6.869 MiniPlaces Challenge Part II

Ryan Sander

15 November 2019

1 Problem 1: Improving NN Performance

For this part of the challenge, I chose ResNet18 as my baseline network, with the following other features that I will use for comparison later:

- No dropout layers
- Stochastic Gradient Descent Optimizer (SGD)

. This network is state-of-the-art, and this network and variations of it have consistently yielded the highest performance for many image classification tasks. The modifications I made to this network:

- Change the neural network architecture
- Adding dropout layers before the fully connected output layer
- Trainining with different optimizers

Some default environment specifications/parameters I used:

- Cloud computer: Amazon EC2 p2.xlarge instance hosted on Ubuntu 18.04
- Epochs: 10
- Learning rates: $\eta \in \{0.001, 0.01, 0.1, 1\}$
- Batch Sizes: 128 (ResNet18), 32 (ResNet101; this smaller batch size was to avoid flooding the GPU memory).

For each of these results, I will show the Top-5 loss plots for training and validation sets corresponding to the modified network and original ResNet18 network, and will show this for different combinations of hyperparameters. **NOTE: I received confirmation from the 6.869 team via Piazza that these plots were an acceptable substitution for the Top-5 error plots.**

1.1 Varying Neural Network Architecture

For this section, I wanted to experiment with a different neural network architecture. Namely, I wanted to use a different, more complex variation of ResNet: ResNet101. This network has substantially more layers and parameters, and thus requires a different approach to tune than the previous networks. Using the default parameters of the other network aside from architecture, training and validation testing yielded the following results for our different learning rate values:

1.1.1 Top-5 Error Comparisons

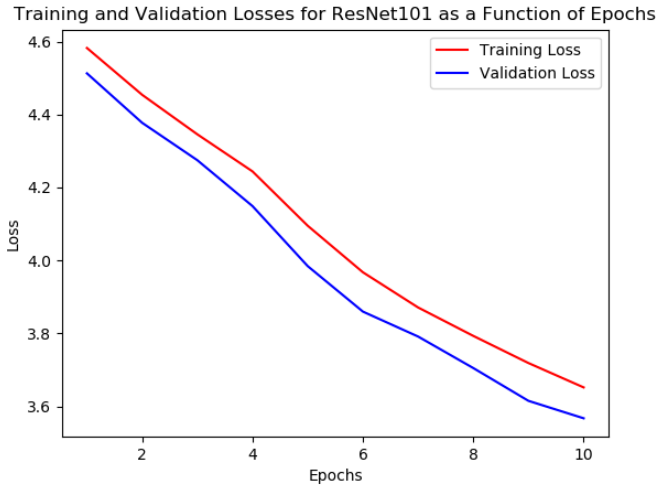

Figure 1: Plot showing the top-5 loss of the ResNet101 network for training and validation sets as a function of epochs. **Architecture**: ResNet101, **Learning Rate**: 0.001, **Optimizer**: SGD, **Dropout Layers**: False

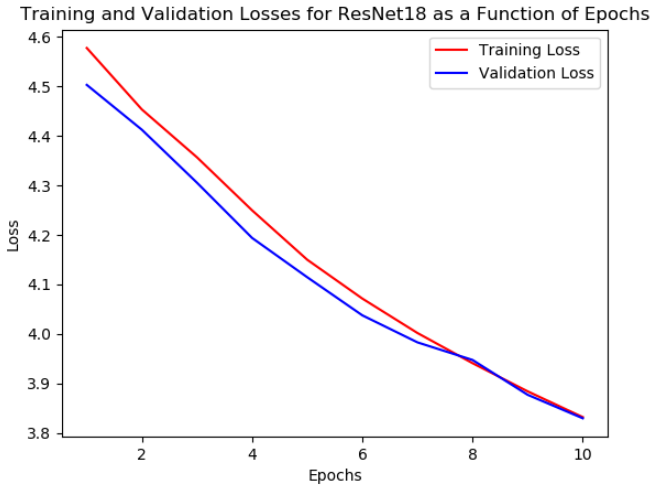

Figure 2: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs. **Architecture**: ResNet18, **Learning Rate**: 0.001, **Optimizer**: SGD, **Dropout Layers**: False

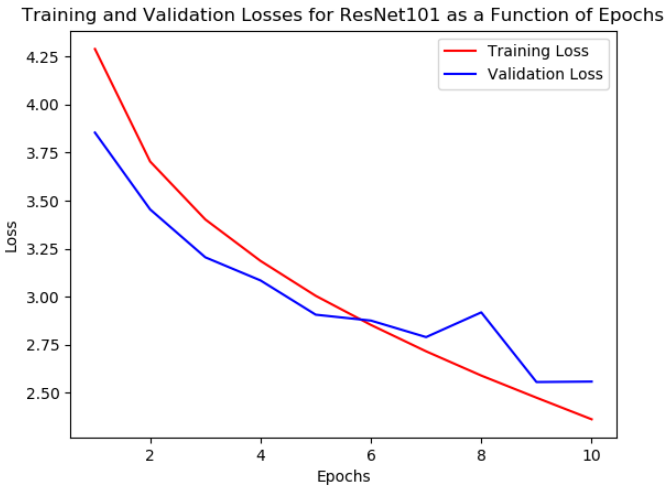

Figure 3: Plot showing the top-5 loss of the ResNet101 network for training and validation sets as a function of epochs. **Architecture**: ResNet101, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: False

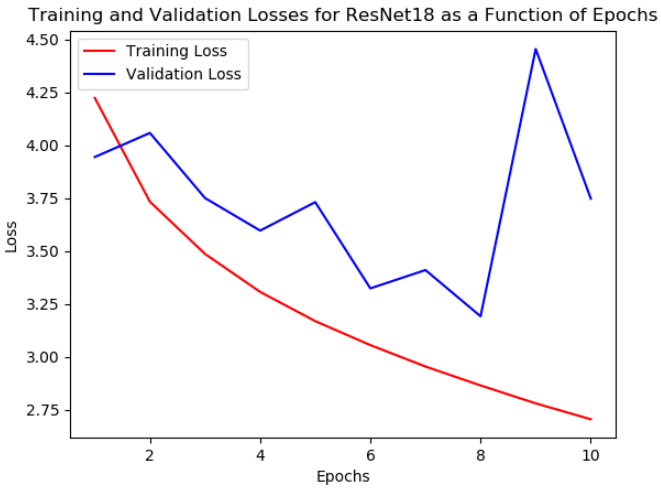

Figure 4: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs. **Architecture**: ResNet18, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: False

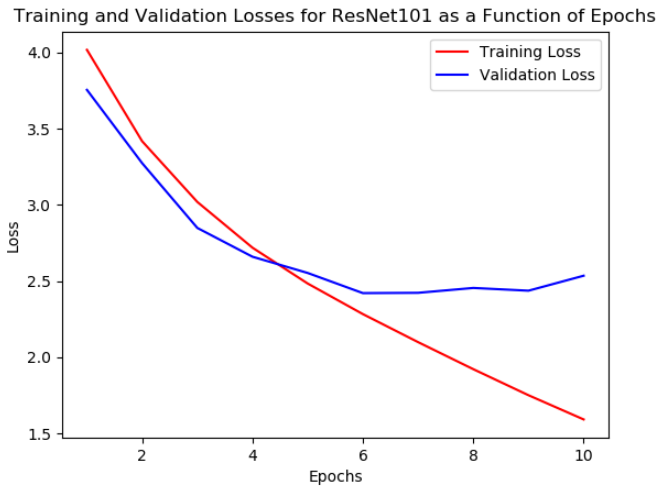

Figure 5: Plot showing the top-5 loss of the ResNet101 network for training and validation sets as a function of epochs. **Architecture**: ResNet101, **Learning Rate**: 0.1, **Optimizer**: SGD, **Dropout Layers**: False

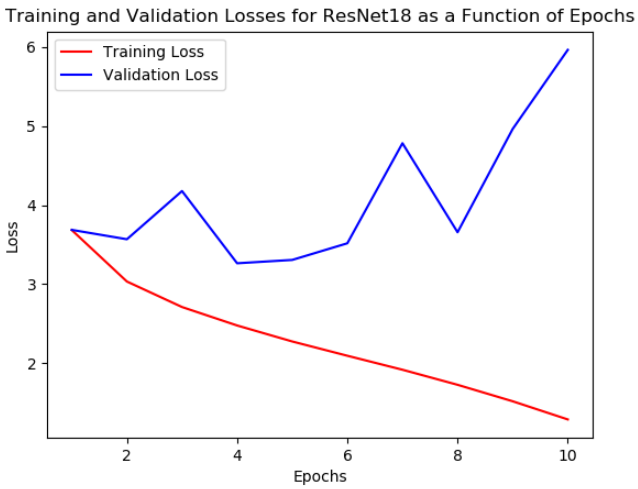

Figure 6: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs. **Architecture**: ResNet18, **Learning Rate**: 0.1, **Optimizer**: SGD, **Dropout Layers**: False

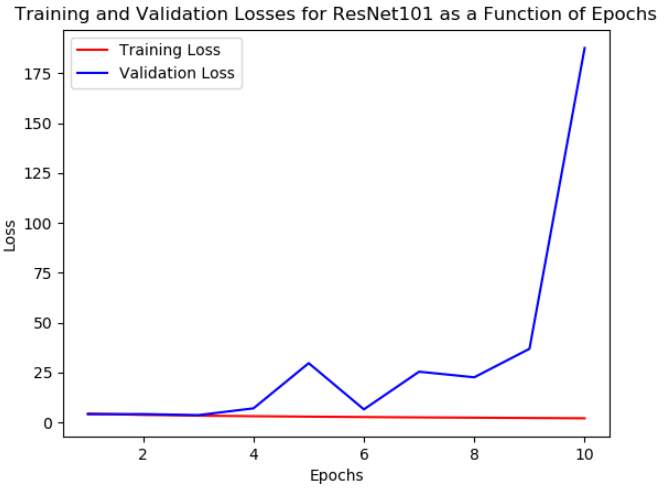

Figure 7: Plot showing the top-5 loss of the ResNet101 network for training and validation sets as a function of epochs. **Architecture**: ResNet101, **Learning Rate**: 1, **Optimizer**: SGD, **Dropout Layers**: False

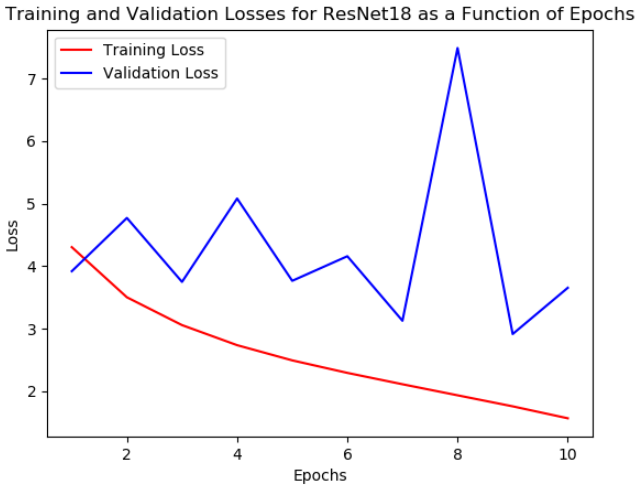

Figure 8: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs. **Architecture**: ResNet18, **Learning Rate**: 1, **Optimizer**: SGD, **Dropout Layers**: False

1.1.2 Assessing our Results

Here, we can see that ResNet101 without dropout and an SGD optimizer outperforms ResNet18 without dropout and an SGD optimizer, except for the largest step size of 1. One reason why the validation loss may have grown out-of-control here is due to the fact that while residual networks excel at mitigating the vanishing gradient problem, they don't really have any features that help to address the exploding gradient problem. This may have resulted in large gradient updates for the network that were tuned really well for the training dataset, but had very poor generalization. A very deep network (101 layers) trained without dropout may have also resulted in the network's parameters becoming "entangled" with one another, once again leading to overfitting on the training set and poor generalization (as is seen by the unbounded validation error).

For small step sizes, however, ResNet101 outperforms ResNet18 (in the case of no dropout and an SGD optimizer). Though step size helps to increase the learning rate, too large of a learning rate can lead to instability when testing on the validation set.

1.2 Adding Dropout Layers

For this section, I wanted to experiment with adding dropout layers to the original ResNet18 neural network. In briefly examining research literature on the use of dropout layers in ResNet, I believed it would be better to add the dropout layer to the fully connected layer, rather than to the convolutional and pooling layers of ResNet. My code for implementing this dropout functionality is given below: Given a ResNet model, it simply modifies the fully connected layer so that it contains both the fully connected layer cascaded after a dropout layer. For the dropout probability, for simplicity I kept this to 0.5.

```
# Code for adding dropout layers
def add_dropout(M, p=0.5, num_ftrs=100):
    """Function that takes in a module M and outputs a new module with a
    dropout layer appended in the fully connected region of the network."""

    M.fc = nn.Sequential(nn.Dropout(0.5), nn.Linear(512, num_ftrs))
```

1.2.1 Top-5 Error Comparisons

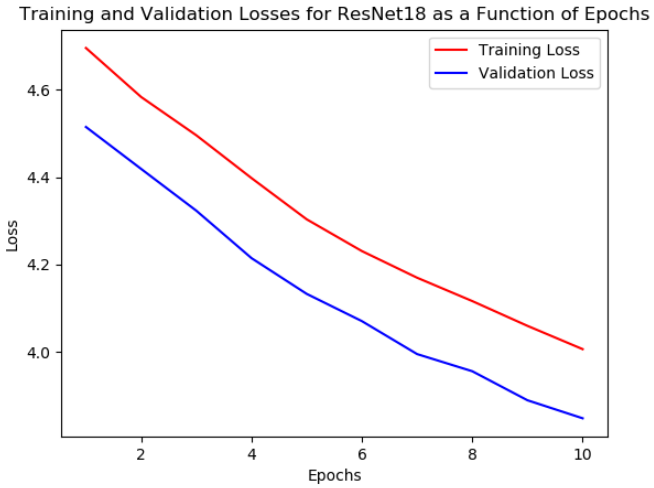

Figure 9: Plot showing the top-5 loss of the ResNet18 dropout network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001,
Optimizer: SGD, **Dropout Layers**: True

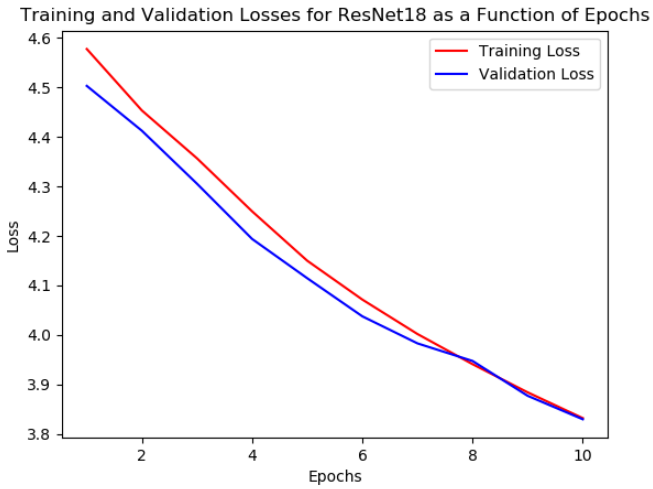

Figure 10: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001,
Optimizer: SGD, **Dropout Layers**: False

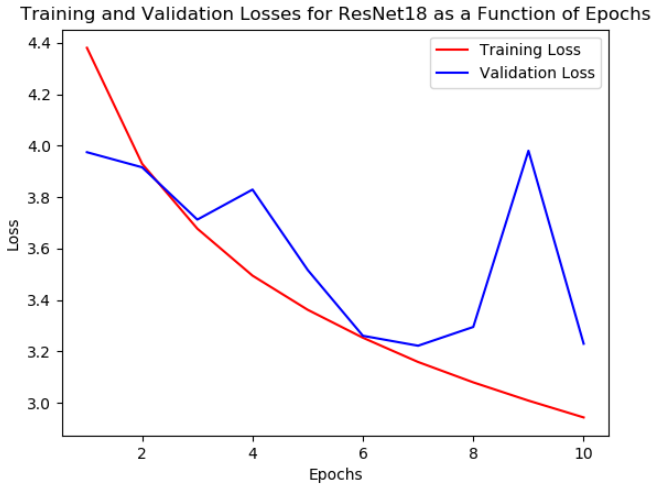

Figure 11: Plot showing the top-5 loss of the ResNet18 dropout network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: True

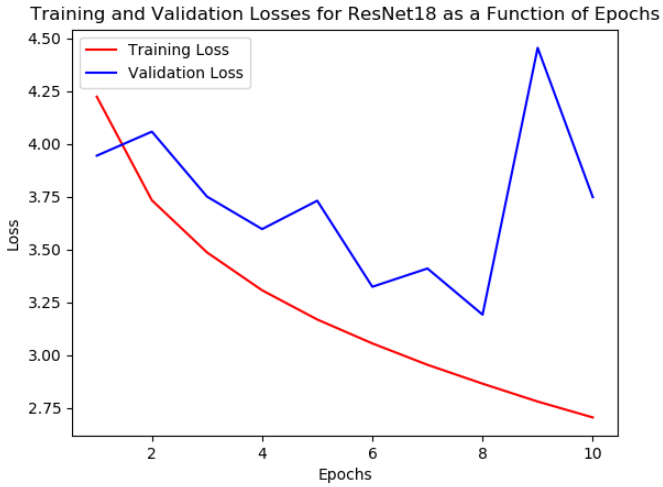

Figure 12: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: False

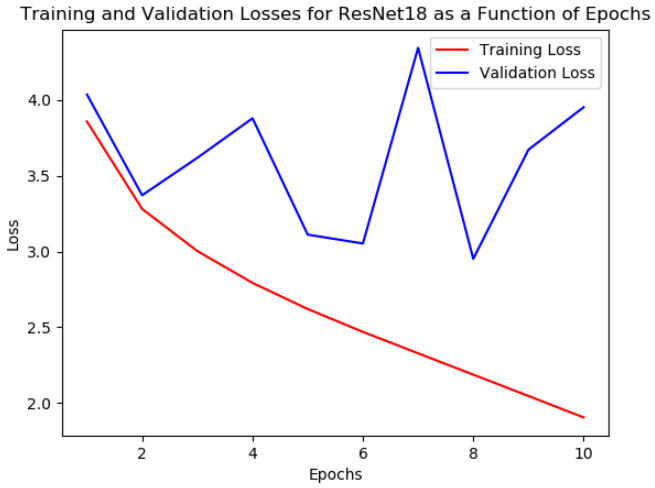

Figure 13: Plot showing the top-5 loss of the ResNet18 dropout network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1, **Optimizer**: SGD, **Dropout Layers**: True

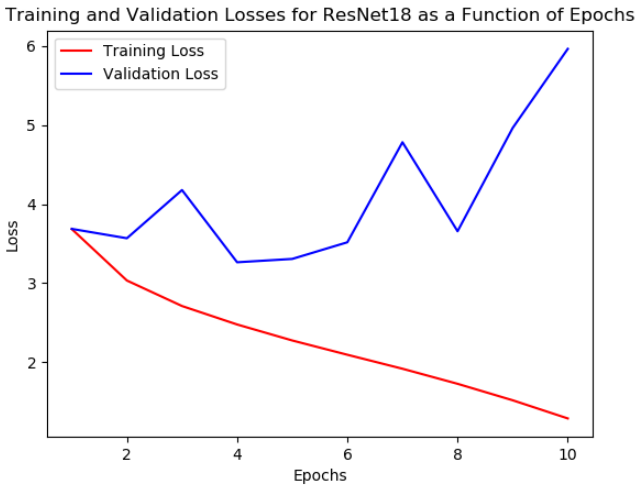

Figure 14: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1, **Optimizer**: SGD, **Dropout Layers**: False

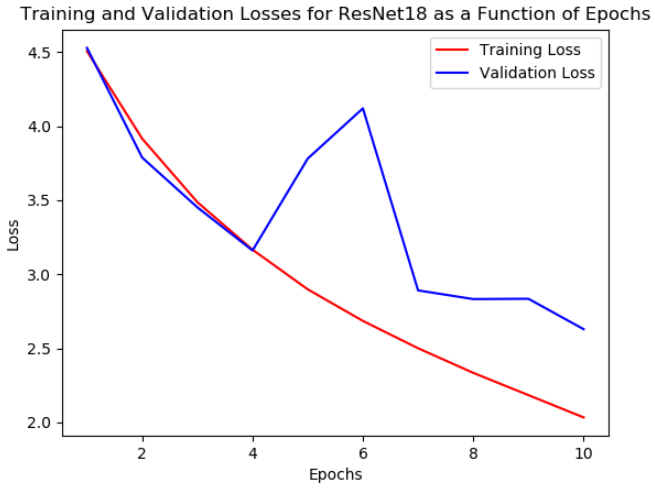

Figure 15: Plot showing the top-5 loss of the ResNet18 dropout network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1, **Optimizer**: SGD, **Dropout Layers**: True

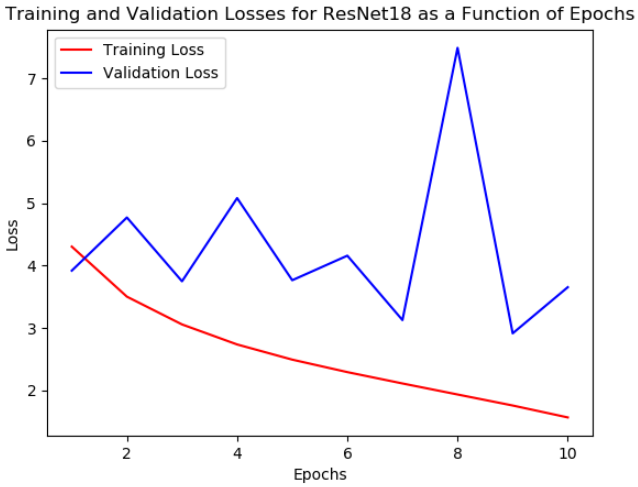

Figure 16: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1, **Optimizer**: SGD, **Dropout Layers**: False

1.2.2 Assessing our Results

Here, we can see that adding a dropout layer to our network leads to an initially slower rate of learning for both training and validation performance. However, eventually, for larger step sizes, this leads to a greater degree of generalization ability (i.e. performance on the validation set), likely because the weights are less coupled with one another through dropout and are forced to function more independently of one another than before. **For smaller step sizes, no dropout slightly outperforms dropout for ResNet18 with an SGD optimizer. For larger step sizes, however, leveraging dropout tends to have better performance. Though step size helps to increase the learning rate, too large of a learning rate can lead to instability when testing on the validation set.**

1.3 Changing Our Optimizer

I also wanted to investigate the use of a different optimizer for this problem. I decided to use ADAM because of this optimizer's ability to create adaptive learning rates using unbiased estimates of first and second-order moments of gradients. Adaptive learning rates are particularly important for optimizing over deep neural networks (i.e. ResNet18 or ResNet101), where vanishing and exploding gradients can prevent meaningful learning (though since we are using residual networks, the impact of vanishing gradients is at least lessened). I kept the learning rates the same as before. The mathematical formulations for the different optimizers used and their respective weight/parameter update steps are given below:

- **Stochastic Gradient Descent (SGD)**: $\theta^{t+1} \longleftarrow \theta^t - \eta_t \nabla_\theta L(x^i; \theta^t)$ [Gradient Update Step Only]
- **Adam**:
  1. **Compute Gradient**: $g_t \longleftarrow \nabla_\theta L(x^i; \theta^t)$
  2. **Compute Biased First-Order Moment Estimate**: $m_t \longleftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t, \;\; \beta_1 \in [0,1]$
  3. **Compute Biased Second-Order Moment Estimate**: $v_t \longleftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \;\; \beta_2 \in [0,1]$
  4. **Compute Unbiased First-Order Moment Estimate**: $\hat{m}_t \longleftarrow \frac{m_t}{1-\beta_1^t}$
  5. **Compute Unbiased Second-Order Moment Estimate**: $\hat{v}_t \longleftarrow \frac{v_t}{1-\beta_2^t}$
  6. **Update Parameters Using First and Second Order Moments**: $\theta_t \longleftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}, \;\; \alpha \in \mathbb{R}$

1.3.1 Top-5 Error Comparisons

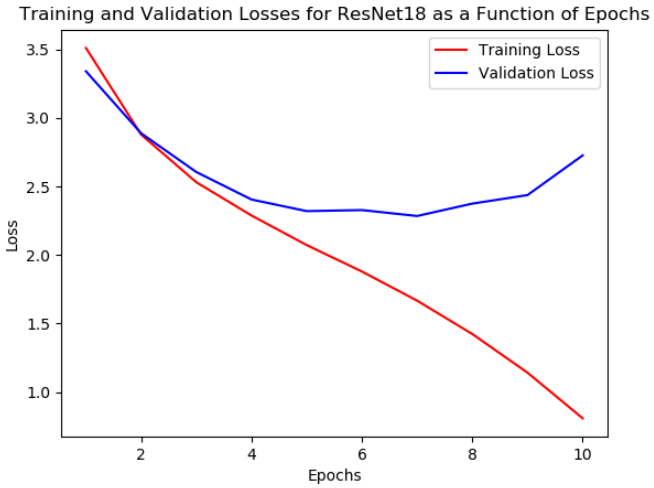

Figure 17: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001, **Optimizer**: ADAM, **Dropout Layers**: False

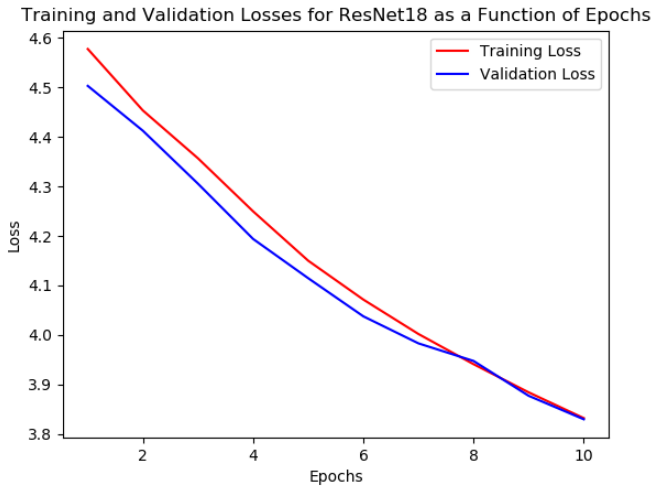

Figure 18: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001, **Optimizer**: SGD, **Dropout Layers**: False

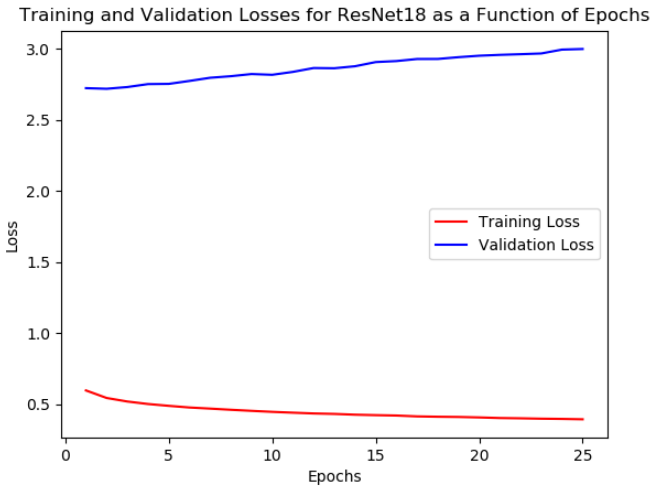

Figure 19: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: ADAM, **Dropout Layers**: False

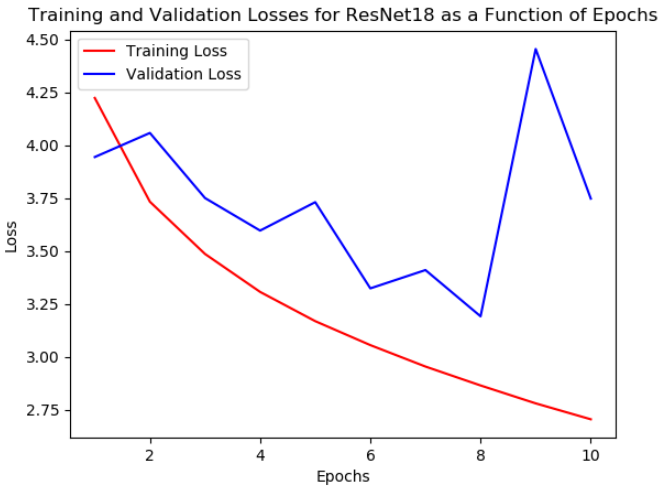

Figure 20: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: False

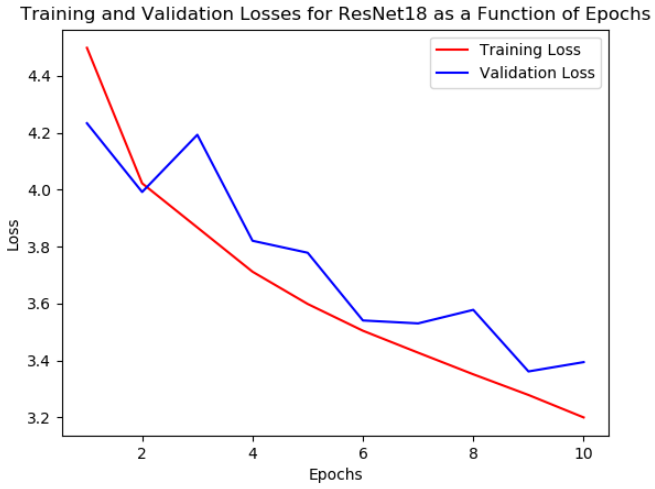

Figure 21: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1,
Optimizer: ADAM, **Dropout Layers**: False

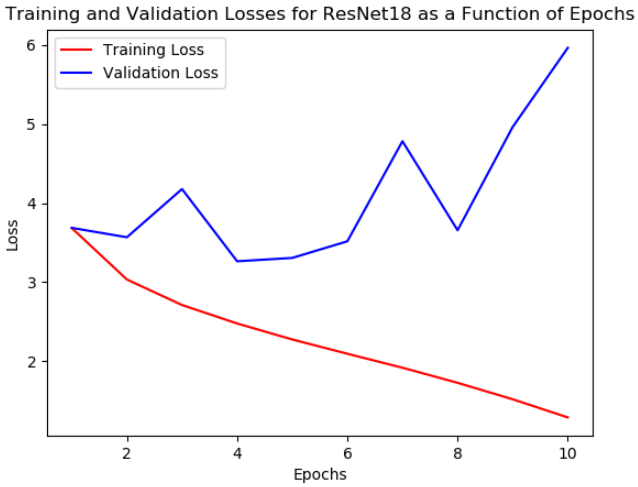

Figure 22: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1,
Optimizer: SGD, **Dropout Layers**: False

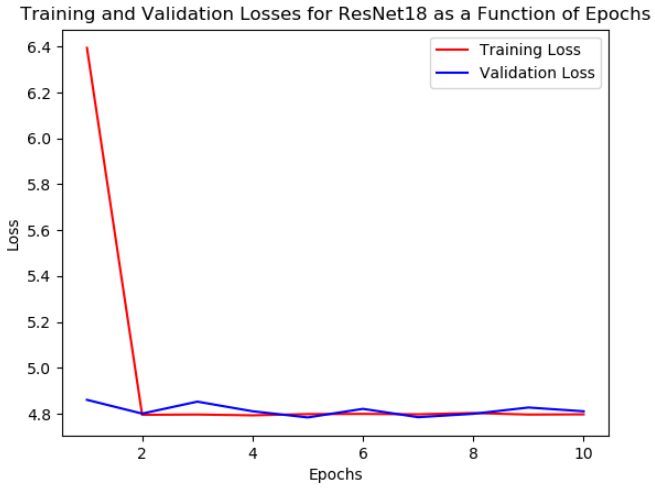

Figure 23: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1,
Optimizer: ADAM, **Dropout Layers**: False

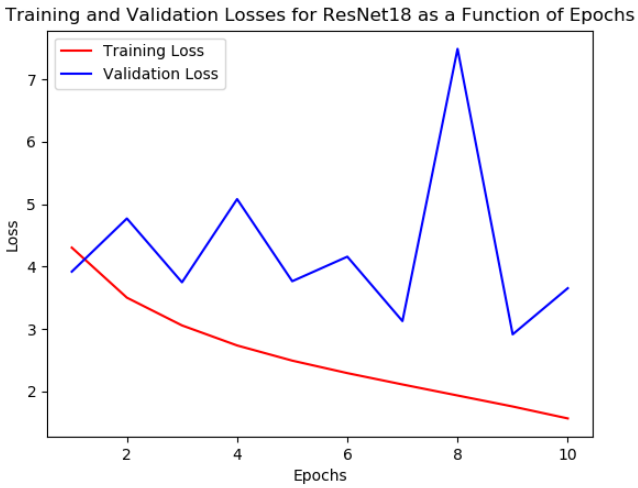

Figure 24: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1,
Optimizer: SGD, **Dropout Layers**: False

1.3.2 Assessing our Results

As a whole, using ADAM as our optimizer does a good job of ensuring that we make updates that generalize reasonably well (i.e. don't drastically overfit to the training data, don't drastically underfit to the training data, and don't create unbounded oscillations for training or testing). This likely arises from Adam's adaptive step size from using unbiased first and second-order moment gradient estimates to ensure we take proper step sizes when updating our parameters.

Using the ADAM optimizer outperforms the Stochastic Gradient Descent (SGD) optimizer when used on ResNet18 without dropout. Step size helps to improve validation top-5 classification ability, until we reach the largest step size of $\eta = 1$, where validation performance begins to diverge - this may be due to either oscillation of parameters or overfitting to the training dataset).

1.4 Modifying Optimizer and Using Dropout

For these sets of training experiments/results, I chose to modify both the optimizer (using ADAM), as well as adding dropout layers (using the process and code described above; here I also added dropout layers to the final layer of the neural network). These results are given below for different learning rates:

1.4.1 Top-5 Error Comparisons

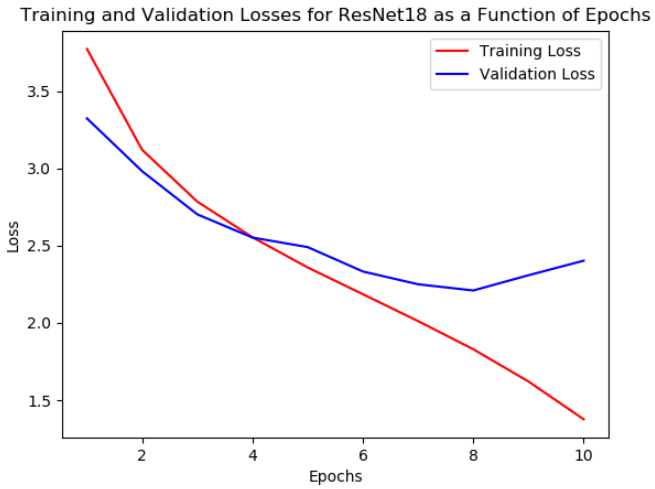

Figure 25: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001, **Optimizer**: ADAM, **Dropout Layers**: False

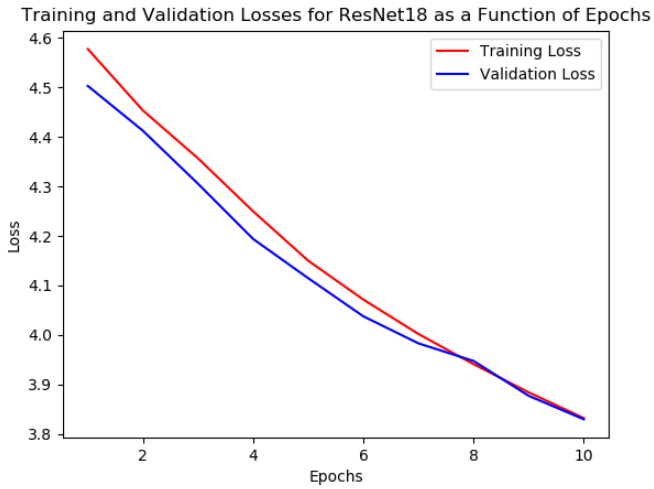

Figure 26: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.001, **Optimizer**: SGD, **Dropout Layers**: False

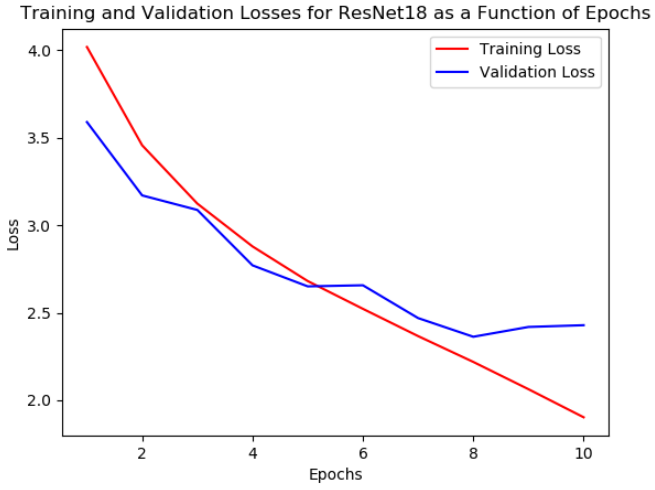

Figure 27: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: ADAM, **Dropout Layers**: False

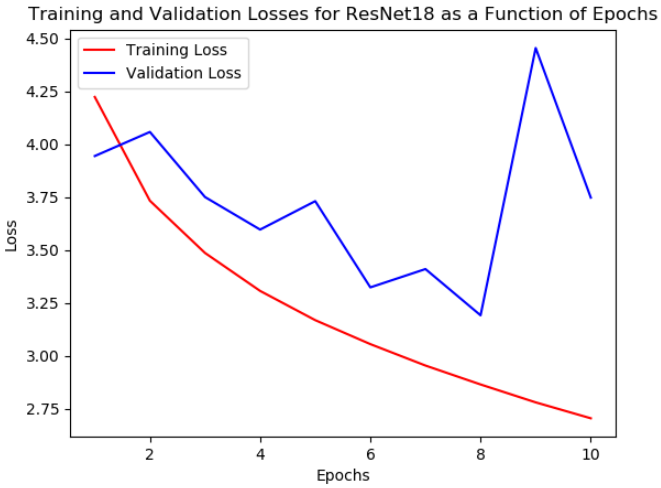

Figure 28: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.01, **Optimizer**: SGD, **Dropout Layers**: False

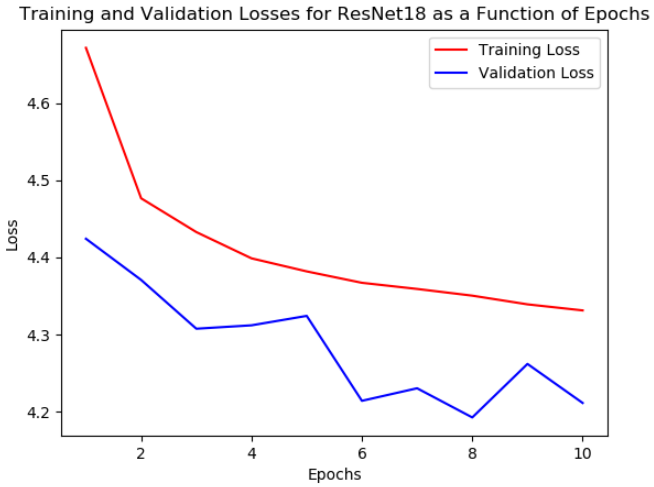

Figure 29: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1,
Optimizer: ADAM, **Dropout Layers**: False

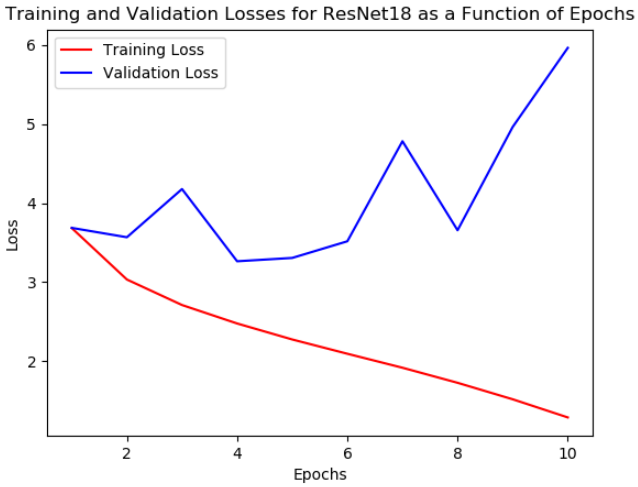

Figure 30: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 0.1,
Optimizer: SGD, **Dropout Layers**: False

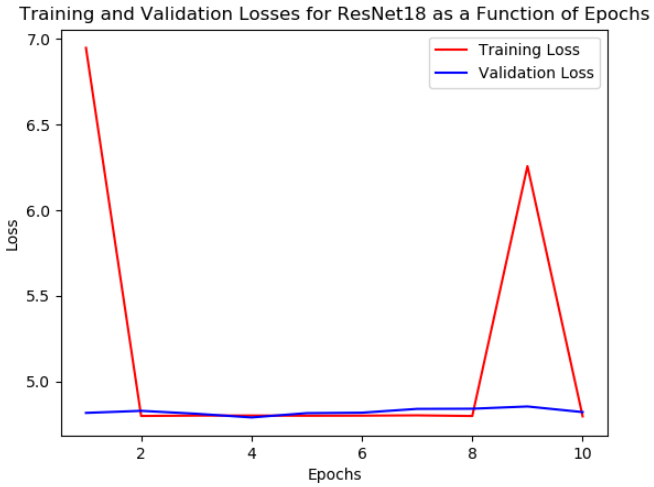

Figure 31: Plot showing the top-5 loss of the ResNet18 ADAM network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1,
Optimizer: ADAM, **Dropout Layers**: False

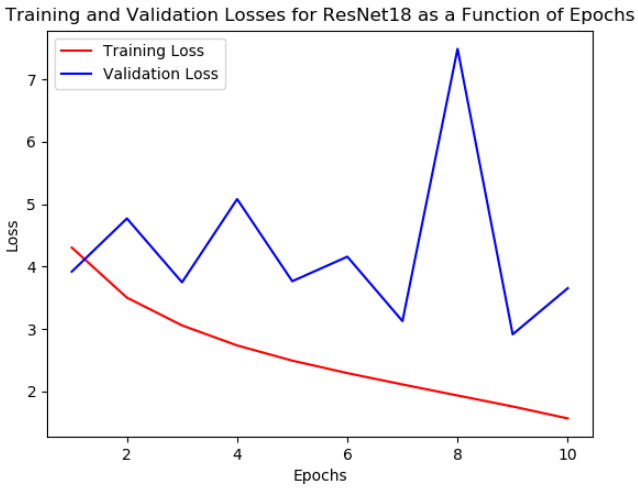

Figure 32: Plot showing the top-5 loss of the original network for training and validation sets as a function of epochs.
Architecture: ResNet18, **Learning Rate**: 1,
Optimizer: SGD, **Dropout Layers**: False

1.4.2 Assessing our Results

Using both the addition of a dropout layer before the fully connected layer as well as an ADAM optimizer, we actually realize optimal performance for this MiniPlaces challenge. Using the combination of these tuning mechanisms helps to enable generalization and stable learning by both decreasing the interdependence of the weights on one another (mitigated by dropout), as well as enable for the taking of more appropriate step sizes when updating the weights/parameters using backpropagation (due to ADAM). **Using the combined alterations of dropout layers and ADAM optimizers, we see significant outperformance (when the architecture is ResNet18) compared to no dropout/SGD optimizers. The optimal step size here (which yielded the overall optimal configuration of parameters as well) was 0.01.**

2 Problem 2: The Mini Places Challenge

2.1 Specification of Final Submission

For my final MiniPlaces challenge submission, I used the following neural network architecture, hyperparameters, and training parameters:

- **Architecture**: ResNet18
- **Dropout Layers**: True
- **Learning Rate**: 0.001
- **Optimizer**: ADAM
- **Training Epochs**: 10
- **Batch Size**: 128

Final Classification Accuracy: The final top-5 classification accuracy for this neural network is **0.7171, or 71%**.

Below is some background information about the neural network architecture I used for this problem, known as ResNet18. This neural network is one of many networks that falls under the class of residual neural networks, a kind of network that utilizes skip connections, such as the ones shown below. These skip connections are utilized so as to mitigate vanishing gradients, and have been proven to speed up the training process by reducing the likelihood of the network finding an incorrect "local minima", as well as mitigating the potential for vanishing gradients.

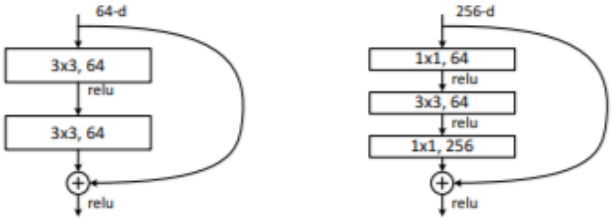

Figure 33: Visualization of the skip connection commonly seen in residual neural networks. These skip connections are typically made over ReLU and batch normalization layers.

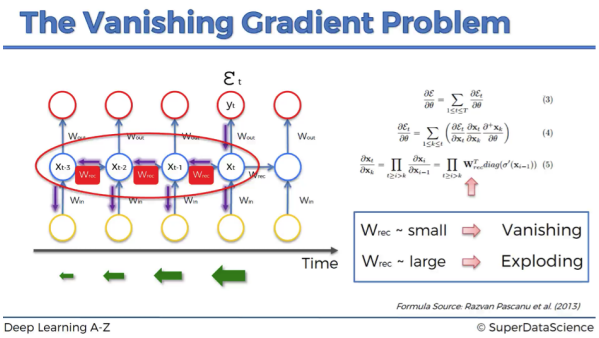

Figure 34: Illustration of the vanishing gradient problem that is especially prevalent when training deep networks, and was one of the main motivations for the development and use of residual neural networks.

The architecture for ResNet18 (an 18-layer residual neural network) is illustrated below:

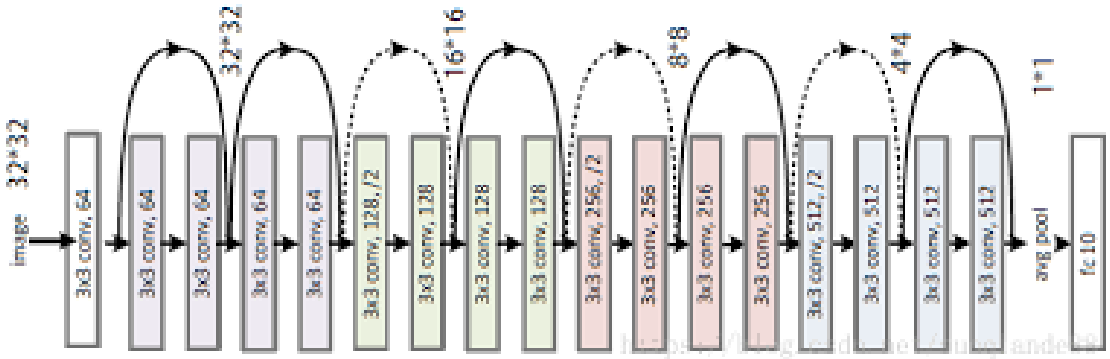

Figure 35: ResNet18 Neural Network architecture. A key innovation used in ResNet for image classification is the use of skip connections - a.k.a. activations that skip layers of the network entirely.

2.2 Model Selection Process

For selecting my final model and combination of parameters, I chose to leverage the resources offered through accelerated GPU computing to test a variety of different combinations of hyperparameters and neural network architectures. I chose to train each of these combinations for 10 epochs, and used Linux's screen feature to run multiple training sessions in parallel. My final decision to determine which network/hyperparameter combination I would use for this assignment was determined by the combination that minimized out-of-sample/validation & test error after 10 epochs. Rather than determining which features worked individually and then simply taking the better of each possible binary set of parameters (e.g. dropout/no dropout, SGD/ADAM, or ResNet18/ResNet101), I decided to use the combination of these different network tuning mechanisms which overall yielded the best performance (which, as I will discuss below, is lowest validation error).

To make more informed choices of parameters/hyperparameters, I used plots of training and validation top-5 error as a function of epochs (the graphs above) to help guide my decisions for which parameters to tune. For example, if I noticed that training and validation error were barely changing as a function of the number of epochs they were trained for, this would be indicative of too small a learning rate; likewise, if the training and validation errors were varying dramatically, or becoming unbounded, this was perhaps indicative of too high a learning rate. These observations were made especially for when training ResNet101, since the vanishing/exploding gradient phenomenon was much more prevalent here (though again, using residual networks can help to mitigate at least the vanishing gradient problem).

This project was a lot of fun; thank you for including it in the course!