CIFAR-10 Classification with CNNs

Joshua LaBranche, Jonathan Lapham, Junghwan Kim

April 27, 2020

Contents

1 Abstract

The proposed convolutional neural network (CNN) model takes inspiration from concepts used within AlexNet, Resnet, and VGG networks in order to find an optimal model to classify the CIFAR-10 dataset. We take the overall layering structure of AlexNet, apply the concept of equivalent receptive fields from VGG, and further include a bottleneck from ResNet within the model in an attempt to achieve high accuracy and low loss in classification. This model achieved an accuracy of 75.72% and loss of 1.6359 without any normalization or regularization methods. The model was then modified and tested using a variety of regularization methods in an attempt to improve the accuracy. The most efficient model had an accuracy of 86.8%, using Batch Normalization, Data Augmentation, and a small mini-batch size of 64 to improve training and reduce overfitting. This paper goes into detail on the various steps in achieving this accuracy, along with the intuition in choosing such regularization and tuning methods to improve our accuracy. This project was implemented using Tensorflow and Keras libraries which allows the building, training, and testing of machine learning models.

2 Introduction

As we move towards an age filled with intelligent machines, the demand for efficient and accurate image classification algorithms is growing. One attempt to solve this classification problem of images is the use of CNNs. Early application of neural networks dates back to the 1980s[1]. Then, neural networks were used for recognizing hand-written text. Fourty years later, and some of the same principles are bing applied to machine learning algorithms for the enhancement of feature extraction from images. CNNs are a popular choice for image classification due to the 2-D data structures present in digital images.

The CNN architecture proposed in this paper was configured with computational limitations, ease of implementation for reproducability purposes, and various state-of-the-art techniques in mind. The computational configuration was restricted to the use of an Nvidia 1080 GTX Ti[1] GPU for training of the network. To maintain portability, the network also makes use of the TensorFlow and Keras libraries due to the popularity of these libraries usage throughout the community. Although the GPU is high-end at the time of this report, it is commercially available such that anyone willing to purchase the card and use the same publically available libraires would be able to reproduce the architecture discussed in this paper. Lastly, we take inspiration from well known architectures published throughout the community such as AlexNet [2], VGG [3], and ResNet [4]. Specifically, the baseline was derived from AlexNet with the addition of layers to deepen the network, the concept of equivalent receptive fields for input layers from VGG, and a bottleneck technique utilized in the ResNet architecture to reduce the computational expense of the algorithm through reducing the dimensionality of inputs. A few additonal commonly used techniques used for development include a rectifier as the activation function,

[1]This card is an 11 Gb DDR5X card with 3584 CUDA cores (TensorFlow library makes use of the CUDA cores). https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080-ti/

data augmentation, a constant and varying rate of drop-out, weight decay, and batch normalization.

Training and performance of the CNN was performed on the also publically available CIFAR-10 dataset[5]. Since many CNNs are trained and tested on the CIFAR-10 and CIFAR-100 dataset [2][3][4], it was decided to focus on the CIFAR-10 dataset due to its populaity. There was no reason not to use the CIFAR-100 dataset, but this CNN was restricted to the CIFAR-10 dataset in this paper. CIFAR is an acronynm for the Canadaian Insistute for Advanced Research, and the CIFAR-10 dataset specifically consists of 60,000 32x32 pixel images divided into ten categories. The ten categories each consist of 6,000 images divided into the following: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The CIFAR-10 dataset is a subset of the even larger 80 Million Tiny Images dataset[2].

The central motivation behind this architecture was a high performing algorithm that can be reproduced without many limitations, but one that utilizes techniques from various previously published research papers.

3 Background

As previously mentioned in section 2, most of the architecture presented is derived from AlexNet, VGG, and ResNet. Early intellectual inspiration is drawn from Bishop's text[6] and [7], with more advanced applications invoked through the aformentioned TensorFlow and Keras libraries.

A few other areas of research were studied. Since an early approach to improving the baseline model was to deepen the model, it was found that the technique presented in [8] can help to improve performance by connecting all layers of a network in a feed-forward fashion. This idea was bulit upon the bottlenecking technique used in ResNet and this paper. Also, the Incepton CNN presented in [9] was studied, but was deemed to surpass the ease of reproducability threshold and not utilized in this architecture. The purpose of the Inception CNN is to increase the depth and width of a CNN while maintaining constant computational resource needs and thus was deemed useful if these properties of the architecture are desirable. This concept was expanded upon and an error rate of 1.00% was reported in [10] for the CIFAR-10 dataset.

4 Approach

The CNN models implemented had been inspired by the three successful CNNs of the AlexNet, VGG net, and ResNet. Our models utilized Stochastic Gradient Descent (SGD) as the CIFAR-10 is a large training set. Additionally, SGD is good with generalization of large datasets, and is commonly used in similar state-of-the-art works [2][3][4]. Within SGD, the learning rate is set to 0.001 and the momentum is set to 0.9. We use a constant mini-batch size of 256 within our models in order to reduce computation time. As our objective is to use supervised learning to classify the CIFAR-10 dataset, we used Categorical Cross Entropy (CCE) in every model as our cost function to evaluate the model at every epoch. The metric that was used to evaluate every model was the

[2]http://groups.csail.mit.edu/vision/TinyImages/

Table 1: Comparison of Proposed Model Architectures

Untuned Model A	**Untuned Model B**	**Untuned Model C**	**Untuned Model D**
2D Conv (256, (3x3)) 2D Conv (256, (3x3)) 2D Conv (256, (3x3)) MaxPool((2x2), stride=2)	2D Conv (64, (3x3)) 2D Conv (64, (3x3)) 2D Conv (64, (3x3)) MaxPool((2x2), stride=2)	2D Conv (256, (3x3)) 2D Conv (256, (3x3)) 2D Conv (256, (3x3)) MaxPool((2x2), stride=2)	2D Conv (256, (3x3)) 2D Conv (256, (3x3)) MaxPool((2x2), stride=2)
2D Conv (128, (3x3)) 2D Conv (128, (3x3)) MaxPool((2x2), stride=2)	2D Conv (64, (3x3)) 2D Conv (64, (3x3)) MaxPool((2x2), stride=1)	2D Conv (256, (3x3)) 2D Conv (256, (3x3)) MaxPool((2x2), stride=2)	2D Conv (256, (3x3)) MaxPool((2x2), stride=1)
2D Conv (64, (3x3)) 2D Conv (64, (3x3)) MaxPool((2x2), stride=2)	2D Conv (32, (3x3)) MaxPool((2x2), stride=2)	2D Conv (256, (1x1)) MaxPool((2x2), stride=1)	2D Conv (256, (1x1)) MaxPool((2x2), stride=1)
NO LAYER	2D Conv (64, (3x3)) 2D Conv (64, (3x3)) MaxPool((2x2), stride=1)	2D Conv (64, (3x3)) MaxPool((2x2), stride=1)	2D Conv (64, (3x3)) MaxPool((2x2), stride=1)
NO LAYER	NO LAYER	2D Conv (256, (1x1)) MaxPool((2x2), stride=2)	2D Conv (256, (1x1)) MaxPool((2x2), stride=2)
Flatten()	Flatten()	Flatten()	Flatten()
Dense(512)	Dense(512)	Dense(512)	Dense(512)
Dense(512)	Dense(512)	Dense(512)	Dense(512)
Dense(10)	Dense(10)	Dense(10)	Dense(10)

accuracy. The four models generated during the design phase are shown in Table 1.

Model A is the baseline model that has been inspired by the layering architecture of the AlexNet and VGG networks. This model aims to capture a 7x7 receptive field of the input image by stacking three 3x3 convolutional layers in a row. The next layer has a receptive field of 5x5, with two 3x3 convolutional layers, along with the following layer. Model A then ends with two dense layers, similar to the AlexNet. The equivalent receptive field is a concept taken from the VGG network which aims to reduce computation time by lowering the number of parameters needed to acquire the same amount of information from an image. It can also be seen that the number of filters decreases by a factor of two as the network gets deeper, which has also been done in the AlexNet.

Model B is the first model which introduces the bottleneck within the architecture. The bottleneck aims to restrict the network in order for the network to learn key features of the images. Having the bottleneck also reduces the complexity of the model, making it less prone to overfitting. This was done by lowering the filter number between the layers to at least a factor of two smaller. Furthermore, the network's width was reduced to 64 for each layer, with 32 at the bottleneck, in order to determine how a smaller width will impact accuracy within this architecture.

Model C is an improvement to Model B in that the network is now wider and deeper. The bottleneck had also been improved by greatly reducing the width at one point and introducing 1x1 convolutions. This idea had been taken from ResNet, as 1x1 convolutions can obtain cross channel information. Additionally, this bottleneck still is capable of learning key features while also remaining efficient in terms of parameter count, improving computation time. The width of the network was increased to 256 as a wider network provided a better accuracy compared to the previous model. As the CIFAR-10 dataset contains small 32x32x3 images, features within those images would be best captured by using smaller kernel sizes.

This led to Model D. Within Model D, the convolutional layers besides the bottleneck layers were reduced by one 3x3 convolution, making the first layer a 5x5 convolutional layer and the next layer a 3x3 convolutional layer. The strides for the Max Pooling operations were also modified in order to coordinate the image size appropriately with the kernel sizes. Layer one has the largest kernel as it deals with a 32x32x3 image. The following layers take information from the now 16x16x256 information, which is why the kernel size for the convolutions were also reduced to have a chance of obtaining feature information. Max Pooling of stride 1 is used in effort to not reduce the image size too much to mask small features. Additionally, the idea of "deeper is better" was the main thought of this model, which is why this model has more layers than the models before it.

4.1 Model Improvements

Model D was chosen for regularization and tuning purposes due to its high accuracy and low loss compared to the other models. It is very common to apply regularization techniques as many models with high parameter count can display overfitting. Overfitting is when the model is very good at classifying the training set, but is very poor at classifying the validation and test sets. This is due to the model not being capable of generalizing, making it poor at classifying new input images. The improvement stage involves first sweeping the model through just weight decay and dropout. Weight decay involves penalizing large weight values with respect to the given lambda value while dropout involves randomly dropping a fraction of nodes within a given layer. Once learning curves were found from those methods, the most efficient model and regularization method will then undergo data augmentation in an attempt to improve the accuracy. Also Batch Normalization had also been explored in order to improve the computational time of the training process and increase the accuracy. The combinations of these methods with the model is further explored within section 5.

Model Name	**Accuracy**	**Loss**	**Epochs**	**Time (m)**	**Batch Size**	**Params (1e6)**
Untuned Model A	70.62	1.5422	50	18.33	256	2.530
Untuned Model B	69.33	2.0220	50	4.898	256	2.580
Untuned Model C	73.86	1.8444	50	14.00	256	4.960
Untuned Model D	**75.72**	**1.6359**	**50**	**14.00**	**256**	**10.07**
Weight Decay	73.96	3.2	150	41.5	256	10.07
Drop	82.87	0.5362	150	48.33	256	10.07
Drop Scaled Aug	80.55	0.5627	300	111.41	256	10.07
Drop Aug	85.2	0.445	300	110.85	256	10.07
Drop Aug Batch Norm	77.16	0.9497	400	369.07	256	10.07
Aug Batch Norm	83.67	0.59409	100	74.58	256	10.07
Aug Batch Norm	**86.8**	**0.558**	**100**	**84.63**	**64**	**10.07**

Table 2: Table of results showing the comparison of methods in the first column. All methods after Model D, utilize Model D as their baseline, as described in section 4. Key: "Drop" refers to the drop-out technique, "Drop Scaled" refers to the same dropout technique coupled with an increasing probability of dropping inputs to sucessive layers, "Aug" refers to the data augmentation technique, and "Batch Norm" refers to batch normalization.

5 Results

In figure 1 the results from Model D are shown. This model was deemed the best performing out of the four baseline models. As shown in the plots, the model is overfitting and is in need of regularization methods to improve the accuracy. This can be seen from the deviation in loss of the test data to the training data results. Couple this with the asymptotic error rate of 24.28% for the test data and this is an indication of model overfitting. This suggests that techniques to combat overfitting will help improve performance.

The first technique used to combat overfitting was the use of drop-out. Here sucessive layers receive a reduced number of inputs, where the reduction size is a function of the probability of drop-out. Effectively this reduces the dimensionality of successive layers and allows for the model to fit this lower dimensional space, and to not overfit the feature space. The depth of the network is a factor into the effectiveness of drop-out. If the network is larger and many features are being extracted, then drop-out can significantly improve the performance loss

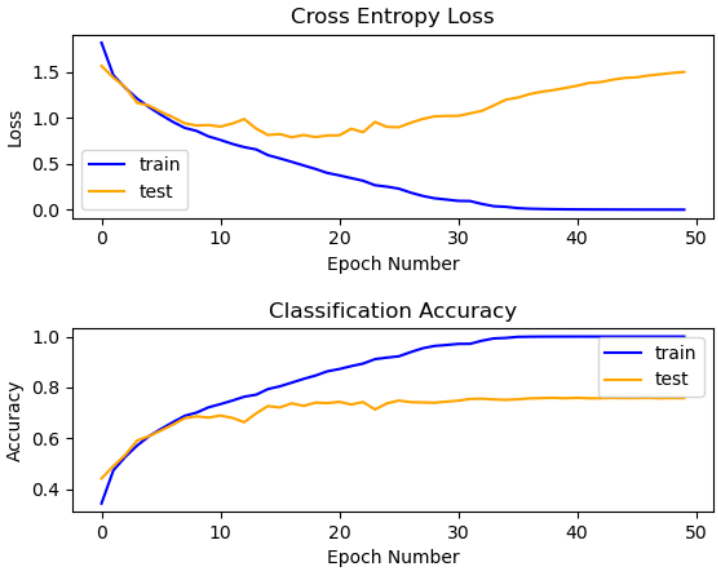

Figure 1: Model D Results

casued by model overfitting. A constant drop-out rate of 0.2 is used.

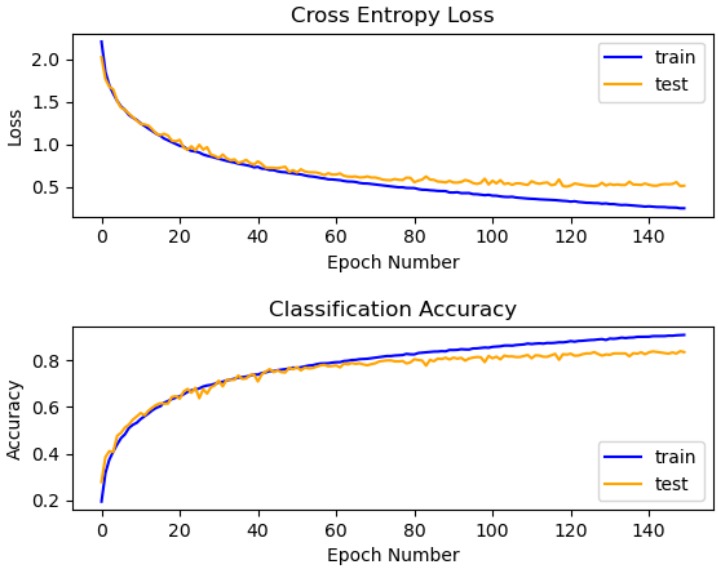

Figure 2: Best Model Results with Drop-Out

In figure 2 we can see an improvement in the deviation of the test data from the training data. However, the loss approaches a minimum above 0 for the 150

epochs of batch size 256 simulated, and an error rate of about 85%. This would suggest an underperfoming architecture, but with reduced overfitting.

Another attempt to combat over fitting is a technique known as weight decay. Weight decay scales the sum of the squares of the output parameter weights and adds this to the loss function. What this does is if we have dominate parameters in the feature space, their large magnitude will increase the magnitude of the loss. Weight decay was applied to Model D. Here, the sum of squares is scaled by a factor of $\lambda = 0.001$.

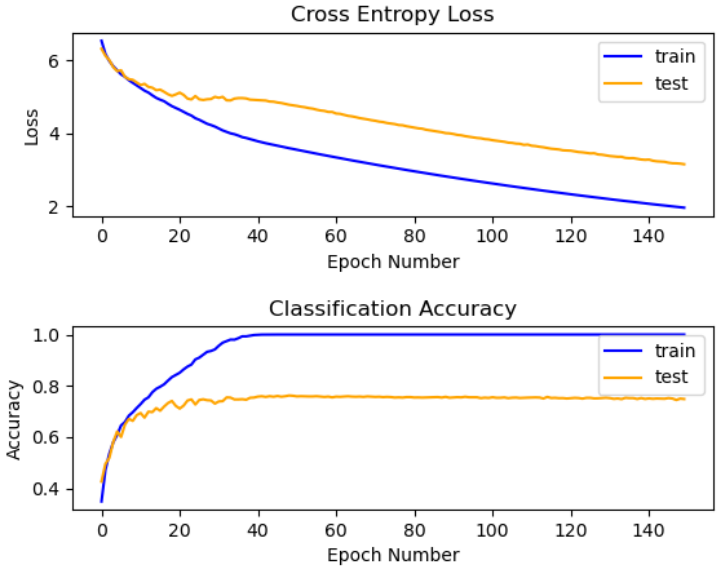

Figure 3: Best Model Results with Weight Decay

In figure 3 the performance of Model D with weight decay is actually worse than without, so this technique is not utilized any further for this implementation. Drop-out is chosen as the way to go and adding the technique of scaling the drop-out factor. This has the effect that as the network becomes deeper the need for increased dropout grows to combat overfitting. In this sense the probability of drop-out is not constant but instead increases with each successive layer. This implementation started with a dropout rate of 0.2 and increases .05 per layer until it reaches 0.5 before the final output layer.

Also introduced is the common technique of data augmentation. Here, the items in the dataset are augmented such that the pixels are modified to create dependent items in the dataset. The reason this was introduced is becasue CNNs are invariant to translations[6]. Effectively, the CNN has more data to process, but since the CNNs are invariant to translations it will produce the same result for translated images. In figure 4 it is seen that data augmentation coupled with scaled drop-out did not improve performance from a traditonal drop-out method. This could possibly be attributed to the constant drop-out factor being a more consistent method to overcome overfitting.

Since the scaling of the drop-out rate decreased performance, this was re-

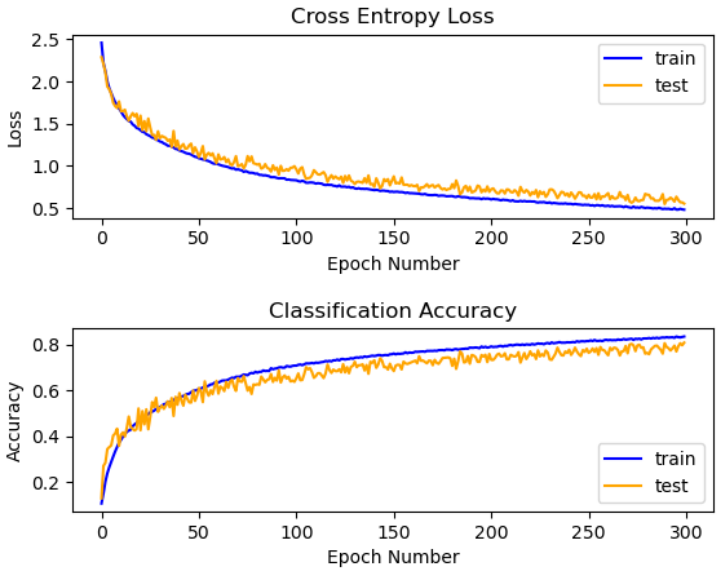

Figure 4: Best Model Results with Scaled Drop-Out and Data Augmentation

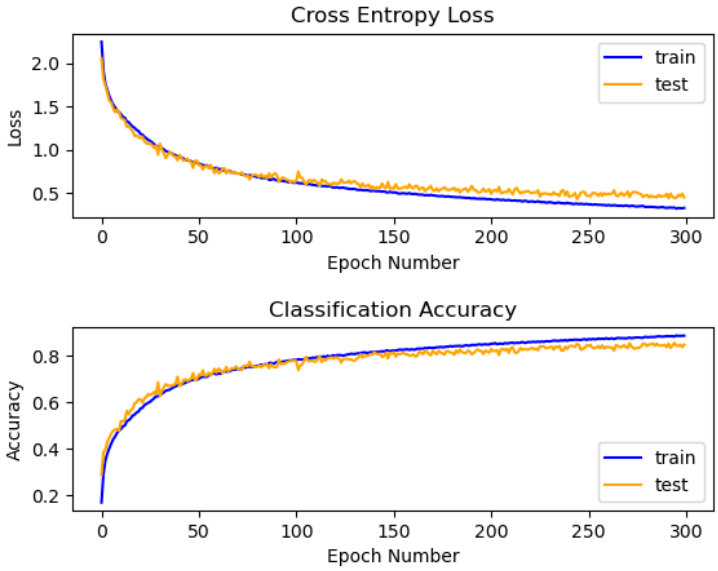

Figure 5: Best Model Results with Drop-Out and Data Augmentation

moved. In figure 5 we can see the best perfromance thus far. The test data achieves an error rate of 14.8% and a loss of 0.445, which is the best performance in terms of minimal loss recorded in these experiments.

Adding batch normalization before each layer of the network was imple-

mented. Batch normalization aims to address a well known issue of covariant drift[11]. Along with constant drop-out and data augmentation, the performance for batch normalization worsened. A thought was then to remove drop-out and only use batch normalization. This did not improve performance over the drop-out with data augmentation case, until the batch size was reduced from 256 to 64.

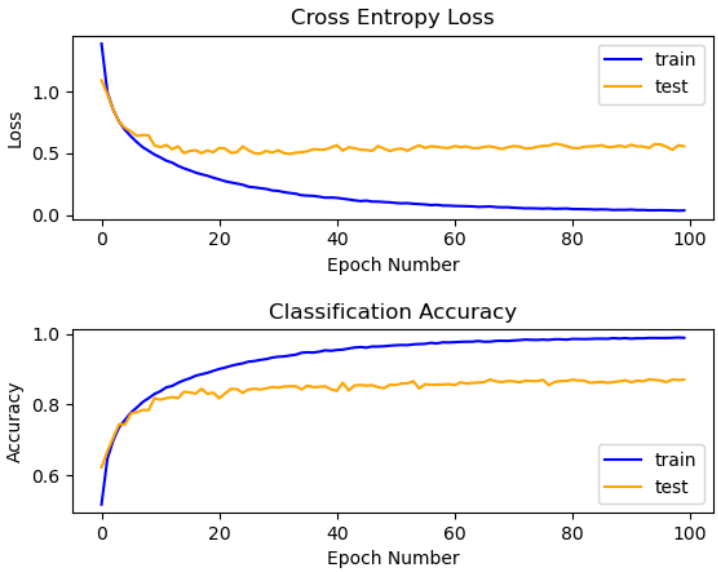

Figure 6: Best Model Results for 64 Image Batch Size with Data Augmentation and Batch Normalization

Not only does this approach give the best performance in terms of error rate. The asymptotice behavior of the test data's measured loss shown in figure 6 indicates that there is room for improvement upon this architecture.

5.1 Discussion

What is key to take away from section 5 is that the addition of drop-out and data augmentation can significantly improve performance by over 10%, however the lowest error rate was achieved without drop-out but instead batch normalization. We propose that should someone continue this work moving forward, they should utilize the batch normalization and data augmentation approach and address the deviation in loss between the testing and training data.

6 Conclusion

Throughout the paper, a mix of AlexNet, VGG, and ResNet technqiues were utilized to construct an easy to set-up, well performing, and original CNN for image classification of the CIFAR-10 dataset. Four baseline models were tested and developed without any regulariztion or normalization methods. It was found that the deepest of the four models, Model D, performed the best and was selected to be improved upon. The intention was to push the performance of the algorithm past this baseline result and techniques were added to do just that. It was found that Model D coupled with data augmentation and batch normalization performed the best with an error rate of 13.2% and a loss of 0.558 in a total simultion time of roughly eighty-five minutes.

It should be highlighted that what was found for this particular configuration is that the addition of data augmentation and drop-out performs slightly worse in terms of error rate at 14.8%, but shows in improvement in loss at 0.445. However, since the error between the training curves and testing curves is much smaller for this case compared to data augmentation and batch normalization, the model that shows the most promise is the architecure that uses data augmentation and batch normalization.[3]

[3]JSL performed research and wrote the final paper, JTL performed research and wrote most of the code, JK perfromed research and tested/verified components of the code

References

[1] K. Fukushima, "Neocognitron: A self-organzing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Cybernetics*, 1980.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[7] Z. Liang, A. Powell, I. Ersoy, M. Poostchi, K. Silamut, K. Palaniappan, P. Guo, M. A. Hossain, A. Sameer, R. J. Maude, J. X. Huang, S. Jaeger, and G. Thoma, "Cnn-based image analysis for malaria diagnosis," pp. 493–496, 2016.

[8] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," pp. 2261–2269, 2017.

[9] C. Szegedy, Wei Liu, and Yangqing Jia, "Going deeper with convolutions," pp. 1–9, 2015.

[10] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *CoRR*, 2018.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.