Deeper Networks for Image Classification

Author: Ostap Orishko
Student ID: 190566678

1. Introduction

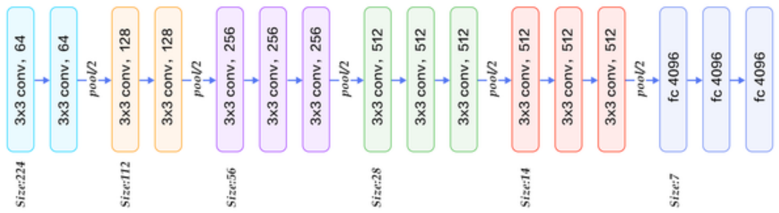
In recent years, Deep Neural Networks have become increasingly popular in solving a wide range of computer vision tasks.
In this paper, I focus on investigating two networks - VGG and ResNet and their effectiveness for image classification on MNIST and CIFAR10 datasets. I implemented each network in using PyTorch and designed it so it is very scalable in terms of choosing the network's depth/flavour.

2. Critical Analysis and Related Work

2.1 VGG

VGG was developed by Simonyan at el. [1] in a paper that addresses the depth on a Convolutional Neural Network with a thorough analysis on ImageNet. The paper focuses on fixing all other parameters of the architecture and increasing the model's depth, by adding a larger amount of 3x3 convolutional layers on top of each other. This paper goes hand in hand with the latest research at the time, that was converging on the idea that deeper networks lead to better performance, like Goodfellow et al. [2], who used deep ConvNets for street number recognition and GoogLeNet by Szegedy et al. [3] that won ILSVRC-2014 using a very complex deep CNN.

One of the main findings of the VGG paper was that the classification error on ImageNet decreased with the increased ConvNet depth - from 11 layers to 19 layers in a network. Another impressive result is that the number of parameters was not greater than in the previously proposed shallow networks.



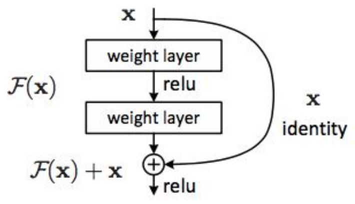
The image above shows a popular network, originating from the paper, which is VGG16. The present convolutional layers gradually reduce the size of the image of the size 224x224 to one of size 7x7. The major advantage of using a stack of layers with smaller kernel size instead of a single layer with a large kernel size is that the former allows to incorporate multiple ReLU making the decision function more discriminative.

Going forward it was important to outline some details in the network's implementation. VGG used mini-batch SGD with batch size of 256 and momentum of 0.9. The regularizer consisted of L2 weight decay with penalty multiplier 0.0005 and two Dropout layers after the first two Fully

Connected layers set to 0.5. Another important detail is the fact they used a learning rate scheduler, with initial Learning Rate of 0.01.

2.2 ResNet

He et al. [4] introduced ResNet as a solution for effectively training increasingly deeper neural networks. The paper addresses the problem of degradation of accuracy found in very deep networks. In order to do so they construct a residual network consisting of the blocks as one shown on the right. Instead of hoping each few stacked layers directly fit a desired underlying mapping, we can now let these layers fit a residual mapping instead.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

A residual network makes a lot of sense since we expect that the deeper network should have training error no greater than its shallower counterpart, if we were to simply add layers that map identity only to a shallower network. Therefore if the identity mappings are optimal, it is easier to map the residual to 0, rather than fitting the underlying mapping identity.

The paper draws direct comparison to VGG - the baseline plain network they construct has fewer filters and lower complexity than VGG. As an example, in comparison to VGG19 has 19.6 billion FLOPs, while both the plain baseline and the residual network with 34 layers have 3.6 billion FLOPs.
Another addition to network architecture the paper provides is including Batch Normalization, which ensures that forward propagated signals have non-zero variance, ensuring that gradient vanishing problem is dealt with in both directions.

In order to implement ResNet, I consulted their adaptation of the network to CIFAR10 dataset. There, they adopt 3x3 convolutions everywhere and include a global average pooling layer. Their hyperparameters include batch size 128 over 200 epochs, with weight decay of 0.0001 and momentum 0.9. They employ a learning scheduler, with initial learning rate of 0.1.
Several data augmentation tricks are employed, including padding 4 pixels on each side and cropping randomly to 32x32 as well as using horizontal flip.

3. Model Description and Evaluation

I focused on investigating VGG-13 and -16 and ResNet-20 and ResNet-44. It is possible to pick the architecture rather easily, but because of page limitation and time constraints I settled for investigating 2 depths for each of the networks.

For VGG, I would like to point out that VGG was not designed for a dataset with such a small input size like MNIST, which has images of the size 28x28, therefore I would like to discuss how I dealt with this first.
I considered two options:

1) Resize MNIST to the size of an ImageNet image, which is 224x224 and make no changes to the network.
2) Resize MNIST to the minimum allowed size for VGG and change the network accordingly.

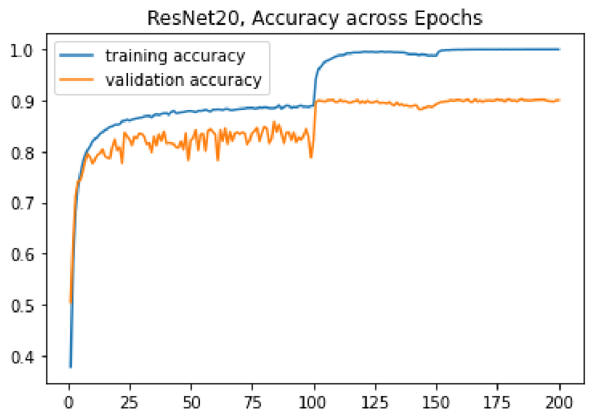
Initially I tried both options, however, on my machine 1 epoch on a fully sized VGG network took 22 minutes, and knowing that I would have to make multiple iterations of hyperparameters, I didn't go with this option.
I ended up resizing MNIST to 32x32, which means that the convolutional layers will gradually reduce image size to be 1x1 with 512 filters. Therefore, when going into fully connected layers, my input size would be 512x1x1 instead of the original 512x7x7. Hence I conducted a number of experiments choosing the output size. My final VGG models include the same number of layers as the paper, with convolutional layers unchanged and fully connected layers as follows:

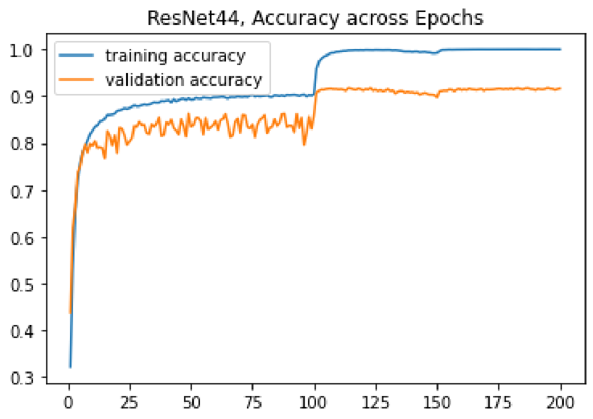
FC(512, 128)-ReLU-Dropout(0.2)-FC(128,128)-ReLU-Dropout(0.2)-FC(128,10)

I will be elaborating on my choices in the next section.

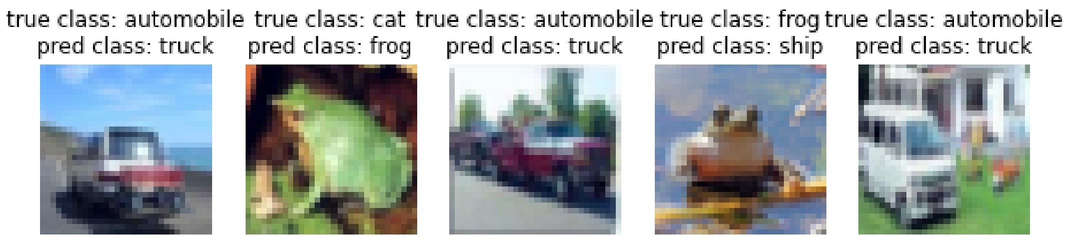
For ResNet, I tried to stay as true to the original as possible. I didn't resize MNIST for ResNet and used random rotations of +-10 degrees for data augmentation. I found that batch size of 32 performed better than the recommended 128 or 256 batch size over 200 epochs on CIFAR.

3.2 CIFAR-10 ResNet evaluation





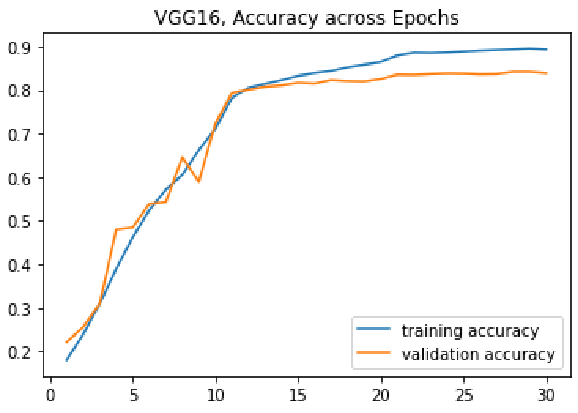
Above one can observe the accuracy change for ResNets over 200 epochs. The initial learning rate was 0.1, dropping to 0.01 after 100 epoch and to 0.001 at 150 epochs. It is possible to observe a sharp increase in accuracy at these milestones, at 100 epochs in particular.
Out of all the models I investigated, ResNet44 had the highest accuracy, and I attached its confusion matrix for CIFAR-10 in the Appendix. According to it, cat and dog the most common examples of misclassification in the model. Below are some examples of misclassified images by my ResNet-44 model.

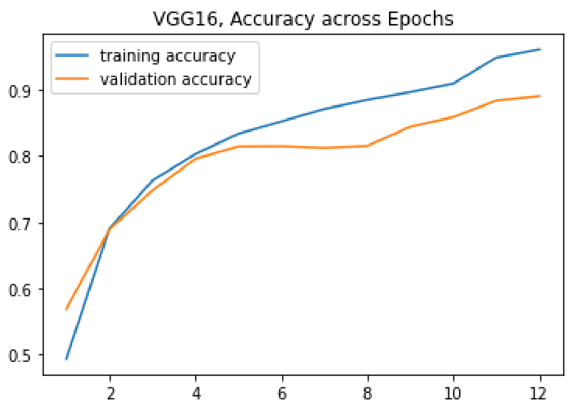


The complexity of CIFAR-10 dataset is the variety of positions and backgrounds as well as the limited amount of detail available in pictures. Automobile and truck have very similar shapes from a variety of angles and the 4th image could be confused with a ship because of the background resembling water and sky and a frog facing directly forward.

3.3 CIFAR-10 VGG Evaluation

For my VGG networks I found that it is unnecessary to train them for a large number of epochs to get top performance. I trained them over 20-30 epochs, but the top performances were obtained by stopping at around 12 epochs and starting at 0.01 learning rate, instead of 0.1 I initially used.





(a) (b)

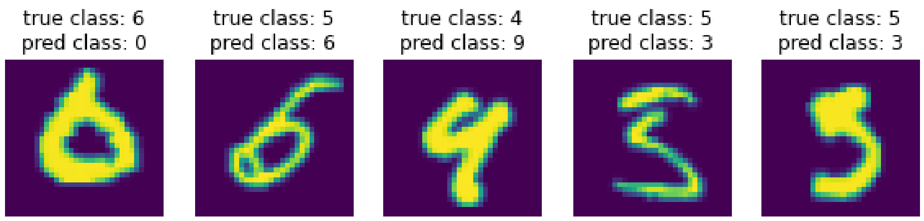
(a) Training VGG16 over 30 epochs with initial lr=0.1; 0.01 after 10 epochs, 0.001 after 20
(b) Training VGG16 over 12 epochs with initial lr=0.01; 0.001 after 10 epochs

This led to a model with 88.35% accuracy after just 12 epochs, beating my previous best results by over 1%. The same behaviour could be observed in VGG13, I attached the appropriate plots in the Appendix.

3.3 Evaluating on MNIST

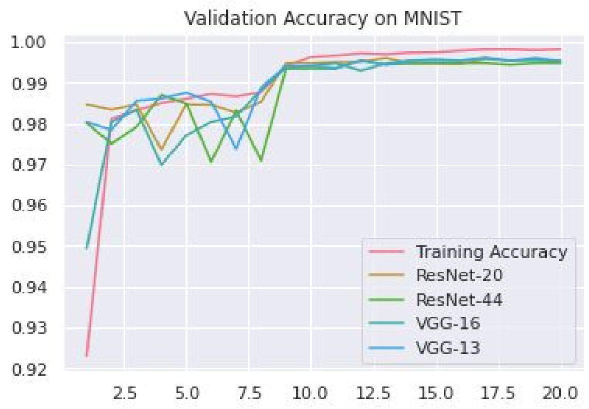
All of the discussion that has been included over CIFAR-10 was easily translated to MNIST, since it is a less complex dataset. The only difference in the datasets is that it's not possible to apply horizontal flips as data augmentation for MNIST, therefore I used very slight random rotations in my training.

Each of the networks does excellently on MNIST with very little room for error. Below are some of common misclassification examples, which occur in all of the networks I trained.



The example above shows that some of the existing images in MNIST are extremely difficult to classify correctly. The most commonly misclassified pairs of classes are 3&5 and 4&9.

The best results on MNIST were obtained by fixing the same optimiser and batch size for all 4 networks over 20 epochs. Batch size = 128, with SGD optimizer and initial learning rate of 0.1, with a schedule to decrease to 0.01 after 8 epochs and to 0.001 after 15 showed the best results for each of the networks I considered. Here is a graph of their validation accuracies against training accuracy, showing how closely each of the networks converges by the 20th epoch.



I am attaching the confusion matrix of the best performing ResNet-20 in the appendix.

4. Experiments and Model Improvements

4.1 Overall Results

Best classification accuracy for each model over CIFAR-10 and MNIST.

Model Name	CIFAR-10	MNIST
VGG-13	88.23	99.71
VGG-16	88.35	99.69
ResNet-20	89.87	**99.72**
ResNet-44	**91.52**	99.65

4.2 VGG Experiments

While trying to get the best performance out of VGG, I tried a variety of settings for the optimizer, the best result ending up the one borrowed from the ResNet architectures.

Optimiser parameters, VGG16	Acc. on CIFAR-10
Adam, lr = 0.1	83.44
SGD, lr = 0.1	83.87
SGD, lr = 0.1, momt= 0.9, wd = 0.0001	84.08
SGD, lr = 0.1, momt=0.9, wd=0.0001 + decreasing on schedule	**85.14**

Where lr is the learning rate, momt is momentum and wd is weight decay. I varied the scheduler by epoch, by each time it multiplies the current learning rate by 0.1 in all cases.
This was critical for my results across both datasets.
Next, I varied the batch size and settled on the sizes of my fully connected layers to arrive at the best model.

Batch size, VGG16	Acc. on CIFAR-10
32	85.14
64	85.19
100	**86.26**
128	86.01

Fully connected layers	Acc. on CIFAR-10
512-4096-4096-10, dr 0.5	84.70
512-128-128-10, dr 0.5	87.22
512-64-64-10, dr 0.5	86.86
512-128-128-10, dr 0.2	**88.35**

Where 512-128-128-10 is
FC(512,128)-ReLU-Dropout(0.2)-FC(128,128)-ReLU-Dropout(0.2)-FC(128,10).
I also performed many iterations to arrive at 0.2 dropout, but I am not including all the results.
Lastly, I found that including Batch Normalization in VGG for CIFAR led to superior performance.

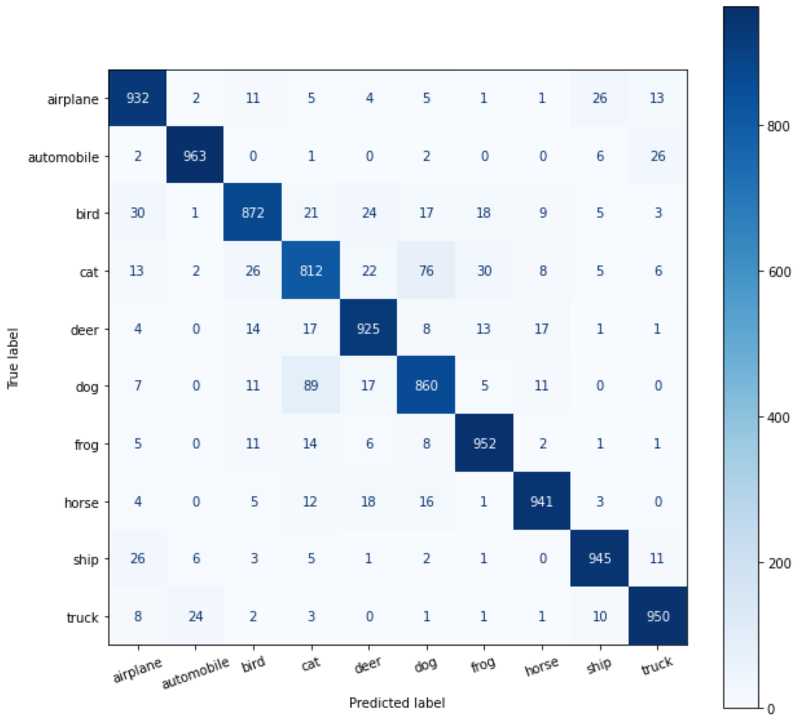
5. Conclusion

The majority of my time was spent tuning the networks for CIFAR-10, since it proved to be a lot more difficult, therefore when approaching MNIST I already had all the answers.
My networks didn't match the performance quoted in papers, hence there must be some room for improvement. One of the things I didn't include was using 4 pixel padding with random cropping, as well as a number of other tricks researchers use to boost the network's performance even more.

References

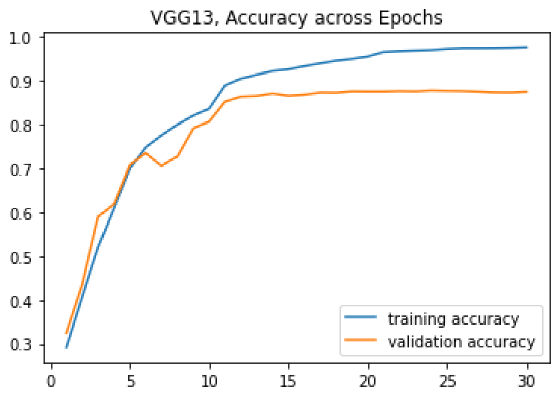
[1] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
[2] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In Proc. ICLR, 2014.
[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.
[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016

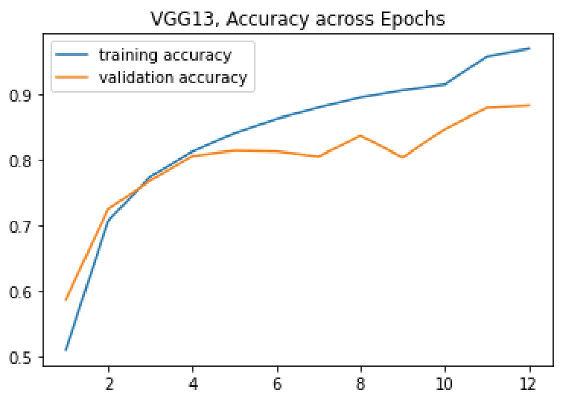
Appendix



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	932	2	11	5	4	5	1	1	26	13
automobile	2	963	0	1	0	2	0	0	6	26
bird	30	1	872	21	24	17	18	9	5	3
cat	13	2	26	812	22	76	30	8	5	6
deer	4	0	14	17	925	8	13	17	1	1
dog	7	0	11	89	17	860	5	11	0	0
frog	5	0	11	14	6	8	952	2	1	1
horse	4	0	5	12	18	16	1	941	3	0
ship	26	6	3	5	1	2	1	0	945	11
truck	8	24	2	3	0	1	1	1	10	950

ResNet-44 confusion Matrix for CIFAR-10

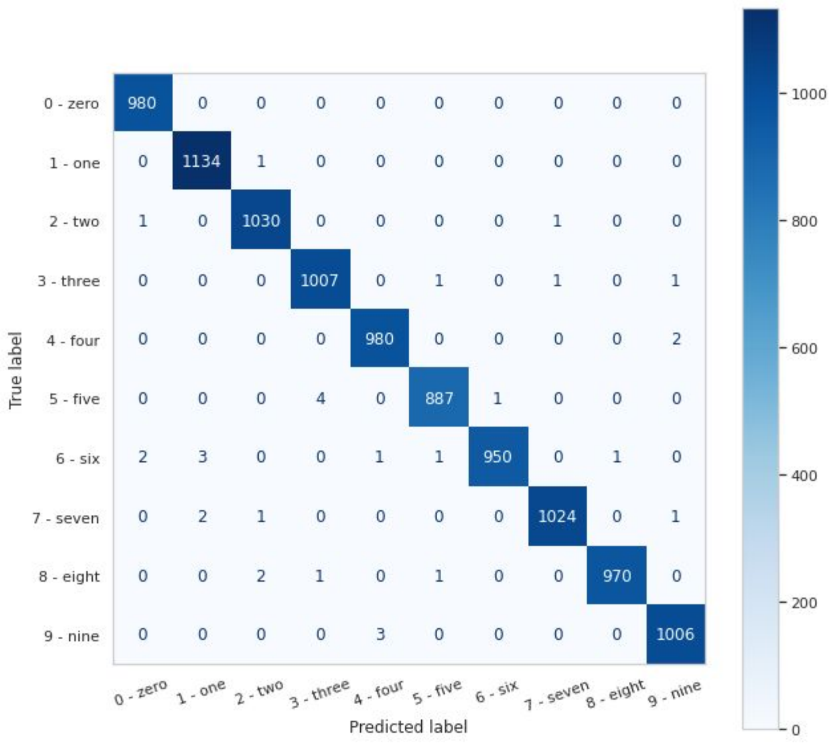




(a) (b)

a. VGG13 over 30 epochs with initial lr=0.1; 0.01 after 10 epochs, 0.001 after 20

b. VGG13 over 12 epochs with initial lr=0.01; 0.001 after 10 epochs



Confusion matrix for best performing ResNet-20 on MNIST