model_resisc45

April 15, 2023

1 Model RESISC 45

[ ]:
```python
import os
import pandas as pd
import cv2
import matplotlib.pyplot as plt
import numpy as np
import logging
import contextlib
import random
import platform
import torch
from torch.utils.data import Dataset, DataLoader
import torchvision
from torchvision.transforms import ToTensor
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import torch.optim as optim
from PIL import Image


logger = logging.getLogger('train')
logger.setLevel(logging.INFO)

print(platform.platform())  # print current platform
print("PyTorch Version: ",torch.__version__)
print("Torchvision Version: ",torchvision.__version__)
```

```
macOS-13.3.1-arm64-arm-64bit
PyTorch Version:  2.0.0
Torchvision Version:  0.15.1
```

1.1 Set constants

[ ]:
```python
DIRPATH = "/Users/cristianion/Desktop/satimg_data/NWPU-RESISC45"

LABELS = [
    'forest',
    'railway_station',
    'tennis_court',
    'basketball_court',
    'river',
    'storage_tank',
    'harbor',
    'terrace',
    'thermal_power_station',
    'golf_course',
    'runway',
    'roundabout',
    'bridge',
    'industrial_area',
    'baseball_diamond',
    'mobile_home_park',
    'overpass',
    'church',
    'chaparral',
    'railway',
    'stadium',
    'medium_residential',
    'sea_ice',
    'intersection',
    'lake',
    'palace',
    'airplane',
    'cloud',
    'sparse_residential',
    'airport',
    'snowberg',
    'parking_lot',
    'commercial_area',
    'rectangular_farmland',
    'island',
    'beach',
    'circular_farmland',
    'dense_residential',
    'ship',
    'mountain',
    'desert',
    'freeway',
    'meadow',
    'wetland',
    'ground_track_field',
]

K_FOLDS = 5
```

```
[ ]: print(len(LABELS))  # number of expected labels
```

```
45
```

2 Prepare dataset

```
[ ]: filelist = [(f"{DIRPATH}/{f}", LABELS[LABELS.index(f)]) for f in os.
      ↪listdir(DIRPATH) if f in LABELS]
     df = pd.DataFrame(filelist, columns=['dirpath', 'label'])
```

```
[ ]: df.head()
```

```
[ ]:                                              dirpath             label
     0  /Users/cristianion/Desktop/satimg_data/NWPU-RE…            forest
     1  /Users/cristianion/Desktop/satimg_data/NWPU-RE…   railway_station
     2  /Users/cristianion/Desktop/satimg_data/NWPU-RE…      tennis_court
     3  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  basketball_court
     4  /Users/cristianion/Desktop/satimg_data/NWPU-RE…             river
```

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   dirpath  45 non-null     object
 1   label    45 non-null     object
dtypes: object(2)
memory usage: 852.0+ bytes
```

Conclusions - 45 directories found corresponding to 45 labels

```
[ ]: data = []
     for i, DIRPATH in enumerate(df["dirpath"]):
         images = os.listdir(DIRPATH)
         images = [f"{DIRPATH}/{img}" for img in images]
         rows = [(img, df['label'][i]) for img in images]
         data.extend(rows)
     data = pd.DataFrame(data, columns=["imgpath", "label"])
```

```
[ ]: data.head()
```

```
[ ]:                                              imgpath   label
     0  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  forest
     1  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  forest
     2  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  forest
```

```
3  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  forest
4  /Users/cristianion/Desktop/satimg_data/NWPU-RE…  forest
```

[ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31500 entries, 0 to 31499
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   imgpath  31500 non-null  object
 1   label    31500 non-null  object
dtypes: object(2)
memory usage: 492.3+ KB
```

Conclusions: - 31500 annotated images

[ ]:
```
data['label'].value_counts()
```

[ ]:
```
label
forest                  700
intersection            700
palace                  700
airplane                700
cloud                   700
sparse_residential      700
airport                 700
snowberg                700
parking_lot             700
commercial_area         700
rectangular_farmland    700
island                  700
beach                   700
circular_farmland       700
dense_residential       700
ship                    700
mountain                700
desert                  700
freeway                 700
meadow                  700
wetland                 700
lake                    700
sea_ice                 700
railway_station         700
medium_residential      700
tennis_court            700
basketball_court        700
river                   700
```

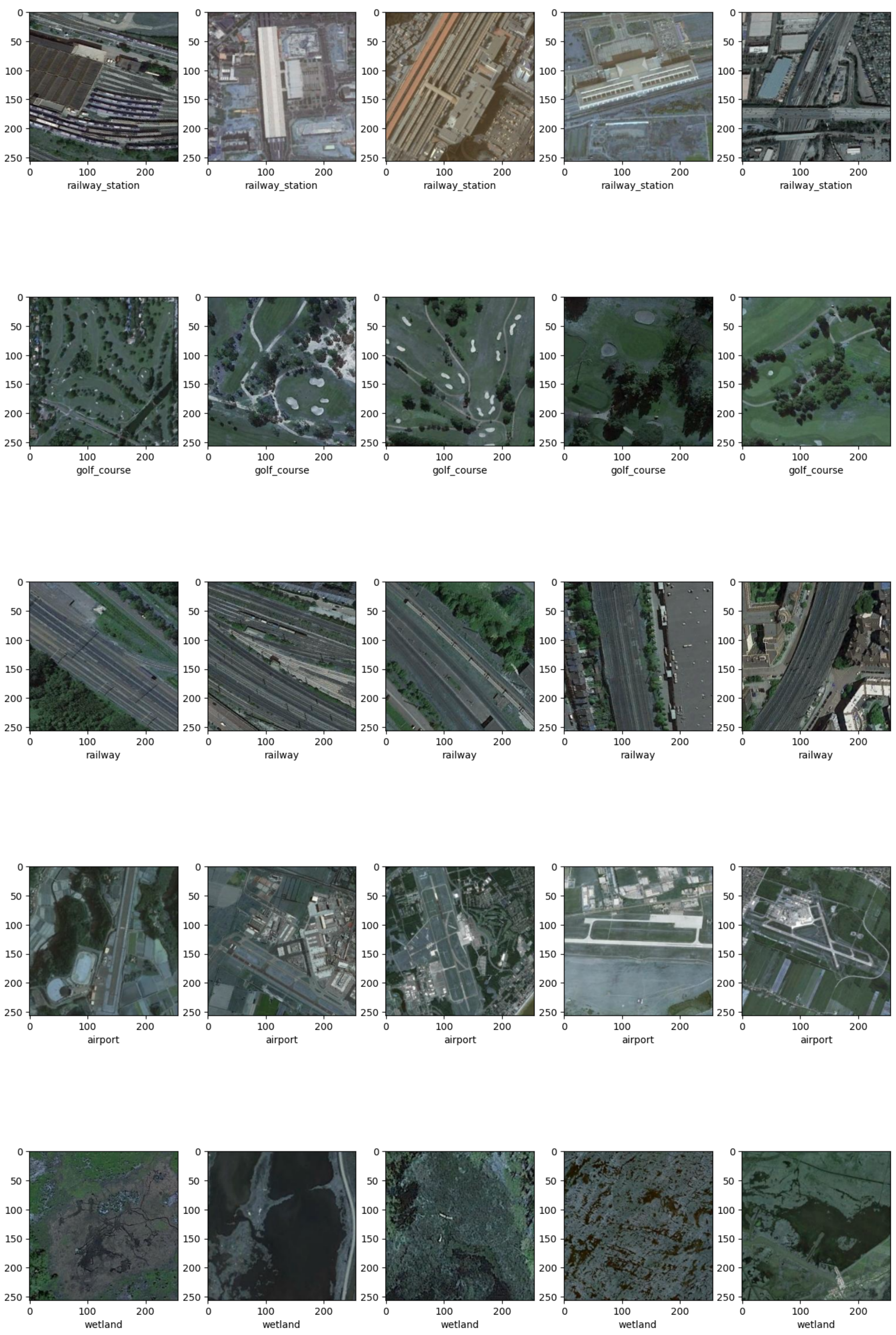
```
storage_tank             700
harbor                   700
terrace                  700
thermal_power_station    700
golf_course              700
runway                   700
roundabout               700
bridge                   700
industrial_area          700
baseball_diamond         700
mobile_home_park         700
overpass                 700
church                   700
chaparral                700
railway                  700
stadium                  700
ground_track_field       700
Name: count, dtype: int64
```

Conclusions: - Each category of the 45 categories has 700 samples

2.1 Sample subset of images from dataset

- show a subset of the images
- select some random labels (max 5 labels)
- show 5 images for each label to show variance
- contains varying spatial resolution ranging from 20cm to more than 30m/px.

[ ]:
```python
random_labels = random.sample(LABELS, min(5, len(LABELS)))
for label in random_labels:
    airplane_dataset = data[data['label'] == label].imgpath
    img_airplane = airplane_dataset.tolist()
    p1 = []
    for img in img_airplane[:5]:
        x = cv2.imread(img)
        p1.append(x)

    plt.figure(figsize=(16,10))
    for i in range(1,6):
        plt.subplot(2, 5, i)
        plt.grid(False)
        plt.imshow(p1[i-1])
        plt.xlabel(label)
    plt.show()
```


railway_station
railway_station
railway_station
railway_station
railway_station
golf_course
golf_course
golf_course
golf_course
golf_course
railway
railway
railway
railway
railway
airport
airport
airport
airport
airport
wetland
wetland
wetland
wetland
wetland

Conclusion - Images have a resolution of 256 by 256 and 3 channels (RGB)

2.2 Partition dataset into folds

- add 5-fold for each sample (1,2,3,4,5): 1st 20%, 2nd 20%, etc.
- save current dataset on disk

```
[ ]: # - add label index for training
label_index = []
for lab in data.label:
    label_index.append(LABELS.index(lab))
data['label_index'] = label_index
```

```
[ ]: def parition_dataset(data, folds=5):
    # partition the dataset into folds
    parition_size = int(100 / folds)
    data = data.sample(frac=1).reset_index(drop=True)  # resample dataset␣
 ↪randomly.
    folds = []
    n = len(data['label_index'])
    fold = 0
    for i in range(n):
        if ((i / n) * 100) % parition_size == 0:  # folds are %20, for 80%␣
 ↪train and 20% val in 5-fold;
            fold += 1
        folds.append(fold)
    data['fold'] = folds
    data.sort_values(by=['label_index'], inplace=True)  # sort values
    return data
```

```
[ ]: data = parition_dataset(data, folds=K_FOLDS)
```

```
[ ]: data.to_csv("dataset_resisc45.csv", index=False)  # save dataset on disk
```

2.3 Data load

```
[ ]: df = pd.read_csv("dataset_resisc45.csv")
```

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31500 entries, 0 to 31499
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   imgpath      31500 non-null  object
 1   label        31500 non-null  object
 2   label_index  31500 non-null  int64
 3   fold         31500 non-null  int64
dtypes: int64(2), object(2)
memory usage: 984.5+ KB
```

[ ]:
```
df["fold"].value_counts()
```

[ ]:
```
fold
3    6300
1    6300
4    6300
2    6300
5    6300
Name: count, dtype: int64
```

- 6300 samples in each fold.

2.4 Train dataset loader

[ ]:
```
# Dataset loader
import enum

class DatasetTypes(enum.Enum):
    train = "train"
    val = "val"
    test = "test"


RES_X = 224
RES_Y = 224

class DatasetResisc45(Dataset):
    def __init__(self, dataset_file, dataset_type=None, val_fold=None,␣
 ↪shuffle=False, transform=None, target_transform=None):
        df = pd.read_csv(dataset_file)
        if shuffle:
            df = df.sample(frac=1).reset_index(drop=True)
        folds = list(df["fold"].unique())  # get all folds
        if val_fold:
            if val_fold not in folds:
                raise Exception("Fold not found.")
            if dataset_type == DatasetTypes.train:
                df.drop(index=df[df["fold"] == val_fold].index, inplace=True) ␣
 ↪# drop the validation fold
            elif dataset_type == DatasetTypes.val:
                df = df[df["fold"] == val_fold]  # keep only the validation␣
↪fold in dataset
        self.img_labels = df
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = self.img_labels.iloc[idx, 0]
        image = cv2.imread(img_path)
        label_index = self.img_labels.iloc[idx, 2]
        label = np.zeros(len(LABELS), dtype=np.float32)
        label[label_index] = 1.0
        if self.transform:
            image = Image.open(img_path)
            image = self.transform(image)
        else:
            image = cv2.resize(image, dsize=(RES_Y, RES_X))
            image = image - 127.5
            image = np.moveaxis(image, -1, 0).astype(np.float32) / 255.0  #␣
↪move channels first (3 x RES_Y x RES_X)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

```
[ ]: data = DatasetResisc45("dataset_resisc45.csv")
```

```
[ ]: data[0][0].std(), data[0][0].mean(), data[0][0].max(), data[0][0].min()
```

```
[ ]: (0.07102675, -0.2668981, 0.17843138, -0.5)
```

```
[ ]: data[0][0]
```

```
[ ]: array([[[-0.34705883, -0.35882354, -0.31176472, …, -0.327451  ,
          -0.30784315, -0.38627452],
         [-0.28431374, -0.31960785, -0.29607844, …, -0.34313726,
          -0.327451  , -0.36666667],
         [-0.2882353 , -0.30784315, -0.28039217, …, -0.33137256,
          -0.3156863 , -0.31176472],
         …,
         [-0.31960785, -0.28431374, -0.32352942, …, -0.19019608,
          -0.2372549 , -0.2647059 ],
         [-0.42156863, -0.44117647, -0.41764706, …, -0.2372549 ,
          -0.3156863 , -0.35490197],
         [-0.45686275, -0.49607843, -0.4372549 , …, -0.34313726,
         -0.37843138, -0.3392157 ]],

        [[-0.2882353 , -0.3       , -0.25686276, …, -0.27254903,
          -0.2529412 , -0.33137256],
         [-0.2254902 , -0.25686276, -0.2372549 , …, -0.2882353 ,
          -0.27254903, -0.31176472],
         [-0.22941177, -0.24901961, -0.22156863, …, -0.2764706 ,
          -0.26078433, -0.25686276],
         …,
         [-0.29215688, -0.24901961, -0.27254903, …, -0.15490197,
          -0.20980392, -0.24117647],
         [-0.39411765, -0.39803922, -0.36666667, …, -0.20196079,
          -0.2882353 , -0.33137256],
         [-0.42941177, -0.46470588, -0.38235295, …, -0.30784315,
          -0.3509804 , -0.3156863 ]],

        [[-0.2764706 , -0.2882353 , -0.24117647, …, -0.28039217,
          -0.2647059 , -0.33529413],
         [-0.21372549, -0.24901961, -0.2254902 , …, -0.29607844,
          -0.28039217, -0.31960785],
         [-0.21764706, -0.2372549 , -0.20980392, …, -0.28431374,
          -0.2647059 , -0.26078433],
         …,
         [-0.30392158, -0.26078433, -0.28431374, …, -0.1392157 ,
          -0.19411765, -0.22156863],
         [-0.4137255 , -0.42156863, -0.38627452, …, -0.18627451,
          -0.27254903, -0.31176472],
         [-0.4490196 , -0.4882353 , -0.40588236, …, -0.29215688,
          -0.33137256, -0.29607844]]], dtype=float32)
```

3 Classification problem

- 5-fold cross validation
- one-vs-all
- softmax

[ ]:
```python
# find CUDA / MPS / CPU device
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

```
Using mps device
```

3.1 Neural Networks Algorithms

[ ]:
```
from torch import nn

# Define model architecture
# need to implement dimensions checking!

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(RES_Y * RES_X * 3, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, len(LABELS))
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits


class SatAlexNet(nn.Module):
    def __init__(self, num_channels=3):
        super(SatAlexNet, self).__init__()

        # add 1x1 conv with stride 1
        # add 3x3 conv with stride 2 in first layer

        # 1. initialize first set of CONV => RELU => POOL layers
        self.conv1 = nn.Conv2d(in_channels=num_channels, out_channels=50,␣
 ↪kernel_size=(7, 7), stride=2)
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2))

        # 2. initialize second set of CONV => RELU => POOL layers
        self.conv2 = nn.Conv2d(in_channels=50, out_channels=100,␣
 ↪kernel_size=(3, 3))
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2))

        # 3. initialize second set of CONV => RELU => POOL layers
        self.conv3 = nn.Conv2d(in_channels=100, out_channels=200,␣
↪kernel_size=(3, 3))
        self.relu3 = nn.ReLU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2))

        # 4. initialize second set of CONV => RELU => POOL layers
        self.conv4 = nn.Conv2d(in_channels=200, out_channels=200,␣
↪kernel_size=(3, 3))
        self.relu4 = nn.ReLU()

        # 5. initialize second set of CONV => RELU => POOL layers
        self.conv5 = nn.Conv2d(in_channels=200, out_channels=100,␣
↪kernel_size=(3, 3))
        self.relu5 = nn.ReLU()

        self.flatten = nn.Flatten()

        # initialize first (and only) set of FC => RELU layers
        self.fc1 = nn.Linear(in_features=8100, out_features=1024)
        self.relu6 = nn.ReLU()
        self.fc2 = nn.Linear(in_features=1024, out_features=len(LABELS))

    def forward(self, x):
        # pass the input through our first set of CONV => RELU =>
        # POOL layers
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.maxpool1(x)
        # pass the output from the previous layer through the second
        # set of CONV => RELU => POOL layers
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.maxpool2(x)
        # pass the output from the previous layer through the second
        # set of CONV => RELU => POOL layers
        x = self.conv3(x)
        x = self.relu3(x)
        x = self.maxpool3(x)
        # pass the output from the previous layer through the second
        # set of CONV => RELU => POOL layers
        x = self.conv4(x)
        x = self.relu4(x)
        # pass the output from the previous layer through the second
        # set of CONV => RELU => POOL layers
        x = self.conv5(x)
        x = self.relu5(x)
        # flatten the output from the previous layer and pass it
        # through our only set of FC => RELU layers
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu6(x)
        x = self.fc2(x)
        return x
```

[ ]:
```
# create model
def create_model_on_device(model_class, *args, **kwargs):
    print(f"device: {device}")
    model = model_class(*args, **kwargs).to(device) # create model on device
    print(str(model))
    return model
```

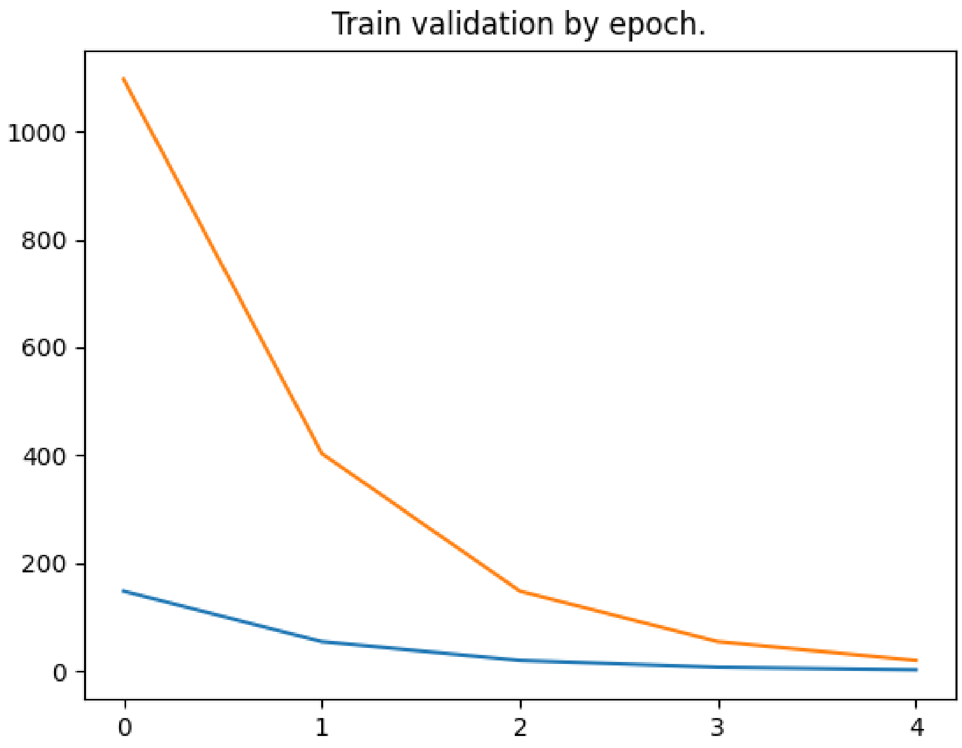
[ ]:
```
demo_model = create_model_on_device(SatAlexNet, num_channels=3)
```

```
device: mps
SatAlexNet(
  (conv1): Conv2d(3, 50, kernel_size=(7, 7), stride=(2, 2))
  (relu1): ReLU()
  (maxpool1): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv2): Conv2d(50, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu2): ReLU()
  (maxpool2): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv3): Conv2d(100, 200, kernel_size=(3, 3), stride=(1, 1))
  (relu3): ReLU()
  (maxpool3): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv4): Conv2d(200, 200, kernel_size=(3, 3), stride=(1, 1))
  (relu4): ReLU()
  (conv5): Conv2d(200, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu5): ReLU()
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=8100, out_features=1024, bias=True)
  (relu6): ReLU()
  (fc2): Linear(in_features=1024, out_features=45, bias=True)
)
```

[ ]:
```
loss_fn = nn.CrossEntropyLoss()

def train_one_epoch(optimizer, dataloader, model):
    num_batches = len(dataloader)

    print("Started train.")
    size = len(dataloader.dataset)
    model.train()  # set model to train mode

    train_loss = 0
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

    train_loss /= num_batches
    return train_loss

def val_one_epoch(dataloader, model):
    print("Started validation.")
    size = len(dataloader.dataset)
    num_batches = len(dataloader)

    model.eval() # set model to evaluation mode

    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            error = (nn.Softmax(dim=1)(pred).argmax(1) == y.argmax(1)).
 ↪type(torch.float)
            correct += error.sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
 ↪{test_loss:>8f} \n")
    return test_loss
```

```
[ ]: %matplotlib ipympl

     def plot_validation_epochs(x, y, y2):
         fig, ax = plt.subplots()
         ax.plot(x, y)
         ax.plot(x, y2)
         #specify axis tick step sizes
         _ = plt.xticks(np.arange(min(x), max(x)+1, 1))
         # _ = plt.yticks(np.arange(0, max(y)+0.1, 0.1))

         ax.set_title("Train validation by epoch.")
         plt.show()
```

```
[ ]: plot_validation_epochs(np.arange(0, 5, 1), np.exp([5, 4, 3, 2, 1]), np.exp([7,␣
       ↪6, 5, 4, 3]))
```



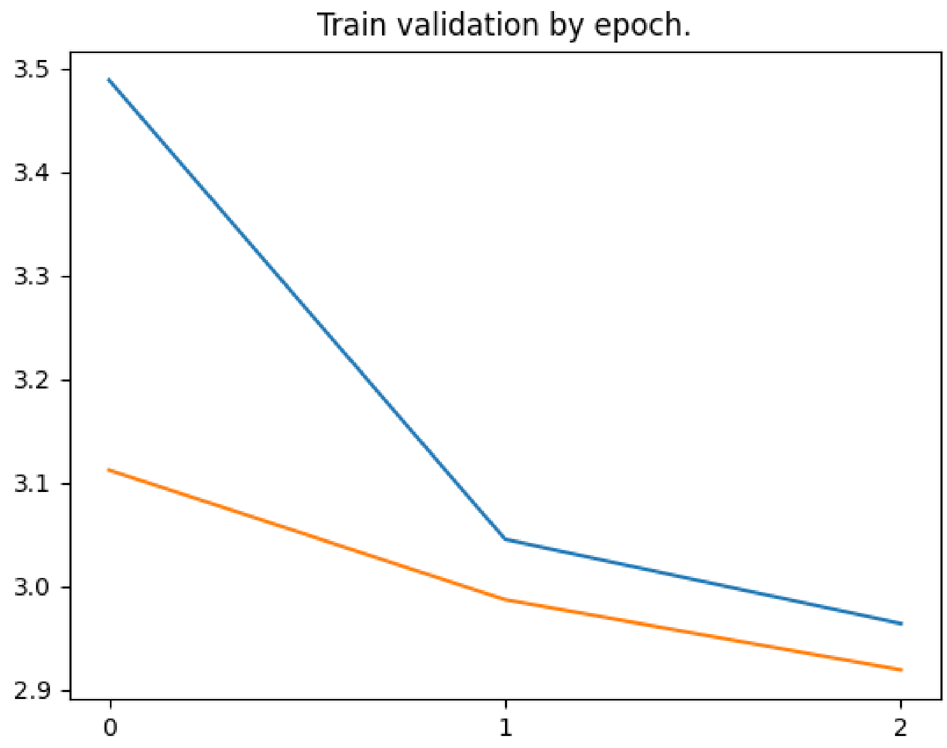
3.2 5 Fold train validation

- trebuie sa reducem timpii de antrenament

```
[ ]: def kfold_train(folds=K_FOLDS):
         epochs = 30
         batch_size = 64

         for val_fold in range(1, folds+1):
             print(f"Fold: {val_fold}\n-------------------------------")
             model = create_model_on_device(SatAlexNet)
             train_set = DatasetResisc45("dataset_resisc45.csv",␣
     ↪dataset_type=DatasetTypes.train, val_fold=val_fold)
             val_set = DatasetResisc45("dataset_resisc45.csv",␣
     ↪dataset_type=DatasetTypes.val, val_fold=val_fold)
             train_dataloader = DataLoader(train_set, batch_size=batch_size,␣
     ↪shuffle=True)
             val_dataloader = DataLoader(val_set, batch_size=batch_size,␣
     ↪shuffle=False)

             optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

             train_losses = []
             val_losses = []
             for t in range(epochs):
                 print(f"Epoch {t+1}\n-------------------------------")
                 train_loss = train_one_epoch(optimizer, train_dataloader, model)
                 val_loss = val_one_epoch(val_dataloader, model)

                 train_losses.append(train_loss)
                 val_losses.append(val_loss)

             plot_validation_epochs([i for i in range(epochs)], train_losses,␣
     ↪val_losses)
```

```
[ ]: kfold_train(folds=1)
```

```
Fold: 1
-------------------------------
device: mps
SatAlexNet(
  (conv1): Conv2d(3, 50, kernel_size=(7, 7), stride=(2, 2))
  (relu1): ReLU()
  (maxpool1): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv2): Conv2d(50, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu2): ReLU()
  (maxpool2): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv3): Conv2d(100, 200, kernel_size=(3, 3), stride=(1, 1))
  (relu3): ReLU()
  (maxpool3): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv4): Conv2d(200, 200, kernel_size=(3, 3), stride=(1, 1))
  (relu4): ReLU()
  (conv5): Conv2d(200, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu5): ReLU()
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=8100, out_features=1024, bias=True)
  (relu6): ReLU()
  (fc2): Linear(in_features=1024, out_features=45, bias=True)
)
Epoch 1
-------------------------------
Started train.
loss: 3.807703  [   64/25200]
loss: 3.781118  [ 6464/25200]
loss: 3.362244  [12864/25200]
loss: 3.196585  [19264/25200]
Started validation.
Test Error:
 Accuracy: 14.2%, Avg loss: 3.112317

Epoch 2
-------------------------------
Started train.
loss: 2.999893  [   64/25200]
loss: 3.412662  [ 6464/25200]
loss: 3.162524  [12864/25200]
loss: 3.148863  [19264/25200]
Started validation.
Test Error:
 Accuracy: 17.8%, Avg loss: 2.987232

Epoch 3
-------------------------------
Started train.
loss: 2.794767  [   64/25200]
loss: 3.020819  [ 6464/25200]
loss: 3.249866  [12864/25200]
loss: 3.042657  [19264/25200]
Started validation.
Test Error:
 Accuracy: 18.9%, Avg loss: 2.919572
```



3.3 Pretrained finetuning

[ ]:
```
model_name = "resnet"
num_classes = len(LABELS)
# Batch size for training (change depending on how much memory you have)
batch_size = 8

# Number of epochs to train for
num_epochs = 15

# Flag for feature extracting. When False, we finetune the whole model,
#   when True we only update the reshaped layer params
feature_extract = True


def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

```
def initialize_model(model_name, num_classes, feature_extract,␣
↪use_pretrained=True):
    # Initialize these variables which will be set in this if statement. Each␣
↪of these
    #   variables is model specific.
    model_ft = None
    input_size = 0

    if model_name == "resnet":
        """ Resnet18
        """
        model_ft = models.resnet18(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)
        input_size = 224

    elif model_name == "alexnet":
        """ Alexnet
        """
        model_ft = models.alexnet(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.classifier[6].in_features
        model_ft.classifier[6] = nn.Linear(num_ftrs,num_classes)
        input_size = 224

    elif model_name == "vgg":
        """ VGG11_bn
        """
        model_ft = models.vgg11_bn(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.classifier[6].in_features
        model_ft.classifier[6] = nn.Linear(num_ftrs,num_classes)
        input_size = 224

    elif model_name == "squeezenet":
        """ Squeezenet
        """
        model_ft = models.squeezenet1_0(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        model_ft.classifier[1] = nn.Conv2d(512, num_classes, kernel_size=(1,1),␣
↪stride=(1,1))
        model_ft.num_classes = num_classes
        input_size = 224

    elif model_name == "densenet":
        """ Densenet
        """
        model_ft = models.densenet121(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.classifier.in_features
        model_ft.classifier = nn.Linear(num_ftrs, num_classes)
        input_size = 224

    elif model_name == "inception":
        """ Inception v3
        Be careful, expects (299,299) sized images and has auxiliary output
        """
        model_ft = models.inception_v3(pretrained=use_pretrained)
        set_parameter_requires_grad(model_ft, feature_extract)
        # Handle the auxilary net
        num_ftrs = model_ft.AuxLogits.fc.in_features
        model_ft.AuxLogits.fc = nn.Linear(num_ftrs, num_classes)
        # Handle the primary net
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs,num_classes)
        input_size = 299

    else:
        print("Invalid model name, exiting...")
        exit()

    return model_ft, input_size

# Initialize the model for this run
model_ft, input_size = initialize_model(model_name, num_classes,␣
 ↪feature_extract, use_pretrained=True)

# Print the model we just instantiated
print(model_ft)

# Send the model to GPU
model_ft = model_ft.to(device)

# Gather the parameters to be optimized/updated in this run. If we are
#  finetuning we will be updating all parameters. However, if we are
#  doing feature extract method, we will only update the parameters
#  that we have just initialized, i.e. the parameters with requires_grad
#  is True.
params_to_update = model_ft.parameters()
print("Params to learn:")
if feature_extract:
    params_to_update = []
```

```python
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            params_to_update.append(param)
            print("\t",name)
else:
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            print("\t",name)

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
```

```
/Users/cristianion/Desktop/satimg_model/.venv/lib/python3.11/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/Users/cristianion/Desktop/satimg_model/.venv/lib/python3.11/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
```

```
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=45, bias=True)
)
Params to learn:
         fc.weight
         fc.bias
```

[ ]:
```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(input_size),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(input_size),
        transforms.CenterCrop(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

[ ]:
```
train_set = DatasetResisc45("dataset_resisc45.csv",␣
 ↪transform=data_transforms['train'], dataset_type=DatasetTypes.train,␣
 ↪val_fold=5)
val_set = DatasetResisc45("dataset_resisc45.csv",␣
 ↪transform=data_transforms['val'], dataset_type=DatasetTypes.val, val_fold=5)
train_dataloader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_set, batch_size=batch_size, shuffle=False)
```

[ ]:
```
train_set[0]
```

[ ]:
```
(tensor([[[-0.9534, -1.1418, -1.2617,  …, -1.4329, -1.5870, -1.6384],
          [-1.1247, -1.1932, -1.2617,  …, -1.1760, -1.4158, -1.6213],
          [-1.3815, -1.5014, -1.6042,  …, -1.0390, -1.1932, -1.3644],
          …,
          [-1.1932, -1.1247, -1.0219,  …, -1.1932, -1.3130, -1.4158],
          [-1.2788, -1.2103, -1.1075,  …, -0.7479, -0.7479, -0.7650],
          [-1.2274, -1.1932, -1.1589,  …, -0.7479, -0.5767, -0.4397]],

         [[-0.9503, -1.1429, -1.2829,  …, -1.3880, -1.5455, -1.5980],
          [-1.1779, -1.2479, -1.3179,  …, -1.1078, -1.3529, -1.5630],
          [-1.4580, -1.5630, -1.6856,  …, -0.9678, -1.1253, -1.3004],
          …,
          [-1.2304, -1.1604, -1.0553,  …, -1.1429, -1.2654, -1.3704],
          [-1.3179, -1.2479, -1.1429,  …, -0.6877, -0.6877, -0.7052],
          [-1.2654, -1.2304, -1.1954,  …, -0.6877, -0.5126, -0.3725]],
         [[-0.9330, -1.1247, -1.2641,  …, -1.4210, -1.5779, -1.6302],
          [-1.1073, -1.1770, -1.2467,  …, -1.1073, -1.3513, -1.5604],
          [-1.3861, -1.4907, -1.6127,  …, -0.9678, -1.1247, -1.2816],
          …,
          [-1.2293, -1.1596, -1.0376,  …, -1.2119, -1.3339, -1.4384],
          [-1.3164, -1.2467, -1.1421,  …, -0.7587, -0.7587, -0.7761],
          [-1.2641, -1.2293, -1.1770,  …, -0.7587, -0.5844, -0.4450]]]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32))
```

[ ]:
```
criterion = nn.CrossEntropyLoss()

# Train and evaluate
train_losses = []
val_losses = []
for t in range(num_epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loss = train_one_epoch(optimizer_ft, train_dataloader, model_ft)
    val_loss = val_one_epoch(val_dataloader, model_ft)

    train_losses.append(train_loss)
    val_losses.append(val_loss)

plot_validation_epochs([i for i in range(num_epochs)], train_losses, val_losses)
```

```
Epoch 1
-------------------------------
Started train.
loss: 4.235198  [    8/25200]
loss: 3.573315  [  808/25200]
loss: 3.178993  [ 1608/25200]
loss: 2.767187  [ 2408/25200]
loss: 2.985579  [ 3208/25200]
loss: 2.483799  [ 4008/25200]
loss: 2.240854  [ 4808/25200]
loss: 2.422409  [ 5608/25200]
loss: 1.886932  [ 6408/25200]
loss: 1.748387  [ 7208/25200]
loss: 1.569471  [ 8008/25200]
loss: 2.523410  [ 8808/25200]
loss: 1.960317  [ 9608/25200]
loss: 1.490180  [10408/25200]
loss: 2.055791  [11208/25200]
loss: 1.412982  [12008/25200]
loss: 1.186982  [12808/25200]
```

```
loss: 0.925945  [13608/25200]
loss: 1.732041  [14408/25200]
loss: 1.796088  [15208/25200]
loss: 1.703750  [16008/25200]
loss: 2.029390  [16808/25200]
loss: 2.298863  [17608/25200]
loss: 1.821632  [18408/25200]
loss: 1.828520  [19208/25200]
loss: 1.127306  [20008/25200]
loss: 0.953950  [20808/25200]
loss: 1.641013  [21608/25200]
loss: 1.291857  [22408/25200]
loss: 1.453563  [23208/25200]
loss: 1.777547  [24008/25200]
loss: 1.577447  [24808/25200]
Started validation.
Test Error:
 Accuracy: 70.4%, Avg loss: 0.987157

Epoch 2
-------------------------------
Started train.
loss: 1.745728  [    8/25200]
loss: 1.677700  [  808/25200]
loss: 1.202552  [ 1608/25200]
loss: 0.831638  [ 2408/25200]
loss: 1.574594  [ 3208/25200]
loss: 0.721324  [ 4008/25200]
loss: 1.747147  [ 4808/25200]
loss: 1.559911  [ 5608/25200]
loss: 1.058604  [ 6408/25200]
loss: 1.170159  [ 7208/25200]
loss: 1.131893  [ 8008/25200]
loss: 1.035291  [ 8808/25200]
loss: 1.515062  [ 9608/25200]
loss: 1.919231  [10408/25200]
loss: 1.286450  [11208/25200]
loss: 0.941319  [12008/25200]
loss: 1.610492  [12808/25200]
loss: 1.607287  [13608/25200]
loss: 1.709679  [14408/25200]
loss: 1.232734  [15208/25200]
loss: 0.548116  [16008/25200]
loss: 3.305570  [16808/25200]
loss: 1.219727  [17608/25200]
loss: 2.099289  [18408/25200]
loss: 0.928743  [19208/25200]
loss: 1.417175  [20008/25200]
```

```
loss: 0.673327  [20808/25200]
loss: 0.274872  [21608/25200]
loss: 1.368964  [22408/25200]
loss: 1.019431  [23208/25200]
loss: 0.863374  [24008/25200]
loss: 0.600804  [24808/25200]
Started validation.
Test Error:
 Accuracy: 73.2%, Avg loss: 0.879267

Epoch 3
-------------------------------
Started train.
loss: 1.525239  [    8/25200]
loss: 1.647601  [  808/25200]
loss: 0.637435  [ 1608/25200]
loss: 0.640737  [ 2408/25200]
loss: 1.097915  [ 3208/25200]
loss: 0.654394  [ 4008/25200]
loss: 0.880570  [ 4808/25200]
loss: 1.207196  [ 5608/25200]
loss: 0.844202  [ 6408/25200]
loss: 1.375883  [ 7208/25200]
loss: 1.863786  [ 8008/25200]
loss: 0.597471  [ 8808/25200]
loss: 0.542997  [ 9608/25200]
loss: 1.298434  [10408/25200]
loss: 1.155591  [11208/25200]
loss: 0.528767  [12008/25200]
loss: 1.408122  [12808/25200]
loss: 1.205272  [13608/25200]
loss: 1.315357  [14408/25200]
loss: 1.826369  [15208/25200]
loss: 0.843555  [16008/25200]
loss: 1.359651  [16808/25200]
loss: 0.740920  [17608/25200]
loss: 0.877828  [18408/25200]
loss: 1.815480  [19208/25200]
loss: 1.441536  [20008/25200]
loss: 0.847763  [20808/25200]
loss: 0.529159  [21608/25200]
loss: 0.837154  [22408/25200]
loss: 1.623312  [23208/25200]
loss: 1.604900  [24008/25200]
loss: 0.851313  [24808/25200]
Started validation.
Test Error:
 Accuracy: 74.2%, Avg loss: 0.849084
```

```
Epoch 4
-------------------------------
Started train.
loss: 0.954241  [    8/25200]
loss: 1.071821  [  808/25200]
loss: 1.058058  [ 1608/25200]
loss: 0.907764  [ 2408/25200]
loss: 1.219106  [ 3208/25200]
loss: 0.809039  [ 4008/25200]
loss: 0.612728  [ 4808/25200]
loss: 0.700763  [ 5608/25200]
loss: 0.755784  [ 6408/25200]
loss: 0.924409  [ 7208/25200]
loss: 0.396317  [ 8008/25200]
loss: 0.415597  [ 8808/25200]
loss: 1.215765  [ 9608/25200]
loss: 2.504439  [10408/25200]
loss: 0.979008  [11208/25200]
loss: 2.592680  [12008/25200]
loss: 1.287935  [12808/25200]
loss: 1.115323  [13608/25200]
loss: 0.817061  [14408/25200]
loss: 2.203200  [15208/25200]
loss: 0.704843  [16008/25200]
loss: 1.143892  [16808/25200]
loss: 0.975930  [17608/25200]
loss: 0.447087  [18408/25200]
loss: 0.297812  [19208/25200]
loss: 1.285548  [20008/25200]
loss: 0.621419  [20808/25200]
loss: 1.763034  [21608/25200]
loss: 1.737665  [22408/25200]
loss: 0.412952  [23208/25200]
loss: 1.438774  [24008/25200]
loss: 1.332282  [24808/25200]
Started validation.
Test Error:
 Accuracy: 75.1%, Avg loss: 0.811321

Epoch 5
-------------------------------
Started train.
loss: 0.772013  [    8/25200]
loss: 0.562190  [  808/25200]
loss: 3.291862  [ 1608/25200]
loss: 0.931256  [ 2408/25200]
loss: 0.557896  [ 3208/25200]
```

```
loss: 1.284098  [ 4008/25200]
loss: 2.556782  [ 4808/25200]
loss: 1.881062  [ 5608/25200]
loss: 0.513958  [ 6408/25200]
loss: 1.236616  [ 7208/25200]
loss: 1.381820  [ 8008/25200]
loss: 0.773038  [ 8808/25200]
loss: 0.991294  [ 9608/25200]
loss: 1.554606  [10408/25200]
loss: 0.500770  [11208/25200]
loss: 1.260070  [12008/25200]
loss: 1.474102  [12808/25200]
loss: 1.127592  [13608/25200]
loss: 1.279178  [14408/25200]
loss: 1.022575  [15208/25200]
loss: 1.009655  [16008/25200]
loss: 0.767781  [16808/25200]
loss: 1.032808  [17608/25200]
loss: 0.738952  [18408/25200]
loss: 1.753260  [19208/25200]
loss: 0.586503  [20008/25200]
loss: 1.531992  [20808/25200]
loss: 0.823398  [21608/25200]
loss: 0.667179  [22408/25200]
loss: 0.257363  [23208/25200]
loss: 1.263420  [24008/25200]
loss: 0.788282  [24808/25200]
Started validation.
Test Error:
 Accuracy: 75.4%, Avg loss: 0.806109

Epoch 6
-------------------------------
Started train.
loss: 2.579009  [    8/25200]
loss: 1.295544  [  808/25200]
loss: 2.107308  [ 1608/25200]
loss: 1.159207  [ 2408/25200]
loss: 1.225102  [ 3208/25200]
loss: 1.357038  [ 4008/25200]
loss: 0.293837  [ 4808/25200]
loss: 1.006881  [ 5608/25200]
loss: 0.962190  [ 6408/25200]
loss: 0.226574  [ 7208/25200]
loss: 0.263327  [ 8008/25200]
loss: 1.387282  [ 8808/25200]
loss: 1.994148  [ 9608/25200]
loss: 0.808767  [10408/25200]
```

```
loss: 0.427519  [11208/25200]
loss: 1.245056  [12008/25200]
loss: 2.378685  [12808/25200]
loss: 2.081524  [13608/25200]
loss: 1.559983  [14408/25200]
loss: 0.691395  [15208/25200]
loss: 2.449813  [16008/25200]
loss: 1.018371  [16808/25200]
loss: 0.887432  [17608/25200]
loss: 1.104313  [18408/25200]
loss: 1.076100  [19208/25200]
loss: 1.190982  [20008/25200]
loss: 0.971923  [20808/25200]
loss: 0.765439  [21608/25200]
loss: 1.491436  [22408/25200]
loss: 1.920276  [23208/25200]
loss: 0.634845  [24008/25200]
loss: 0.314191  [24808/25200]
Started validation.
Test Error:
 Accuracy: 76.0%, Avg loss: 0.788206

Epoch 7
-------------------------------
Started train.
loss: 1.429339  [    8/25200]
loss: 3.175894  [  808/25200]
loss: 1.146268  [ 1608/25200]
loss: 1.724091  [ 2408/25200]
loss: 1.434451  [ 3208/25200]
loss: 0.353380  [ 4008/25200]
loss: 0.763157  [ 4808/25200]
loss: 1.316129  [ 5608/25200]
loss: 1.751049  [ 6408/25200]
loss: 0.936889  [ 7208/25200]
loss: 0.414818  [ 8008/25200]
loss: 1.987554  [ 8808/25200]
loss: 1.869974  [ 9608/25200]
loss: 2.235788  [10408/25200]
loss: 1.068395  [11208/25200]
loss: 1.241704  [12008/25200]
loss: 1.804383  [12808/25200]
loss: 1.209724  [13608/25200]
loss: 0.860779  [14408/25200]
loss: 2.191154  [15208/25200]
loss: 1.630217  [16008/25200]
loss: 1.097403  [16808/25200]
loss: 0.911218  [17608/25200]
```

```
loss: 0.284380  [18408/25200]
loss: 0.918811  [19208/25200]
loss: 1.171835  [20008/25200]
loss: 1.386243  [20808/25200]
loss: 0.884975  [21608/25200]
loss: 0.483948  [22408/25200]
loss: 0.943258  [23208/25200]
loss: 1.091119  [24008/25200]
loss: 0.795000  [24808/25200]
Started validation.
Test Error:
 Accuracy: 76.7%, Avg loss: 0.768637

Epoch 8
-------------------------------
Started train.
loss: 0.851881  [    8/25200]
loss: 0.867061  [  808/25200]
loss: 1.470282  [ 1608/25200]
loss: 0.541640  [ 2408/25200]
loss: 1.747303  [ 3208/25200]
loss: 1.609906  [ 4008/25200]
loss: 1.416789  [ 4808/25200]
loss: 0.421959  [ 5608/25200]
loss: 0.697495  [ 6408/25200]
loss: 1.259277  [ 7208/25200]
loss: 1.058897  [ 8008/25200]
loss: 0.698516  [ 8808/25200]
loss: 1.918081  [ 9608/25200]
loss: 2.057371  [10408/25200]
loss: 0.705721  [11208/25200]
loss: 1.414919  [12008/25200]
loss: 0.996540  [12808/25200]
loss: 0.690201  [13608/25200]
loss: 0.281828  [14408/25200]
loss: 1.021650  [15208/25200]
loss: 2.123637  [16008/25200]
loss: 1.017958  [16808/25200]
loss: 1.361350  [17608/25200]
loss: 0.705806  [18408/25200]
loss: 2.073182  [19208/25200]
loss: 0.963266  [20008/25200]
loss: 1.947176  [20808/25200]
loss: 0.532775  [21608/25200]
loss: 1.070579  [22408/25200]
loss: 0.751444  [23208/25200]
loss: 1.616042  [24008/25200]
loss: 1.598641  [24808/25200]
```

```
Started validation.
Test Error:
 Accuracy: 77.6%, Avg loss: 0.741684

Epoch 9
-------------------------------
Started train.
loss: 2.424690  [    8/25200]
loss: 0.892867  [  808/25200]
loss: 1.204110  [ 1608/25200]
loss: 1.256358  [ 2408/25200]
loss: 0.280959  [ 3208/25200]
loss: 1.619173  [ 4008/25200]
loss: 0.715723  [ 4808/25200]
loss: 0.360773  [ 5608/25200]
loss: 0.811739  [ 6408/25200]
loss: 1.927245  [ 7208/25200]
loss: 1.164632  [ 8008/25200]
loss: 1.206754  [ 8808/25200]
loss: 0.465576  [ 9608/25200]
loss: 1.052724  [10408/25200]
loss: 1.114631  [11208/25200]
loss: 1.125302  [12008/25200]
loss: 0.223339  [12808/25200]
loss: 1.477065  [13608/25200]
loss: 0.798754  [14408/25200]
loss: 1.180329  [15208/25200]
loss: 2.352105  [16008/25200]
loss: 1.105843  [16808/25200]
loss: 1.174492  [17608/25200]
loss: 0.773589  [18408/25200]
loss: 0.617418  [19208/25200]
loss: 0.449244  [20008/25200]
loss: 0.419981  [20808/25200]
loss: 0.635731  [21608/25200]
loss: 1.896612  [22408/25200]
loss: 0.340485  [23208/25200]
loss: 0.591762  [24008/25200]
loss: 0.494604  [24808/25200]
Started validation.
Test Error:
 Accuracy: 75.9%, Avg loss: 0.793421

Epoch 10
-------------------------------
Started train.
loss: 0.743021  [    8/25200]
loss: 0.313890  [  808/25200]
```

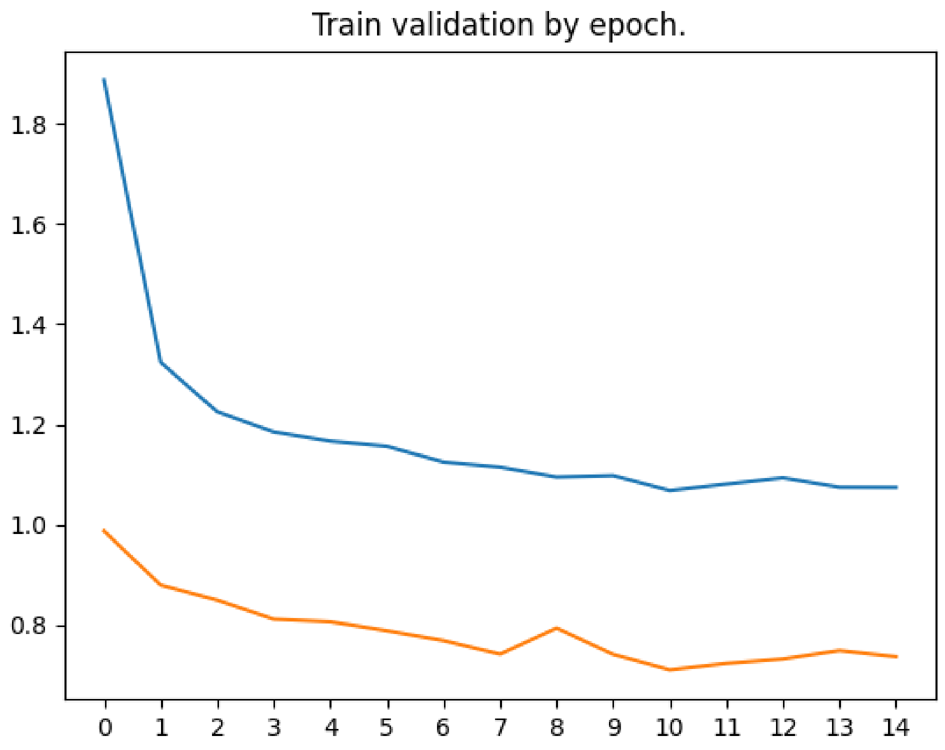
```
loss: 1.311780  [ 1608/25200]
loss: 1.073589  [ 2408/25200]
loss: 0.494511  [ 3208/25200]
loss: 1.796522  [ 4008/25200]
loss: 2.081824  [ 4808/25200]
loss: 1.031786  [ 5608/25200]
loss: 0.178365  [ 6408/25200]
loss: 1.248342  [ 7208/25200]
loss: 0.785931  [ 8008/25200]
loss: 1.505507  [ 8808/25200]
loss: 0.917121  [ 9608/25200]
loss: 1.137240  [10408/25200]
loss: 1.427951  [11208/25200]
loss: 0.918296  [12008/25200]
loss: 0.338738  [12808/25200]
loss: 1.570283  [13608/25200]
loss: 1.524523  [14408/25200]
loss: 0.852725  [15208/25200]
loss: 0.674294  [16008/25200]
loss: 1.467068  [16808/25200]
loss: 0.538925  [17608/25200]
loss: 0.756391  [18408/25200]
loss: 1.837482  [19208/25200]
loss: 1.110972  [20008/25200]
loss: 2.117155  [20808/25200]
loss: 1.629425  [21608/25200]
loss: 1.068895  [22408/25200]
loss: 1.352091  [23208/25200]
loss: 0.905642  [24008/25200]
loss: 1.048516  [24808/25200]
Started validation.
Test Error:
 Accuracy: 77.4%, Avg loss: 0.741207

Epoch 11
-------------------------------
Started train.
loss: 1.294993  [    8/25200]
loss: 0.640666  [  808/25200]
loss: 0.925078  [ 1608/25200]
loss: 0.867275  [ 2408/25200]
loss: 0.864298  [ 3208/25200]
loss: 1.269502  [ 4008/25200]
loss: 1.725043  [ 4808/25200]
loss: 0.488038  [ 5608/25200]
loss: 1.022462  [ 6408/25200]
loss: 1.736880  [ 7208/25200]
loss: 1.126996  [ 8008/25200]
```

```
loss: 1.324253  [ 8808/25200]
loss: 0.772619  [ 9608/25200]
loss: 0.976272  [10408/25200]
loss: 1.224224  [11208/25200]
loss: 1.510202  [12008/25200]
loss: 0.672113  [12808/25200]
loss: 1.215505  [13608/25200]
loss: 1.485055  [14408/25200]
loss: 0.687782  [15208/25200]
loss: 3.108798  [16008/25200]
loss: 1.313692  [16808/25200]
loss: 0.800166  [17608/25200]
loss: 0.625035  [18408/25200]
loss: 0.414350  [19208/25200]
loss: 1.415639  [20008/25200]
loss: 1.384153  [20808/25200]
loss: 0.887641  [21608/25200]
loss: 0.816809  [22408/25200]
loss: 0.263452  [23208/25200]
loss: 1.651235  [24008/25200]
loss: 0.538639  [24808/25200]
Started validation.
Test Error:
 Accuracy: 78.3%, Avg loss: 0.710174

Epoch 12
-------------------------------
Started train.
loss: 0.976293  [    8/25200]
loss: 1.503797  [  808/25200]
loss: 1.469231  [ 1608/25200]
loss: 0.370644  [ 2408/25200]
loss: 0.278530  [ 3208/25200]
loss: 1.293297  [ 4008/25200]
loss: 1.301512  [ 4808/25200]
loss: 2.207334  [ 5608/25200]
loss: 1.050458  [ 6408/25200]
loss: 0.608001  [ 7208/25200]
loss: 1.415084  [ 8008/25200]
loss: 1.587996  [ 8808/25200]
loss: 0.888445  [ 9608/25200]
loss: 1.028068  [10408/25200]
loss: 1.070373  [11208/25200]
loss: 1.324372  [12008/25200]
loss: 0.353027  [12808/25200]
loss: 1.127598  [13608/25200]
loss: 1.159891  [14408/25200]
loss: 1.234193  [15208/25200]
```

```
loss: 0.306367  [16008/25200]
loss: 1.309225  [16808/25200]
loss: 2.317075  [17608/25200]
loss: 0.955117  [18408/25200]
loss: 1.201392  [19208/25200]
loss: 1.478246  [20008/25200]
loss: 0.624154  [20808/25200]
loss: 0.778271  [21608/25200]
loss: 1.122135  [22408/25200]
loss: 0.552724  [23208/25200]
loss: 0.541644  [24008/25200]
loss: 0.712418  [24808/25200]
Started validation.
Test Error:
 Accuracy: 78.3%, Avg loss: 0.722925

Epoch 13
-------------------------------
Started train.
loss: 1.221581  [    8/25200]
loss: 1.833614  [  808/25200]
loss: 0.602057  [ 1608/25200]
loss: 2.000625  [ 2408/25200]
loss: 0.774749  [ 3208/25200]
loss: 1.361415  [ 4008/25200]
loss: 1.761897  [ 4808/25200]
loss: 1.126355  [ 5608/25200]
loss: 1.219370  [ 6408/25200]
loss: 3.963800  [ 7208/25200]
loss: 0.837898  [ 8008/25200]
loss: 1.775590  [ 8808/25200]
loss: 0.538849  [ 9608/25200]
loss: 1.444894  [10408/25200]
loss: 0.572245  [11208/25200]
loss: 0.845410  [12008/25200]
loss: 0.141636  [12808/25200]
loss: 1.220652  [13608/25200]
loss: 1.064067  [14408/25200]
loss: 0.562579  [15208/25200]
loss: 1.298440  [16008/25200]
loss: 1.182702  [16808/25200]
loss: 0.854738  [17608/25200]
loss: 0.721246  [18408/25200]
loss: 1.259927  [19208/25200]
loss: 0.971981  [20008/25200]
loss: 2.529023  [20808/25200]
loss: 0.238600  [21608/25200]
loss: 1.987256  [22408/25200]
```

```
loss: 1.367934  [23208/25200]
loss: 1.212104  [24008/25200]
loss: 0.601594  [24808/25200]
Started validation.
Test Error:
 Accuracy: 77.8%, Avg loss: 0.731903

Epoch 14
-------------------------------
Started train.
loss: 0.224807  [    8/25200]
loss: 1.337244  [  808/25200]
loss: 0.666663  [ 1608/25200]
loss: 0.837662  [ 2408/25200]
loss: 2.209371  [ 3208/25200]
loss: 1.666738  [ 4008/25200]
loss: 0.743037  [ 4808/25200]
loss: 1.632318  [ 5608/25200]
loss: 0.579718  [ 6408/25200]
loss: 1.089319  [ 7208/25200]
loss: 1.977990  [ 8008/25200]
loss: 0.578554  [ 8808/25200]
loss: 0.812236  [ 9608/25200]
loss: 0.175909  [10408/25200]
loss: 1.278362  [11208/25200]
loss: 1.089429  [12008/25200]
loss: 0.747847  [12808/25200]
loss: 0.633426  [13608/25200]
loss: 1.943150  [14408/25200]
loss: 0.863685  [15208/25200]
loss: 1.839129  [16008/25200]
loss: 1.630465  [16808/25200]
loss: 0.382859  [17608/25200]
loss: 0.623017  [18408/25200]
loss: 0.671669  [19208/25200]
loss: 0.784496  [20008/25200]
loss: 1.234680  [20808/25200]
loss: 1.799416  [21608/25200]
loss: 0.742027  [22408/25200]
loss: 0.428757  [23208/25200]
loss: 0.875678  [24008/25200]
loss: 1.272220  [24808/25200]
Started validation.
Test Error:
 Accuracy: 77.4%, Avg loss: 0.748688

Epoch 15
-------------------------------
```

```
Started train.
loss: 2.489374  [    8/25200]
loss: 1.825793  [  808/25200]
loss: 0.202243  [ 1608/25200]
loss: 2.334979  [ 2408/25200]
loss: 0.532373  [ 3208/25200]
loss: 1.237943  [ 4008/25200]
loss: 0.372692  [ 4808/25200]
loss: 0.364006  [ 5608/25200]
loss: 1.640793  [ 6408/25200]
loss: 1.683649  [ 7208/25200]
loss: 0.588175  [ 8008/25200]
loss: 0.837094  [ 8808/25200]
loss: 0.605451  [ 9608/25200]
loss: 0.862012  [10408/25200]
loss: 1.277416  [11208/25200]
loss: 0.540496  [12008/25200]
loss: 0.509171  [12808/25200]
loss: 0.886002  [13608/25200]
loss: 1.428193  [14408/25200]
loss: 0.341075  [15208/25200]
loss: 0.721711  [16008/25200]
loss: 0.864051  [16808/25200]
loss: 1.104101  [17608/25200]
loss: 0.608998  [18408/25200]
loss: 0.173288  [19208/25200]
loss: 1.031734  [20008/25200]
loss: 1.727535  [20808/25200]
loss: 1.554444  [21608/25200]
loss: 1.093528  [22408/25200]
loss: 0.981523  [23208/25200]
loss: 0.411151  [24008/25200]
loss: 1.155440  [24808/25200]
Started validation.
Test Error:
 Accuracy: 77.8%, Avg loss: 0.736493
```


Train validation by epoch.
1.8
1.6
1.4
1.2
1.0
0.8
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

```
[ ]:
```