David Vaughan

Machine learning

Juntao Huang

4/30/2023

Final Paper

Project 2 Residual Network

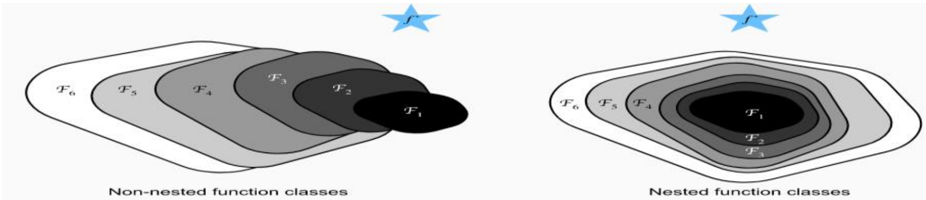
The goal of project two was to apply a Residual Network to the image classification problem on the CIFAR-10 dataset and to compare the performance with AlexNet. "The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes.[3] The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class." (Wikipedia CIFAR-10) Using an AlexNet approximation for the midterm project I was able to achieve an accuracy of around 70%. AlexNet was a convolutional neural network (CNN) used in the 2012 ImageNet Large Scale Visual Recognition Challenge and the AlexNet network achieved a top-5 error of 15.3% (Wikipedia AlexNet). AlexNet helped start the trend of bigger and bigger neural networks. As CNNs have gotten bigger and the layer counts increase one new method used to keep improving their accuracy is Residual Networks. Residual Networks are modern CNN that make use of massive data sets, GPUs, and new techniques to help make CNN's even more accurate for solving the image classification problem.

To solve the image classification problem a CNN is usually used. CNNs use a convolutional layer, which performs a convolution (element-wise matrix multiplication, summation of all the elements, and add bias) on the input data to extract features = {W - K +

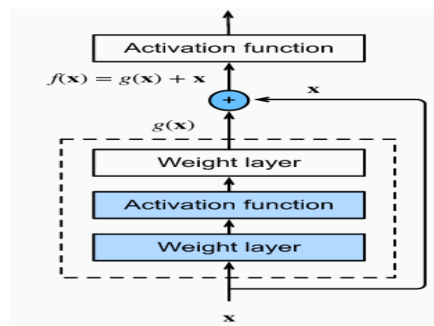
2P}/{S} + 1, O: output size, W: input size, K: kernel size, P: same padding (non-zero), S: stride. The convolution operation involves sliding a small filter, or kernel, over the input data and computing a dot product between the filter and each patch of the input. The product then will probably go through a ReLU activation function (Wikipedia Contributors, "Convolutional Neural Network"). The kernel is changed and adjusted through backpropagation to get optimal values. A CNN can have padding: adding zeros around the matrix (for example 4x4 pad becomes 6x6). A CNN can have a Pooling layer, which reduces the dimensions of the hidden layer by combining the outputs of neuron clusters at the previous layer into a single neuron in the next layer. O = {W - K}/{S} + 1. W: input size, K: kernel size, S: stride size = kernel size.

Older versions of CNNs like LeNet used to use sigmoid as an activation function. This was not ideal because non-linear functions can take too long to compute for large neural networks. (Wikipedia "Convolutional Neural Network") "In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned." (Jason Brownlee) ReLU was chosen to fulfill these criteria because is simple to compute (if input > 0: return input else: return 0). Because of ReLU speed and the fact that is somewhat linear, it allows for deeper CNN to be made.

As our Neural network gets larger, and as the function class gets bigger, we want the function to be more accurate and to move closer to the correct function for the neural network. But with non-nested functions, we can slowly drift away from the correct function, and we will end up with an incorrect answer.



Also, large CNNs suffer from the vanishing gradient problem. The vanishing gradient problem is encountered when neural networks use backpropagation, which tries to find the minimum of a function with derivatives and weights. As the Network gets deeper and deeper the weights get smaller and smaller until backpropagation no longer works effectively. To solve the vanishing gradient problem and the non-nested classes, we make our neural network have a nested function class. To make this we have a function F(x) = G(x) + x. G(x) is a residual block which can consist of a convolution, batch normalization, activation function then another convolution and batch normalization. Batch normalization is used to solve internal covariate shift which is "the distribution of each layer's inputs changes during training, as the parameters of the previous layers change". (Ioffe) Internal covariate shift slows down the neural network, so batch normalization is used throughout the network to help speed it up. Batch normalization in Pytorch is a function y= β + (γ *(x – E[x]))/sqrt (Var[x]+ϵ). (BatchNorm2d — PyTorch). The x input is the input layer that travels through a residual connection past the residual block to be added with the result of that residual box g(x). (*D21*). To add x correctly no matter the size the Residual class uses "… two types of networks: one where we add the input to the output before applying the ReLU nonlinearity whenever use_1x1conv=False, and one where we adjust channels and resolution by means of a convolution before adding." (*D2l*). After F(x) is found it goes through the activation function



RELU and then goes on to the next block. Because x is added to F(x) it helps to solve the vanishing gradient problem. With this, we can achieve nested function classes which help make the neural network more accurate.

A problem encountered when implementing the Res-net the d21 website wants to define the class ResNet (d2l.Classifier). When trying to implement this, there were errors, and it did not work properly. It may have something to do with Google Collab or maybe it was not implementing it correctly. Using a GPU/cuda was useful to test quickly whether a change to the neural network was going to increase accuracy or not. Increasing the neuron count to above 4000 caused an error where there was not enough GPU RAM. Using multiple blocks above 1000 neurons caused the accuracy to plummet to 10%. Increasing the block count from 5 to 10 increased the accuracy from about 73% to 76%. Adding another convolution and batch normalization to the residual block g(x) did not improve accuracy and it caused a significant slowdown to the overall performance. The accuracy of the network stopped increasing by around 5000 iterations. There was no discussion with other team members because I was the only team member.

The result of the project was an accuracy of around 75%. It is around 5% better than my version of AlexNet but a Residual Network would probably be better as you scaled up the network more and more. With bigger machines more code blocks could be added which would add some accuracy to the final output.

Sources

Works Cited

"8.6. Residual Networks (ResNet) and ResNeXt — Dive into Deep Learning 1.0.0-Beta0 Documentation." *D2l.ai*, d2l.ai/chapter_convolutional-modern/resnet.html#resnet-model. Accessed 29 Apr. 2023.

"BatchNorm2d — PyTorch 2.0 Documentation." *Pytorch.org*, pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html#torch.nn.BatchNorm2d. Accessed 30 Apr. 2023.

"BatchNorm2d: How to Use the BatchNorm2d Module in PyTorch." *Www.datascienceweekly.org*, www.datascienceweekly.org/tutorials/batchnorm2d-how-to-use-the-batchnorm2d-module-in-pytorch. Accessed 30 Apr. 2023.

"CIFAR-10." *Wikipedia*, 29 Mar. 2021, en.wikipedia.org/wiki/CIFAR-10.

Ioffe, Sergey. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

Jason Brownlee. "A Gentle Introduction to the Rectified Linear Unit (ReLU) for Deep Learning Neural Networks." *Machine Learning Mastery*, 20 Apr. 2019, machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

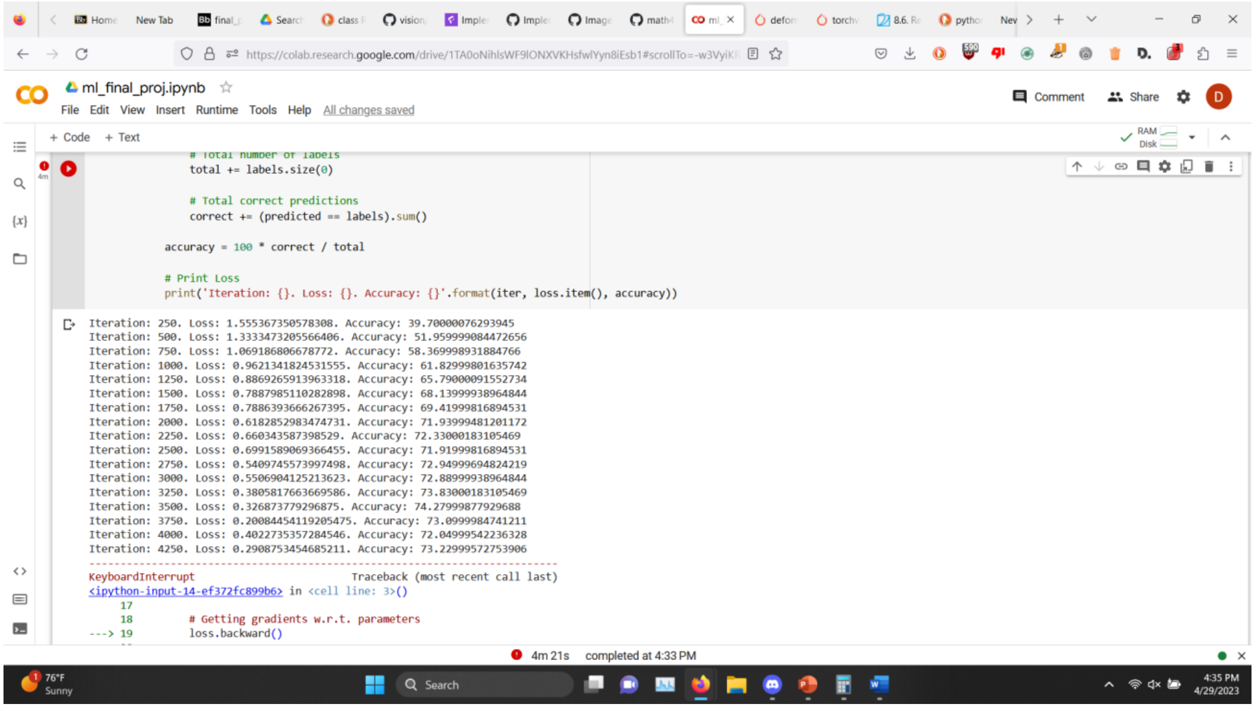
Mayur. "Image Classification Using Deep Learning." *GitHub*, 25 Apr. 2023, github.com/Mayurji/Image-Classification-PyTorch/blob/main/ResNet.py. Accessed 29 Apr. 2023.

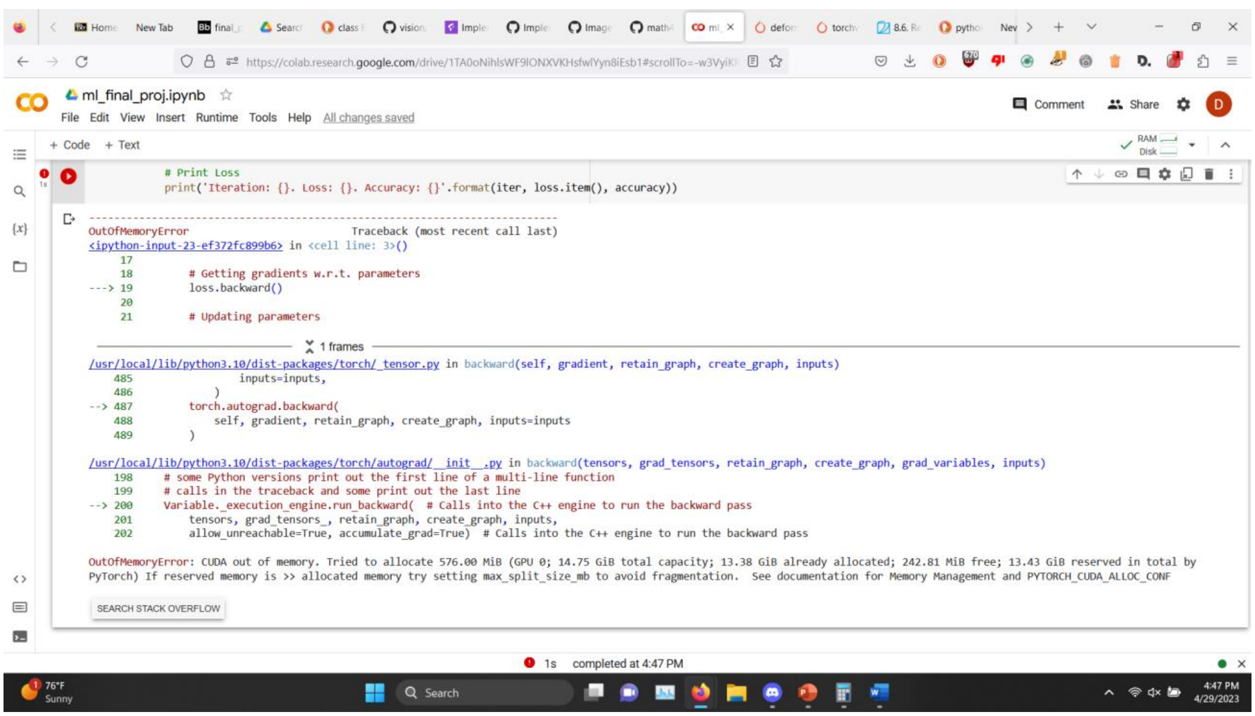
Wikipedia Contributors. "AlexNet." *Wikipedia*, Wikimedia Foundation, 18 Oct. 2019, en.wikipedia.org/wiki/AlexNet.

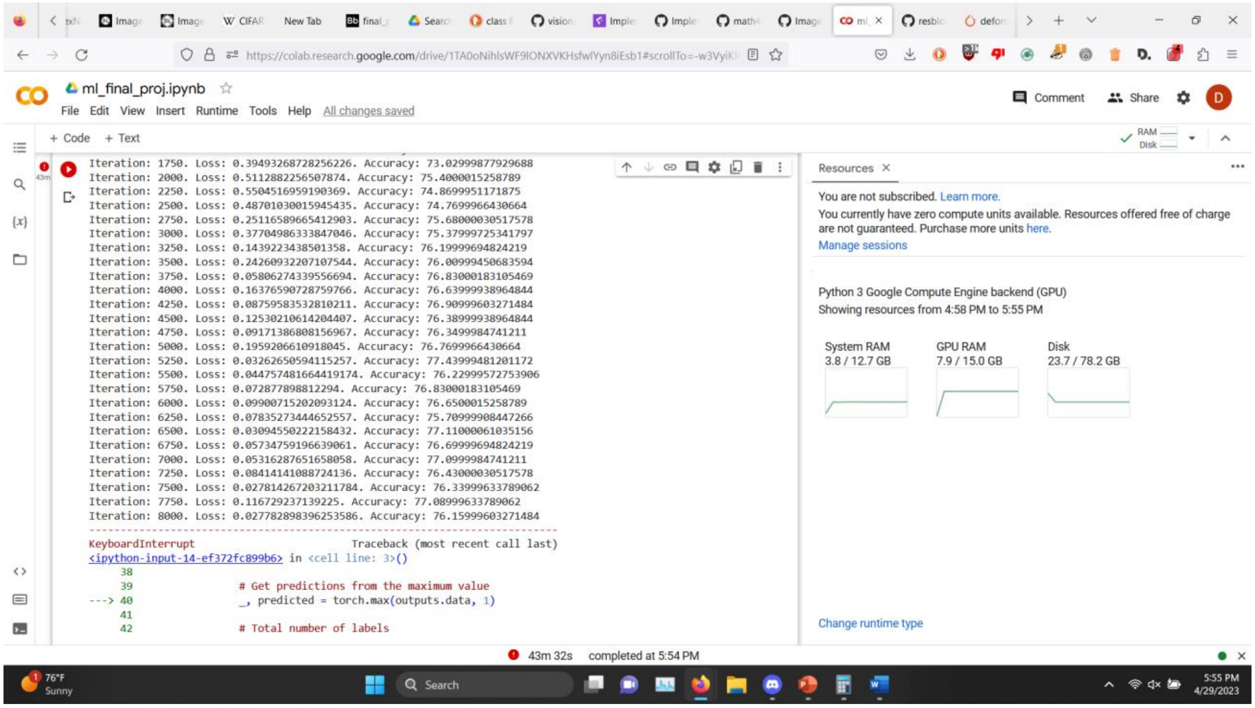
---. “Convolutional Neural Network.” *Wikipedia*, Wikimedia Foundation, 27 Feb. 2019, en.wikipedia.org/wiki/Convolutional_neural_network.

Other screenshots.

```
+ Text
Iteration: 500. Loss: 1.0641311407089233. Accuracy: 60.29999923706055
Iteration: 750. Loss: 1.035887360572815. Accuracy: 63.44999694824219
Iteration: 1000. Loss: 0.8321187496185303. Accuracy: 67.5
Iteration: 1250. Loss: 0.7875125408172607. Accuracy: 70.02999877929688
Iteration: 1500. Loss: 0.643134593963623. Accuracy: 72.36000061035156
Iteration: 1750. Loss: 0.5252638459205627. Accuracy: 72.83000183105469
Iteration: 2000. Loss: 0.512503981590271. Accuracy: 74.0999984741211
Iteration: 2250. Loss: 0.5846725702285767. Accuracy: 73.58999633789062
Iteration: 2500. Loss: 0.2877025008201599. Accuracy: 75.3699951171875
Iteration: 2750. Loss: 0.38471177220344543. Accuracy: 75.0199966430664
Iteration: 3000. Loss: 0.3463422954082489. Accuracy: 75.43999481201172
Iteration: 3250. Loss: 0.31843501329421997. Accuracy: 74.68000030517578
Iteration: 3500. Loss: 0.10883495211601257. Accuracy: 75.06999969482422
Iteration: 3750. Loss: 0.201067253947258. Accuracy: 74.68000030517578
Iteration: 4000. Loss: 0.20712165534496307. Accuracy: 75.48999786376953
Iteration: 4250. Loss: 0.08115792274475098. Accuracy: 75.68999481201172
Iteration: 4500. Loss: 0.18091925978660583. Accuracy: 76.08000183105469
Iteration: 4750. Loss: 0.09614812582731247. Accuracy: 75.4000015258789
Iteration: 5000. Loss: 0.21980181336402893. Accuracy: 75.68999481201172
Iteration: 5250. Loss: 0.06515239179134369. Accuracy: 75.54000091552734
Iteration: 5500. Loss: 0.14491713047027588. Accuracy: 75.52999877929688
Iteration: 5750. Loss: 0.07748299837112427. Accuracy: 76.50999450683594
Iteration: 6000. Loss: 0.07954555749893188. Accuracy: 75.91999816894531
Iteration: 6250. Loss: 0.11358100920915604. Accuracy: 75.69999694824219
Iteration: 6500. Loss: 0.12273155152797699. Accuracy: 76.40999603271484
Iteration: 6750. Loss: 0.027821458876132965. Accuracy: 76.25999450683594
Iteration: 7000. Loss: 0.039936259388923645. Accuracy: 75.83000183105469
Iteration: 7250. Loss: 0.052797265350818634. Accuracy: 75.00999450683594
Iteration: 7500. Loss: 0.11800876259803772. Accuracy: 76.25
Iteration: 7750. Loss: 0.04934846609830856. Accuracy: 75.77999877929688
Iteration: 8000. Loss: 0.018781498074531555. Accuracy: 76.77999877929688
Iteration: 8250. Loss: 0.02274753525853157. Accuracy: 76.87999725341797
Iteration: 8500. Loss: 0.026616979390382767. Accuracy: 76.43999481201172
Iteration: 8750. Loss: 0.020248951390385628. Accuracy: 76.20999908447266
Iteration: 9000. Loss: 0.013367736712098122. Accuracy: 76.52999877929688
1h 5
```


ml_final_proj.ipynb
File Edit View Insert Runtime Tools Help All changes saved
total += labels.size(0)
Total correct predictions
correct += (predicted == labels).sum()
accuracy = 100 * correct / total
# Print Loss
print('Iteration: {}. Loss: {}. Accuracy: {}'.format(iter, loss.item(), accuracy))
Iteration: 250. Loss: 1.555367350578308. Accuracy: 39.70000076293945
Iteration: 500. Loss: 1.3333473205566406. Accuracy: 51.959999084472656
Iteration: 750. Loss: 1.069186806678772. Accuracy: 58.369998931884766
Iteration: 1000. Loss: 0.9621341824531555. Accuracy: 61.82999801635742
Iteration: 1250. Loss: 0.8869265913963318. Accuracy: 65.79000091552734
Iteration: 1500. Loss: 0.7887985110282898. Accuracy: 68.13999938964844
Iteration: 1750. Loss: 0.7886393666267395. Accuracy: 69.41999816894531
Iteration: 2000. Loss: 0.6182852983474731. Accuracy: 71.93999481201172
Iteration: 2250. Loss: 0.660343587398529. Accuracy: 72.33000183105469
Iteration: 2500. Loss: 0.6991589069366455. Accuracy: 71.91999816894531
Iteration: 2750. Loss: 0.5409745573997498. Accuracy: 72.94999694824219
Iteration: 3000. Loss: 0.5506904125213623. Accuracy: 72.88999938964844
Iteration: 3250. Loss: 0.3805817663669586. Accuracy: 73.83000183105469
Iteration: 3500. Loss: 0.326873779296875. Accuracy: 74.27999877929688
Iteration: 3750. Loss: 0.20084454119205475. Accuracy: 73.0999984741211
Iteration: 4000. Loss: 0.4022735357284546. Accuracy: 72.04999542236328
Iteration: 4250. Loss: 0.2908753454685211. Accuracy: 73.22999572753906
KeyboardInterrupt Traceback (most recent call last)
<ipython-input-14-ef372fc899b6> in <cell line: 3>()
17
18 # Getting gradients w.r.t. parameters
---> 19 loss.backward()
4m 21s completed at 4:33 PM
76°F Sunny
4:35 PM 4/29/2023


ml_final_proj.ipynb
File Edit View Insert Runtime Tools Help All changes saved
# Print Loss
print('Iteration: {}. Loss: {}. Accuracy: {}'.format(iter, loss.item(), accuracy))
OutOfMemoryError Traceback (most recent call last)
<ipython-input-23-ef372fc899b6> in <cell line: 3>()
17
18 # Getting gradients w.r.t. parameters
---> 19 loss.backward()
20
21 # Updating parameters
1 frames
/usr/local/lib/python3.10/dist-packages/torch/_tensor.py in backward(self, gradient, retain_graph, create_graph, inputs)
485 inputs=inputs,
486)
--> 487 torch.autograd.backward(
488 self, gradient, retain_graph, create_graph, inputs=inputs
489)
/usr/local/lib/python3.10/dist-packages/torch/autograd/__init__.py in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables, inputs)
198 # some Python versions print out the first line of a multi-line function
199 # calls in the traceback and some print out the last line
--> 200 Variable._execution_engine.run_backward(# Calls into the C++ engine to run the backward pass
201 tensors, grad_tensors_, retain_graph, create_graph, inputs,
202 allow_unreachable=True, accumulate_grad=True) # Calls into the C++ engine to run the backward pass
OutOfMemoryError: CUDA out of memory. Tried to allocate 576.00 MiB (GPU 0; 14.75 GiB total capacity; 13.38 GiB already allocated; 242.81 MiB free; 13.43 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
SEARCH STACK OVERFLOW
1s completed at 4:47 PM
76°F Sunny
4:47 PM 4/29/2023



ml_final_proj.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Iteration: 1750. Loss: 0.39493268728256226. Accuracy: 73.02999877929688
Iteration: 2000. Loss: 0.5112882256507874. Accuracy: 75.4000015258789
Iteration: 2250. Loss: 0.5504516959190369. Accuracy: 74.8699951171875
Iteration: 2500. Loss: 0.48701030015945435. Accuracy: 74.7699966430664
Iteration: 2750. Loss: 0.25116589665412903. Accuracy: 75.68000030517578
Iteration: 3000. Loss: 0.37704986333847046. Accuracy: 75.37999725341797
Iteration: 3250. Loss: 0.1439223438501358. Accuracy: 76.19999694824219
Iteration: 3500. Loss: 0.24260932207107544. Accuracy: 76.00999450683594
Iteration: 3750. Loss: 0.05806274339556694. Accuracy: 76.83000183105469
Iteration: 4000. Loss: 0.16376590728759766. Accuracy: 76.63999938964844
Iteration: 4250. Loss: 0.08759583532810211. Accuracy: 76.90999603271484
Iteration: 4500. Loss: 0.12530210614204407. Accuracy: 76.38999938964844
Iteration: 4750. Loss: 0.09171386808156967. Accuracy: 76.3499984741211
Iteration: 5000. Loss: 0.1959206610918045. Accuracy: 76.7699966430664
Iteration: 5250. Loss: 0.03262650594115257. Accuracy: 77.43999481201172
Iteration: 5500. Loss: 0.044757481664419174. Accuracy: 76.22999572753906
Iteration: 5750. Loss: 0.072877898812294. Accuracy: 76.83000183105469
Iteration: 6000. Loss: 0.09900715202093124. Accuracy: 76.6500015258789
Iteration: 6250. Loss: 0.07835273444652557. Accuracy: 75.70999908447266
Iteration: 6500. Loss: 0.03094550222158432. Accuracy: 77.11000061035156
Iteration: 6750. Loss: 0.05734759196639061. Accuracy: 76.69999694824219
Iteration: 7000. Loss: 0.05316287651658058. Accuracy: 77.0999984741211
Iteration: 7250. Loss: 0.08414141088724136. Accuracy: 76.43000030517578
Iteration: 7500. Loss: 0.027814267203211784. Accuracy: 76.33999633789062
Iteration: 7750. Loss: 0.116729237139225. Accuracy: 77.08999633789062
Iteration: 8000. Loss: 0.027782898396253586. Accuracy: 76.15999603271484
KeyboardInterrupt Traceback (most recent call last)
<ipython-input-14-ef372fc899b6> in <cell line: 3>()
38
39 # Get predictions from the maximum value
---> 40 _, predicted = torch.max(outputs.data, 1)
41
42 # Total number of labels
Resources
You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions
Python 3 Google Compute Engine backend (GPU)
Showing resources from 4:58 PM to 5:55 PM
System RAM
3.8 / 12.7 GB
GPU RAM
7.9 / 15.0 GB
Disk
23.7 / 78.2 GB
Change runtime type
43m 32s completed at 5:54 PM
76°F
Sunny
Search
5:55 PM
4/29/2023