CIFAR 10 Classification

Xinyue Yu, Binghong Yu

xiy210@ucsd.edu, biy011@ucsd.edu

Abstract

In this paper, we investigated how different methods of tuning hyper-parameters affect the performance of CNN models. In particular, we applied different activation functions, pooling layers, optimizers and transformations on AlexNet and ResNet, based on the CIFAR10 dataset[1]. The top-performing model was inspired by ResNet-18[10][14]. To avoid overfitting, we split the validation set from the train set. The network performance was analyzed by evaluating train accuracy and test accuracy.

1. Introduction

Machine learning gradually plays a significant role in many technology fields. Its application has become quite broader, as Jordan mentioned in his article, "robotics and autonomous vehicle control, speech processing and natural language processing, neuroscience research, and applications in computer vision.[2]" Image classification is one of the hotly discussed topics in machine learning as it gives computer vision, which is one of the most important senses in human beings. Image classification is the process of predicting the classification of images based on the pictures feed in. As Bonner stated in his article, the convolutional neural network (CNN) is a class of deep learning neural networks, which is mainly used for analyzing visual imagery and training image classification[3]. According to Bonner, CNN's application has been widespread, including "image and video recognition, image classification, medical image analysis, etc"[3]. Therefore, in this paper, we studied two popular CNN architectures: AlexNet and ResNet.

AlexNet was built to improve the results of the ImageNet challenge. As Anwar introduced in his article, it was one of the first deep convolutional networks that gained high accuracy on the 2012 ImageNet LSVRC-2012 challenge[4].

On the other hand, ResNet was introduced to overcome the problem that as layers go deeper, classical CNN had bad performance after reaching a threshold[5]. ResNet introduces the idea of Identity shortcut and Projection shortcut to overcome this problem.

The dataset we used is CIFAR-10. This is a classic dataset for training CNN. We have three main goals for our project. First, we want to compare the performance between AlexNet and ResNet. Second, investigate the influence of fine-tuning of models, such as the application of different activation functions, different optimizers, and different pooling functions on model performance. Finally, we studied how transformation affects the performance of the networks. Since we only have limited numbers of train data, we use transformation such as horizontal flip and rotation to create variations. We built our models based on the inspiration from AlexNet[13] and ResNet-18[14].

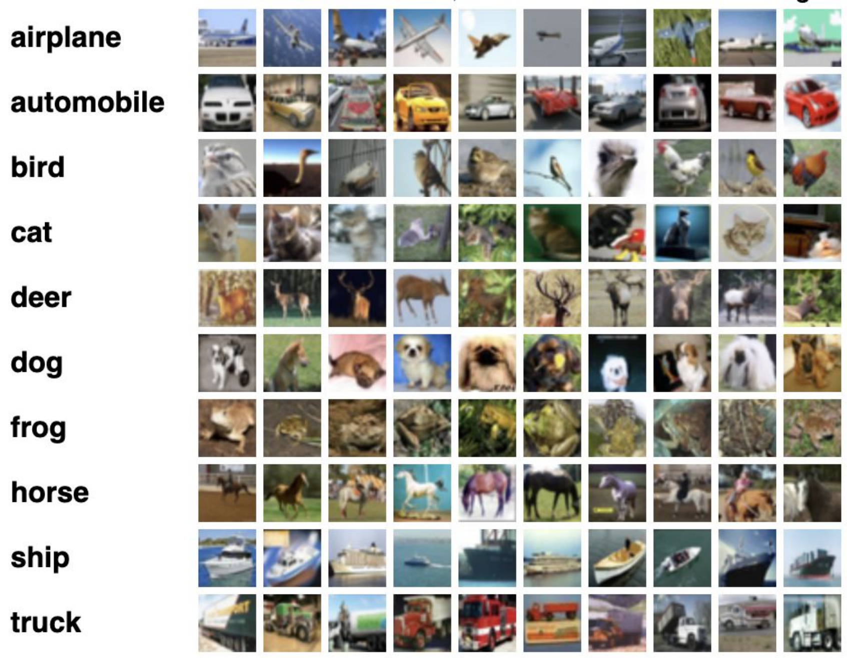
In the Method and Experiment sections, more details related to modifications on model and analysis of the results will be presented.

2. Method

2.1 Dataset: CIFAR-10

The CIFAR-10 is labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton[1]. The CIFAR-10 dataset consists of 60000 color images, which includes 50000 for training, 10000 for testing. There are 10 classes in total, which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. According to Krizhevesky's description of the dataset, to avoid confusion, these classes are completely mutually exclusive. Thus, there is no overlap between automobiles and trucks[1].

2.2 Data example



Alex Krizhevsky (2009). Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html.

2.3 Data preprocessing and data augmentation

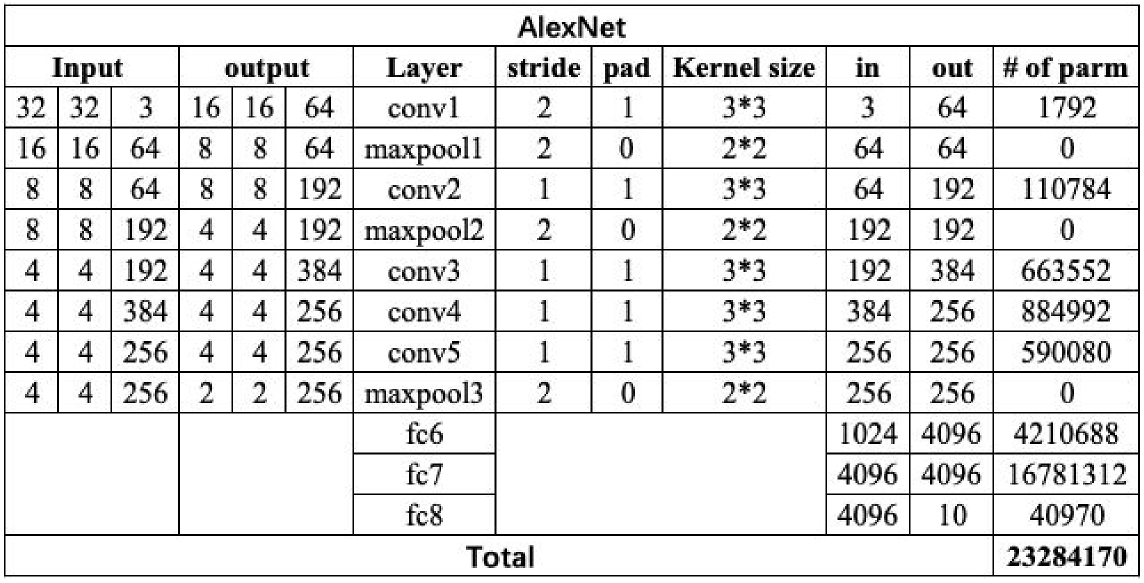
We use PyTorch as our platform to train the deep learning network. We load the data through the data loader from torchvision. To avoid the overfitting of models, we manually split the original training set into a training set and a validation set[12]. We sample indices randomly without replacement, then shuffle the training set according to those indices, and split the training set with a validation set rate of 0.02. That is, after processing with the dataset, we have 49000 training images and 1000 validation images. We also define the augment into five states[10]: first state with random horizontal flip, second state with a random vertical flip, third state with random rotation of 45 degrees, fourth state which contains all of the combinations and fifth state with none of these variations. In all of the five states, we normalize the data.

2.4 AlexNet & ResNet

2.4.1 AlexNet

As mentioned in the introduction section, AlexNet was one of the first deep convolutional networks that gained high accuracy on the 2012 ImageNet LSVRC-2012 challenge[4]. Anwar introduced AlexNet as "The spatial correlation was explored using convolutional layers and receptive fields"[4]. At that time, AlexNet had two major breakthroughs: overcoming the overfitting problem and the vanishing gradient problem[4]. AlexNet solves the over-fitting problem by applying drop-out layers where a connection is dropped during training, while it solves the vanishing gradient problem by introducing Local Response Normalization (LRN), i.e: "normalize the local neighborhood of thus magnifying the excited neuron while restricting the surrounding neurons at the same time" [4].

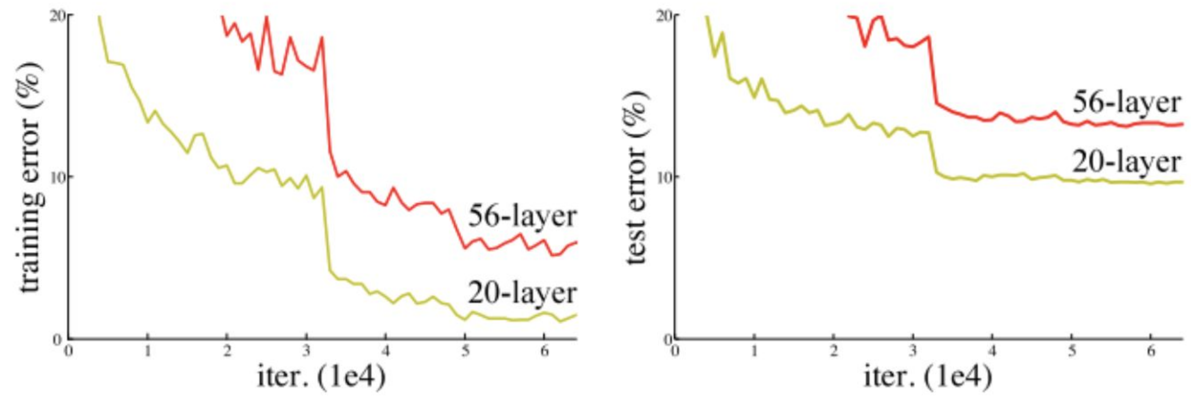
The AlexNet in our research is a modification of the original AlexNet. It consists of five convolutional layers and three fully connected layers. Detailed information of input size, output size and other parameters are shown in the table below. To summarize briefly, we first applied two convolutional blocks. Each of them contains one 3 x 3 convolutional layer, one ReLu activation layer, and one 2 x 2 max pooling layer. Then, we applied three 3 x 3 convolutional layers and three ReLU functions, followed by one max pooling function. The training process above converts the original 32 x 32 x 3 input size into a 2 x 2 x 256 output size. Next, we flattened the output into a 1024-dimensional vector and passed it through a total of three linear layers and three ReLU functions. Finally, we output a 10-dimensional vector and put it into the softmax function that extracts the entry with the highest possibility. In total, we trained about 23 million parameters.



AlexNet												
Input			output			Layer	stride	pad	Kernel size	in	out	# of parm
32	32	3	16	16	64	conv1	2	1	3*3	3	64	1792
16	16	64	8	8	64	maxpool1	2	0	2*2	64	64	0
8	8	64	8	8	192	conv2	1	1	3*3	64	192	110784
8	8	192	4	4	192	maxpool2	2	0	2*2	192	192	0
4	4	192	4	4	384	conv3	1	1	3*3	192	384	663552
4	4	384	4	4	256	conv4	1	1	3*3	384	256	884992
4	4	256	4	4	256	conv5	1	1	3*3	256	256	590080
4	4	256	2	2	256	maxpool3	2	0	2*2	256	256	0
						fc6				1024	4096	4210688
						fc7				4096	4096	16781312
						fc8				4096	10	40970
Total												**23284170**

2.4.2 ResNet

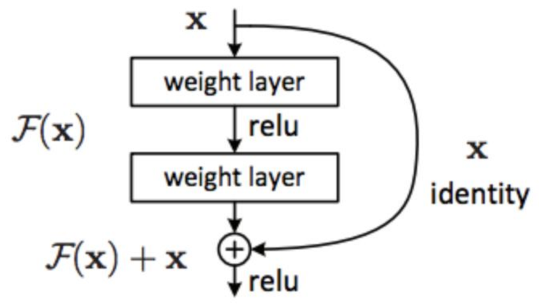
Since the presence of AlexNet, layers of CNN are going deeper and deeper. While AlexNet has only 5 convolutional layers, other popular networks such as VGG have 19 layers. However, they both encounter the problem that increasing network depth leads to worse performance because of the vanishing gradient problem[9].



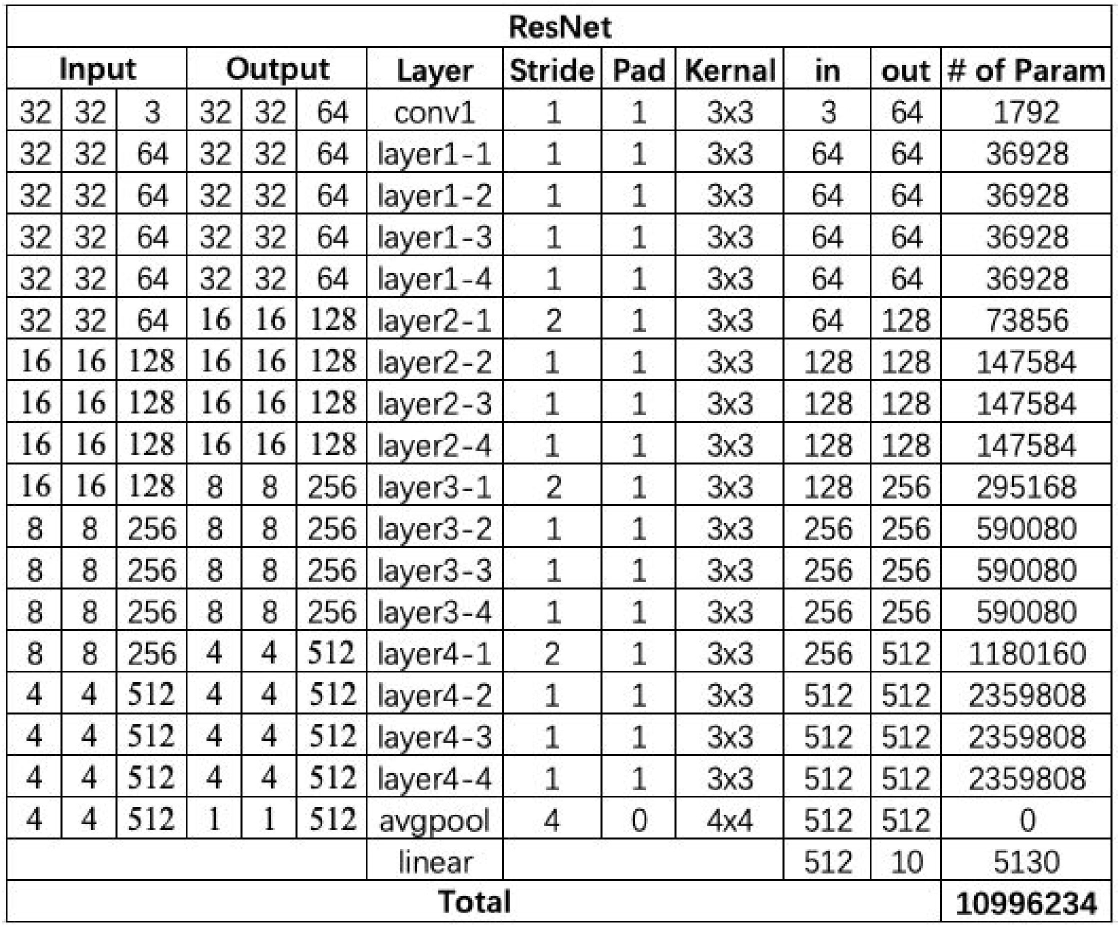
Vincent Fung (2009). Retrieved from https://towardsdatascience.com/an-overview-of-ResNet-and-its-variants-5281e2f56035

ResNet was introduced to overcome the problem that the performance of classical CNN drops after passing a certain threshold[5]. ResNet introduces the idea of "Identity shortcut and Projection shortcut", thus allowing the training of deeper networks.[9] This identity mapping is unique at outputting a combination of the f(x) as well as the input x to the next layer. The residual mapping could also be thought of "as the amount of error which can be added to input to approximate the final function”[10]. The image below shows the math formula and structure image for the residual block.

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


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

The model we used is called ResNet-18 which means there are a total of 18 layers. It consists of 17 convolutional layers and one fully connected layer. Detailed information of input size, output size and other parameters are shown in the table below. To summarize briefly, we first applied one convolutional layer with batch normalization and ReLU activation. Then we self-defined four layers. Each of them contains four 3 x 3 convolutional layers. The training process above in layer 1 converts the original 32 x 32 x 3 input size into a 32 x 32 x 64 output size. In layer 2 , 32 x 32 x 63 are converted into 16 x16 x128. In layer 3, 16 x 16 x 128 are converted to 8 x 8 x 256. And in layer 4, 8 x 8 x 256 are converted to 4 x 4 x 512. After applying a 4 x 4 average pooling, we flattened the output into a 512-dimensional vector and passed it through a linear layer that output a 10-dimensional vector. Finally, we load them into the softmax function. The total number of parameters we trained in this network is around 11 million.

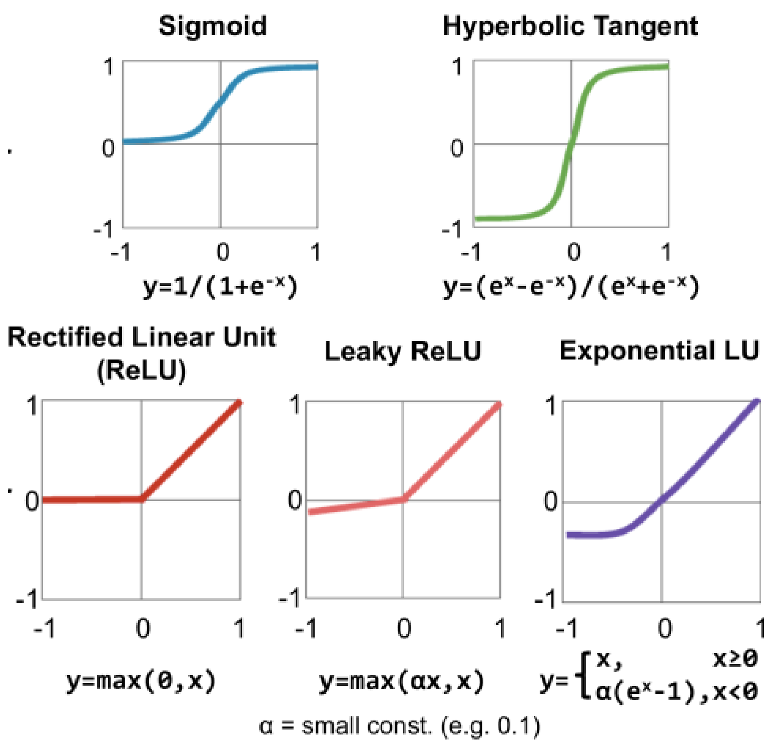


ResNet												
Input			Output			Layer	Stride	Pad	Kernal	in	out	# of Param
32	32	3	32	32	64	conv1	1	1	3x3	3	64	1792
32	32	64	32	32	64	layer1-1	1	1	3x3	64	64	36928
32	32	64	32	32	64	layer1-2	1	1	3x3	64	64	36928
32	32	64	32	32	64	layer1-3	1	1	3x3	64	64	36928
32	32	64	32	32	64	layer1-4	1	1	3x3	64	64	36928
32	32	64	16	16	128	layer2-1	2	1	3x3	64	128	73856
16	16	128	16	16	128	layer2-2	1	1	3x3	128	128	147584
16	16	128	16	16	128	layer2-3	1	1	3x3	128	128	147584
16	16	128	16	16	128	layer2-4	1	1	3x3	128	128	147584
16	16	128	8	8	256	layer3-1	2	1	3x3	128	256	295168
8	8	256	8	8	256	layer3-2	1	1	3x3	256	256	590080
8	8	256	8	8	256	layer3-3	1	1	3x3	256	256	590080
8	8	256	8	8	256	layer3-4	1	1	3x3	256	256	590080
8	8	256	4	4	512	layer4-1	2	1	3x3	256	512	1180160
4	4	512	4	4	512	layer4-2	1	1	3x3	512	512	2359808
4	4	512	4	4	512	layer4-3	1	1	3x3	512	512	2359808
4	4	512	4	4	512	layer4-4	1	1	3x3	512	512	2359808
4	4	512	1	1	512	avgpool	4	0	4x4	512	512	0
						linear				512	10	5130
Total												**10996234**

**** These two structure graphs are manually calculated by ourselves.***

2.5 Activation functions: ReLU, Tanh, LeakyReLU, Elu, Sigmoid

The purpose of activation function is to introduce nonlinearity into our neural networks. Adding nonlinearity can increase a model's ability to approximate complex functions. In our project, we compare the performance of five kinds of activation functions in our self-defined AlexNet while keeping all other variables the same. The following five activation functions are: ReLU, Tanh, LeakyReLU, Elu and Sigmoid.



Sagar Sharma (2017). Retrieved from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Previous to our experiment, we know that both Sigmoid and Tanh kill gradients. Applying them on a large positive value or large negative value will lead to near-zero gradients. Thus, it becomes useless in the process of backpropagation and it is hard for the network to learn something. ReLU is expected to have a better performance because it does not saturate in the positive region. In addition, the computation speed of ReLU is faster than Sigmoid and Tanh. LeakyReLU and Elu don't saturate even in the negative region, so we expect they have similar performance like ReLU does.

2.6 Optimizer: Stochastic gradient descent, Adam

Optimizers' function is to update the model weights thus enhancing the model performance and minimize its error rate[7]. In other words, optimizers fit your model into its most accurate possible form by updating weights through training. While updating weights, the loss function decides whether the optimizer is moving in the right or wrong direction.

In this project, we mainly use two most common optimizers: Stochastic gradient descent and Adam, to update weights and minimize the loss.

Stochastic gradient descent (SGD) is one type of gradient descent. Instead of performing computations on the whole dataset, SGD only computes on a small part of data examples[7]. SGD has several advantages including computationally more efficient, less memory demanding, and avoids bad local optimal.

While the other optimizer: Adam, could be considered as a “combination of RMSprop and Stochastic Gradient Descent with momentum”[8]. From Bushaev’s article, he introduces Adam optimizer as“computes individual learning rates for different parameters”[8]. One characteristic of Adam is its fast speed to train, which greatly enhances the training efficiency.

2.7 Pooling function: Max pooling, Average pooling

The goal of the pooling function is to reduce the spatial size of the images between layers, therefore decreasing the number of parameters and computation demand in the model[3]. The most common approach used in pooling is max pooling. In this project, we apply both max pooling and average pooling as our pooling methods. The idea behind it is to take average or maximum from the self-defined kernel size, thus output the most important information and transport it to the next layer.

3. Experiment

3.1 AlexNet vs ResNet

From our experiment, we found that ResNet and AlexNet have similar best training accuracy, which are 0.89 and 0.91, respectively. The best validation accuracy occurs at epoch 10 and 7, respectively. While keeping all other parameters and conditions the same, ResNet takes a longer time to train. During test time, ResNet has an accuracy of 0.70, which is a much better performance than AlexNet which only reaches 0.47. For more details please refer to the table below. For the graphs and code output please refer to the appendix.

	ResNet	**AlexNet**
Best Train Accuracy	0.8930	0.9062
Best Validation Accuracy	0.83	0.764
Epoch of Best Validation Accuracy	10	7
Best Test Accuracy	0.7029	0.4743
Time	34m 23s	23m 22s

3.2 AlexNet Modification

After several turns of modification on the parameters on the AlexNet, we gained the result presented in the table below. We found that the combination of LeakyReLU, SGD and max pooling produced the highest test accuracy, which is 0.48. Other model combinations such as ReLU, SGD, max pooling and ReLU, SGD, average pooling also produced similar results at test time, which are 0.47 and 0.45 respectively. But the Adam optimizer, Elu activation function, Tanh activation function and Sigmoid activation functions did not perform well.

AlexNet	**Best Train accuracy**	**Best Validation accuracy**	**Best Test accuracy**
ReLU, SGD, max pooling	0.977	0.768	**0.4661**
ReLU, Adam, max pooling	0.4919	0.532	0.2305
ReLU, SGD, avg pooling	0.8338	0.767	**0.4470**
LeakyReLU, SGD, max pooling	0.9085	0.749	**0.4848**
Elu, SGD, max pooling	0.652	0.666	0.3979
Tanh, SGD, max pooling	0.564	0.650	0.3937
Sigmoid, SGD, max pooling	0.101	0.110	0.100

We also applied fine-tune augmentation to our base model of AlexNet. From the table below, we see that originally we have a test accuracy of 0.47. However, after applying random vertical flip, the accuracy was increased to 0.49. It means that adding random vertical flip gives variation to our training process and successfully increases the model's ability at test time. Besides, the combination of horizontal flip, vertical flip and rotation of 45 degrees also produced a higher test accuracy than our base model. Other transformations did not help increase test accuracy in our experiment.

AlexNet Transformation	**Best Train accuracy**	**Best Validation accuracy**	**Best Test accuracy**
BASE: ReLU, SGD, max pooling	0.977	0.768	0.4661
Random horizontal flip	0.7630	0.759	0.4534
Random vertical flip	0.7644	0.777	**0.4939**
Random rotation with 45 degrees	0.7686	0.778	0.4332
Random horizontal & vertical flop & rotation with 45 degrees	0.7599	0.775	0.4895

3.3 ResNet Modification

After applying modifications to our base model of ResNet, we gained the result presented in the table below. We see that both switching the activation function from ReLU to LeakyReLU and switching the optimizer from SGD to Adam increase the test accuracy.

ResNet	**Best Train accuracy**	**Best Validation accuracy**	**Best Test accuracy**
ReLU, SGD, IdentityPadding	0.8930	0.83	0.7029
LeakyReLU, SGD, IdentityPadding	0.8943	0.826	**0.7106**
ReLU, Adam, IdentityPadding	0.9184	0.838	**0.7644**

Like what we did in AlexNet, we also apply transformation to our ResNet model. From the table here, we see that random horizontal flip increases the test accuracy from 0.7029 to 0.7391. However, all of the other augmentations lead to worse performance than the original base model.

ResNet Transformation	**Best Train accuracy**	**Best Validation accuracy**	**Best Test accuracy**
Base: ReLU, SGD, IdentityPadding	0.8930	0.83	0.7029
Random horizontal flip	0.862	0.848	**0.7391**
Random vertical flip	0.790	0.783	0.6721
Random rotation with 45 degrees	0.748	0.779	0.5551
Random horizontal & vertical flop & rotation with 45 degrees	0.658	0.699	0.4739

4. Conclusion

As mentioned, we have three main goals in this project. First, comparing the performance of AlexNet and ResNet. Second, fine-tuning hyper-parameters to investigate their influence on the test accuracy. Third, applying different augmentations to see if any of them can improve our network performance at test time.

We drew the following conclusions from this research. First We found that ResNet has a generally better performance than AlexNet during test time. Second, ReLu and LeakyReLU are usually the best activation functions to use. Third, max pooling and average pooling have similar performance, but max pooling is slightly better. Fourth, different optimizers have different performance on two of our models. SGD is good for AlexNet, while Adam is good for ResNet in this specific research. Finally, some transformations such as random horizontal flip and random vertical flip help create variation to the data, thus enhancing the performance at test time.

5. Bonus

We did a thorough comparison in the networks. We calculate the detailed parameters and structure of the two networks (see two tables in the 2.4 section). In addition, we also recorded a presentation video.

6. Contribution

We worked together throughout the whole process of project outlining, researching, coding, paper writing, copy-editing, video presenting and video editing. We contributed equally to this project.

7. References

[1]Krizhevsky, A. (2009). Chapter 3. In Learning Multiple Layers of Features from Tiny Images. Retrieved June 12, 2020, from https:/www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[2]Jordan, M. I., & Mitchell, T. M. (2015, July 17). Machine learning: Trends, perspectives, and prospects. Retrieved from https://science.sciencemag.org/content/349/6245/255.full

[3]Bonner, A. (2019, June 01). The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks. Retrieved June 13, 2020, from https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[4]Anwar, A. (2020, May 13). Difference between AlexNet, VGGNet, ResNet and Inception. Retrieved June 13, 2020, from https://towardsdatascience.com/the-w3h-of-AlexNet-vggnet-ResNet-and-inception-7baaaecccc96

[5]Shorten, C. (2019, May 15). Introduction to ResNets. Retrieved June 13, 2020, from https://towardsdatascience.com/introduction-to-ResNets-c0a830a288a4

[6]Wu, J., Zhang, Q., & Xu, G. (2017). Tiny ImageNet Challenge. Retrieved 2020, from http://cs231n.stanford.edu/reports/2017/pdfs/930.pdf

[7]Algorithmia (2020, May 14). Introduction to Optimizers. Retrieved June 13, 2020, from https://algorithmia.com/blog/introduction-to-optimizers

[8]Bushaev, V. (2018, October 24). Adam - latest trends in deep learning optimization. Retrieved
June 13, 2020, from https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

[9]Fung, V. (2017, July 15). An Overview of ResNet and its Variants - Towards Data Science. Retrieved June 13, 2020, from https://towardsdatascience.com/an-overview-of-ResNet-and-its-variants-5281e2f56035

[10]He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/cvpr.2016.90

[11]Moon, T. (2018, November 23). Tjmoon0104/pytorch-tiny-imagenet. Retrieved June 14, 2020, from https://github.com/tjmoon0104/pytorch-tiny-imagenet

[12]Zakka, K. (2017). Train, Validation and Test Split for torchvision Datasets. Retrieved June

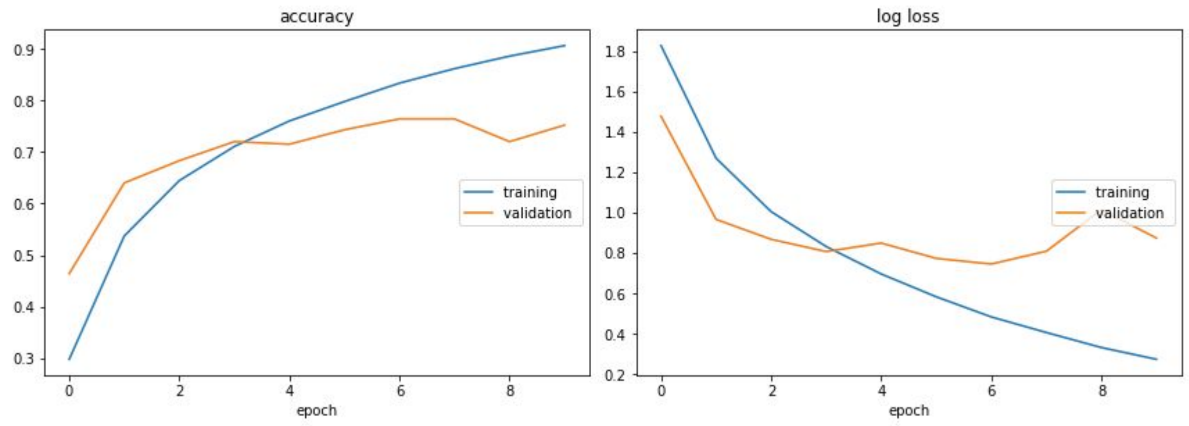
14, 2020, from https://gist.github.com/kevinzakka/d33bf8d6c7f06a9d8c76d97a7879f5cb

[13]Luo, Z. (2018, September 21). Icpm/pytorch-cifar10. Retrieved June 14, 2020, from https://github.com/icpm/pytorch-cifar10/tree/master/models

[14]Xiangjianxiaolu(2018, September 25). Pytorch入门实战：ResNet18图像分类（Cifar10）. Retrieved June 14, 2020, from http://flyrie.top/2018/09/26/Pytorch_ResNet18_Cifar10/

8. Appendix

1. AlexNet Base Model (ReLU + SGD + Max Pooling)

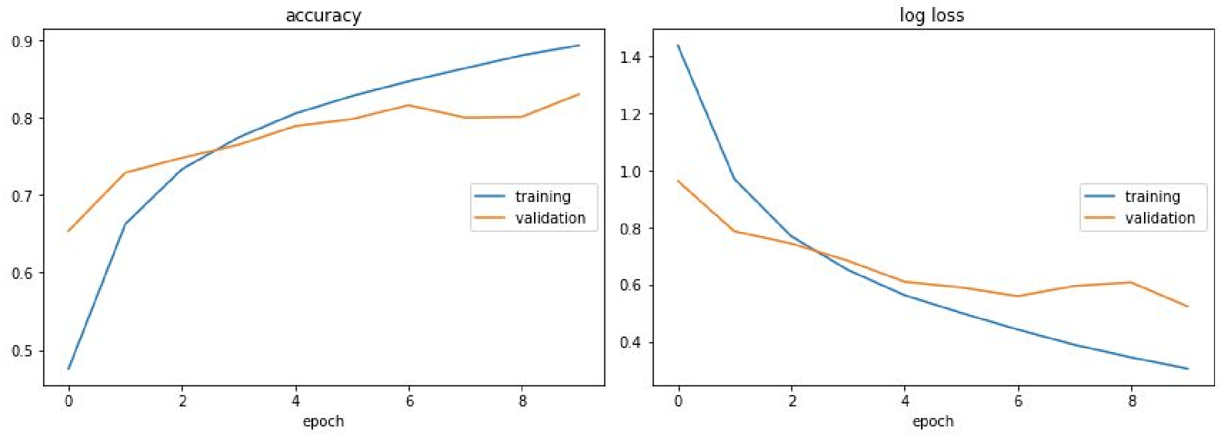


```
accuracy
      training          (min:    0.298, max:    0.906, cur:    0.906)
      validation        (min:    0.464, max:    0.764, cur:    0.752)
log loss
      training          (min:    0.271, max:    1.826, cur:    0.271)
      validation        (min:    0.744, max:    1.476, cur:    0.872)

Train Loss: 0.2715 Acc: 0.9062
Val Loss: 0.8724 Acc: 0.7520
Best Validation Accuracy: 0.764, Epoch: 7

Test Loss: 1.9669 Acc: 0.4743
```

2. ResNet Base Model (ReLU + SGD + Identity Pooling)



```
accuracy
      training          (min:    0.476, max:    0.893, cur:    0.893)
      validation         (min:    0.654, max:    0.830, cur:    0.830)
log loss
      training           (min:    0.306, max:    1.437, cur:    0.306)
      validation         (min:    0.523, max:    0.962, cur:    0.523)

Train Loss: 0.3057 Acc: 0.8930
Val Loss: 0.5232 Acc: 0.8300
Best Validation Accuracy: 0.8300000000000001, Epoch: 10

Test Loss: 0.9124 Acc: 0.7029
```