CS771hw2

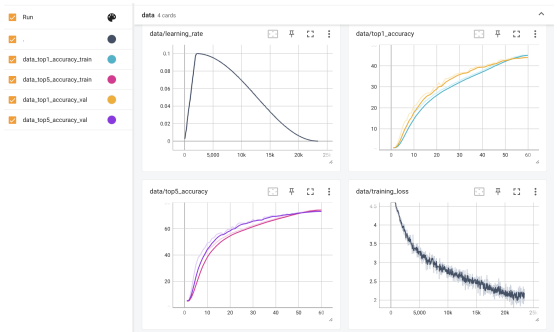
Yuchen Dou, Keshav Sharan, Kaiwalya Shukla

October 2023

1 Understand Convolutions

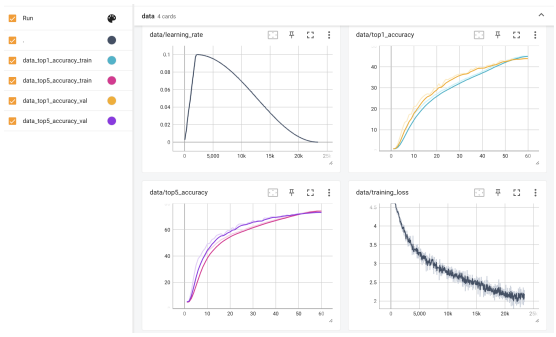
In this part of the code, we implement a simple 2D convolution graph. In the forward function, the convolution is defined as Y = W * X + b, w is the weight, x is the input feature and b is the bias. First, we check the validation of inputs. Then, we check if the padding value is larger than 0. It ensures the spatial dimensions of the feature map don't reduce too quickly as we apply convolutions. After that, we unfold and reshape the output.
In the backward part, the gradients are computed. If the gradient with respect to the input is needed, we reshape and transpose the weight. Then, we adjust the gradient of output to prepare for later matrix multiplying. After that, we can get the new gradient input value based on gradient output and new weight. Then we fold it. the next part is gradient of the loss of the convolution kernel weight. First, we check that the weight is needed, then we apply similar actions like we did in the last part, and then we can get the gradient weight.
After we implement the 2D convolution, we run the test and it passes it.

2 A Simple Convolutional Network



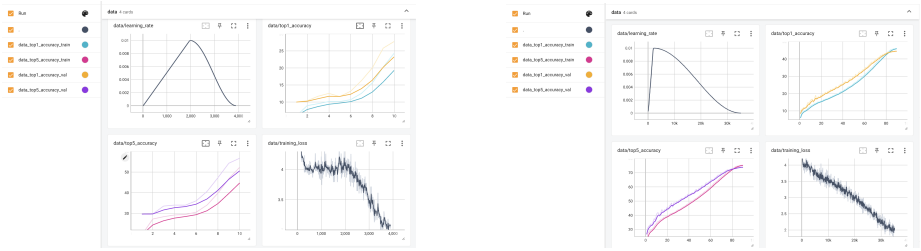
Above is the visualization of the training process of the simple convolution part. The top one accuracy reaches 40+ so we know that the training goes successfully.

3 Train with Your Own Convolutions



This is the visualization of my own convolution, we can see that the learning rate reaches the peak later than the pytorch one. Also, the training speed did not decrease significantly at the end of the training process comparing to the pytorch one. One of the reason maybe that the number of epoch is significantly less than the pytorch training. Also, since the custom convolution implementation uses the unfold technique, it leads to high memory consumption. The loss curve of these two are similar since the mathematical operation are similar.

4 Implement a Vision Transformer



(a) 10 epoch (b) 90 epoch

Figure 1: Since it works well, we decided to move forward and train it with an epoch number of 90

First, the model is break into smaller patches, then changes it into numbers. In the transformer block part, each block helps the model know how different parts of the image relate to each other. If we decide to use position, we adjusts it to make sure it starts in the right way. In the forward part, the image is broken down, processed through several blocks, averaged, and then a decision is made on its category.
The model is trained using the AdamW optimizer. In order to adjust it, we

decrease the learning rate and increase the weight decay to stablize the training and help to regularizing the model. At the same time, we have to increase the number of epoch. Considering the number of epoch, we start by training the model with 10 epoches.

Since it works well, we decided to move forward and train it with an epoch number of 90.

The final visualization is shown in figure 2, we can see that the top 1 accuracy reaches 40+ which give a performance level similar to the simple convolutional network. This proves that our model meets the expectation.

5 Design Your Own Network

In this part, we choose to focus on convolutional network, which is improving the SimpleNet class. In the first attempt, we wanted to improve the network by adding batch normalization in the coding part to fasten convergence during training. Also, I chosed to decrease the learning rate and increase training epoch in order to achieve higher accuracy. However, the accuracy did not improve significantly from the simpleNet one.

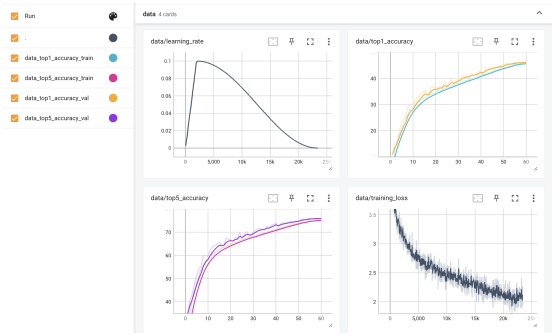


Figure 2: first attempt

Based on the experience before, I learned that applying simple batch normalization is not enough to increase accuracy significantly. So, in the second attempt, I decided to apply more actions to improve accuracy. In this model, besides of batch normalization, I also increased the number of layers so we can capture more features from the input image. What is more, we apply some changes in kernel size, stride, and padding to potentially improve the performance of the model.

From the final visualization, we can see that the top 1 accuracy reaches 50+ so we can know that our new training model works better than the old one so our improvement did work. However, considering the gap between validation accuracy and training accuracy, our model has some overfitting problem; however, since the accuracy increases significantly, we can still say that our attempt is successful.

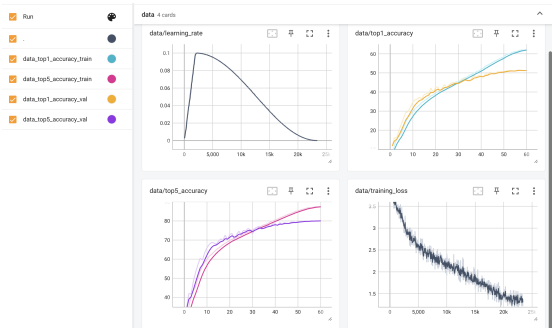
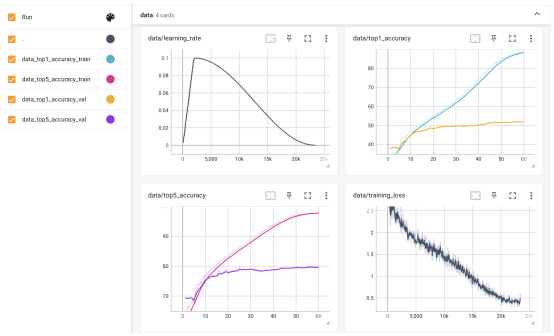


Figure 3: second attempt

6 Fine-Tune a Pre-trained Model



Here is the visualization of the residual network pre-trained. Comparing to my code, the validation accuracy of top 1 and top 5 are similar. It means that our model has a similar performance to their residual network pre-trained model. However, the training accuracy of the residual network is a lot higher than mine. It shows that the residual network has a worse overfitting problem, which means that it is too specialized in learning the training data and starts to capture noise or random variations in the training data rather than the underlying patterns that generalize to new data.

7 Attention and Adversarial Samples

7.1 Saliency Maps:

7.1.1 Method:

We computed the input gradient while minimizing the loss of the model's most confident prediction, generating attention maps using a trained model. When

these gradients are displayed, they draw attention to regions of the picture that are crucial to the model's judgment.

7.1.2 Results:

Red attention maps superimposed on the supplied images show the regions where the model focused its attention. High contrast areas, the main items in sceneries, and the clear borders or edges in structured environments are examples of notable zones. The model's efficacy is demonstrated by its attention to significant characteristics, which corresponds with human perception. Still, the distribution of attention in some photographs points to possible areas where the model could be improved. Furthermore, the model may be susceptible to deliberate adversarial perturbations because to its emphasis on high contrast regions.

(a) Original Input images (b) Attention Maps

Figure 4: Difference between original and images with superimposed attention maps

7.2 Adversarial Samples:

7.2.1 Differences between Adversarial and Original photos:

The differences between the adversarial samples and original photos appear to be introduction of noisy artifacts. However, these small changes can fool our model, showing how susceptible neural networks are to these kinds of disruptions.

7.2.2 Effect of More Iterations and Lower Error Bound:

The adversarial perturbations get more sophisticated when the PGD iterations are increased and the error bound is lowered. This can further reduce the accuracy of the model, highlighting the need for stronger defenses. A lower

Epsilon also would result in images that look more like the original photos due to the smaller perturbations made while increased number of iterations further decrease the accuracy.

7.2.3 Method:

Using the rapid gradient sign method in several phases and clipping results to keep them inside the designated ε-neighborhood, we constructed PGD under the ℓ^∞ norm. We decided to push the classification model towards the least confident class. Since this class is frequently the "furthest" from the model's present prediction, it makes intuitive sense to drive the model towards it. As a result, a model that is now unable to recognize the supplied image will probably have more distinguishing features that it correlates with the least confident class. You're severely altering the model's decision boundaries by boosting those absent features.The simpleNet was modified to accommodate PGD. We generate an adversarial sample based on the input using our attacker model (self.attacker.perturb) and then blend the original input x with using a 50-50 weighted average. This approach, which balances maintaining some of the original data's features with introducing adversarial perturbations, can enhance our model's robustness against adversarial attacks. By avoiding the direct use of , we mitigate the risk of the model becoming overly specialized in handling perturbed inputs, thus safeguarding its generalization ability on clean data.

(a) Original Input images

(b) Generated Adversarial Samples

Figure 5: Difference between original and adversarial images

7.2.4 Results:

The adversarial attacks caused a 9% decline in accuracy, demonstrating how vulnerable our approach is to these kinds of dangers. Additional illustrations

of the differences between the adversarial and original samples can be found in the images where the drop in the accuracy is very steep and noticeable.

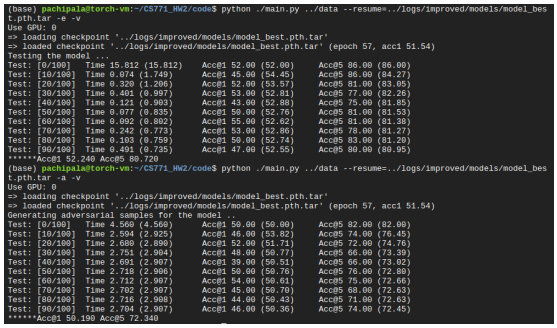


Figure 6: Terminal output showing drop a in accuracy

8 Adversarial Training

In the modified forward function of SimpleNet, a mechanism for adversarial training has been integrated. This is achieved by invoking the perturb method from the attack attribute of the model, which is using the defaultattack (PGDAttack class). A unique flag, isadversarial, is temporarily set to ensure that this adversarial generation process doesn't recursively invoke itself, thereby avoiding infinite loops. Once the adversarial sample, advx, is generated, it replaces the original input x for the subsequent layers of the network. After generating the adversarial sample, the isadversarial flag is deleted to reset the state. This approach ensures that during training, the model is exposed to and learns from both original and adversarial samples, thereby enhancing its robustness against adversarial attacks. To speed up training we modify the numofsteps in perturb method to 3 and then train the model for 10 epochs which took about 2.5 (hours).

9 Contribution

Yuchen Dou: Responsible for Understand Convolutions, A Simple Convolutional Network, Train with Your Own Convolutions, Implement a Vision Transformer, Design Your Own Network.
Keshav Sharan: Responsible for Saliency Maps, Adversarial Samples
Kaiwalya Shukla: Responsible for Fine-Tune a Pre-trained Model, Adversarial Training. Helped in Design Your Own Network.