0.001. With some tweaking of the other generalization and optimization methods, the *ResNet* was able to achieve a 93.36% test accuracy, as seen in Fig.3b.

5.2.2 Deeper ResNet

Before implementing the learning rate schedule, a deeper *ResNet* did not improve the performance. With the learning rate schedule implemented, the depth was investigated again and this time a deeper network improved the performance and the results can be seen in Fig.3c. The network managed to reach a test accuracy of 93.78%.

5.2.3 ResNet performance on CIFAR-100.

Mostly for curiosity's sake, we trained a *ResNet* with the same structure as our best performing *ResNet* (the only difference being the output layer containing 100 nodes instead of 10) on the *CIFAR-100* dataset. The results can be seen in Fig.3d, and it achieved a test accuracy of 71.39%.

5.3 Conclusions

To conclude, *CNN's* can be quite powerful when it comes to image classification, especially with a *ResNet* architecture. The *ResNet* with a *learning rate schedule* could achieve better performance than the regular *CNN*, in a shorter amount of epochs. The best performing *CNN* reached a test accuracy of 88.62% in 100 epochs while the best performing *ResNet* managed to reach 93.78% in the same amount of epochs. In addition to its good performance on the *CIFAR-10* dataset, the *ResNet* performed surprisingly well on the *CIFAR-100*, without any customization. If further work was to be made, it would be of interest to see if a deeper network would improve the results further on the *CIFAR-10* and *CIFAR-100* data sets. Other generalization methods that showed interesting results but that we did not have sufficient time to investigate further was data augmentation on the test data and noisy labels.

References

[1] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: `2010.11929`. URL: `https://arxiv.org/abs/2010.11929`.

[2] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: `10.48550/ARXIV.1512.03385`. URL: `https://arxiv.org/abs/1512.03385`.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[4] Jason Brownlee. *How to Develop a CNN From Scratch for CIFAR-10 Photo Classification*. May 2019. URL: `https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/`.

[5] Alexander Kolesnikov et al. "Large Scale Learning of General Visual Representations for Transfer". In: *CoRR* abs/1912.11370 (2019). arXiv: `1912.11370`. URL: `http://arxiv.org/abs/1912.11370`.

[6] C. Nebauer. "Evaluation of convolutional neural networks for visual recognition". In: *IEEE Transactions on Neural Networks* 9.4 (1998), pp. 685–696. DOI: `10.1109/72.701181`.

[7] Papers with code. *Image Classification on CIFAR-10*. 2022. URL: `https://paperswithcode.com/sota/image-classification-on-cifar-10`.

[8] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747`.

[9] Hwanjun Song et al. "Learning from Noisy Labels with Deep Neural Networks: A Survey". In: *CoRR* abs/2007.08199 (2020). arXiv: `2007.08199`. URL: `https://arxiv.org/abs/2007.08199`.

[10] Hugo Touvron et al. "Going deeper with Image Transformers". In: *CoRR* abs/2103.17239 (2021). arXiv: `2103.17239`. URL: `https://arxiv.org/abs/2103.17239`.