Report

alikhanimjd

November 2020

Contents

Abstract

With ever more promising advancements, neural networks are widely adopted as the solution to many problems in the realm of artificial intelligence. Significant leaps in performance owed to innovations in architecture in both computer vision [12, 9, 19] and Natural language processing [24, 1] have been witnessed to corroborate the effectiveness of neural networks. However, applying even out of the box models[27] requires experience, great attention to details and knowledge of various techniques to successfully train and optimize the models. Obstacles such as shortage of data, sub-optimal local minima [11], over-fitting, etc., are specific to each individual problem and cannot use a general all-purpose solution. Moreover, improving accuracy of a solution after establishing a functional baseline demands problem specific engineering. This work is an attempt to investigate and explore pivotal innovations in computer vision architectures and some engineering techniques to enhance the performance of the model on reference datasets.

1 Introduction

This section of the report briefly discusses the problem and the objectives. This course touches on a wide range of topics in modern neural network, especially in the image processing branch from a practical point of view. Each of these topics is the subject of seminal literature and deserves to be studied separately. The goal is to get familiar with common practice in application of techniques and discuss the results.

Neural networks research has been experiencing staggering progress the likes of which is never seen before. Each year many innovations that are aimed at shortcomings of the previous methods prove effective and are reflected in the literature. Number of published articles containing the query word "deep learning" goes as high as 28000 in the year 2016 [29] and only continues to grow year after year. A very interesting characteristic of the field is the applicability of the latest break-throughs in many industries with small latency after publication. Moreover, many innovations require engineering extensive rounds of experiments and trials to exhibit favorable results. This engagement between research and practice makes it inevitable for deep learning researchers to be practically keen and meticulous as well.

Here we focus on the state of the art deep learning models for image processing. The success of AlexNet [12] invited a lot of attention to be invested in image processing with neural networks. Especially application of Convolutional Neural Networks(CNNs) became the focus. Models such as VGG net [19], U-Net [18], Inception [21] are examples of CNNs to enhance the performance in various image processing tasks. These models principally focus on deeper sequences of CNNs to take advantage of higher level feature representations. Very deep models although very promising, are notoriously difficult to train [6]. ResNet architecture [8], discussed in details below, overcomes certain difficulties by a very transferable solution. The versatility of the model is an appeal to implement and conduct experiments with.

Besides the architecture of the model which explores solution spaces, there are intricacies in the optimization process that can speed up the convergence and prevent issues such as overfitting. A ResNet model is used to explore these methods.

The datasets used are Fashion-MNIST [28] and CIFAR-10 [26]. The details and examples of the datasets are provided in the background section.

Finally, the YOLO paper [15] as well as it's implementation is discussed. The choice of YOLO to dive into is made because of comprehensiveness and integration of many techniques.

2 Background

2.1 ResNet

Research has shown the promise of deeper models [19]. Training very deep models has its own difficulties as mentioned before. Deep Residual Learning for Image Recognition [8] introduces an architecture which allows for very deep model

to be tractable and live up to their promises.

Simply stacking layers together can be a straightforward alternative for complicated architectures. Normalization and various weight initialization methods obviate the problem of vanishing and exploding gradients [6] to a good extent. However, very deep models have appeared to suffer a saturation and degradation of accuracy even on the training set [7]. Interestingly enough, a deep model constructed from an intermediate model by only adding identity layers exhibits better performance compared to other more complicated deep models. To elaborate, suppose a model with n layers that works well on a data-set. A new model with m layers ($m > n$), due to larger capacity, is supposed to be better at learning the mapping. To ensure comparable performance, the models are from the same family with the only difference being in the number of layers. To be at least as good as the shallower model, the deep model can assign the first n layers to be copied from the smaller model and the remaining $m - n$ layers can learn identity mappings. To defy our intuition, this larger model exhibits lower train and test accuracy. This new roadblock calls for a solution other than normalizing layers.

The problem stems from intractability of a large model in optimization. Intuitively, it is harder for the larger model to learn the mapping because there are more parameters to be learned. Suppose a mapping $H(x)$ needs to be learned through a sequence of layers. Setting aside the what has been learned so far, the sequence of layers can focus on the residual that needs to be learned. Formally the mapping $F(x) = H(x) - x$ would be the focus of current layers to learn. By allowing the model to directly copy from previous layers (as shown in fig 1), the current sequence of layers may have an easier time finding the optimal set of parameters. In the extreme case if the desired mapping is identity function, it is easier to drive the parameters in $F(x)$ to zero rather than directly finding a constellation of $H(x)$ parameters that render the identity mapping. This identity mapping is a type of what is referred to as

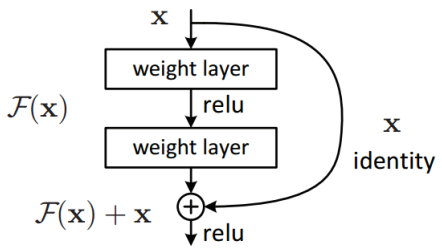


Figure 1: Residual Connection. From [8]

"shortcut connection" and shortcut connections are not a fundamentally new idea [2, 16, 20]. A computational upside of this connection is addition of no extra parameters or computational complexity.

It has been shown that with this approach, better accuracies can obtained simply by increasing the depth of residual nets on ImageNet and CIFAR-10 datasets [8].

Previously literature had explored the potential of reformulation in optimization problems such as solving PDEs [23, 25] and image recognition [10]. Moreover, shortcut connections have been experimented with before. In [17, 30] a linear layer is used to directly connect the input to the output. The famous Inception model makes use of a shortcut branch among other branches [22]. Highway networks utilize shortcut connections between layers with gating functions where these gating functions vary with the data. This adds more computation and may amount to zero depending on the incoming data. Highway connections have not shown improvement with the increase with increase of depth.

Residual learning The motivations is to find the validity of the hypothesis that learning $F(x) = H(x) - x$ is actually easier than $H(x)$. The fact that adding more layers even though they only need to learn an identity mapping yields a larger error is suggestive of difficulty of imitating identity mappings with non-linearities. It is needless to say that identity mappings do not cover a comprehensive set of functions that we wish to learn. However, it would be worthwhile to find out if treating them separately provides a suitable preconditioning.

Fomrally, a residual connection works as follows:

$$H = y = F(x, \{W_i\}) + x \tag{1}$$

This adds no additional burden on computations offering opportunities for comparison between networks with and without residual connection. F and x would have to be of the same dimension. To preserve degrees of freedom. Sometimes a mapping to the desired dimension is preformed by multiplication $W_s x$:

$$y = F(x, \{W_i\}) + W_s x \tag{2}$$

Experiments show that this additional mapping is not essential for learning and is simply performed to change dimensions.

2.1.1 Architecture

The core of network is from the family of VGG networks [19]. At least 18 layers of convolutional and pooling units are sequenced. The convolutions mostly have a kernel size of 3. For downsampling, a maxpool with window size 2 is used. If the feature map size is modified, the number of channels will be adjusted to offset the change in computation and keep it's cost constant. Figure 2 is a summary of ResNet architectures.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2: ResNet Architectures. From [8]

The 34-layer for example starts with a convolutional layer of size 7 with 64 filters that downsample the high resolution input image. The output undergoes a series of 64-filter 3×3 convolutions with a shortcut connection between each two. The following set of convolutions starts with 128 filters of 3×3 convolutions that downsample the feature map by a factor of two, using an average pooling layer. Same set of convolution is performed without downsampling for 7 times. After the initial 7×7 convolution, a shortcut connects input and the output of every two subsequent layers. This is implemented as "block" object in our implementation to avoid redundancy. Eventually the predictions are made after a 1000-way fully connected layer. Batch normalization and ReLu nonlinearities are carried out after each convolution block respectively. Dimensions of the feature map are adopted after each downsampling. 1- Simply padding with zeros. 2- projecting via 1×1 convolutions.

One of the interesting results of the [8] is table 1 where result of a ResNet and it's plain counter part on ImageNet is shown. Although the solution space for 34 layer plain neural network includes that of 18-layer plain network, its error rates are higher indicating the difficulty of optimization. Finding the main reason for this difficulty in optimization was left for future studies. Moreover, both 18 layer plain and residual networks exhibit similar errors. The difference is in the convergence time. When it comes to type of shortcut layers, experiments show comparably good results from paddings as well as projecting the feature map ($W_s x$) to change the size. Therefore, projections implementing the ($W_s x$) term in equation are only used when downsampling. The ResNet model implemented in this project is ResNet-50 version which is analogous to ResNet-34 only with each block consisting of 3 convolutions rather than 2. A batch normalization after each convolution is performed along the filters dimensions. For both datasets (CIFAR-10, MNIST-Fashion), the original size of pictures are too small so they are upsampled before feeding to the model. The optimization algorithm used is AdamW with the initial learning rate of 0.001. Different initialization

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	25.03

Table 1: Table 2 from [8]. Results on ImageNet.

Figure 3: examples of fmnist

methods are experimented with to observe the effects of each. For the MNIST-Fashion dataset, the model is trained for 10 and then 20 epochs for augmentations, and tested on the test dataset. Train and test batch sizes are chosen to be 32 and 16 respectively. Cross Entropy loss is the choice for loss function.

2.2 Data

MNIST-Fashion dataset [28] is from family of MNIST [13] and consists of gray scale 28×28 images for classification into 10 classes of fashion products such as shoes, dresses, etc. The data set contains 60000 images divided into train and test sets with ratio of 5 to 1. A sample of images is shown in figure 3.

CIFAR-10 datasets contains the same number of train and test images. The images are RGB-scale of size 32×32 with each example containing an object (Animal, tree, car, etc.) and the name of object as label. Figure 4 shows a sample of CIFAR-10 images:

3 Experiments

3.1 ResNet vs NonResNet

A ResNet-50 architecture shown figure in 2 and it's non-ResNet counter part are implemented to observe the effects of the ResNet on the CIFAR-10 dataset. Li et al [14] studies visualiation techniques to observe the landscape of neural networks' loss function and sure enough, the non-ResNet curve for a deep network is more difficult to traverse than that of the ResNet model illustrated in figure 5. Comparison of ResNet and non-ResNet implementations in our experiment are in line with figure 5. For an arbitrary layer, histograms for the activation values and gradient values are depicted in figure 6. The axis into the screen is the epoch number of the training. The activation values for the ResNet model move in a more stable, consistent fashion towards the desired distribution as a result of traverse a smoother loss curve. Meanwhile, the values for the non-ResNet counter-part change abruptly with some oscillations, indicating the spiky surface of the loss curve. Looking at the gradient values, we can notice the larger range of values for the no-ResNet version which can again be attributed to the bumpy surface with large gradients. This results in larger steps during training and therefore oscillations in output values of the layer. The smoother surface of the ResNet version helps with lower gradient values and small consistent steps towards the minimum.

3.1.1 MNIST-Fashion Classification

As mentioned the model is a ResNet-50 trained on MNIST-fashion. With only 10 epochs, the network is already overfitting as a result of a simple distribution in gray-scale. Figure 8 depicts the train and evaluation losses. The

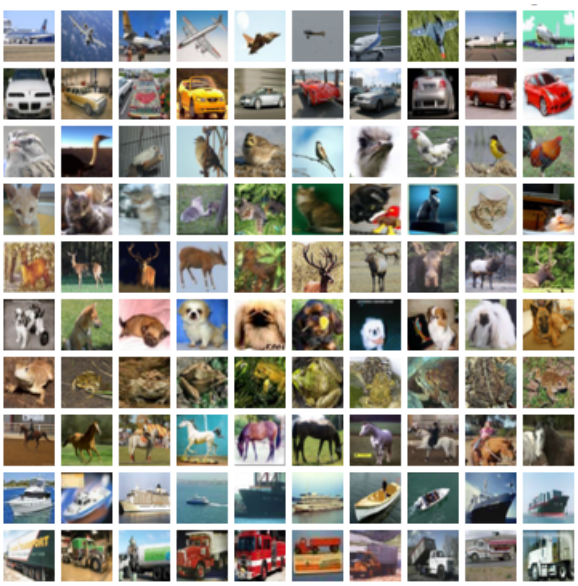


Figure 4: examples of CIFAR-10.

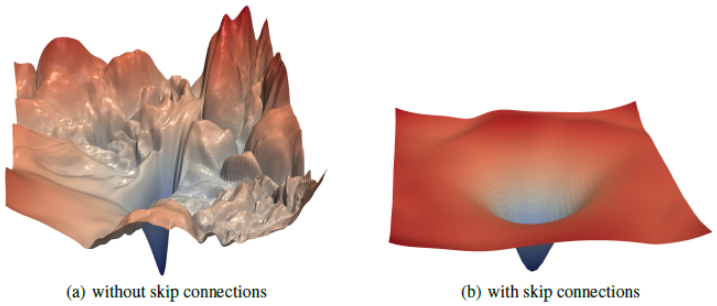
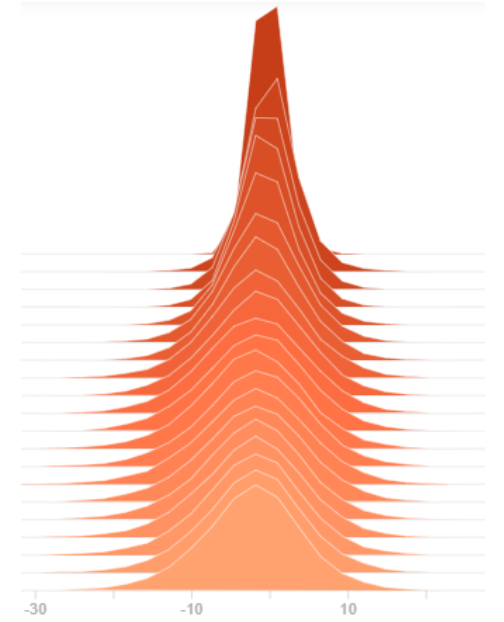
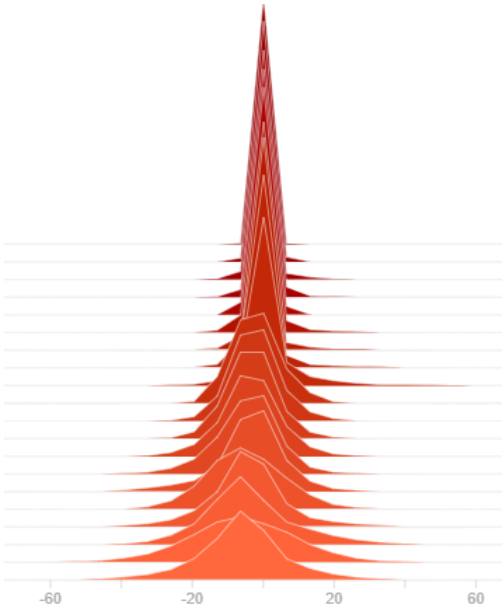


Figure 5: Loss curve landscapes from [14]

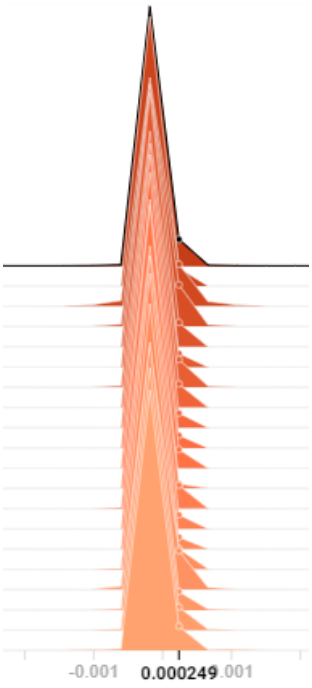


ResNet activations

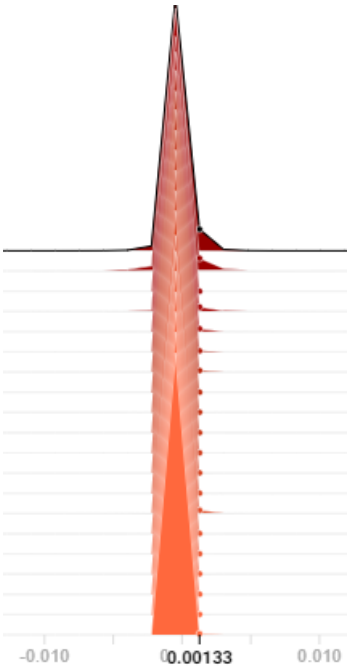


non-ResNet activations

Figure 6: Comparison of activations between a ResNet and non-ResNet model

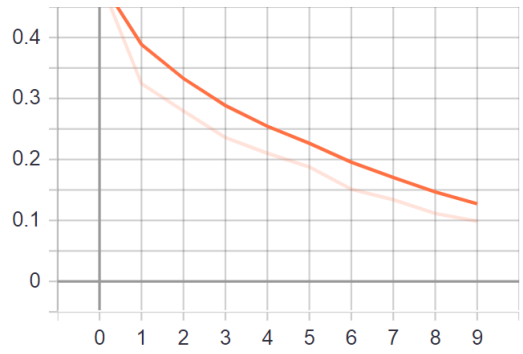


ResNet activations

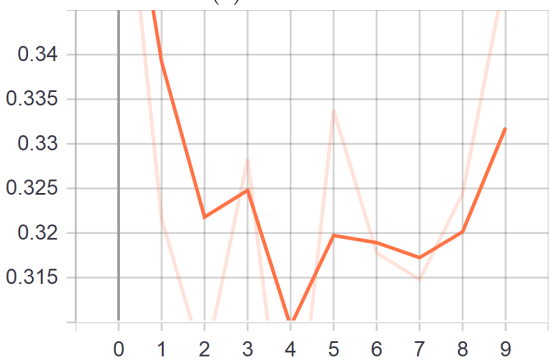


non-ResNet activations

Figure 7: Comparison of gradients between a ResNet and non-ResNet model



(a) train loss



(b) validation loss

Figure 8: train loss for for MNIST-Fashion without augmentation

best performance occurs on epoch number 5 before overfitting happens with accuracy of 89.55% as shown in figure . The training loss decreases constantly. However, while the evaluation loss increases after overfitting the accuracy keeps increasing too. This can be attributed to some flaws in accuracy metrics. For example in here, for a training point with label y the accuracy will increase as long as out of the softmax function is larger than 0.5 for y. However this still adds up to the loss value since for example probability 0.51 is not a very confident prediction. Please refer to my article at Medium-article for more details.

Next experiments with various augmentations are discussed. Train time augmentations are random transformations that preserve the label of the image while making enough changes to the image to be considered a new training example. Here, 6 set of augmentations are tested. Each set of augmentations is chosen so that they perform similar operation in order to make comparisons. For each training batch there is a 60% chance that any augmentation operation will be carried out.

Augmentation 1 Each example will go through two sequential steps in random order. The first is either a dropout of pixels with probability between $(0.01, 0.1)$ or coarse dropout of pixels with probability in range $(0.03, 0.15)$. A coarse dropout would set rectangular areas in the image to zero while dropout only acts on square patches. This is predicted not to change much as a result of gray-scale color channel. The second step in this sequence is a combination of horizontal flip, rotation up to 5 degrees and cropping or padding by 25% on each side. The rotation and cropping or padding is done 50% of the time. The combination may include none or all of the augmentations. An example of this first set of augmentations is shown in figure 10. Figure 11 shows the results. An important observation is the smoothness of validation loss. Although the validation accuracy suffers a negligible drop, the loss plots are more stable and model is learning consistently. It is not overfitting even in larger epochs and exhibits better generalization. Although the model may have a harder time in learning, having seen perturbations in images arms the model with ability to recognize new images even in presence of occlusion, rotation, etc.

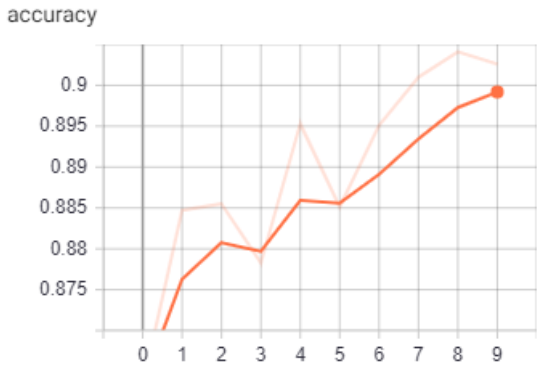


Figure 9: Accuracy for MNIST-Fashion without augmentation

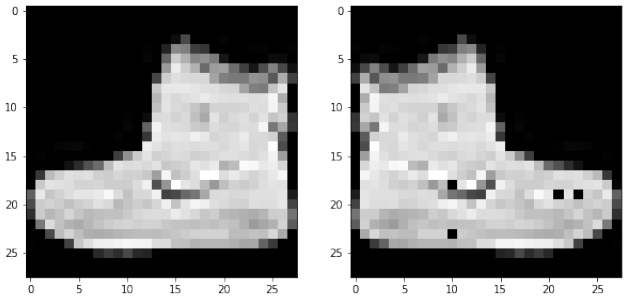
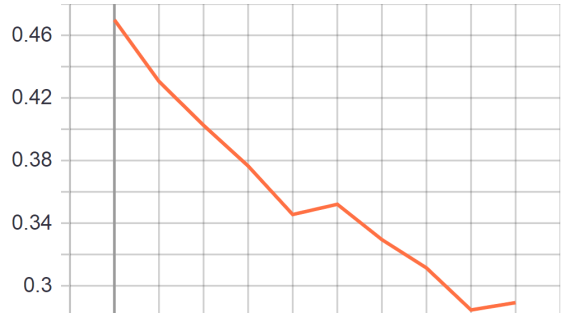


Figure 10: Example of augmentation 1

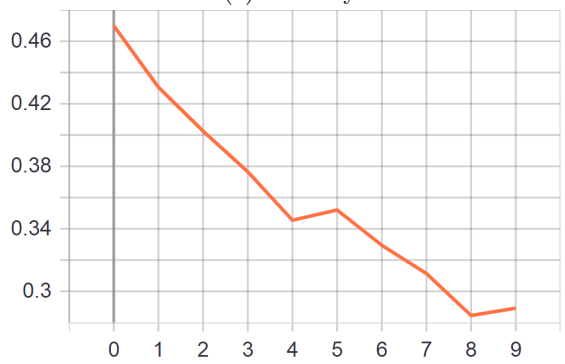
Augmentation 2 This augmentation is a sequence of up to 3 transformations from the following. Adding to each pixel a value between (-10, 10). Addition of gaussian noise with scale between (0, 0.2×255). Multiply each pixel value with a value between 0.5 and 1.5. Drop out with maximum probability of 0.1. Filling out up to two random areas in the image with value 255. The final item in the sequence is replacing 30% of pixel values v with value (255 - v) (Called inverting). The result proves improved generalization ability as the validation loss is below that of training loss for 9 epochs. Example of this set is shown in figure 12 The accuracy corresponding to best validation loss drops down to 78% (fig 13). This suggests that a milder version of augmentations will make less changes which helps the parameters to be more tuned for he validation distribution. Also this suggest for stranger augmentations it may be beneficial to increase the epochs.

Augmentation 3 This next set of augmentations performs one of Gamma contrast, Sigmoid contrast and Logarithmic contrast. The pixel values are adjusted according to each distribution. An example of this augmentation is shown in figure 14. As a feedback from previous experiments the number of training epochs is increased to 20. The plots in figure 15 show the sufficiency of this increase to 20 as the model overfits past epoch 14. Almost the same trend happens to the accuracy plot with maximum accuracy of 91.26% occurring in epochs 13. Addition of contrast helps the model as it would help human eye to distinguish the object from the borders and dark regions.

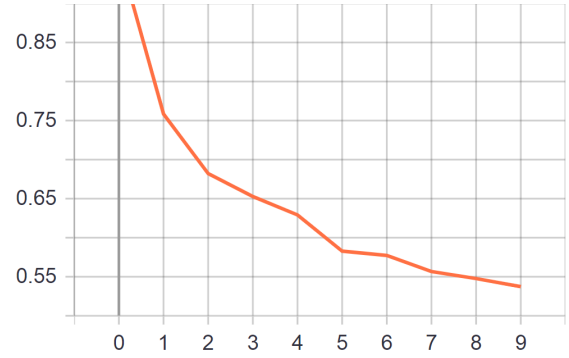
Augmentation 4 This set of augmentations operate similar to convolutions kernels and emphasize on detection of the edges. Up to two transformations from the following are selected to operate. A sharpening with $\alpha \in (0.0, 1.0)$ and a lightness between 0.75 and 2.0. An augmentation that embosses the image with $\alpha \in (0.0, 1.0)$ and strength value in (0.5, 1.5). Edge detect is another transformation which detects the edges via turning image into a black and white version and overlays the results on the original one. Lastly, direct-edge detection would embolden the edges in certain directions and again overlay the results on the original image. Figre 16 is an example of an original image with augmented version side by side. This set of augmentations stands out among the rest in that it does not actually perturb the distribution in images to help the model get familiar with irregular data points. It, on the other hand, makes learning easier for the model by providing additional information with regards to position of the



(a) accuracy



(b) train loss



(c) validation loss

Figure 11: Loss and accuracy for 1st set of augmentations on MNIST-Fashion

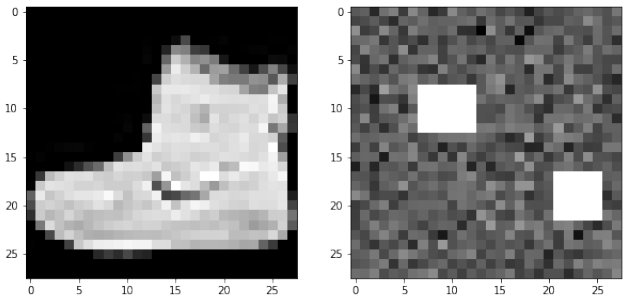
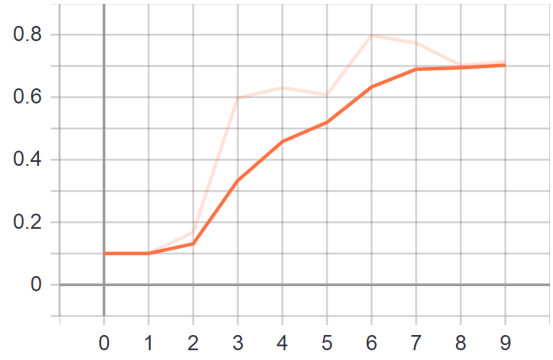
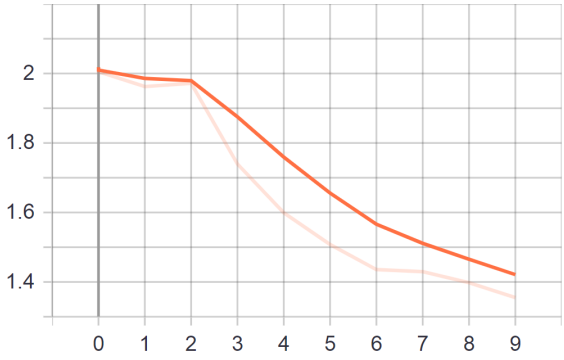


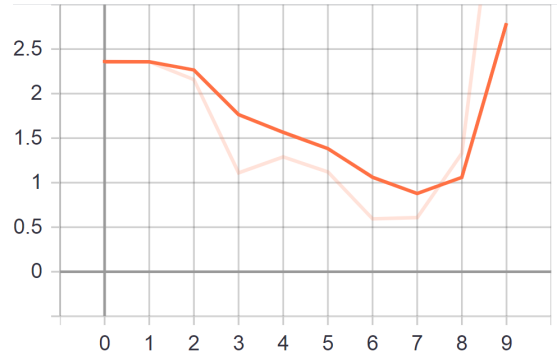
Figure 12: Example of augmentation 2



(a) accuracy



(b) train loss



(c) validation loss

Figure 13: Loss and accuracy for 2nd set of augmentations on MNIST-Fashion

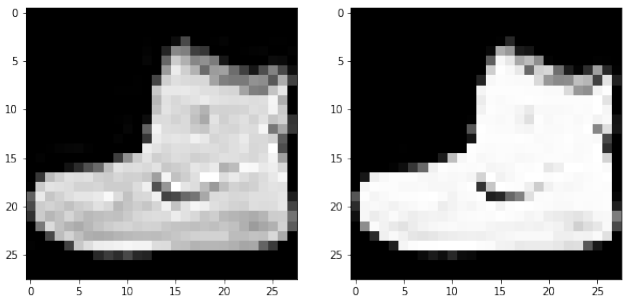
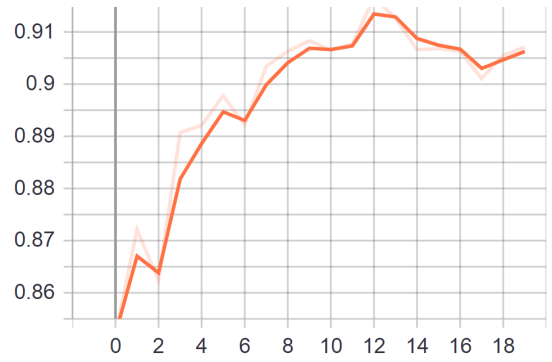
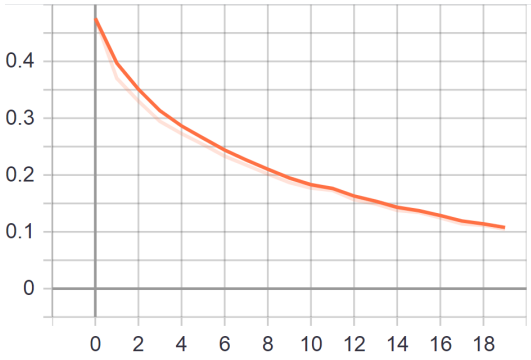


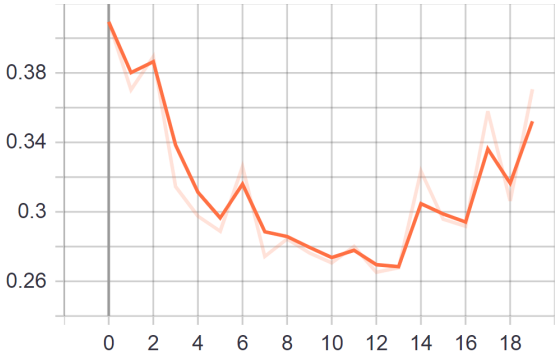
Figure 14: Example of augmentation 3



(a) accuracy



(b) train loss



(c) validation loss

Figure 15: Loss and accuracy for 3rd set of augmentations on MNIST-Fashion

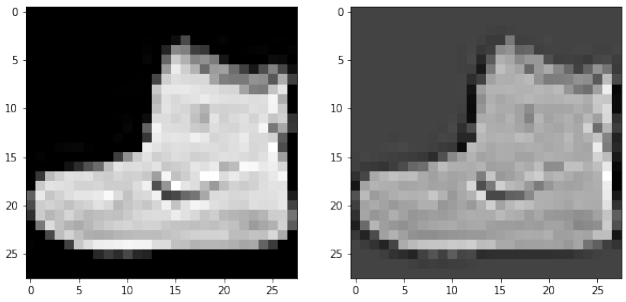


Figure 16: Example of augmentation 4

edges. The transformation operate similar to convolution filters which are removed during validation. This in effect presents the model with a larger quality set of images in validation resulting in high validation loss, overfitting and low accuracy:

Augmentation 5 This family of augmentations would only flip the image over the horizontal or vertical axis. Either one or both of these flips are performed. The accuracy and loss plots are shown in figure 18. Epoch 13 marks the highest accuracy of 90.87% before overfitting. These results are not very different from the non-augmentation training.

Augmentation 6 The next set of augmentations is from family of geometrically affine transformations. Three of the following set of augmentations are applied in a random order. First is a scaling transformation with a factor between 0.5 and 1.5. Next is translation of image in both x and y direction up to 20%. Next on the list is rotation up to 45 degrees followed by shearing images up to 16 degrees in either x or y dimension. The last two are piece-wise affine transformation that only operate on a random patch of image. These are piece-wise scales up to 5% and padding up to 10%. An example is shown in figure 19 The validation loss does not really follow its training counter part although decreasing overall (20). This could be due to heavy changes in distribution after the augmentations. The first speculation is that the epoch during which validation loss increases, especially the one with overshoot, piece-wise augmentations are selected multiple times. These along with for example a 45 degree rotation construct new images which makes it hard for the model to draw similarity to the original image example. These changes in training examples is especially problematic for high capacity models such as ResNet50 on a rather modest MNIST-Fashion dataset.

3.2 Weight Decay

Similar to data augmentation during training, weight decay can partially mitigate overfitting. Very large models have numerous parameters exposing them to generate a very complex and high degree function to fit simple datasets. Reducing or removing a random subset of those parameters can can reduce model's complexity. This for example, is achieved by drop out layers. Analogous to dropping out a weight parameter, one can reduce the weights to very small values via penalizing weights through an added term in loss function.

$$Loss(w)^{new} = Loss(w) + \sum \frac{\lambda}{2} \times w_i^2 \tag{3}$$

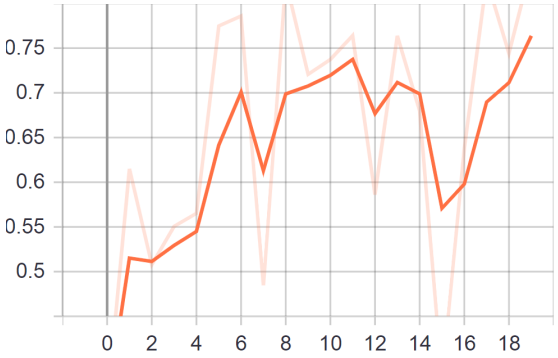
Equation 3 shows an example of loss function with the added term $\sum \frac{\lambda}{2} \times w_i^2$ to penalize large weights values (w). This term will efficiently limit model's freedom to overfit noise and outliers. This same penalizing term is added in learning Fashion-MNIST classification. It can be seen in figure 21 weight values of layer 34 in the version using weight decay are limited in a smaller range the right end of which is almost 10 times smaller than its counterpart with no with decay. This along with similar pattern in other layers results in less overfitt supported by the plots in figure 22

3.3 Initialization methods

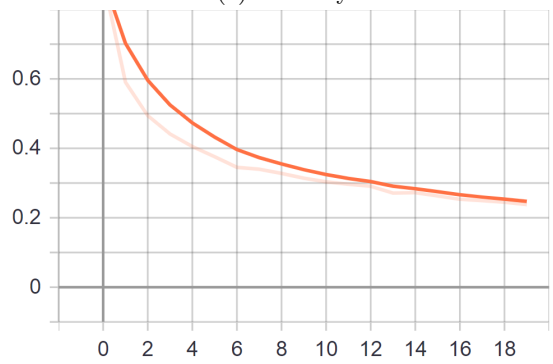
4 initialization methods are tried and results are compared to the experiment without any augmentations. A good initialization can reduce overfitting and help with generalization. This is due to the fact that initialization methods attempt to prevent the variance of each layer's output from blowing up or diminishing.

Normal(Standard) Initialization Does not help much. This initialization sets the mean and standard deviation of weight in each layer to zero and one respectively. It overfits almost immediately. Validation loss and turnes U-shape and train loss almost plateaus after the 5^th or 6^th epoch. Finally validation accuracy was not better than 61.38% (fig 23).

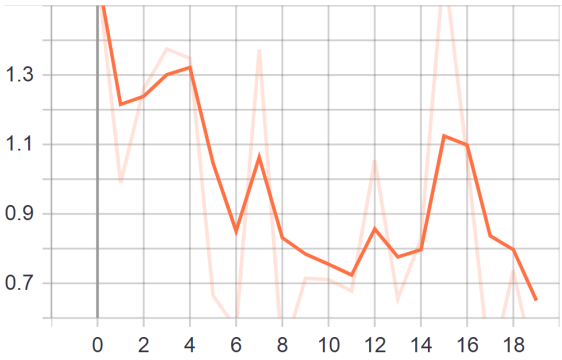
Xavier Initialization Xavier initialization actually mitigated overfitting although not much changed in terms of accuracy. The plots in fig. 24 suggest that training for more epochs will lead to better validation loss and accuracy.



(a) accuracy

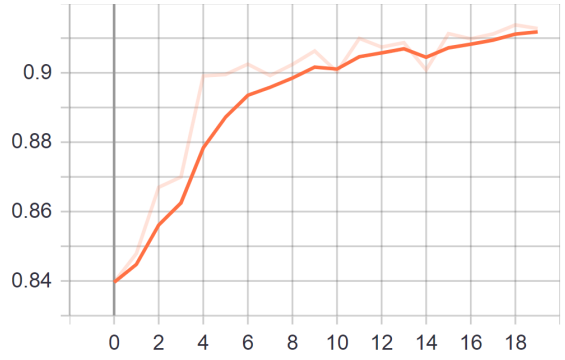


(b) train loss

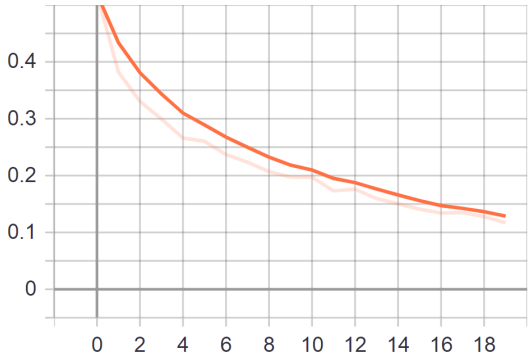


(c) validation loss

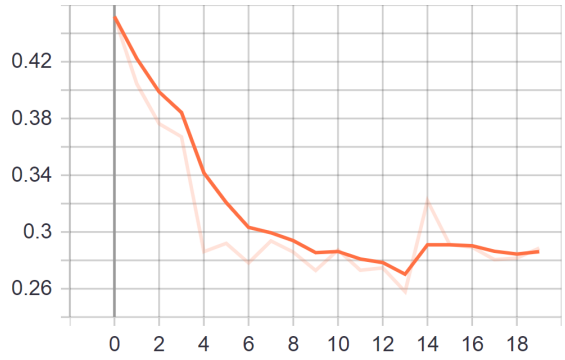
Figure 17: Loss and accuracy for 4th set of augmentations on MNIST-Fashion



(a) accuracy



(b) train loss



(c) validation loss

Figure 18: Loss and accuracy for 5th set of augmentations on MNIST-Fashion

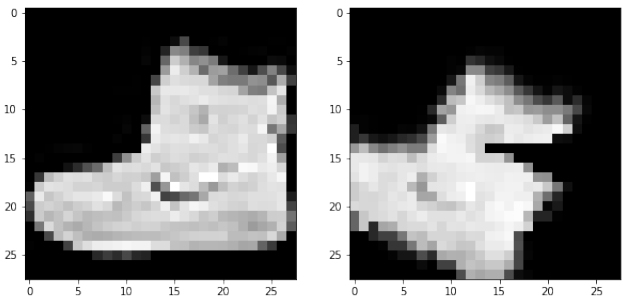
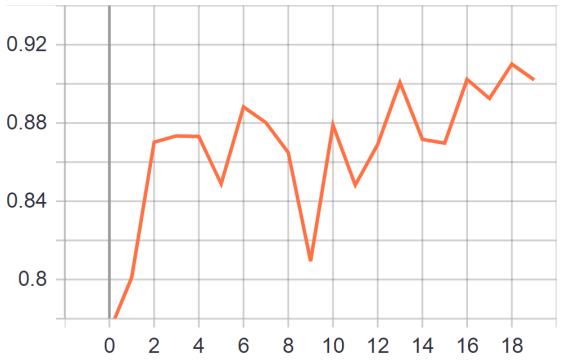
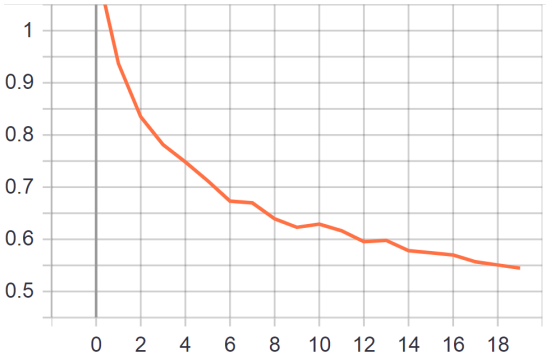


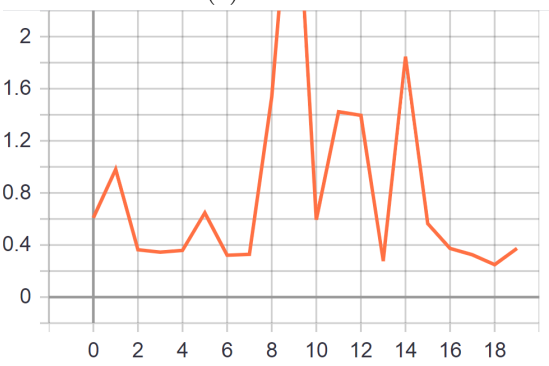
Figure 19: Example of augmentation 6



(a) accuracy

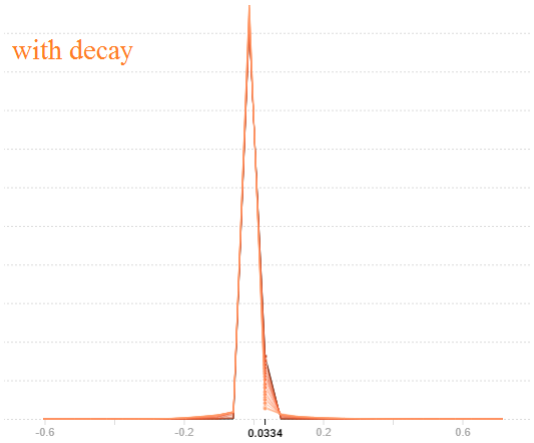


(b) train loss

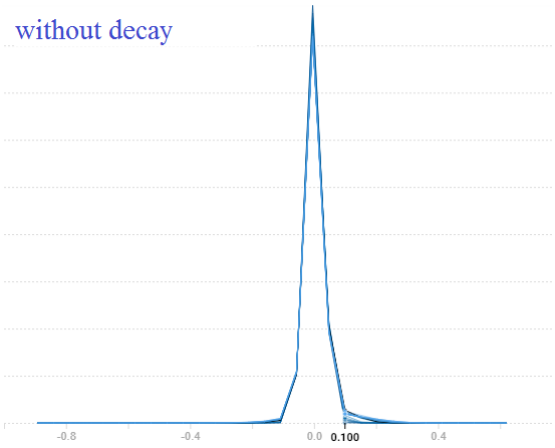


(c) validation loss

Figure 20: Loss and accuracy for 6th set of augmentations on MNIST-Fashion

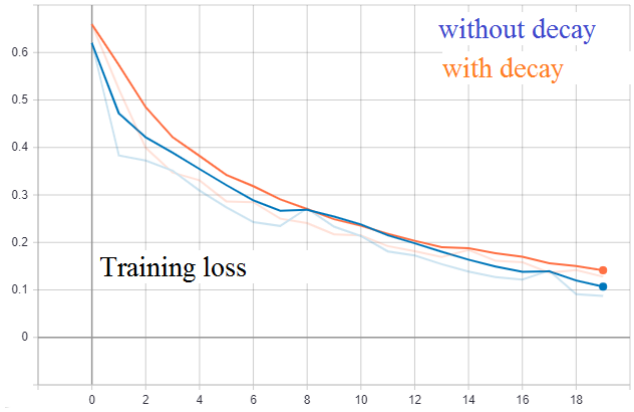


(a) weight distribution in layer 34 with weight decay

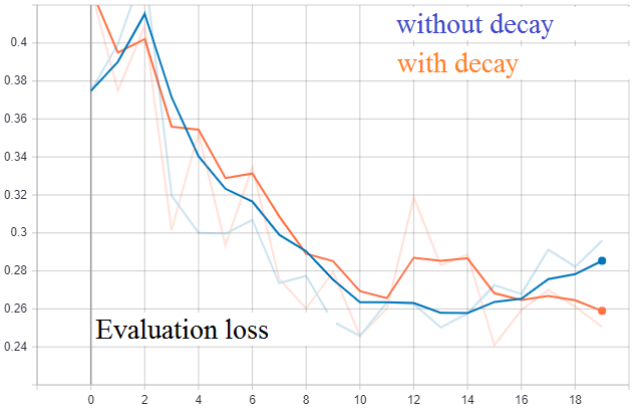


(b) weight distribution in layer 34 with weight decay

Figure 21: comparison of weight distributions with and without weight decay

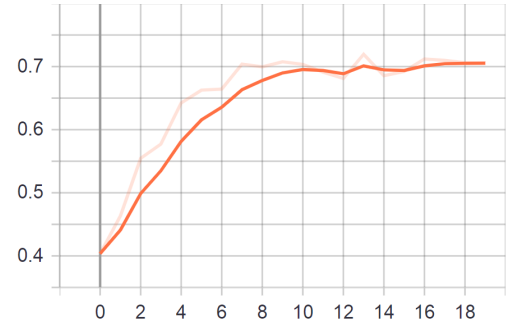


(a) training loss with and without weight decay

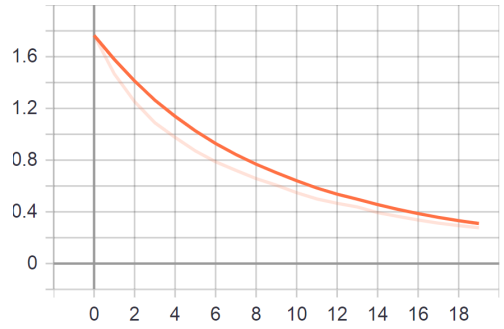


(b) Evaluation loss with and without weight decay

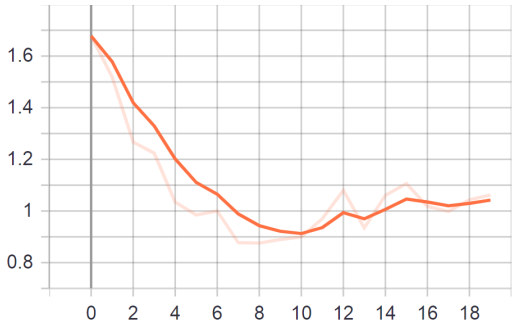
Figure 22: Overfitting- Without weight decay vs With weight decay



(a) accuracy

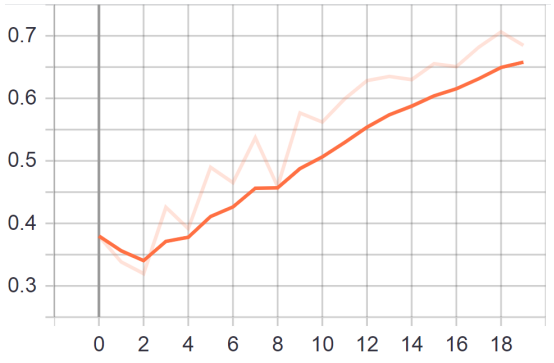


(b) train loss

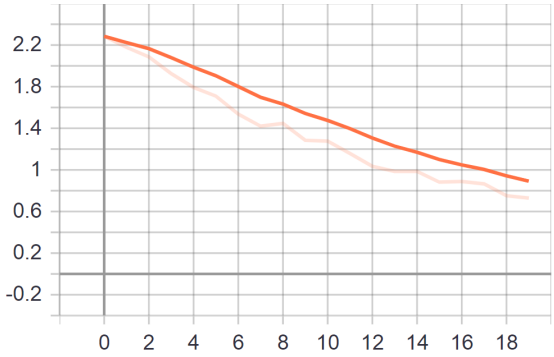


(c) validation loss

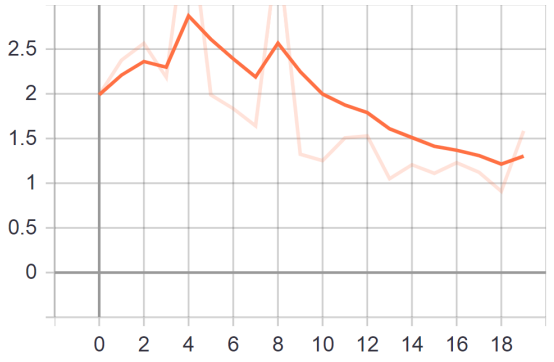
Figure 23: Loss and accuracy for Normal Initialization on CIFAR-10



(a) accuracy

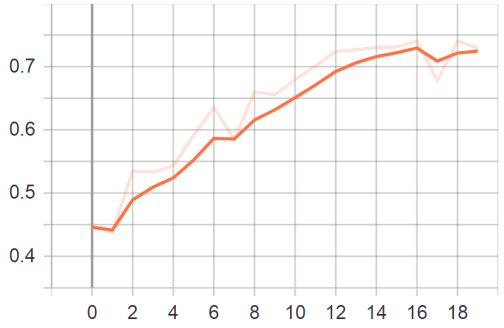


(b) train loss

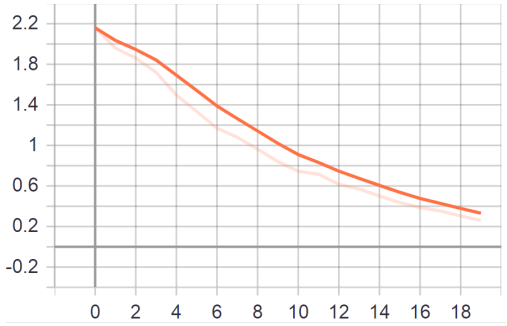


(c) validation loss

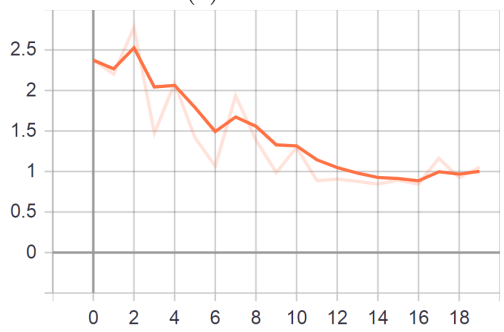
Figure 24: Loss and accuracy for Xavier Initialization on CIFAR-10



(a) accuracy



(b) train loss

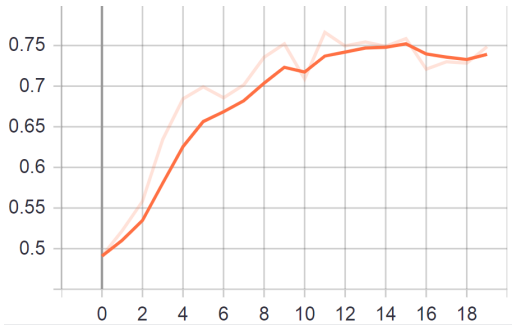


(c) validation loss

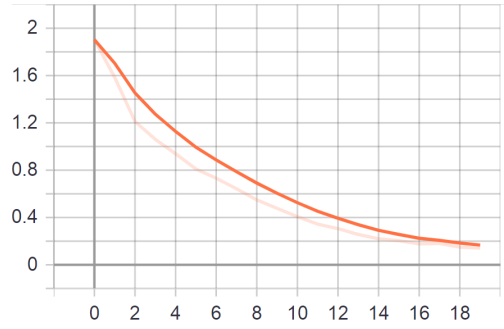
Figure 25: Loss and accuracy for Kaiming Initialization on CIFAR-10

Kaiming Initialization Works even better than Xavier, mostly because it is geared for ReLU which is the activation function of choice for ResNet. The accuracy enjoys a leap to 74.05%. The training loss shows constant improvement and never plateuos. It goes to show that gradients are not vanishing. The more stable validation loss indicates no explosion of gradients (fig 25).

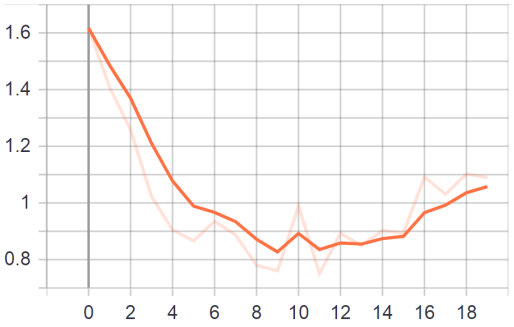
LSUV initialization The LSUV initialization as shown in fig 26 does converge very earlier compared to other initialization. Epochs 10-12 have the least validation loss and error with an accuracy of 76.62 which is better than all the rest of the initialization. Once again the training curve is stable with perceivable room for improvement. Investing in a good set of augmentations can probably be worthwhile in larger epochs to help with generalization. Figure 27 depicts the train loss plots for all initialization. Xavier, Kaiming and LSUV have initial losses close to loss value for normal initialization around the 7th epoch. All the former 3 initializations have similar initial values, however, LSUV is the one to better preserve the standard deviation due to its iterative algorithm, and have lower loss values through the end. To have a better understanding of different results due to different initialization we look at the evolution of output distributions through a few layers. Namely, the activation distribution for layers 1, layer 16, layer 48 for Normal, Kaiming and LSUV initialization are shown in fig 28. Xavier initialization is skipped due to similarity with Kaiming method and the fact that Kaiming is geared towards ReLU function used in experiments.



(a) accuracy



(b) train loss



(c) validation loss

Figure 26: Loss and accuracy for LSUV Initialization on CIFAR-10

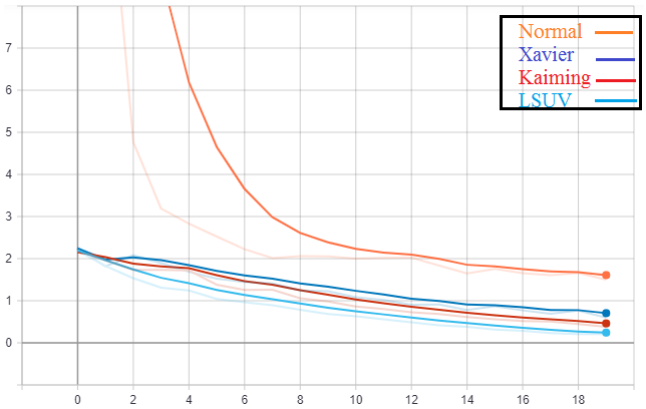


Figure 27: Train loss plots for Normal, Xavier, Kaiming and LSUV initialization methods

Inspecting the range of values closely, we can see how the distribution blows up from layer 1 to the 16th layer with range of values growing by factor of 5 for the normal initialization. Layer 48 is has also a scale up distribution compared to the first layer where the maximal values move even further away to the proximity of 200. This situation is alleviated in Kaiming and LSUV initialization. For Kaiming, less values are located in range (-50, -10) and (10, 50) in layer 16. The final layer's distribution is shrunk too with maximal values decreasing. LSUV initialization manages to preserve almost the same range of standard deviation through all layers. The big blow up in layer 16 does not happen. The largest values after 47 layers are only around 20.

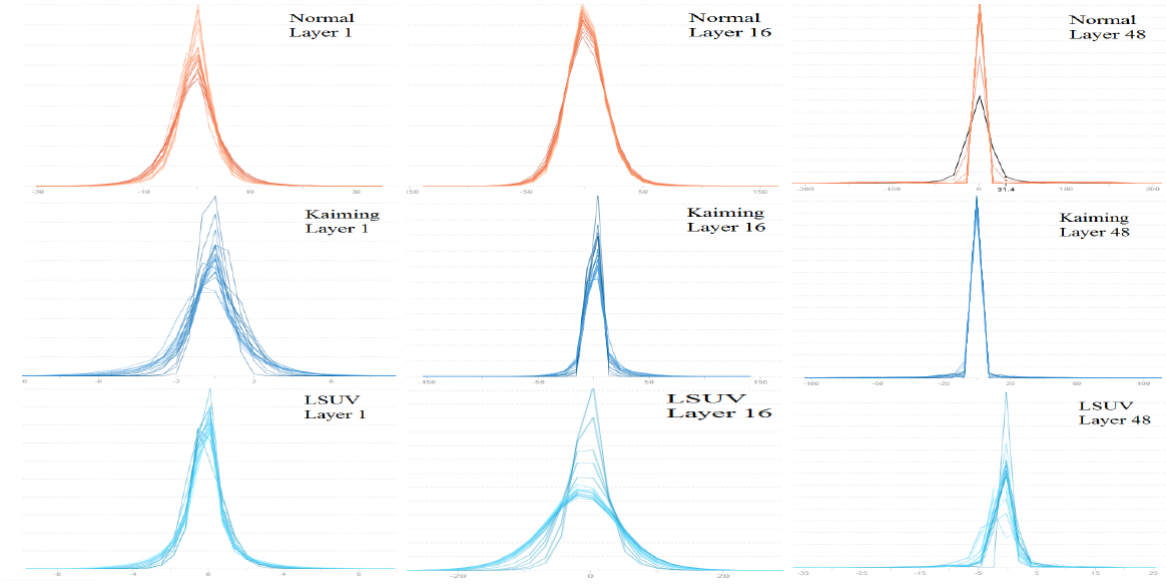


Figure 28: Evolution of distributions through layers for each initialization

3.4 Batch Normalization

Another issue with the way neural network's parameters are optimized is the sequential update of layers. For example, in fig 29, updating parameters in the second layer is carried out assuming the weights and values in previous layer remain constant. This is an inaccurate assumption. Weights in the first layer will update in the next step and as a result, the output value of the layer which is the input to the second layer will be different. This event is a surprise to the second layer as it not was updated for this new set of inputs. To suppress the surprise batch normalization is usually used to control the distribution of inputs to the layer. This, to some extent, will prevent the change of inputs. An experiment on an arbitrary layer in the implemented ResNet-50 illustrates the how values are more preserved in the same range with batch normalization (fig 30). On the contrary, these values grow asymmetrically towards negatives. Without batch normalization, the model is stuck in a loop, chasing an ever changing target. Therefore convergence is supposed to happen faster with batch normalization.

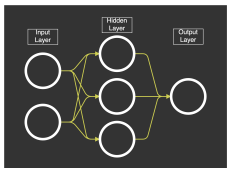


Figure 29: asynchronous update of layers

3.5 Test time augmentations

For this part a new model is trained on CIFAR-10 dataset for 20 epochs. Then the best model is used for inference on the test set for several iterations with each iteration conducting a set of augmentations and pooling. The idea is to provide the model with different variations of the input image to help better distinguish the label. So that for example if the model made a mistake on the image itself, it would correct that wrong prediction upon seeing other variations of the example. Test time augmentations are supposed to be deterministic as opposed to train time augmentations as a result of the difference

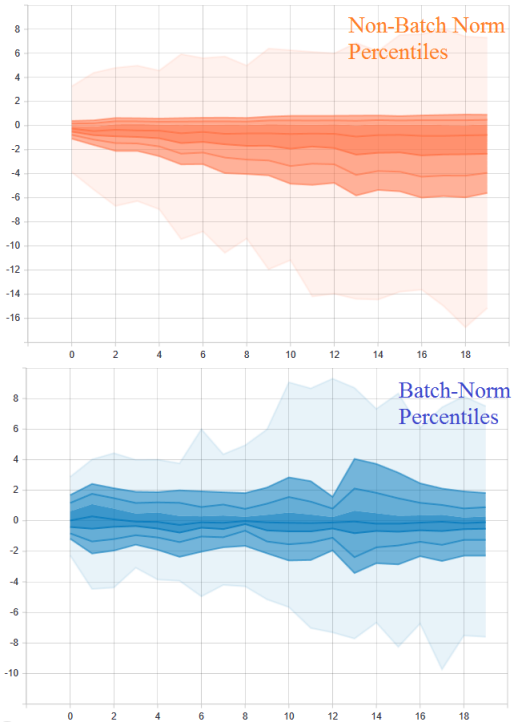


Figure 30: Evolution of percentiles through time for an arbitrary intermediate layer, with and without batch normalization

in the purpose of each. They are chosen carefully with little to no randomness from a set that seems more likely to enhance the model prediction. For example, changes in colors can impact the models accuracy as colors are usually dependable features to draw an inference. Here, the set of augmentations include: 2 degree counter-clockwise rotation; scaling by 1.2 along the horizontal axis; scaling by 1.2 in along the vertical axis; cropping 2 pixels from each border, etc. Each transformation is observed individually at first. Later on, combinations of individual augmentations were tried, however, individual augmentations outperformed. Even combining two combinations that improve accuracy individually decreased the accuracy compared to the superior individual. It is important to remember that usually test time augmentations are data dependant. Meaning, what works for one data-set may not necessarily be helpful on another, even with the same model. The baseline accuracy is 70.55% which has room for improvement, however, since we are only interested in comparison results, this baseline is moved on with. Fig 31 depicts the accuracy bars for each transformation.

The black bar corresponds to the baseline. An interesting observation is the scaling augmentations where magnifying the images results in improvement as opposed to reducing the size by 0.8. This is similar to effects on human vision. Rotations, even in the limit of 2 degrees is detrimental. Looking at scaling augmentations, increases in height of image yields more gains rather than increase in width. Translations of image are beneficial in both directions. It can be due to the fact that target objects are located more so in the center rather than the borders. Translation then removes some of the border pixels entailing unhelpful information. Similar effect is observed in cropping. Finally the outstanding gain comes from the horizontal flip. This could be the result of less invariance perceived by the model as relative to other augmentations. The same set of experiments are conducted with "average" polling and "max" pooling. The results in figures 32 show the average pooling to be slightly outperforming the max pooling. The better performance of the same model during inference as a result of test time augmentation comes at the price of inference time. For each image, multiple versions are run through the model. This extra time is not always affordable especially in real-time applications such as self-driving vehicles where a small lag can be disastrous. In the second set of experiments, transformations are incorporated accumulatively to illustrate the increase in time over the whole test set. Inference time almost doubles after adding the first transformation (fig 33).

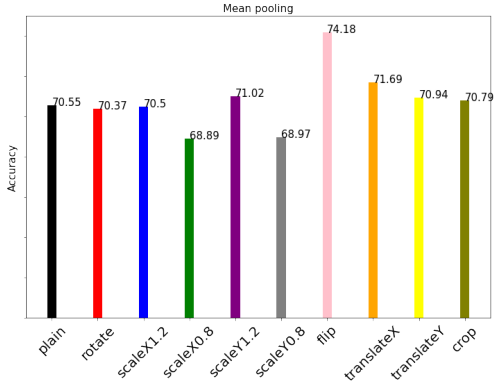


Figure 31: Individual test time augmentations with average pooling

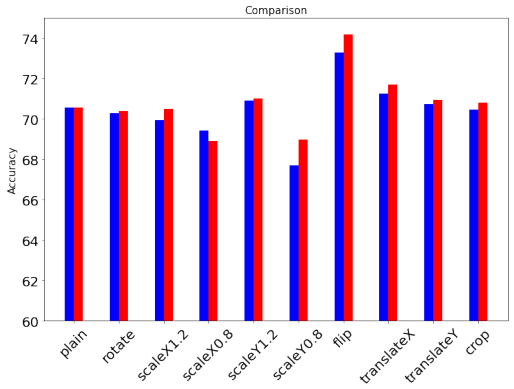


Figure 32: Individual test time augmentations with max pooling

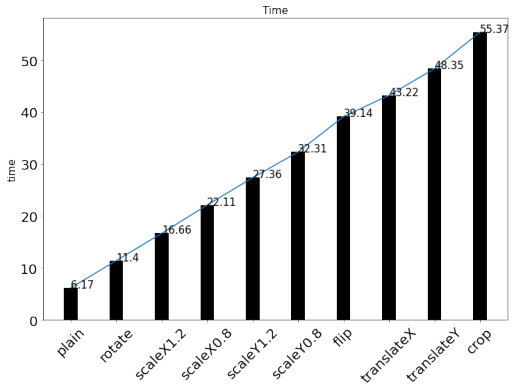


Figure 33: Inference time after adding augmentations accumulatively

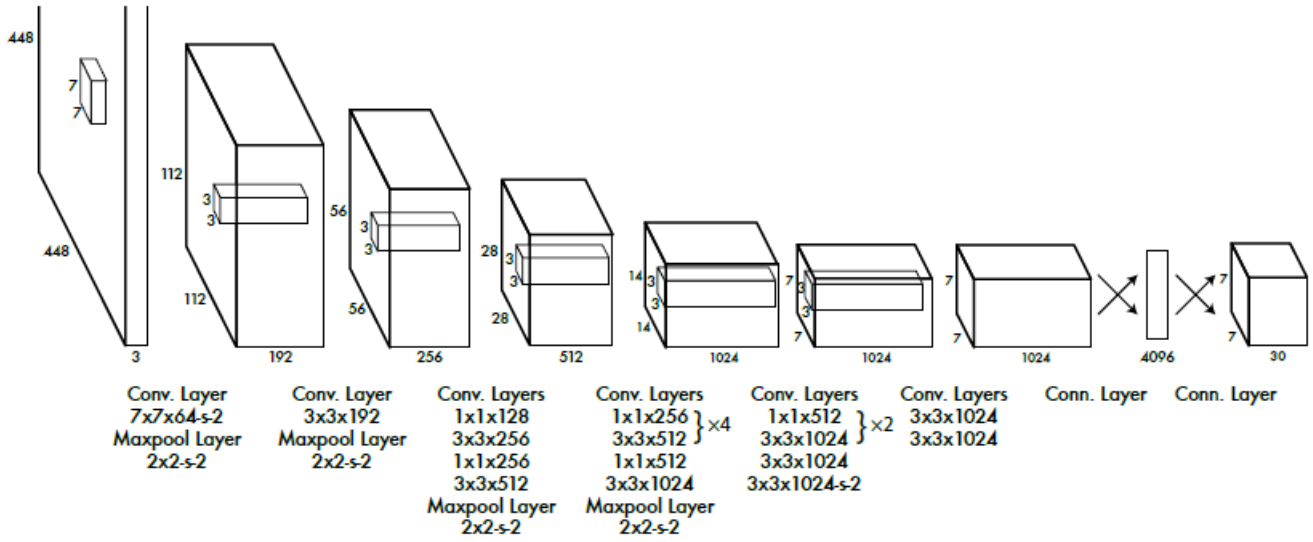


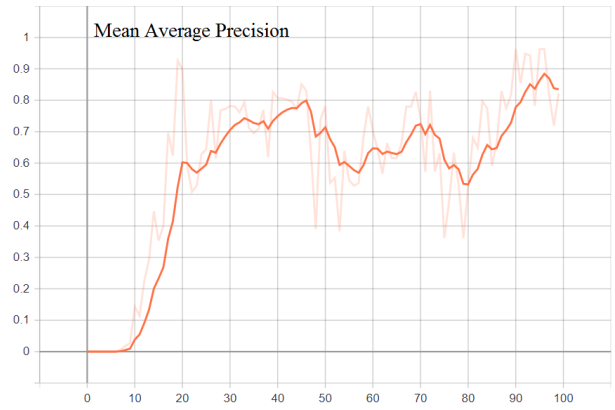
Figure 34: YOLO architecture

3.6 YOLO implementation

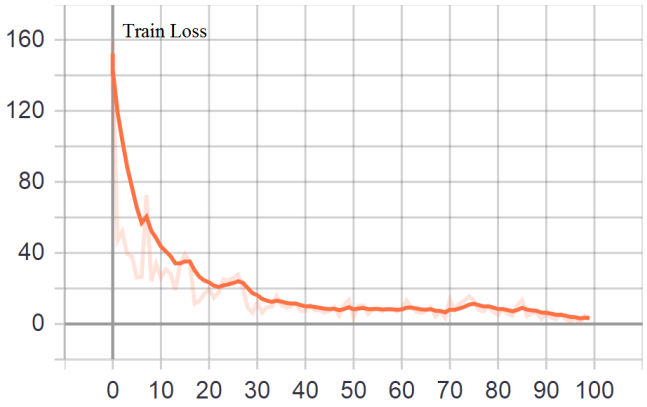
The final attachment to this report is the implementation of yolo algorithm for object detection. YOLO stands for "You Only Look Once"; an abbreviation fir a recent object detection method by Ali Farhadi et al. [15]. As the name indicates, the model is able to detect objects in an image processing it only once contrary to prior models. Previous models usually repurpose a classifier to run through multiple patches of the image. For example RCNN [5] would propose regions to look at before making final detections. Discriminatively trained part-based models need to traverse an input image with an sliding classifier to perform detection [4]. One of the promising features of YOLO is being fast at inference resulting in applicability in real time tasks. This improvement is accompanied by over twice the mean average precision [15] of previous real-time systems. Struggle to learn very close or grouped objects such as flocks of birds and struggle to generalize to new aspect ratios are of limitations this model.

YOLO splits each image into and $S \times S$ grid (7×7 in the implementation). Each object is assigned to only one cell in which the center of the object falls. This cell will be responsible for its' prediction. Each grid cell can output a number of coordinates and corresponding confidence of where it would guess the object is location. Each of these so called "bounding boxes" is to represent a certain shape (aspect ration) that target object will likely fit in.There are 2 "bounding boxes" for each grid cell in this implementation. Overall a cell would output probability scores for each class and bounding the box predictions. The coordinates of a bounding box are determined by relative horizontal and vertical distance (x, y) from top left corner of the cell and width and height (w, h) values relative to width and height of the cell. The ground truth labels in VOC dataset [3] a subset of which is chosen to train the model on provide the coordinate values relative to the whole image which then requires a simple conversion to fit in out training scheme.

The model itself consists of 22 convolutional and 2 fully connected layers (figure **??**). Activation function of choice is Leaky Relu and batch normalization is applied after each convolutional layer. Finally, the innovation in loss function is measure the loss for each prediction as a regression loss. Therefore mean squared error is calculated for the class scores, confidence scores for each bounding box and coordinate values (multiplied by different weights). Figure **??** depicts the mean average precision value through 100 iterations on a small subset of VOC dataset besides the training loss.



(a) mean average precision



(b) train loss

Figure 35: Mean average precision and train loss for the implemented YOLO

4 conclusion

In conclusion, details in both theory and implementations of two seminal deep learning architectures namely, ResNets and YOLO are learned. Effectivness of techniques such as shortcut connections, batch normalization, and popular weight initialization such as LSUV are observed through integration with ResNet model. Data Augmentation and how it differs in training and test time is practiced with to improve the learning and inference performance respectively. As future works, data augmentation in other domains and optimizing data augmentation can be investigated. It will also be interesting to find about expansion of ResNets into newer architectures and also how innovations in YOLO is combined with other object detection systems for mitigate its' limitations.

5 References

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[4] Pedro F Felzenszwalb et al. "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2009), pp. 1627–1645.

[5] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[6] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[7] Kaiming He and Jian Sun. "Convolutional neural networks at constrained time cost". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5353–5360.

[8] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[10] Herve Jegou et al. "Aggregating local image descriptors into compact codes". In: *IEEE transactions on pattern analysis and machine intelligence* 34.9 (2011), pp. 1704–1716.

[11] Kenji Kawaguchi. "Deep learning without poor local minima". In: *Advances in neural information processing systems*. 2016, pp. 586–594.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[13] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[14] Hao Li et al. "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems*. 2018, pp. 6389–6399.

[15] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[16] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[17] Adriana Romero et al. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014).

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[19] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[20] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway networks". In: *arXiv preprint arXiv:1505.00387* (2015).

[21] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[22] Richard Szeliski. "Fast surface interpolation using hierarchical basis functions". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.6 (1990), pp. 513–528.

[23] Richard Szeliski. "Locally adapted hierarchical basis preconditioning". In: *ACM SIGGRAPH 2006 Papers*. 2006, pp. 1135–1143.

[24] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[25] Tommi Vatanen et al. "Pushing stochastic gradient towards second-order methods–backpropagation learning with transformations in nonlinearities". In: *International Conference on Neural Information Processing*. Springer. 2013, pp. 442–449.

[26] Wikipedia contributors. *CIFAR-10 — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=CIFAR-10&oldid=981745305`. [Online; accessed 29-November-2020]. 2020.

[27] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* (2019), arXiv–1910.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[29] Mutlu Yapıcı, Adem Tekerek, and Nurettin Topaloglu. "Literature Review of Deep Learning Research Areas". In: 5 (Dec. 2019), pp. 188–215. DOI: 10.30855/gmbd.2019.03.01.

[30] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.