REPORT **Date** May 23, 2018 **Reference number** Deepl-18 Project

Created by
W. Skagerström, T. Price, N. Lindqvist

Classification of ImageNet with Convolutional Neural Networks

A study in techniques to improve CNN's

Abstract

This report is our final piece of work in the course DD2424, Deep Learning in Data Science. The overarching purpose is to investigate convolutional neural networks (CNNs) while applying knowledge and techniques which we've acquired during the course.

To thoroughly investigate the topic, a CNN will be constructed. The architecture will be based on research of previous state of the art implementations on the topic. The network will be optimized iteratively by more sophisticated initialization, changing activation functions and additional optimization and regularization such as dropout and batch normalization. The effect of each modification will be documented and analyzed. The results are then discussed on their own and in context of other ImageNet classifiers.

We implemented a VGGNet which initially suffered immensely by overfitting. By tinkering with parameters of dropout, L2 regularization and batch normalization we countered the effect to some extent but not fully. We achieved a maximum accuracy of 29%. All in all we learned a lot and conclude that optimization of CNNs is a difficult craft.

Contents

1 Introduction

Image classification is one of the main branches within Computer Vision. The task can be summarized as extracting features from a picture, which takes the shape of a pixel matrix containing either 1 or 3 channels (for gray-scale or RGB, respectively). Back in 2012, Convolutional Neural Networks (CNNs) began replacing the previously manually written algorithms for feature detection, and continue to hold the title as a state of the art method for image classification. The reason for the resurgence of CNNs was partly due to the winning entry of the AlexNet in the 2012 ImageNet competition [3]. Convolutional Neural Networks have existed for almost 20 years, but was previously limited by the availability of data and computer hardware. However, due to the great advancement of modern computers the ability to train increasingly deep and complex types of neural networks have been enabled.

In 2014, another progressive leap in the development of image classification occurred with the introduction of VGGNet and GoogleNet [6], [7]. Due to the possibility of creating CNNs of varying architecture and size, a modern industry standard for evaluating the performance of a network is by benchmarking it through the ImageNet dataset [5].

2 Previous work

Among the many entries to the ImageNet classification challenge, deep convolution network architectures such AlexNet, VGGNet and GoogleNet have proven themselves to be highly effective and has created a renewed interest in the development and optimization of such architectures. However, these different types of network comes with some individual strengths and weaknesses.

2.1 AlexNet

AlexNet is the network architecture that renewed the industry interest in Convolutional Neural networks. AlexNet consists of a total of 8 layers, which are 5 convolutional layers which are stacked on each other, followed by three fully connected layers. The network features two sets of of batch normalization, which are applied following each of the first two convolution layers, and a dropout occurs after each of the last two fully connected layers. Compared to the other networks mentioned here, it performs slightly worse in terms of accuracy as seen in figure 1.

2.2 GoogLeNet

GoogLeNet uses what is called a inception module, which works by calculating several different convolutions of different dimension within the same module combining them before propagating them to the next layer of the network[7]. This causes GoogLeNet to be much more space efficient due to the less amount of parameters. This is only further amplified by the usage of average pooling instead of fully connected layers, which further reduces the dimensionality of the total number of network parameters, see figure 1.

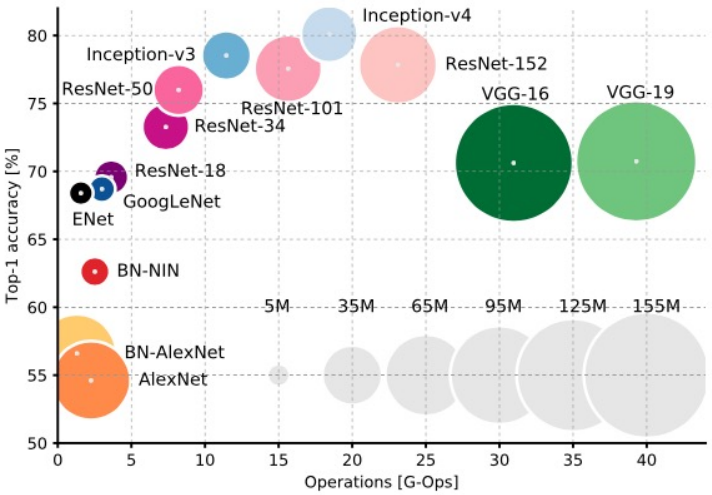


Figure 1: Comparision of performance, amount of parameters and number of operations between different CNNs.

2.3 ResNet

ResNet was added to the different architectures of networks for image classification in 2015 and won the 1st place on the ILSVRC 2015 classification task. As deeper networks are more difficult to train a residual learning framework eases the training for deep networks by adding a shortcut between layers. The shortcut decreases the complexity of the networks and witch allows more layers [2].

2.4 VGGNet

VGGNet was introduced in 2014 and one of its most common characteristics is the simplicity of the network, while still achieving good performance on complicated classification tasks such as the ImageNet challenge [6]. While being simple in its design, the network is computationally demanding to train due to the heavy number of parameters, again visualized in figure 1.

3 Method

This section covers our CNN, the methods used for improving its performance, and the tools utilized for the implementation.

3.1 Dataset

The data used was the ImageNet Tiny dataset, which is a reduced variant of the ImageNet set. The Tiny variation consists of 200 different classes. The resolution of the images has been reduced to $64 \times 64 \times 3$ (from $224 \times 224 \times 3$). The dataset consists of 100 000 training samples, 10000 validation samples, and 10000 test samples. Some images are in grayscale, these are ignored as they only constitutes 2% of the dataset. Figure 2 visualizes a small sample of pictures included in the dataset.

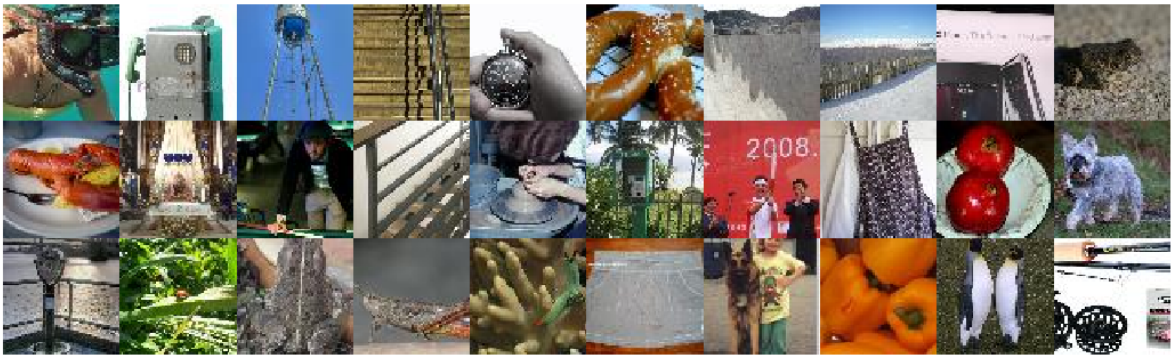


Figure 2: 30 Samples from the Tiny ImageNet dataset.

3.2 CNN implementation

We chose to experiment with different variations of the VGGNet architecture due to its simple implementation and the availability of hardware from the Google Cloud Computing services obtained by KTH for the DD2424 course.

3.2.1 Implementation specifics

The network was written and implemented in Python 3.5.2, using the Keras Framework which runs on top of Tensorflow. Numpy was used for data management and Matplotlib to create the graphs. To assist in training and evaluating the networks, we used Google Cloud services for computation power. The specifications of the hardware used was:

- 1 x NVIDIA Tesla K80
- An unspecified dual core CPU (Unknown CPU Platform on the Google Cloud Platform).
- 32 GB RAM memory
- 40 GB Disk space.

3.3 Normalization

Normalization of the input data occurs as two different methods: At first, some models are trained with data where the color channels are divided by the maximum output of the RGB color palettes (255), resulting in the the colors being represented as numbers in $\mathbb{R} \in [0, 1]$. This was later changed to subtracting the mean from the data set, channel wise, in addition to using Batch Normalization instead for the later training iterations.

3.4 Evaluating architectures

The basis of the CNN architecture is inspired by [4], it is summarized in figure 3. However, the network by [4] was implemented on the full sized ImageNet dataset. Inspired by architectures mentioned in Previous work 2, we decided to test four configurations, summarized in table 1.

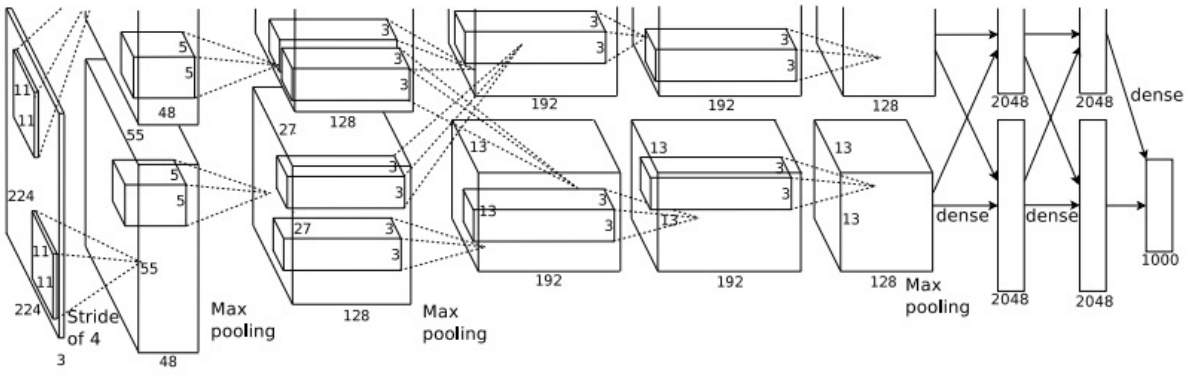


Figure 3: Overview of network architecture presented by [4].

Training was done on the entire Tiny ImageNet dataset, containing 100 000 images and their respective labels. The composition of the dataset is 200 classes with 500 images each. The training data is shuffled before the training process begins. For evaluation, the entire validation dataset was used, containing 10000 images.

ConvNet Configuration			
A	B	C	D
SGD, RUI	Adam, RUI, BN	Adam, RUI, Dropout	Adam, HEI, Dropout, BN
input ($64 \times 64 \times 3$ RGB $image$)			
conv-32, 2x2, ReLu	conv-32, 2x2, ReLu, BN	conv-32, 2x2, ReLu	conv-32, 2x2, ReLu, BN
conv-32, 2x1, ReLu	conv-32, 2x1, ReLu, BN	conv-32, 2x1, ReLu	conv-32, 2x1, ReLu, BN
conv-32, 1x2, ReLu	conv-32, 1x2, ReLu, BN	conv-32, 1x2, ReLu	conv-32, 1x2, ReLu, BN
MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2
conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN	conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN
conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN	conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN
conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN	conv-48, 2x2, ReLu	conv-48, 2x2, ReLu, BN
MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2
conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN	conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN
conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN	conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN
conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN	conv-80, 2x2, ReLu	conv-80, 2x2, ReLu, BN
MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2	MaxPooling, 2x2
FC-2048, ReLu	FC-2048, ReLu, BN	FC-2048, ReLu	FC-2048, ReLu, BN
FC-200, SoftMax	FC-200, SoftMax	Dropout 0.3	Dropout 0.3
		FC-200, SoftMax	FC-200, SoftMax

Table 1: Configuration of the four different CNNs tested initially. They differ in terms of solver (SGD/Adam), batch normalization (BN), Dropout and initializer (Random Unit Initialization/He Initialization).

4 Results

4.1 First training iteration: Declaring war on overfitting

Running over all 4 models presented in table 1 the training and validation loss proved immense after around 5-10 epochs. See figure 4. We then realized that most of the architectures lacked any form of regularization, except the ones with dropout. That is, without regularization these results are to be expected.

To combat the overfitting we began tinkering with the different methods we had in our toolbox by testing different combinations of:

- L2 regularization
- Decay rate
- Dropout
- Batch normalization

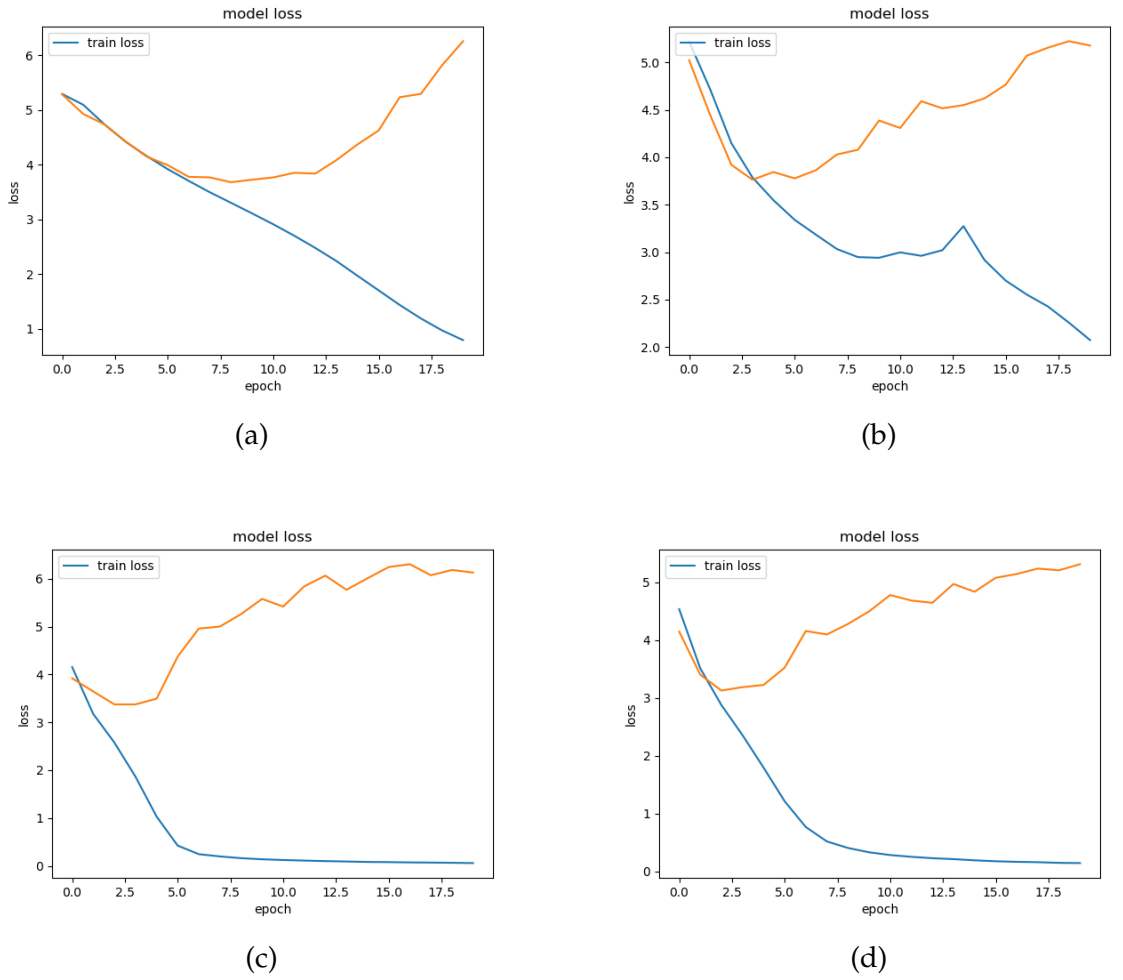


Figure 4: Training and validation loss of 20 first epochs of respective models presented in table 1. Each suffer from immense overfitting after 5-10 epochs of training.

We also added some additional preprocessing to the dataset by subtracting the mean value of the dataset, feature-wise.

Manually tuning these parameters we found some minor improvements, which is displayed in figure 5. In this version, we took model D from table 1 and added an L_2 regularization term of $1 * 10{-}3$ after each convolutional and fully connected layer, and also increased the number of fully connected layers to 3. In these fully connected layers we increased the dropout to 70%.

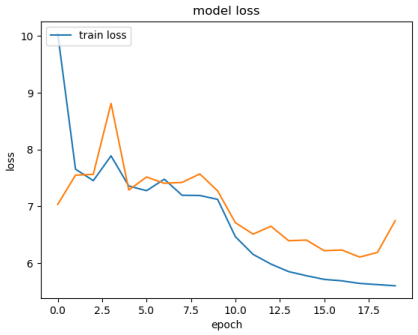


Figure 5: Slightly improved convergence of loss functions(not overfitting to the training data) while tinkering with regularization, but significantly increased loss value.

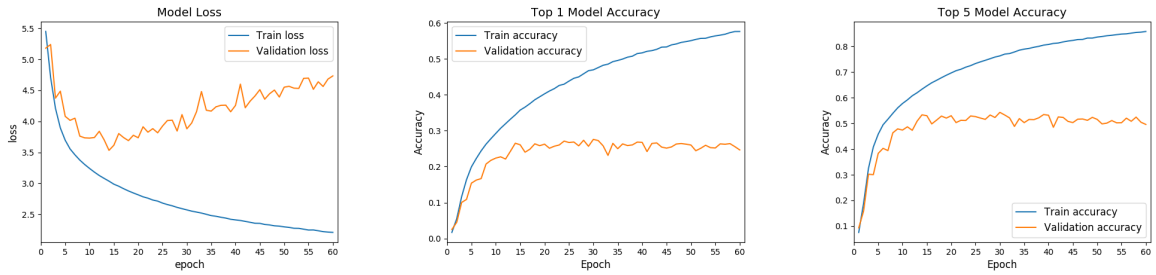
The new model showed a slight improved convergence of the validation loss, but showed an atrocious learning capability due to the high loss values, and thus we found that adding batch normalization, dropout in addition to L_2 regularization caused the network to shift over to the other side of the spectrum, where instead of overfitting, it had problems learning things. We decided to remove dropout that occurs after the fully connected layers, and instead rely on the L_2 regularization and add a decay to the learning rate of the network. The decay rate was set to $1 * 10^{-6}$. The Adam solver was initialized with a learning rate of $1 * 10^{-2}$, and we now let it train for longer, 60 epochs in total.

4.2 Final results

The final results found was achieved with two different models that has the structure presented in table 2. We tested two different approaches: Model E kept the fully connected layers in the end and combined with L2 regularization to help it prevent overfitting to some extent. During our testing it proved a steady learning the 20 first epochs, then it stagnates and begin to overfit. Ideally if we had more time another run with more stricter regularization hopefully would prove more consistent in learning.

ConvNet Configuration	
E	F
Adam, HEI, BN, L2	Adam, HEI, BN, L2
input ($64 \times 64 \times 3$ RGB $image$)	
conv-32, 2x2, ReLu, BN	conv-32, 2x2, ReLu, BN
conv-32, 2x1, ReLu, BN	conv-32, 2x1, ReLu, BN
conv-32, 1x2, ReLu, BN, L2	conv-32, 1x2, ReLu, BN, L2
MaxPooling, 2x2	MaxPooling, 2x2
conv-48, 2x2, ReLu, BN	conv-48, 2x2, ReLu, BN
conv-48, 2x2, ReLu, BN	conv-48, 2x2, ReLu, BN
conv-48, 2x2, ReLu, BN, L2	conv-48, 2x2, ReLu, BN, L2
MaxPooling, 2x2	MaxPooling, 2x2
conv-80, 2x2, ReLu, BN	conv-80, 2x2, ReLu, BN
conv-80, 2x2, ReLu, BN	conv-80, 2x2, ReLu, BN
conv-80, 2x2, ReLu, BN, L2	conv-80, 2x2, ReLu, BN, L2
MaxPooling, 2x2	MaxPooling, 2x2
FC-4096, ReLu, BN, L2	FC-200, SoftMax
FC-4096, ReLu, BN, L2	
FC-4096, ReLu, BN, L2	
FC-200, SoftMax	

Table 2: Configuration of the two models E and F. They differ in the F does not have any fully connected layers leading up to the output, and has a higher degree of L_2 regularization, a lower learning rate, anda higher learning decay decay.).



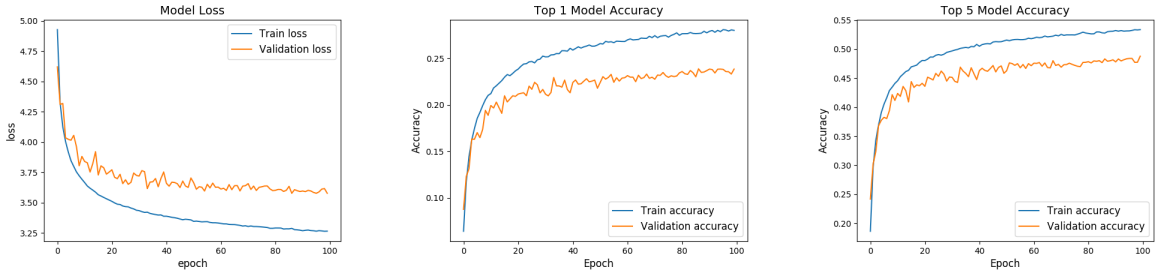
(a) (b) (c)

Figure 6: Training and validation loss of 60 first epochs from model E, specified in 2. (a) Training and validation loss. (b) Training and validation top-1 accuracy. (c) Training and validation top-5 accuracy.

The top-1 validation accuracy was 25% and top-5 validation accuracy 50% for model E. Plot 6 (a) represents training and validation loss, even though there are fluctuations the validation loss initially follows the training loss nicely, the expanding gap in the higher epochs may as mentioned be countered by increasing regularization. Plot 6 (b) illustrates the evolution of accuracy which, in conjunction with the loss, converges towards the maximal accuracy found.

As a final network, model F, we attempted to add some additional reguralization in an attempt to reduce the degree of overfitting to the training data that occured. The L_2 regularization term was increased to $5 * 10^{-3}$, and we opted to remove the

fully connected layers between the last max-pooling layer and the output layer, and the initial learning rate of the Adam algorithm was reduced to $1 * 10^{-3}$. We let this network train for even longer, 100 epochs in total.



(a) (b) (c)

Figure 7: Training and validation loss of the 100 first epochs from model F, specified in 2. (a) Training and validation loss. (b) Training and validation top-1 accuracy. (c) Training and validation top-5 accuracy.

The training of the network shows less signs of overfitting, but the accuracy ended at a top 1 validation accuracy of 24% and a top 5 vaildation accuracy of 48%. It is likely that the network can be further optimized to still reguarlize enough to not overfit, but that would require additional experimentation.

Summing up all tested models during the experiments, the results of the these models can be found in table 3 presented with top-1 accuracy and top-5 accuracy. It's noteworthy that our last two model performed worse than the earlier in terms of accuracy. However, we reason that the last models (E) and (F) had a more consistent behavior during many epochs of training. That is, if model (D) had been trained further is would faster overfit to the training data and generate a non generalizable network. Model (E) and (F) on the other hand manifests consistent learning for a longer duration implying minor tweaking of learning parameters may produce a more optimal network.

Model	**Top-1 val**	**Top-5 val**
A: SGD, RUI	17%	37%
B: Adam, RUI, BN	22%	45%
C: Adam, RUI, Dropout	6%	20%
D: Adam, HEI, Dropout, BN	29%	55%
E: Adam, HEI, BN, L2	25%	50%
F: Adam, HEI, BN, L2	24%	48%

Table 3: Accuracy of models.

5 Discussion

5.1 Evaluation of method

There are several flaws in our method and if we where to recreate the experiment we would use a different base architecture for the network. As seen in figure 1 it is clear that using an VGG structure is not optimal in regard to accuracy, operations and amount of parameters. A residual network structure seems have the highest potential today with regard to accuracy and operation. We would probably start with a basic ResNet structure instead of the VGG-net if we were to redo the experiments. The reason for using a VGG- net was foremost it's easy interpretation and simple implementation. However, the struggle to get an high accuracy due to slow training rate most likely slowed down the process more then the time required to implement and interpret the ResNet structure. With ResNet in place fewer computations would mean more time on our hands to search for optimal parameters.

Ignoring all gray-scale images in training could have an slight impact on our test accuracy. If we were to recreate the method we would try to figure out an optimal way of handling both grayscale images and RGB images in the same method. One approach would be to reduce all images with 3 channels to grayscale images and create a model optimal for only 1 channel images. Another approach would be to use a assemble methodology with one network adapted for grayscale images and one for RGB images. The lack of classification of the grayscale images also resulted in inability to send our predictions for testing which is made externally by providers of ImageNet.

5.2 Evaluation of results

Looking at our result we can see that overfitting is an major issue for our network. All figures in 3 show that the validation loss increases in the final epochs of the models which clearly indicates overfitting. To combat this issue more regularization was added in the model and the adaption shows results of less overfitting seen in figure 6 and 7. From table 3 it is clear that adding regularization and thus increasing the complexity also increased the models validation accuracy.

If we would continue the implementations the next step would be further try to balance the parameters between learning and regularization, allowing to efficiently train over more epochs without overfitting, perhaps by broadly searching a parameter space then progressing to more fine coarse searches, to find optimal parameters of this model.

5.3 Tiny ImageNet State of the art

In [8] it was showed that using a ResNet18 with data augmentation, dropout and Snapshot Ensembleling they could receive an accuracy of top-5 validation of 81.4%, top-1 validation of 60.2% and top-1 test of 53.6%. This shows with a quite simple network, i.e. not so deep, high accuracy can be achieved when utilizing suitable parameters.

[1] achieve an accuracy of approximately 65% top-1 error rate on the validation set and an error on the test set of 25.6%. As of June 12th, 2017 their result was first on the leaderboard for TinyImagenet. They used an ensemble of three models: ResNet101, ResNet152 and DenseNet121. They apply normalization, marginalization and MAP inference in the ensemble architecture.

Our results definitely comes up short comparing against these. A difference lies in number of epochs, we ran 60 which is less than number of epochs used by the implementations mentioned above. But ultimately, the fact our parameters are suboptimal results in our network being prone to overfitting and returns a network that becomes overtrained and generalizes poorly to validation data.

6 Conclusion

Making neural networks is a form of art. With so many architectures to choose from the real craftsmanship lies in tuning and tweaking the network accordingly to work properly. We're satisfied with the project as we've learned a lot about CNNs, even though we were hoping to achieve a higher accuracy.

References

[1] Victor Cheung. Denresnet: Ensembling dense networks and residual networks.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[8] Frank Cipollone Zach Barnes and Tyler Romero. Techniques for image classification on tiny-imagenet.