I couldn’t get the code to work, but this is straight from the reference of <https://levelup.gitconnected.com/fruits-vegetables-and-deep-learning-c5814c59fcc9>

I tried plugging it into Google Colab, but there seems to be files missing.

**#CODE:**

**import** os

**import** torch

**import** torchvision

**import** tarfile

**import** torch**.**nn **as** nn

**import** numpy **as** np

**import** torch**.**nn**.**functional **as** F

**from** torchvision**.**datasets**.**utils **import** download\_url

**from** torchvision**.**datasets **import** ImageFolder

**from** torch**.**utils**.**data **import** DataLoader

**import** torchvision**.**transforms **as** tt

**from** torch**.**utils**.**data **import** random\_split

**from** torchvision**.**utils **import** make\_grid

**import** matplotlib**.**pyplot **as** plt

**%**matplotlib inline

**from** tqdm**.**notebook **import** tqdm

**import** torchvision**.**models **as** models

# Load the directory paths to the dataset

DATA\_DIR **=** '../content/fruits-360'

TRAIN\_DIR **=** DATA\_DIR **+** '/Training'

TEST\_DIR **=** DATA\_DIR **+** '/Test'

train\_tfms **=** tt**.**Compose**([**tt**.**RandomCrop**(**100**,** padding**=**10**,** padding\_mode**=**'reflect'**),**

tt**.**RandomHorizontalFlip**(),**

tt**.**RandomRotation**(**20**),**

tt**.**ToTensor**()**

**])**

valid\_tfms **=** tt**.**Compose**([**tt**.**ToTensor

train\_ds **=** ImageFolder**(**TRAIN\_DIR**,** train\_tfms**)**

valid\_ds **=** ImageFolder**(**TEST\_DIR**,** val id\_tfms**)**

batch\_size\_custom **=** 32 # Batch size for custom CNN model

batch\_size\_resnet **=** 32 # Batch size for resnet CNN model

random\_seed **=** 42

torch**.**manual\_seed**(**random\_seed**);**

train\_dl\_custom **=** DataLoader**(**train\_ds**,** batch\_size\_custom**,** shuffle**=True,** num\_workers**=**3**,** pin\_memory**=True)**

valid\_dl\_custom **=** DataLoader**(**val id\_ds**,** batch\_size\_custom**\***2**,** num\_workers**=**3**,** pin\_memory**=True)**

train\_dl\_resnet **=** DataLoader**(**train\_ds**,** batch\_size\_resnet**,** shuffle**=True,** num\_workers**=**3**,** pin\_memory**=True)**

valid\_dl\_resnet **=** DataLoader**(**valid\_ds**,** batch\_size\_resnet**\***2**,** num\_workers**=**3**,** pin\_memory**=True)**

**def** get\_default\_device**():**

"""Pick GPU if available, else CPU"""

**if** torch**.**cuda**.**is\_available**():**

**return** torch**.**device**(**'cuda'**)**

**else:**

**return** torch**.**device**(**'cpu'**)**

**def** to\_device**(**data**,** device**):**

"""Move tensor(s) to chosen device"""

**if** **isinstance(**data**,** **(list,tuple)):**

**return** **[**to\_device**(**x**,** device**)** **for** x **in** data**]**

**return** data**.**to**(**device**,** non\_blocking**=True)**

**class** **DeviceDataLoader():**

"""Wrap a dataloader to move data to a device"""

**def** \_\_init\_\_**(**self**,** dl**,** device**):**

self**.**dl **=** dl

self**.**device **=** device

**def** \_\_iter\_\_**(**self**):**

"""Yield a batch of data after moving it to device"""

**for** b **in** self**.**dl**:**

**yield** to\_device**(**b**,** self**.**device**)**

**def** \_\_len\_\_**(**self**):**

"""Number of batches"""

**return** **len(**self**.**dl**)**

# Device Data Loader for Custom CNN Model

train\_dl\_custom **=** DeviceDataLoader**(**train\_dl\_custom**,** device**)**

valid\_dl\_custom **=** DeviceDataLoader**(**valid\_dl\_custom**,** device**)**

train\_dl\_resnet **=** DeviceDataLoader**(**train\_dl\_resnet**,** device**)**

valid\_dl\_resnet **=** DeviceDataLoader**(**valid\_dl\_resnet**,** device**)**

**def** accuracy**(**outputs**,** labels**):**

\_**,** preds **=** torch**.max(**outputs**,** dim**=**1**)**

**return** torch**.**tensor**(**torch**.sum(**preds **==** labels**).**item**()** **/** **len(**preds**))**

**class** **ImageClassificationBase(**nn**.**Module**):**

**def** training\_step**(**self**,** batch**):**

images**,** labels **=** batch

out **=** self**(**images**)**

loss **=** F**.**cross\_entropy**(**out**,** labels**)** # Calculate training loss

**return** loss

**def** validation\_step**(**self**,** batch**):**

images**,** labels **=** batch

out **=** self**(**images**)** # Generate predictions

loss **=** F**.**cross\_entropy**(**out**,** labels**)** # Calculate validation loss

acc **=** accuracy**(**out**,** labels**)** # Calculate accuracy

**return** **{**'val\_loss'**:** loss**.**detach**(),** 'val\_acc'**:** acc**}**

**def** validation\_epoch\_end**(**self**,** outputs**):**

batch\_losses **=** **[**x**[**'val\_loss'**]** **for** x **in** outputs**]**

epoch\_loss **=** torch**.**stack**(**batch\_losses**).**mean**()** # Combine losses

batch\_accs **=** **[**x**[**'val\_acc'**]** **for** x **in** outputs**]**

epoch\_acc **=** torch**.**stack**(**batch\_accs**).**mean**()** # Combine accuracies

**return** **{**'val\_loss'**:** epoch\_loss**.**item**(),** 'val\_acc'**:** epoch\_acc**.**item**()}**

**def** epoch\_end**(**self**,** epoch**,** result**):**

**print(**"Epoch [{}], last\_lr: {:.10f}, train\_loss: {:.4f}, val\_loss: {:.4f}, val\_acc: {:.4f}"**.format(**

epoch**,** result**[**'lrs'**][-**1**],** result**[**'train\_loss'**],** result**[**'val\_loss'**],** result**[**'val\_acc'**]))**

**def** conv\_block**(**in\_channels**,** out\_channels**,** pool**=False):**

layers **=** **[**nn**.**Conv2d**(**in\_channels**,** out\_channels**,** kernel\_size**=**3**,** padding**=**1**),**

nn**.**BatchNorm2d**(**out\_channels**),** # Batch Normalization

nn**.**ReLU**(**inplace**=True)]**

**if** pool**:** layers**.**append**(**nn**.**MaxPool2d**(**2**))**

**return** nn**.**Sequential**(\***layers**)**

**class** **CustomCNN(**ImageClassificationBase**):**

**def** \_\_init\_\_**(**self**,** in\_channels**,** num\_classes**):**

**super().**\_\_init\_\_**()**

self**.**conv1 **=** conv\_block**(**in\_channels**,** 128**)** # 3 x 64 x 64

self**.**conv2 **=** conv\_block**(**128**,** 256**,** pool**=True)** # 128 x 32 x 32

self**.**res1 **=** nn**.**Sequential**(**conv\_block**(**256**,** 256**),** conv\_block**(**256**,** 256**))** # 256 x 32 x 32

self**.**conv3 **=** conv\_block**(**256**,** 512**,** pool**=True)** # 512 x 16 x 16

self**.**conv4 **=** conv\_block**(**512**,** 1024**,** pool**=True)** # 1024 x 8 x 8

self**.**res2 **=** nn**.**Sequential**(**conv\_block**(**1024**,** 1024**),** conv\_block**(**1024**,** 1024**))** # 1024 x 8 x 8

self**.**conv5 **=** conv\_block**(**1024**,** 2048**,** pool**=True)** # 256 x 8 x 8

self**.**conv6 **=** conv\_block**(**2048**,** 4096**,** pool**=True)** # 512 x 4 x 4

self**.**res3 **=** nn**.**Sequential**(**conv\_block**(**4096**,** 4096**),** conv\_block**(**4096**,** 4096**))** # 512 x 4 x 4

self**.**classifier **=** nn**.**Sequential**(**nn**.**MaxPool2d**(**4**),** # 9216 x 1 x 1

nn**.**Flatten**(),** # 9216

nn**.**Linear**(**9216**,** num\_classes**))** # 131

**def** forward**(**self**,** xb**):**

out **=** self**.**conv1**(**xb**)**

out **=** self**.**conv2**(**out**)**

out **=** self**.**res1**(**out**)** **+** out # Residual Block

out **=** self**.**conv3**(**out**)**

out **=** self**.**conv4**(**out**)**

out **=** self**.**res2**(**out**)** **+** out # Residual Block

out **=** self**.**classifier**(**out**)**

**return** out

**class** **ResNetCNN(**ImageClassificationBase**):**

**def** \_\_init\_\_**(**self**):**

**super().**\_\_init\_\_**()**

# Use a pretrained model

self**.**network **=** models**.**resnet34**(**pretrained**=True)** # You can change the resnet model here

# Replace last layer

num\_ftrs **=** self**.**network**.**fc**.**in\_features

self**.**network**.**fc **=** nn**.**Linear**(**num\_ftrs**,** 131**)** # Output classes

**def** forward**(**self**,** xb**):**

**return** torch**.**sigmoid**(**self**.**network**(**xb**))**

**def** freeze**(**self**):**

# To freeze the residual layers

**for** param **in** self**.**network**.**parameters**():**

param**.**require\_grad **=** **False**

**for** param **in** self**.**network**.**fc**.**parameters**():**

param**.**require\_grad **=** **True**

**def** unfreeze**(**self**):**

# Unfreeze all layers

**for** param **in** self**.**network**.**parameters**():**

param**.**require\_grad **=** Tru

*@torch***.**no\_grad**()**

**def** evaluate**(**model**,** val\_loader**):**

**print(**'Evaluating Model ...'**)**

model**.eval()**

outputs **=** **[**model**.**validation\_step**(**batch**)** **for** batch **in** tqdm**(**val\_loader**)]**

**return** model**.**validation\_epoch\_end**(**outputs**)**

**def** get\_lr**(**optimizer**):**

**for** param\_group **in** optimizer**.**param\_groups**:**

**return** param\_group**[**'lr'**]**

**def** fit\_one\_cycle**(**epochs**,** max\_lr**,** model**,** train\_loader**,** val\_loader**,**

weight\_decay**=**0**,** grad\_clip**=None,** opt\_func**=**torch**.**optim**.**SGD**):**

torch**.**cuda**.**empty\_cache**()**

history **=** **[]**

# Set up cutom optimizer with weight decay

optimizer **=** opt\_func**(**model**.**parameters**(),** max\_lr**,** weight\_decay**=**weight\_decay**)**

# Set up one-cycle learning rate scheduler

sched **=** torch**.**optim**.**lr\_scheduler**.**OneCycleLR**(**optimizer**,** max\_lr**,** epochs**=**epochs**,**

steps\_per\_epoch**=len(**train\_loader**))**

**for** epoch **in** **range(**epochs**):**

# Training Phase

model**.**train**()**

train\_losses **=** **[]**

lrs **=** **[]**

**print(**'\nTraining Model ...'**)**

**for** batch **in** tqdm**(**train\_loader**):**

loss **=** model**.**training\_step**(**batch**)**

train\_losses**.**append**(**loss**)**

loss**.**backward**()**

# Gradient clipping

**if** grad\_clip**:**

nn**.**utils**.**clip\_grad\_value\_**(**model**.**parameters**(),** grad\_clip**)**

optimizer**.**step**()**

optimizer**.**zero\_grad**()**

# Record & update learning rate

lrs**.**append**(**get\_lr**(**optimizer**))**

sched**.**step**()**

# Validation phase

result **=** evaluate**(**model**,** val\_loader**)**

result**[**'train\_loss'**]** **=** torch**.**stack**(**train\_losses**).**mean**().**item**()**

result**[**'lrs'**]** **=** lrs

model**.**epoch\_end**(**epoch**,** result**)**

history**.**append**(**result**)**

Return history

epochs **=** 10

max\_lr **=** 1e-3

grad\_clip **=** 1e-1

weight\_decay **=** 1e-4

opt\_func **=** torch**.**optim**.**Adam

input\_channels **=** 3

output\_classes **=** 131

custom\_model **=** to\_device**(**CustomCNN**(**input\_channels**,** output\_classes**),** device**)**

custom\_model

history\_CustomCNN **=** **[**evaluate**(**custom\_model**,** valid\_dl\_custom**)]**

history\_CustomCNN

**%%**time

history\_CustomCNN **+=** fit\_one\_cycle**(**epochs**,** max\_lr**,** custom\_model**,** train\_dl\_custom**,** valid\_dl\_custom**,**

grad\_clip**=**grad\_clip**,**

weight\_decay**=**weight\_decay**,**

op

resnet\_model **=** to\_device**(**ResNetCNN**(),** device**)**

resnet\_model

history\_ResNetCNN **=** **[**evaluate**(**resnet\_model**,** valid\_dl\_resnet**)]**

history\_ResNetCNN

resnet\_model**.**freeze**()**

**%%**time

history\_ResNetCNN **+=** fit\_one\_cycle**(**5**,** 1e-2**,** resnet\_model**,** train\_dl\_resnet**,** valid\_dl\_resnet**,**

grad\_clip**=**grad\_clip**,**

weight\_decay**=**weight\_decay**,**

op t\_func**=**opt\_func**)**

resnet\_model**.**unfreeze**()**

**%%**time

history\_ResNetCNN **+=** fit\_one\_cycle**(**5**,** 1e-3**,** resnet\_model**,** train\_dl\_resnet**,** valid\_dl\_resnet**,**

grad\_clip**=**grad\_clip**,**

weight\_decay**=**weight\_decay**,**

opt\_func**=**opt\_func**)**