#### CMPT 762 X100, Fall 2024, Computer Vision

#### Project 2: Deep learning by PyTorch

Name: Wenhe Wang

Student Number: 301586596

Email: wwa118@sfu.ca

##### Improving BaseNet on CIFAR100

**Best accuracy on validation set: 77%**

**Best accuracy on 10% test set showing in Kaggle leaderboard: 78.2%**

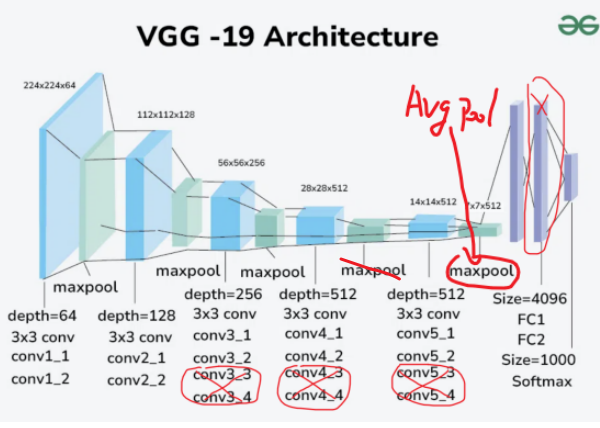
**Final model Architecture Table**

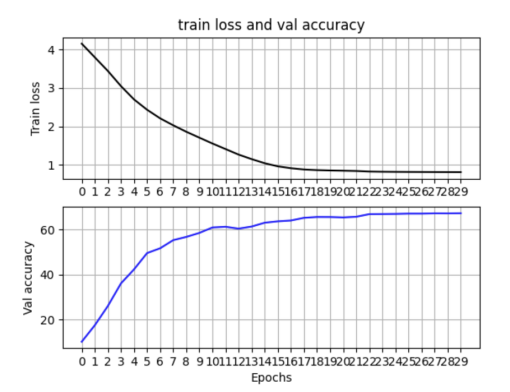
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer No. | Layer Type | Kernel size | Input|Output Dimension | Input|Output Channels |
| 1 | Conv2d | 3 | 128|128 | 3|64 |
| 2 | BatchNorm2d | - | 128|128 | - |
| 3 | ReLU | - | 128|128 | - |
| 4 | Conv2d | 3 | 128|128 | 64|64 |
| 5 | BatchNorm2d | - | 128|128 | - |
| 6 | ReLU | - | 128|128 | - |
| 7 | Maxpool2d | 2 | 128|64 |  |
| 8 | Conv2d | 3 | 64|64 | 64|128 |
| 9 | BatchNorm2d | - | 64|64 | - |
| 10 | ReLU | - | 64|64 | - |
| 11 | Conv2d | 3 | 64|64 | 128|128 |
| 12 | BatchNorm2d | - | 64|64 | - |
| 13 | ReLU | - | 64|64 | - |
| 14 | Maxpool2d | 2 | 64|32 | - |
| 15 | Conv2d | 3 | 32|32 | 128|256 |
| 16 | BatchNorm2d | - | 32|32 | - |
| 17 | ReLU | - | 32|32 | - |
| 18 | Conv2d | 3 | 32|32 | 256|256 |
| 19 | BatchNorm2d | - | 32|32 | - |
| 20 | ReLU | - | 32|32 | - |
| 21 | Maxpool2d | 2 | 32|16 | - |
| 22 | Conv2d | 3 | 16|16 | 256|512 |
| 23 | BatchNorm2d | - | 16|16 | - |
| 24 | ReLU | - | 16|16 | - |
| 25 | Conv2d | 3 | 16|16 | 512|512 |
| 26 | BatchNorm2d | - | 16|16 | - |
| 27 | ReLU | - | 16|16 | - |
| 28 | AdaptiveAvgPool2d | - | 16|1 | - |
| 29 | Flatten | - | 1|512 | - |
| 30 | Linear | - | 512|4096 | - |
| 31 | ReLU | - | 4096|4096 | - |
| 32 | Linear | - | 4096|100 | - |

My customized model is inspired by VGG-19 and ResNet-18 but modified for improved performance and efficiency. It consists of five convolutional blocks, each with BatchNorm layers to stabilize training. MaxPooling layers are omitted to retain more spatial information, and AdaptiveAvgPool is applied after the fifth block for global feature extraction. The fully connected layers are reduced to streamline the model, and BatchNorm is removed from the linear layers. This design balances complexity and generalization. While testing popular models such as VGG-19, ResNet-18, and GoogleNet, my customized VGG-19 consistently outperformed them without modifying their original architectures.

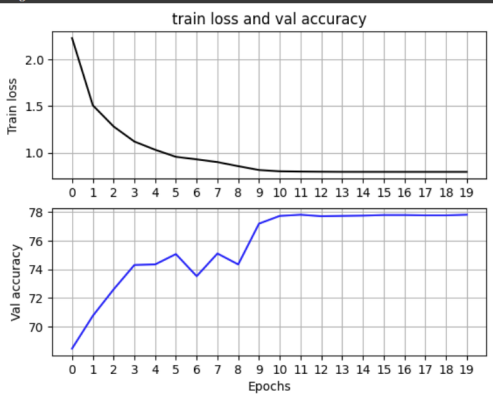
**Training and Validation Loss and Accuracy**

Training with optimized VGG-19 model + the normalized + resized 128x128 images (as the 14th step in ablation study):





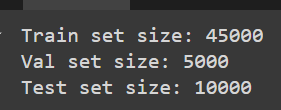
Final round training with 700 epochs data augmentation (as the 15th step in ablation study):



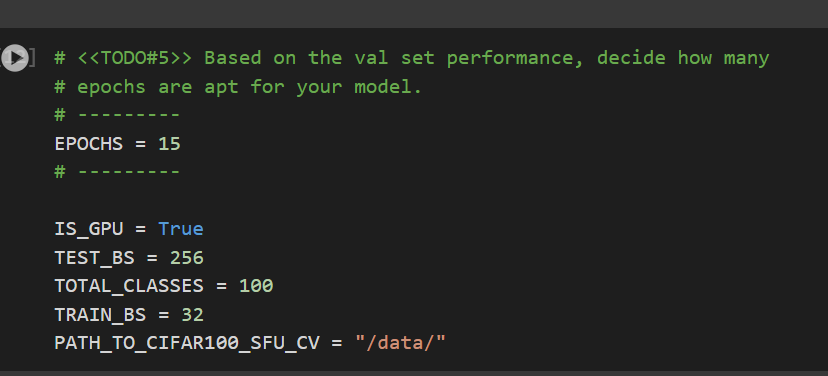
Because the training plateaued after epoch 10 in our last round of training with the original images, so to prevent overfitting, I manually stopped the finetuning process at epoch 20.

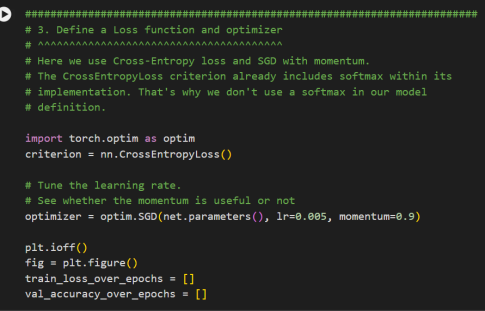
**Ablation Study**

Dataset Size:

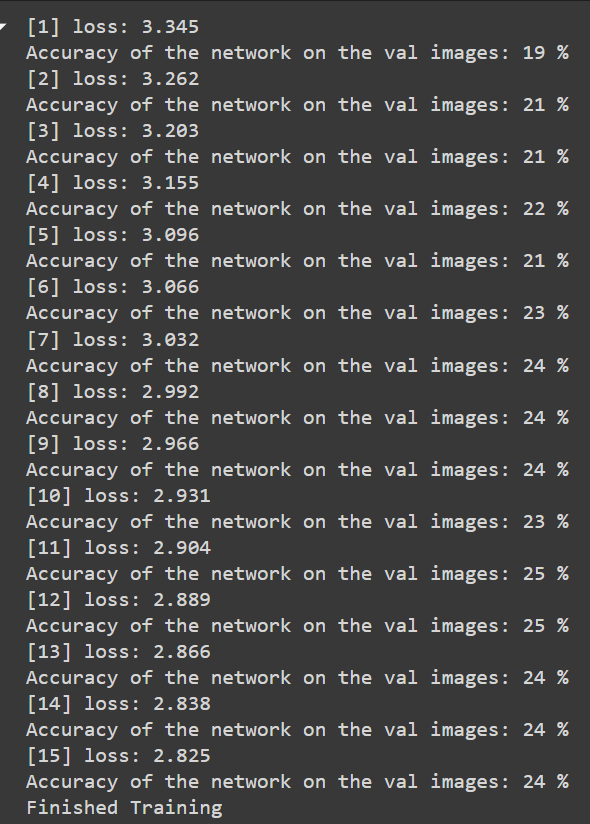


Baseline Hyper-parameters:





Baseline model (default BaseNet) :



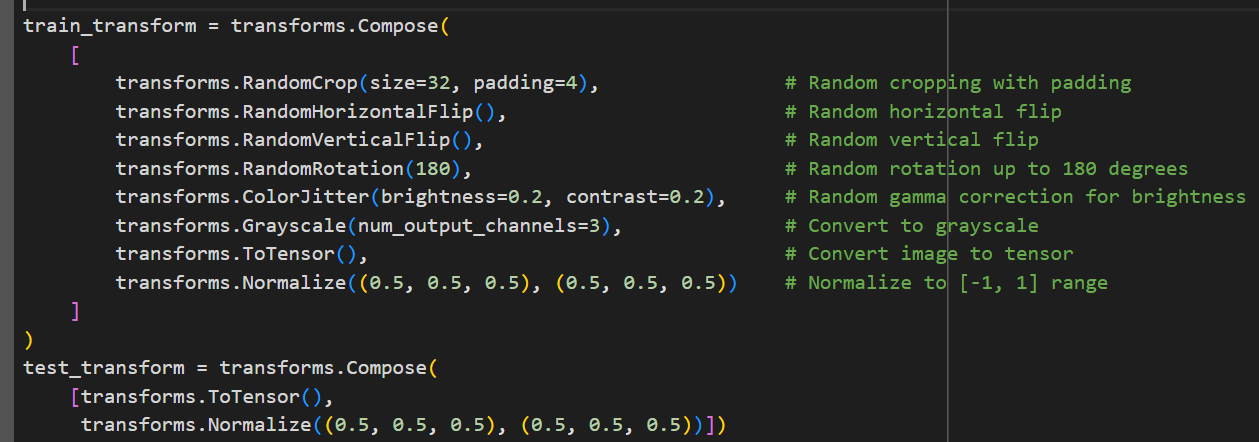
1. Baseline + data normalization ((0, 0, 0), (1, 1, 1))

Accuracy on val: 24% -> 26%

1. Baseline + data normalization ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

Accuracy on val: 24% -> 26%

1. Baseline + data normalization + data augmentation:



Accuracy on val: 24% -> 8%

The initial baseline may be too simple to capture features with multiple augmentations applied simultaneously. A better approach could be to apply each augmentation separately across the entire training set, resulting in 7 variations of the 45,000 images. However, this would significantly increase training time. For now, we'll hold off on this and focus on designing a new model architecture. Even with 45,000 images and multiple augmentations using default hyper-parameters, the baseline takes around 13 minutes to train with 15 epochs.

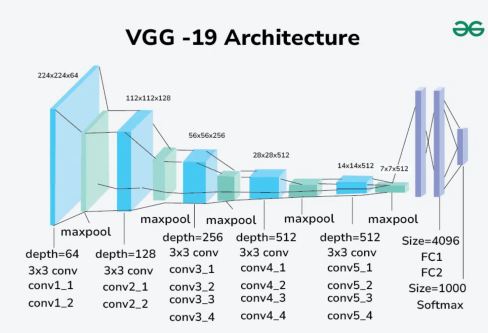
1. Baseline + data normalization + data augmentation (without grayscale):

Accuracy on val: 24% -> 19%

The grayscale method often used in segmentation to capture shape/edge information, isn't suitable for our classification task as it removes color information, which is crucial when combined with other augmentation methods.

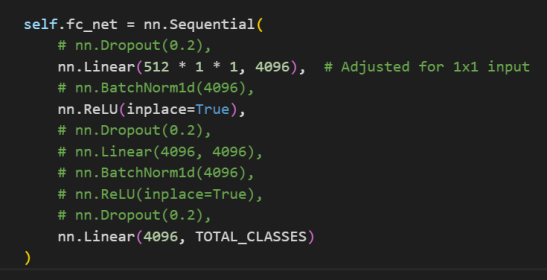
1. VGG-19 + data normalization:

Accuracy on val: 26% -> 49%



1. VGG-19 (delete the middle FC layer) + data normalization:

Accuracy on val: 49% -> 51%



In the experiment with the classification head design, I tested applying BatchNorm1d after each FC layer and after all FC layers, along with adding dropout. However, none of these approaches improved accuracy; instead, accuracy significantly dropped to around 25-30%.

1. VGG-19 (delete the middle FC layer and Max Pooling layers) + data normalization:

Accuracy on val: 51% -> 40%

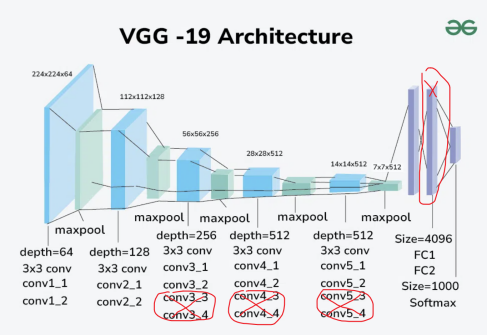
In the experiment with the VGG body design, I tested deleting Max Pooling layer after each convolution block and after all blocks. However, accuracy dropped to around 35-40%.

1. VGG-19 (delete the middle FC layer) + data normalization + He’s initialization:

Accuracy on val: 51% -> 38%

1. VGG-19 (delete the middle FC layer and 6 Conv layers and 6 Batch Norm layers) + data normalization:

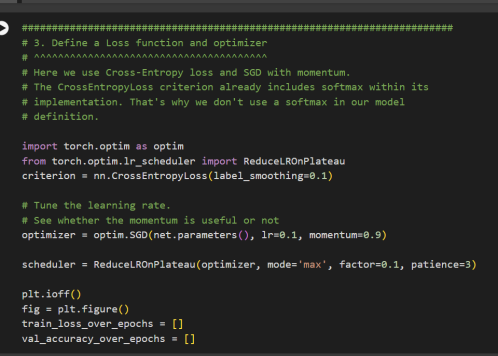
Accuracy on val: 51% -> 56%



1. Step 9 + ReduceLROnPlateau scheduler + 30 epochs:

Accuracy on val: 56% -> 58%

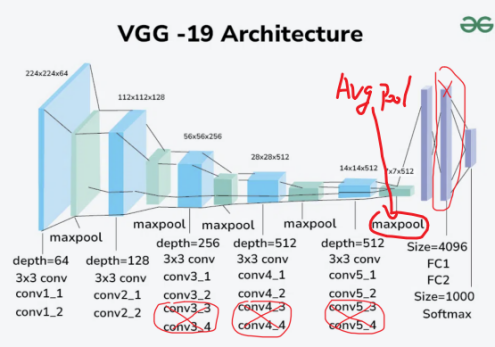
The learning rate will decrease when the validation accuracy plateaus.



1. Step 10 - the maxpool after block 5 + AdaptiveAvgPool2d:

Accuracy on val: 58% -> 58%

Although the validation accuracy didn't improve, using average pooling after the fifth Conv block led to faster and more stable convergence to 58% accuracy during training compared to max pooling. Referring to ResNet-18 design, average pooling at the end provides a more generalized and smooth representation of the learned features, helps reduce overfitting.



1. Step 11 + image resize to 64:

Accuracy on val: 58% -> 62%

Training time increased significantly: training with 32x32 images took around 25-30 seconds per epoch, while 64x64 images took about 1.5 minutes per epoch.

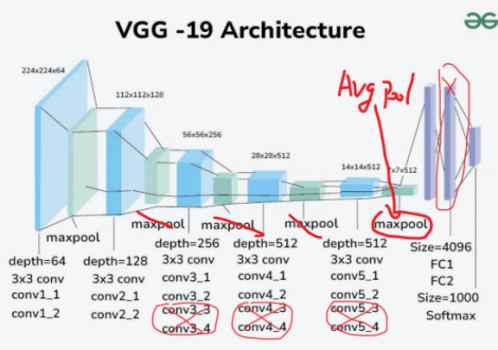
1. Step 11 + image resize to 128:

Accuracy on val: 58% -> 67%

Training with 128x128 images took about 4.5 minutes per epoch. To save time and resources, we will just keep our image size to 128x128 in our following steps without keep exploring resize to higher resolution.

1. Step 11 - the maxpool after block 2, 3, 4:

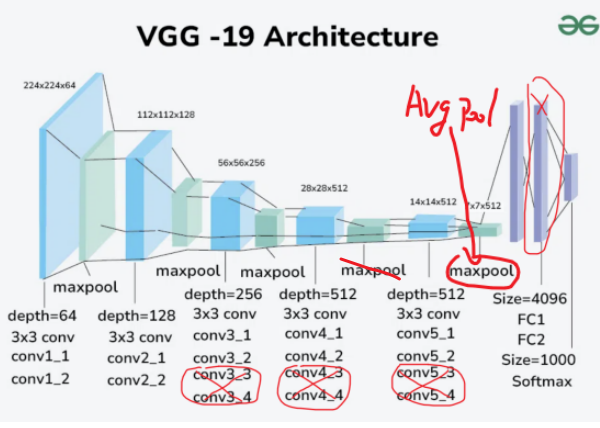
Accuracy on val: 58% -> 61%

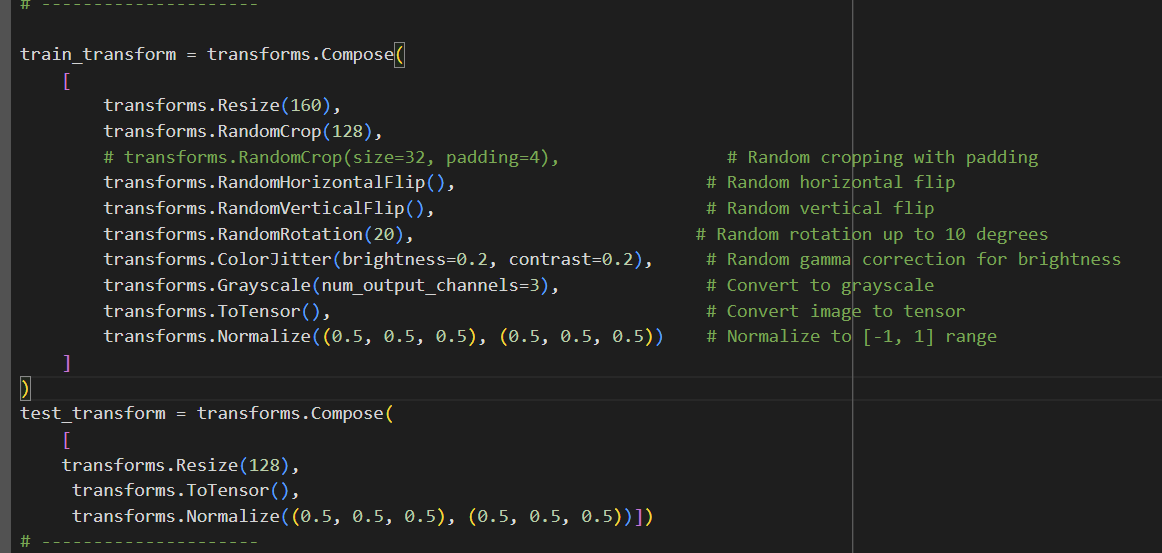


With only the first one maxpool layer as shown in the image, training on one epoch and 64 batch size and one A100 GPU takes 3 min and 36.3 GB GPU RAM. So we have to keep the maxpool layer 1, 2 and 3 to reduce the computation cost.

1. Step 14 + maxpool after block 2, 3 + image resize to 128 + data augmentation(6 methods individually):

Accuracy on val: 67% -> 77%



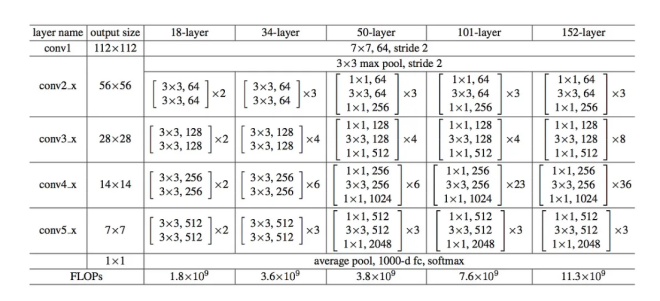


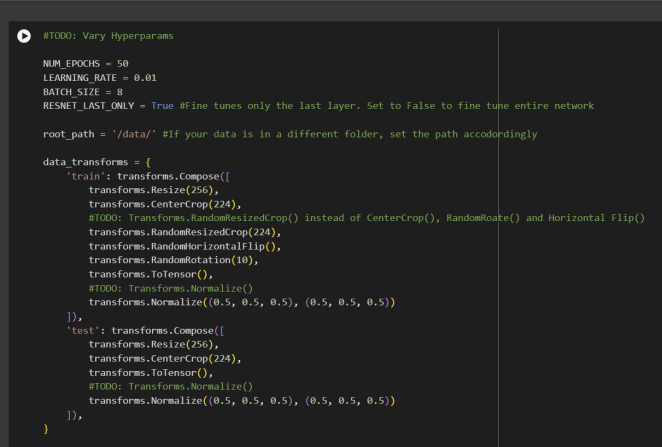
When training VGG-19, ResNet-18, VGG-19-ResNet-18 Mix, VGG-19-ResNet-18-Concat Mix, GoogleNet, or ResNetV2 with mixed data augmentations (random crop, horizontal/vertical flip, rotation, gamma correction, grayscale), the models showed little improvement, with accuracy stuck around 68-71%. However, when training with each augmentation separately for 100 epochs (a total of 700 epochs), the models showed significant improvement.

##### Transfer Learning

**Model and Hyper-parameters**

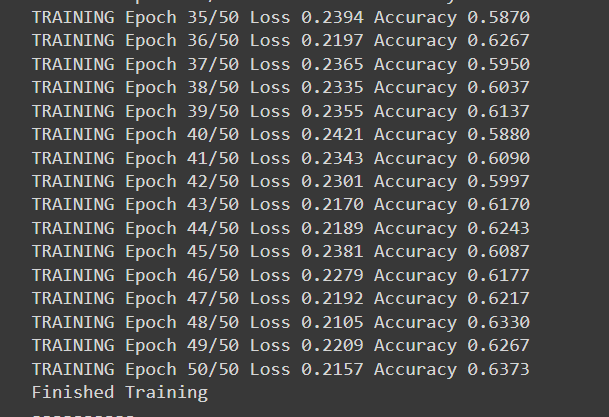
Because the size of ResNet-18, ResNet-34 and ResNet-50 are pretty similar, so I just used ResNet-50 as my backbone model. And as per the hyper-parameters and data augmentation methods I used in my first part, I just used the same settings here.



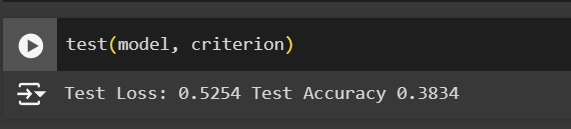


**Using the ResNet-50 as a fixed feature extractor (RESNET\_LAST\_ONLY = True)**

Training accuracy: 63.73%

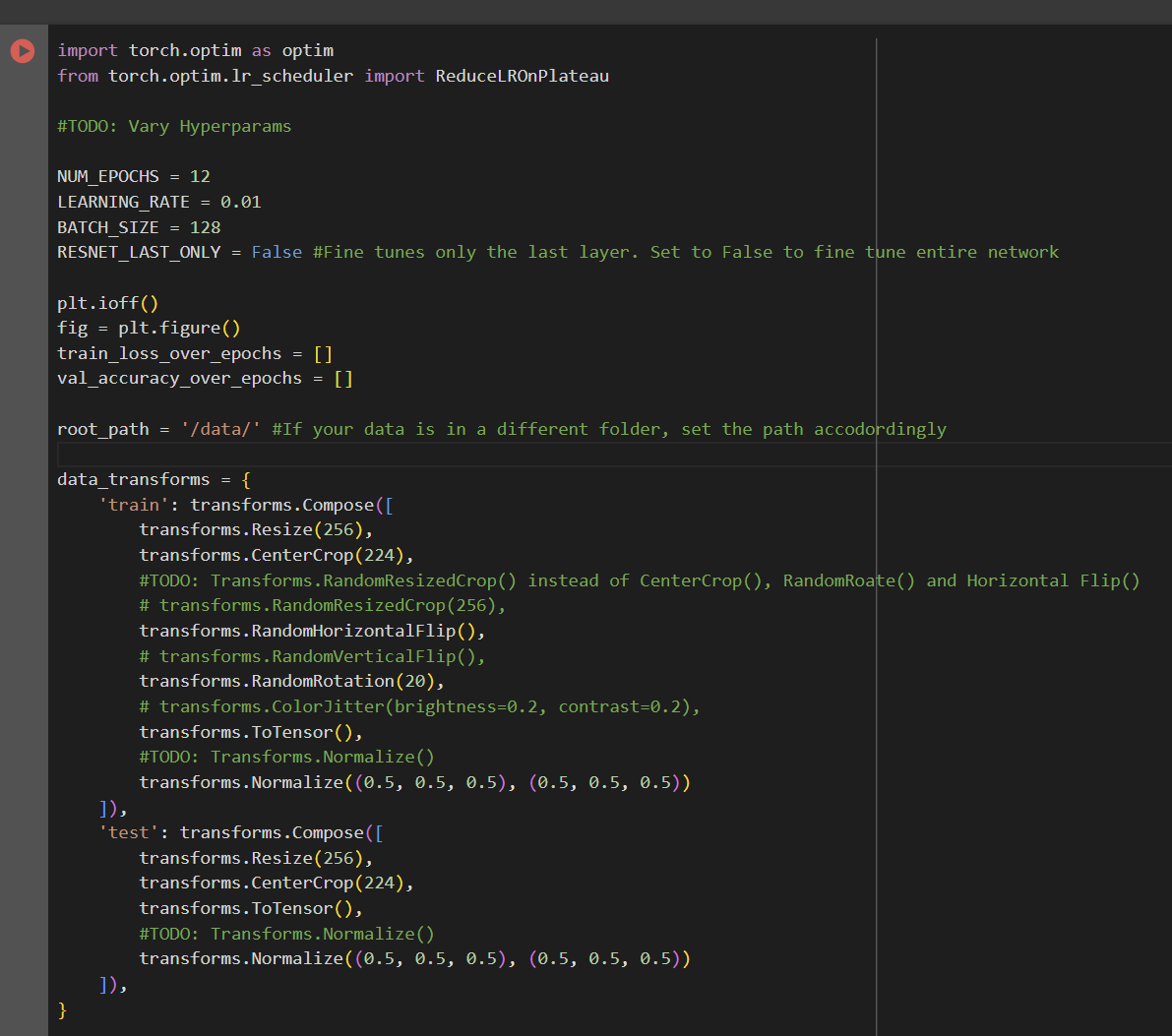


Test accuracy: 38.34%

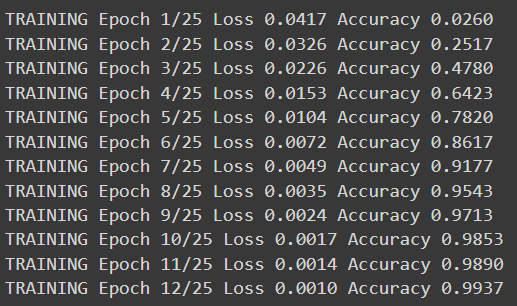


**Fine-tuning the whole network (RESNET\_LAST\_ONLY = False)**

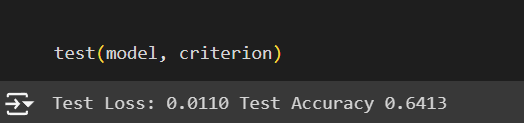
After many experiments, this hyper-parameter settings worked the best.



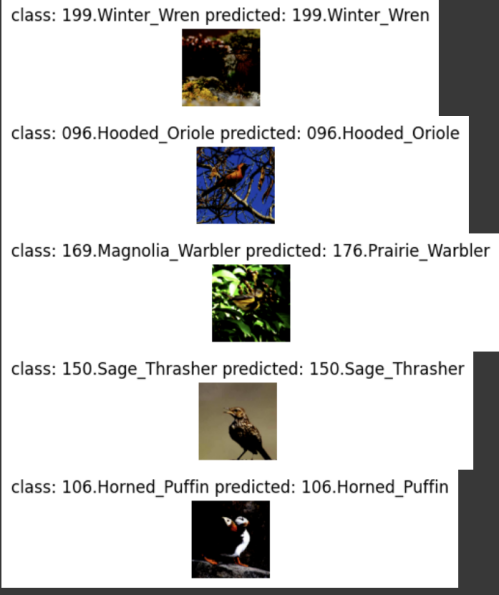
Training accuracy: 99.37%



Test accuracy: 64.13%



**Visualizing the model predictions**



**Reference**

1. VGG-Net Architecture Explained. (2024, June 07). Retrieved October 12, 2024, from https://www.geeksforgeeks.org/vgg-net-architecture-explained/
2. Big Transfer (BiT): General Visual Representation Learning. (2021, June 18). Retrieved October 12, 2024, from https://github.com/google-research/big\_transfer
3. Structure of the Resnet-18 Model. (2022, December). Retrieved October 12, 2024, from https://www.researchgate.net/figure/Structure-of-the-Resnet-18-Model\_fig1\_366608244
4. Pytorch-cifar100. (2022, March 27). Retrieved October 12, 2024, from https://github.com/weiaicunzai/pytorch-cifar100
5. GoogleNet. (2020, May 5). Retrieved October 12, 2024, from https://github.com/pytorch/vision/blob/6db1569c89094cf23f3bc41f79275c45e9fcb3f3/torchvision/models/googlenet.py#L62
6. Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky (2009). Retrieved October 12, 2024, from https://www.cs.toronto.edu/%7Ekriz/cifar.html
7. ResNet (34, 50, 101)…what actually it is ?, chandini (2020). Retrieved October 13, 2024, from https://medium.com/@aschandinip/resnet-34-50-101-what-actually-it-is-c63da24ba695