# COS 424, Final Project

**Matt Harrington**
BSE COS '19
mah5@princeton.edu

**Brahmnoor Chawla**
BSE COS '21
bchawla@princeton.edu

**Carla Haignere**
BSE '20
haignere@princeton.edu

**Joel Ritossa**
BSE COS '20
jritossa@princeton.edu

## Abstract

Image classification is a well-motivated, and thoroughly-studied, field in machine learning: sporting a multitude of practical purpose in real-world applications. One such example is Google Lens. The most prominent models for image classifications, in recent years, have been Convolutional Neural Networks (CNNs). These networks mimic human perception by using convolutional filter layers to extract features from groups of nearby pixels.

In this work, we consider three popular CNN architectures: VGG-16, ResNet(50), and the GoogLe / Inception Net. We construct these models, then train them on Google's Landmark dataset from `kaggle.com`. By analyzing the accuracy of these different models on a range of sparse class sample-sizes, we can better understand the effects number of samples per class has on classification accuracy.

## 1 Introduction

In this paper, we explore the effects of sample sparsity across three popular CNNs in order to deduce an optimal model for classifying the Google Landmarks Dataset. We start by exploring the distribution of the landmark classes in our dataset to better understand how to reduce our training data such that we retain a comparable indicator of the accuracy of our model on the entire set. Using this information, we subdivide the data into percentile ranges corresponding to the number of samples per class. We make sure to correlate the distribution of the number of classes in each percentile range with the number of classes we use in our reduced training data.

Next, we build and train three popular image classification CNNs on our sub-divided training data and plot the accuracy and loss for each epoch. Through this analysis, we are able to see which model performs the best on each of the sparsity ranges, and thus observe how sparsity effects our models accuracy and loss. Once we have our optimal model, we work on optimizing its performance across the sample size spectrum by cross-validating the optimal number of epochs in the hopes of deducing a singular model that classifies with significant accuracy across the full spectrum of sparsity.

## 2 Related Work

Computer Vision is an ever-growing discipline with wide-spread impact. Fundamentally, CV concerns any intention to extract information from pixels. This information is often classified as geometric, semantic, or both.

CV techniques are powerful tools, but difficult to leverage effectively: the difference between an image's representation to a human and a machine (*referred to as the semantic gap*) makes pixel data challenging for computers to interpret.

While “Old School” Computer Vision leveraged computational techniques like SIFT features and edge-detection, a new wave of Computer Vision has been born from Machine Learning approaches. Herein, we compare the performance of some of the most innovative Convolution Neural Network (CNN) architectures of their times.

Breakthroughs in CNN architecture design often target reducing the number of trainable parameters in a given model. Not only do more parameters tend to lead towards model over-fitting, the surplus of variables bloats the training time. Researchers on the GoogLeNet / Inception Network cite the following:

> *[...] in a deep vision network, if two convolution layers are chained, any uniform increase in the number of their filters results in a quadratic increase of computation. If the added capacity is used inefficiently (for example, if most weights end up to be close to zero), then a lot of computation is wasted.* [3]

Generalized, reducing a single $n \times n$ convolution into two smaller convolutions of size $n \times 1$, then $1 \times n$. This reduces the number of trainable variables from $n^2$ to $2n$ (see Figure 7 in the appendix for a depiction. There, $n = 3$)[4].

## 3 Models

### 3.1 VGG-16

Published initially in 2014, VGG-16 is seen the first “very deep” architecture. The model consists of a series of 16 to 19 fully-connected convolutional layers (Figure 1). Though VGG employs only small filters ($3 \times 3$), the model totals over 135 millions parameters.

With this large of a parameter space, VGG understandably takes a long time to train. The initial ICLR conference paper claims the following:

> *On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2-3 weeks depending on the architecture.*

While built for the ImageNet challenge, authors Simoyan and Zisserman illustrate in their appendix how the trained architecture generalizes to “*a wide range of tasks and datasets*” [2].

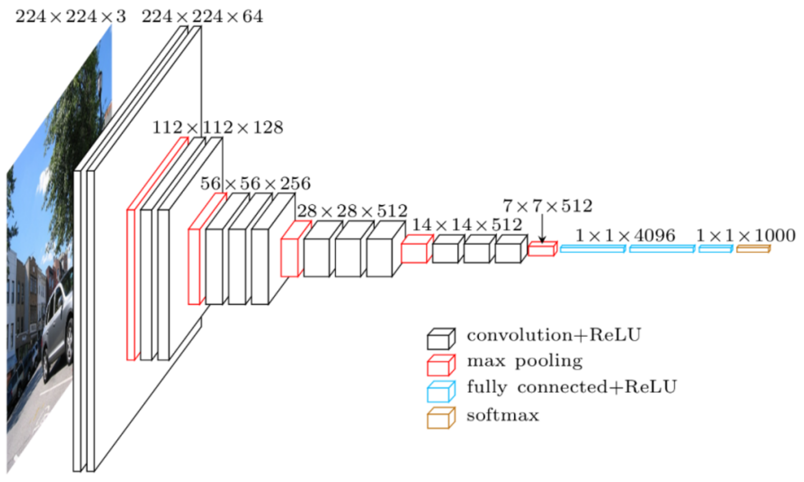


Figure 1: A depiction of the VGG-16 architecture.

### 3.2 ResNet

ResNet (2015) leverages a series of 16 “residual blocks”: groupings of layers that allow later convolutions to directly operate on the outputs of earlier layers. After noticing that adding more layers eventually increased the error rate, residual networks were created to adjust the input feature map for higher quality features by creating residual blocks in which intermediate layers learn from the input (Figure 2). This allowed ResNet to reduce the number of parameters from the aforementioned 135 million (*in VGG-16*) to 25 million (*in ResNet-50*) [1].

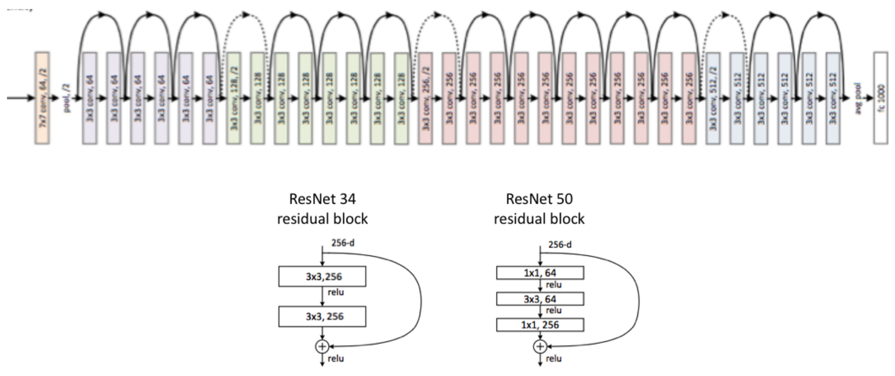
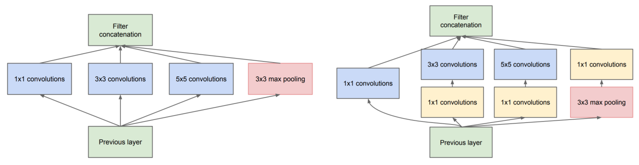


Figure 2: (Top) The outline of ResNet-34. (Bottom) Depictions of a residual block. Our project implements ResNet-50, which replaces each 2-layer Residual Block (left) with 3 convolutional layers (right).

### 3.3 Inception Net

GoogLeNet, or Inception Net, was introduced as an entry to the 2014 ImageNet competition. The network is constructed as a series of "inception cells" on which several convolutions of different scales are run and results are then aggregated. To these are added auxiliary loss outputs to aid backpropogation (Figure 3) [3].



(a) Inception module, naïve version (b) Inception module with dimension reductions

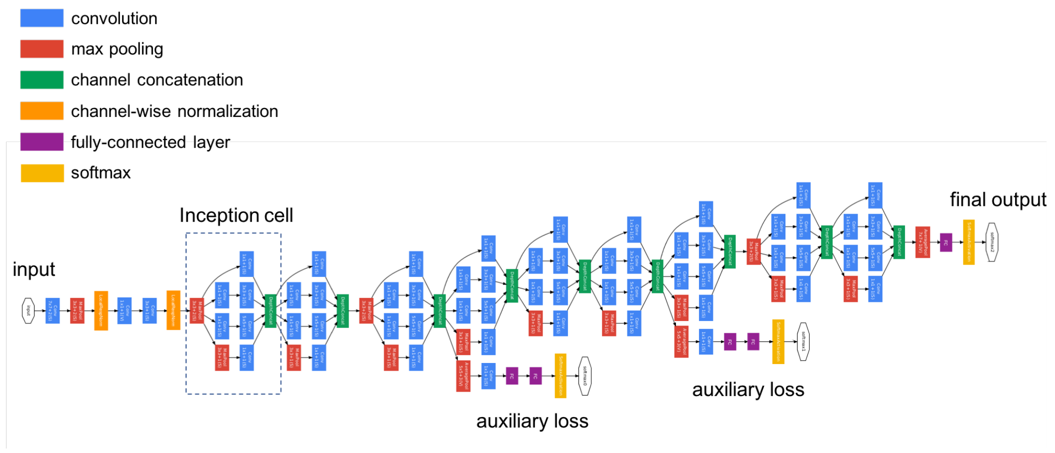


Figure 3: Depictions of the insights and construction of the GoogLeNet / Inception Net architecture.

## 4 Dataset Processing

The Google Landmarks training dataset consists of $1.2 \cdot 10^6$ samples across 14,700 landmark classes. Considering time constraints associated with this work, the dataset's size presented us with a problem. We addressed this issue in two ways:

First, we hosted our scripts on Google Cloud's Deep Learning virtual machines. This allowed us to be able to run our tests on machines with variable resources, allowing us to observe the effect of increasing the computing power. Our virtual machine ran on 8 vCPUs with 30GB memory, along with 2 NVIDIA Tesla K80 GPUs.

The dataset's sheer size still posed a problem, and we had limited credits on Google Cloud. Therefore, we sub-divided the training data according to sample-size percentile ranges, and used these sub-divisions to train and run our models. This allowed us to both:

1. Successfully run our CNNs on the Google Landmark training data
2. Better understand the effects of sparsity on our CNNs

### Understanding our Data

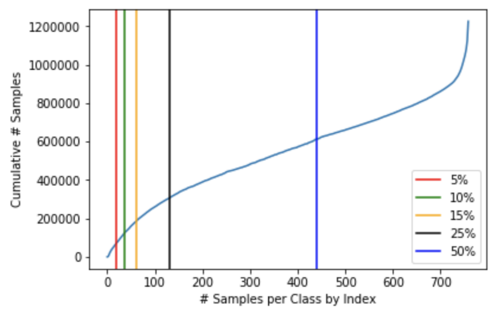
To begin data analysis, we first create two arrays $X = [x_0, x_1, \ldots, x_N]$, $Y = [y_0, y_1, \ldots, y_N]$. These are constructed in accordance with the following:

$x_i =$ The number of samples (images) present in the dataset labeled with class (landmark) ID $i$. X is sorted in ascending order (formally, $x_i \leq x_{i+1}, \forall i$).

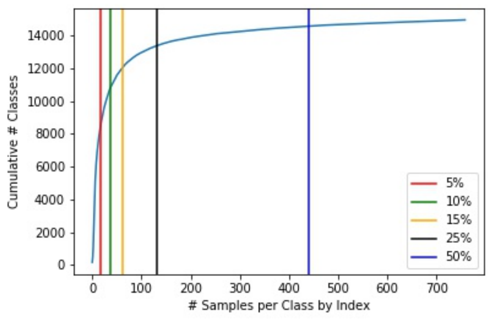
$y_i =$ The cumulative sum of $x_i - x_N$. Formally, $y_i = \sum_{j=i}^{N} x_j$.

$N =$ $14,700$: the number of total classes (landmarks).

We plot both $X$ and $Y$ below: (Figure 4a)



(a) Number of samples per landmark

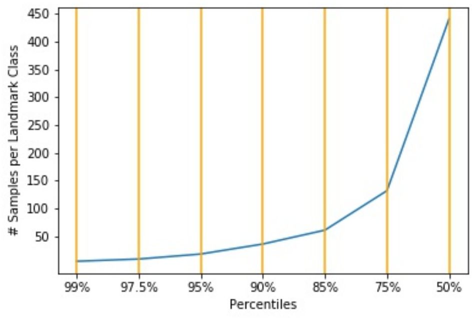


(b) Cumulative sum of samples per landmark

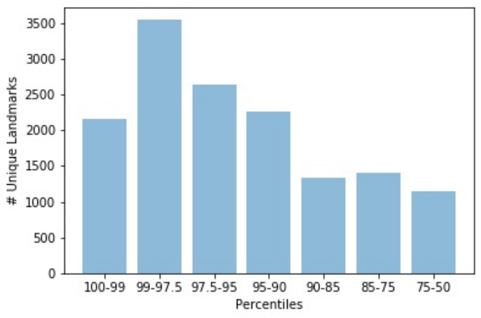
Figure 4

Note the vertical lines, they highlight the cutoffs for the following sample saprsity percentiles of interest: $95\%,\ 90\%,\ 85\%,\ 75\%,\ 50\%$ We see a large number of low sample-size classes distributed over a number of samples between the 99th and 85th percentiles, as well as a small number of classes with a significantly larger number of samples between the 25th and 50th percentiles (and beyond). Given our interest in the relationship between sparsity and CNN accuracy & loss rates, we segment the data according to the percentiles defined above. We will use these segmented datasets to test our CNNs.

Next, we inspect the number of samples per landmark class for each of our specific percentile values as displayed in Figure 5a, where the yellow lines show the segmentation of sample sizes for our respective percentile ranges. We will be creating our segmented data training sets from landmarks that fall within the ranges displayed in the figure. This enables us to test our CNNs on an array of specific class sample-size distributions. Since we wish to use our findings to deduce an optimal model for the complete Google Landmarks training set, we want to retain the proportion of class distributions over our sub-divided training sets. Figure 5b shows the distribution of classes within our percentile ranges. For our sub-divided training sets, we will be taking some proportion x from our percentile segmented data such that for each range we will choose $x \cdot |classes|$ for all ranges (the proportion of classes per percentile range in the complete test set will be the same as the proportion of classes per percentile range in our sub-divided test sets).



(a) Number of samples per landmark per percentile



(b) Cumulative number of classes per percentile range

Figure 5

### Image Formatting

In order to be able to run the keras library models on the files described above, the format of the images had to be changed. These CNN's require specific inputs. The first step was to resize the images in 224x224 shape. Then we turned the images into numpy arrays and turn it into a numpy array, with dtype float32. The values inside the images will lie between 0 and 255. These are then normalized between 0 and 1. All the images are then added to an array. The labels of the different classes (each individual landmark) have to be one-hot-encoded. The `ImageDataGenerator()` `Karas.io` preprocessing function then generates batches of tensor image data with real-time data augmentation. The benefits of this are two fold, the first being the ability to generate more data from limited data (since we only use a subset of the full training data) and secondly it prevents over fitting.

## 5 Results

### Initial CNN Evaluations

We start by testing three separate CNN models on a subset of our sub-divided percentile range test sets. Specifically, we test VGG, ResNet 50, and GoogLe Net on our [(95-90%), (90-85%), (85-75%)] percentile range test sets. We simplify the initial evaluation as our aim is to find an optimal CNN from which we can further analyze with respect to our wider-sparsity-range test sets. The accuracy and loss plots can be found in the appendix, in Figures 9-17.

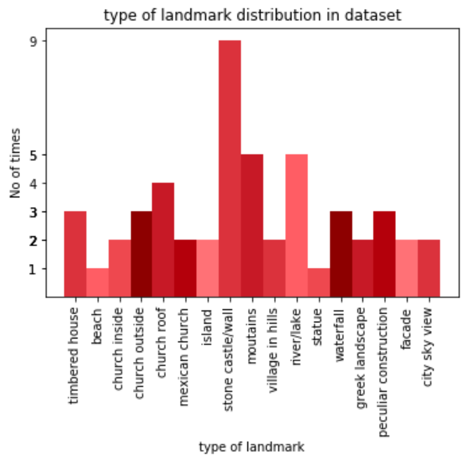
Upon first inspection, we immediately see that VGG performs the worst of the three CNNs, with a negligible accuracy. Thus, we focus on ResNet 50 and GoogLe Net. Looking at Figures (9, 10, 12, 13, 15, 16), we can analyze the differences between the performance of ResNet 50 and GoogLe Net across the range of training set sparsity:

- First, we notice that both ResNet 50 and GoogLe Net have somewhat comparable accuracy and loss scores insofar as they achieve a significant classification accuracy on our ranges.
- However, some of our models appear to be under-fitted as the validation loss remains large, while others appear to fluctuate between over-fitting and under-fitting. We believe this to be due to too few epochs present, and therefore the correlations between images are yet to be fully fitted in our CNNs

- In relation to this, we found the large distribution of changes in validation accuracy and validation loss in GoogLe Net across the percentile range to be particularly interesting, and thus we were interested in understanding more about how sparsity effects these metrics for the GoogLe Net CNN specifically.

## Landmark Class Similarities

We thought the difference between results on the three datasets could be due to the fact that we chose the images in them at random and some datasets could have more similar landmarks that were harder to differentiate whereas others could have very different types of landmark (beach, tower, mountain etc...) which are easier to classify. We went through the datasets and realized all three of them had very similar distributions of types of landmarks. We classified these landmarks in 17 types that are present in proportional quantities in the three folders. These types of landmarks and the number of times they show up in our total data are shown in the graph below:



This allowed to eliminate the type of landmarks present in each dataset as potential unwanted noise that could distort our analysis of the date. However, these types of landmarks do matter since across datasets, consistently, some had lower accuracies than others. The most striking example are mountains. Classification on mountains consistently performed worse than on other types of landmarks and different mountainous landscapes got mistaken for each other (Figure 8 in the Appendix illustrates one such example).

This is evidence that even though mean accuracy through classes is very high when using GoogleNet, it isn't consistent throughout types of landmarks. For man built landmarks, they are more easily identified and separated from the background. Buildings are also always in focus in the picture since people specifically intend to photograph them. For landscape landmarks however, they are the background and they are sometimes "noise polluted" by people standing in the middle of the picture. We hypothesize that different angles are also harder to process for landscapes than for buildings. Depending on the dataset these networks are applied to we could not expect to get the same accuracy (for instance if someone were to run it only on landscapes).

## GoogLe Net sparsity Evaluation

To further explore the GoogLe Net CNN model, and its relation to sparsity in sample sizes and the resulting validation metrics, we ran our model on the following sparsity ranges: [(95-90), (90-85), (85-75), (75-50)] with a larger number of samples per interval. The results can be found in Figures 18-21.

We immediately notice that the fluctuation between low and high validation accuracy/loss values is inconsistent across the sparsity percentile range. To clarify, there was no single number of epochs that would result in consistency high accuracy and low loss values across the sparsity spectrum. We did, however, observe that we were able to successfully classify our images with a high accuracy and low loss at some epoch value $\leq 10$ for all percentile ranges.

We also noticed that there was a strong correlation between sparsity and optimal validation accuracy. Specifically, we found that as sparsity decreased (and the number of samples per class increased), that we achieved a larger validation accuracy and lower validation loss. Furthermore, the less sparse the training set, the more epochs were required to reach an optimally fitted model. This coincides with our beliefs as the more complex the training set becomes, the more training steps would be required to fit the model correctly.

## 6 Discussion and Conclusion

We started this project with the hope of using the information we gathered by running our models on different sparsity levels to create a single model that would perform well across the entire landmark training set. Through our analysis of GoogLe Net, we found that this is not possible as the fitting effect at each epoch resulted in different validation accuracy and loss metric values for each of our sparsity ranges.

We did, however, find interesting information about how sparsity effects the validation metrics of our models. There was a strong correlation between sparsity and accuracy/loss. Specifically, as we increased the number of samples per class, we saw an increase in the validation accuracy and decrease in validation loss. Furthermore, there was a correlation between sparsity and optimal number of epochs insofar as the larger the sample space became, the more epochs were required in order to achieve a large accuracy.

**We summarize our understandings as the following:**

Given a training set of images with a given sparsity, we can roughly predict the number of epochs required to fit the model and the validation accuracy and loss at the given sparsity level. Data that is extremely sparse is difficult to fit in such a way that results in a high accuracy and low loss metric values. This is due to the minimal amount of information from which our model can deduce relationships between. Conversely, with low sparsity our model has more data from which it can properly fit, and thus results in a higher accuracy and loss values.

Similarly, we found that the more sparse the training data, the fewer epochs were required to find an optimally fit model as fewer steps were required to find a relationship. With less sparse data, there is more information due to the increase in images. Although we can achieve a similar accuracy at a similar number of epochs to data which is more sparse (we could be hypothetically using the same amount of image data), we find that achieving an optimally accurate model with less sparse data requires more epochs as more complex relationships are realized.

## 7 Further directions

The first extension that comes to mind when working on this project is of course to run the models we studied on much larger datasets which we were unable to do given the limited computing power to which we have access. Further developing and fine tuning the models, especially building our own weights instead of using the models with pretrained weights on the imagenet dataset would probably be more effective. In addition, maybe running the model on more epochs would reveal an optimal number of epochs for all shapes and sizes of data for the googlenet model. Finally, here we have used a validation set that did not contain images of ”non-landmarks”. The next step would be to be able to classify these in a specific non-landmark class despite the fact that there are none in the training data.

## Acknowledgments

We'd like to acknowledge and thank the authors of the three models we considered. Their explanations and analysis, in the papers cited in this work, enabled us to easily interpret and recreate their models.

Moreover, we'd like to thank Professor Englehart and all course TA's for your work this semester. We hope this final report reaches you well.

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. Apr 2015.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

## Appendix

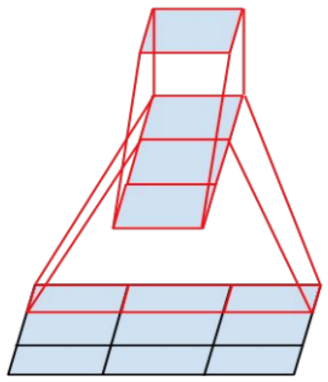


Figure 7: Depicting how a $3 \times 3$ convolution can be reduced into a $3 \times 1$ and $1 \times 3$ [4]

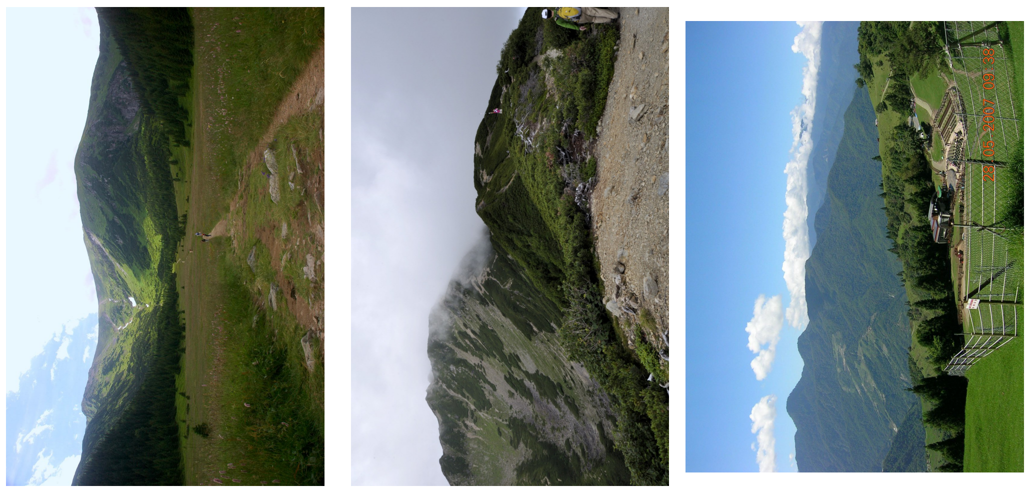
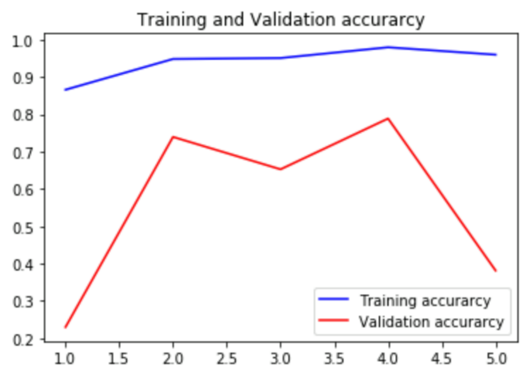
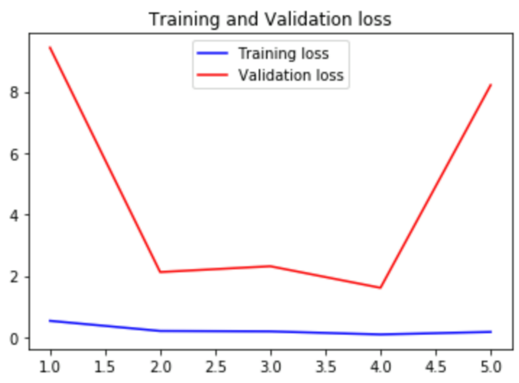


Figure 8: Easily-mistaken mountain landscapes

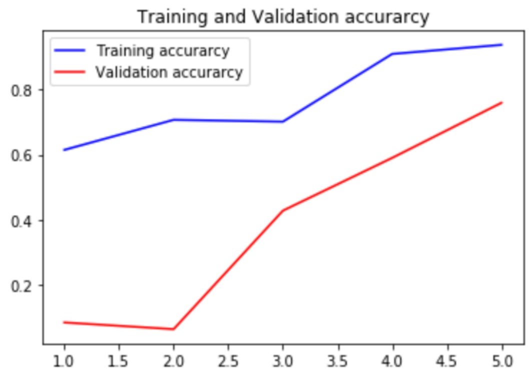


(a) 85-75% GoogLe Net Accuracy

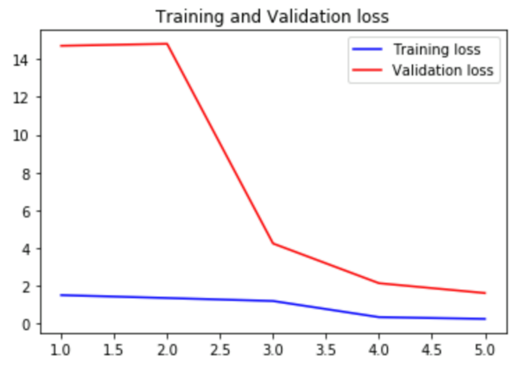


(b) 85-75% GoogLe Net Loss

Figure 9: GoogLe Net metrics: 85-75%

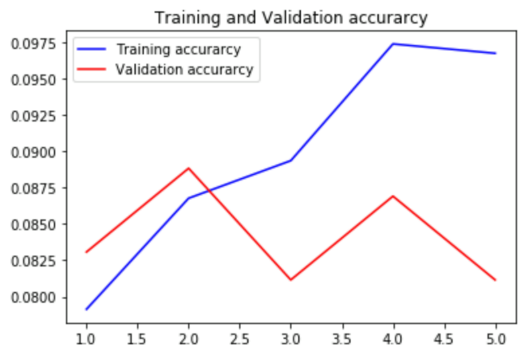


(a) 85-75% ResNet 50 Accuracy

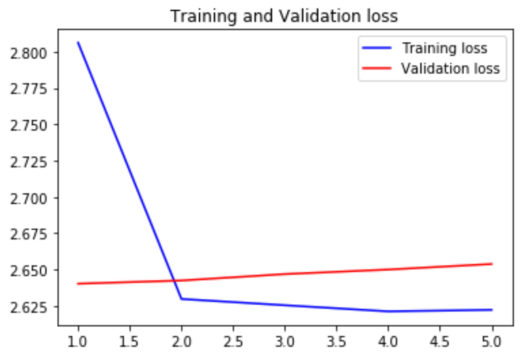


(b) 85-75% ResNt 50 Loss

Figure 10: ResNet 50 metrics: 85-75%

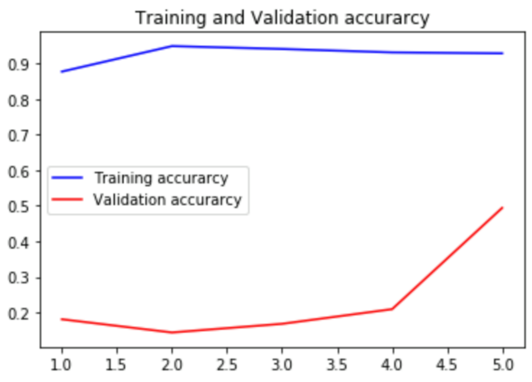


(a) 85-75% VGG Accuracy

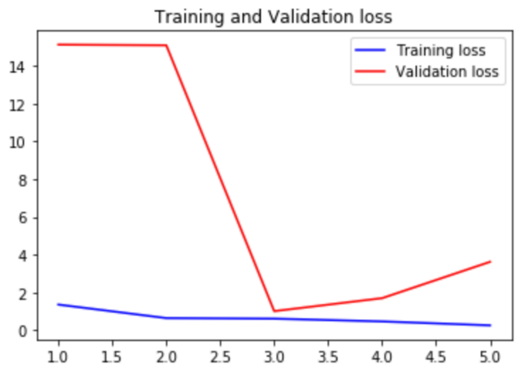


(b) 85-75% VGG Loss

Figure 11: VGG metrics: 85-75%

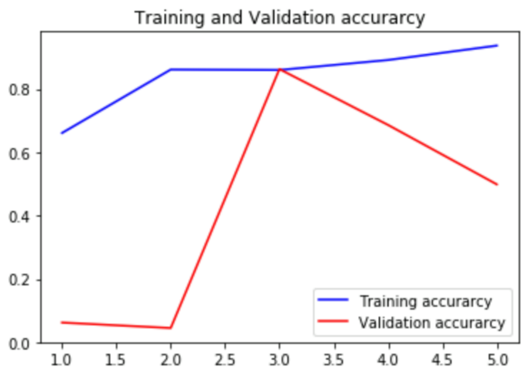


(a) 90-85% GoogLe Net Accuracy

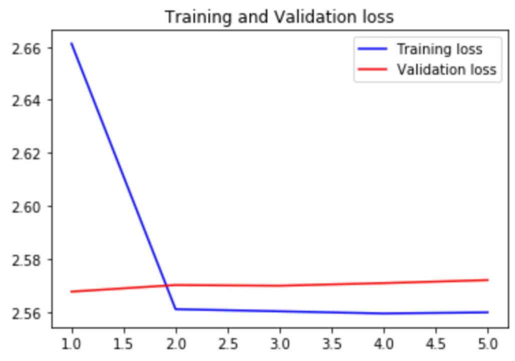


(b) 90-85% GoogLe Net Loss

Figure 12: GoogLe Net metrics: 90-85%

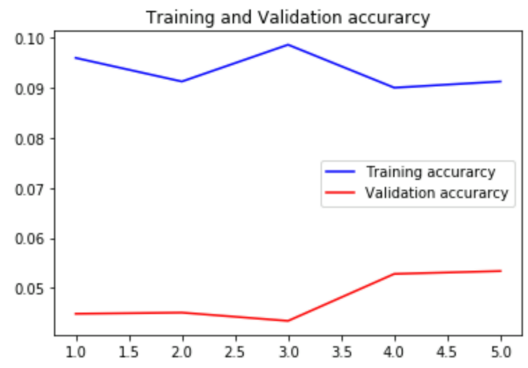


(a) 90-85% ResNet 50 Accuracy

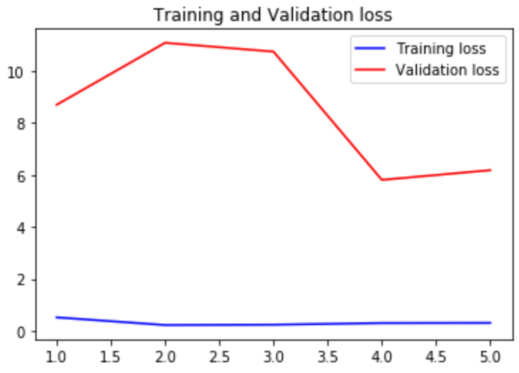


(b) 90-85% ResNet 50 Loss

Figure 13: ResNet 50 metrics: 90-85%

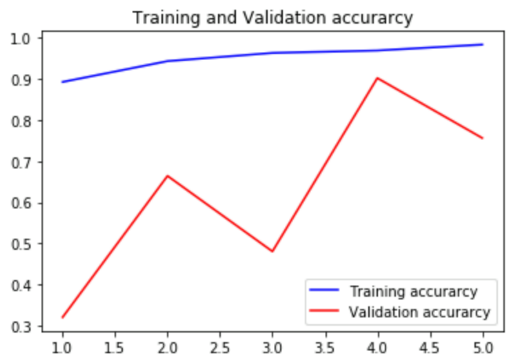


(a) 90-85% ResNet 50 Accuracy

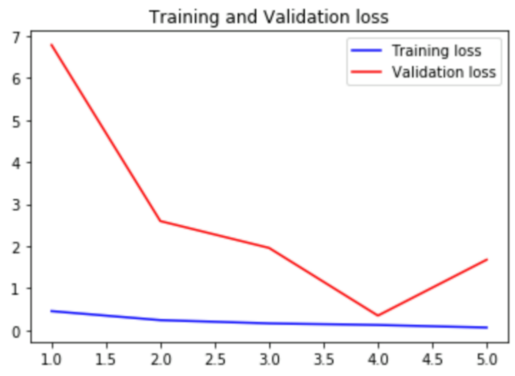


(b) 90-85% ResNet 50 Loss

Figure 14: VGG 50 metrics: 90-85%

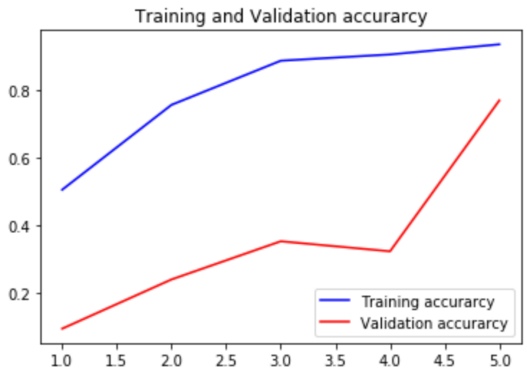


(a) 95-90% GoogLe Net Accuracy

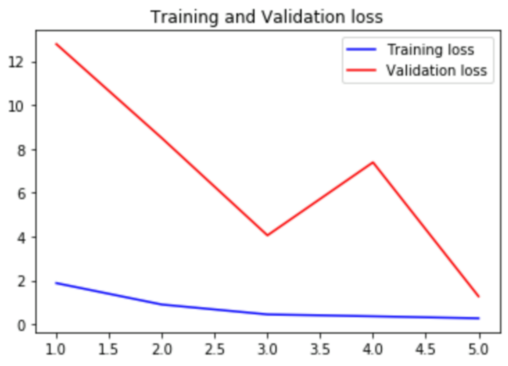


(b) 95-90% GoogLe Net Loss

Figure 15: GoogLe Net metrics: 95-90%

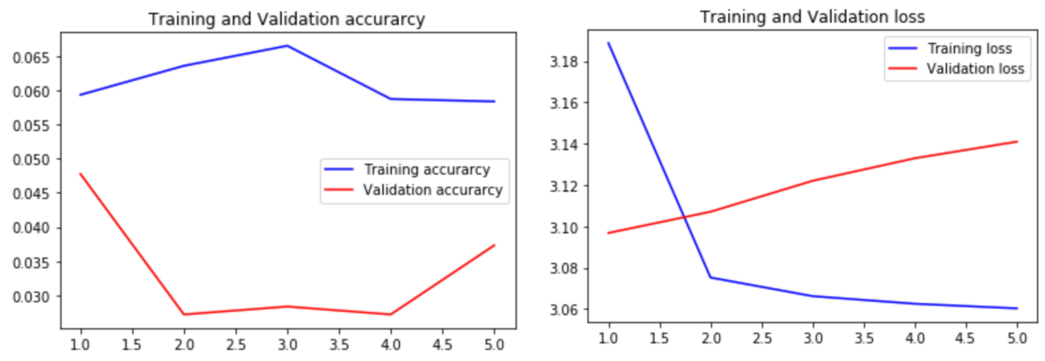


(a) 95-90% ResNet 50 Accuracy



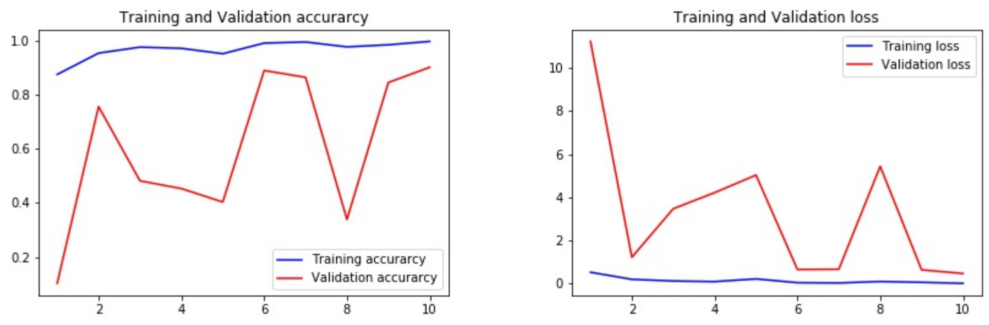
(b) 95-90% ResNet 50 Loss

Figure 16: ResNet 50 metrics: 95-90%



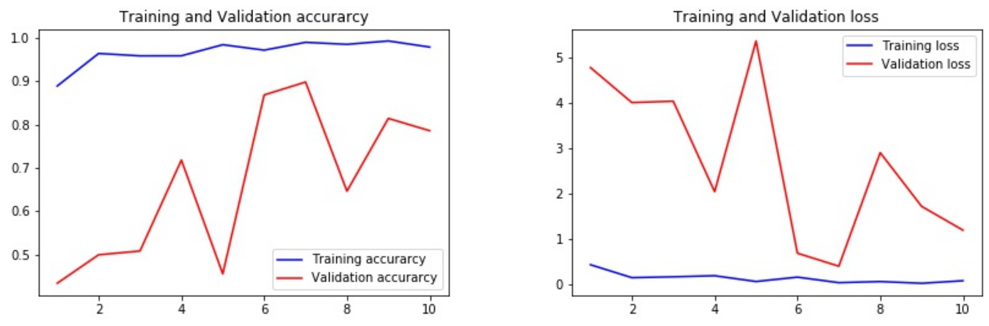
(a) 95-90% ResNet 50 Accuracy (b) 95-90% ResNet 50 Loss

Figure 17: VGG 50 metrics: 95-90%



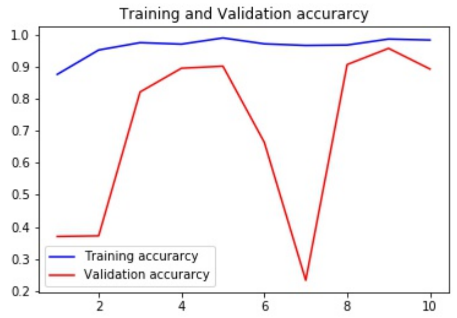
(a) 95-90% GoogLe Net Accuracy (b) 95-90% GoogLe Net Loss

Figure 18: GoogLe Net metrics: 95-90%

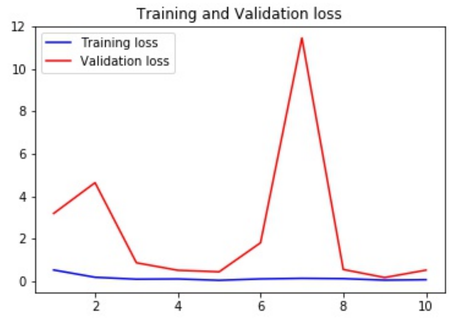


(a) 90-85% GoogLe Net Accuracy (b) 90-85% GoogLe Net Loss

Figure 19: GoogLe Net metrics: 90-85%

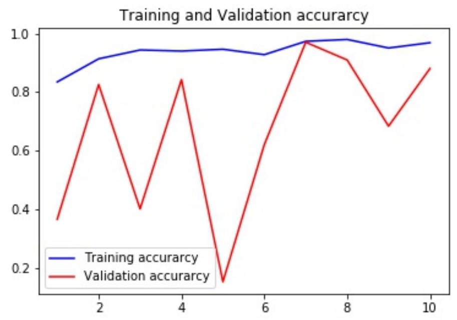


(a) 85-75% GoogLe Net Accuracy

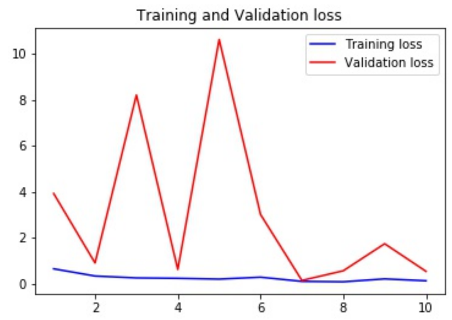


(b) 85-75% GoogLe Net Loss

Figure 20: GoogLe Net metrics: 85-75%



(a) 75-50% GoogLe Net Accuracy



(b) 75-50% GoogLe Net Loss

Figure 21: GoogLe Net metrics: 75-50%