model_resisc45

April 12, 2023

1 Model RESISC 45

[ ]:
```
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
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt

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

print(platform.platform())  # print current platform
```

```
macOS-13.3.1-arm64-arm-64bit
```

1.1 Set constants

[ ]:
```
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
```

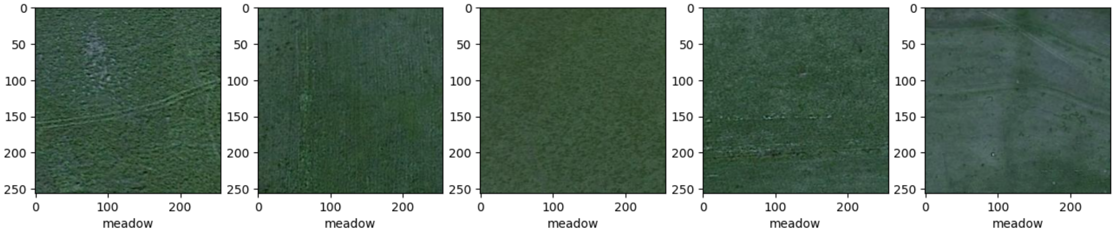
Conclusions: - Each category of the 45 categories has 700 samples

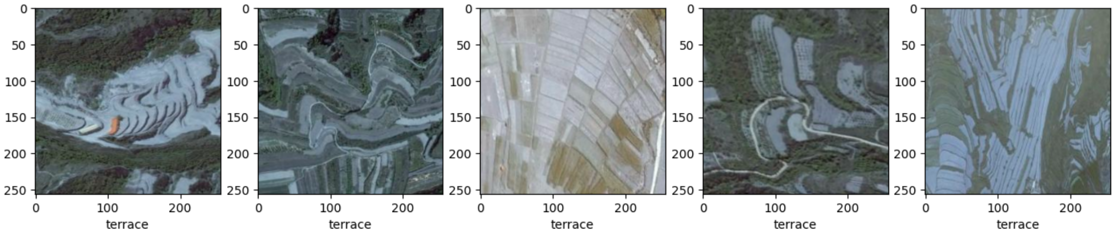
2.1 Sample subset of images from dataset

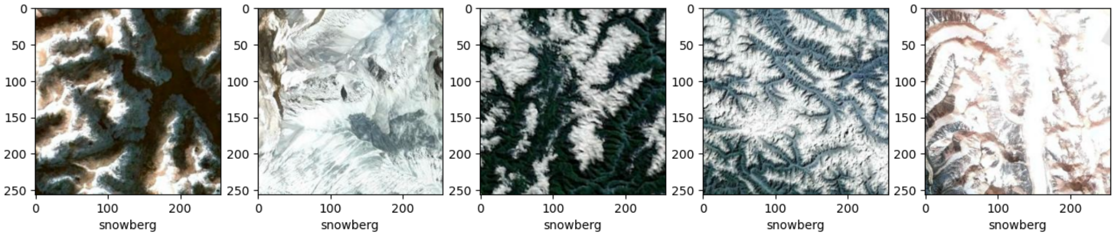
- show a subset of the images
- select some random labels (max 5 labels)
- show 5 images for each label to show variance

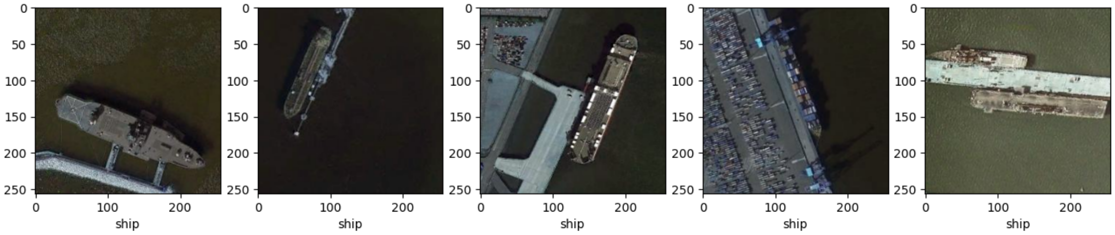
[ ]:

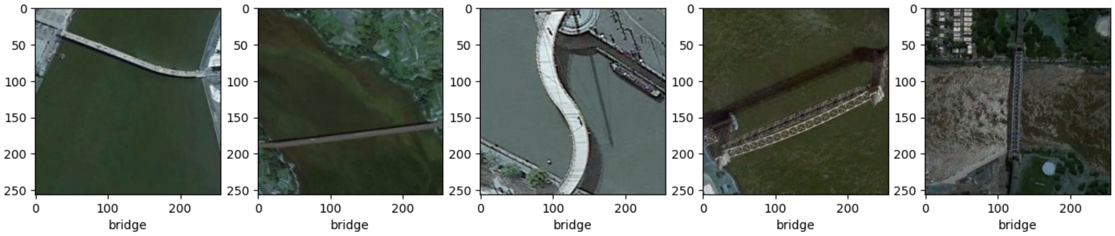
```
random_labels = random.sample(LABELS, min(5, len(LABELS)))
for label in random_labels:
    airplane_dataset = data[data['label'] == label].imgpath
    img_airplane = airplane_dataset.tolist()
    p1 = []
    for img in img_airplane[:5]:
        x = cv2.imread(img)
        p1.append(x)

    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()
```











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

2.4 Train dataset loader

[ ]:
```
# Dataset loader
import enum

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


RES_X = 256
RES_Y = 256

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)
        image = cv2.resize(image, dsize=(RES_Y, RES_X))
        image = image - 128.5
        image = np.moveaxis(image, -1, 0).astype(np.float32) / 255.0  # move␣
 ↪channels first (3 x RES_Y x RES_X)
        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 = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

3 Classification problem

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

[ ]:

```
# 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()
        self.maxpool4 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2))

        # 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.maxpool5 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2))
        self.flatten = nn.Flatten()

        # initialize first (and only) set of FC => RELU layers
        self.fc1 = nn.Linear(in_features=2500, 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)
        x = self.maxpool4(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)
        x = self.maxpool5(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)
        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=(3, 3), stride=(1, 1))
  (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()
  (maxpool4): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv5): Conv2d(200, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu5): ReLU()
  (maxpool5): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=2500, out_features=45, bias=True)
)
```

[ ]:

```python
loss_fn = nn.CrossEntropyLoss()

def train_one_epoch(dataloader, model):
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

    print("Started train.")
    size = len(dataloader.dataset)

    model.train()  # set model to train mode

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

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")
```

3.2 5 Fold train validation

- trebuie sa reducem timpii de antrenament

[ ]:

```
def kfold_train():
    epochs = 5
    batch_size = 64

    for val_fold in range(K_FOLDS):
        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)

        for t in range(epochs):
            print(f"Epoch {t+1}\n-------------------------------")
            train_one_epoch(train_dataloader, model)
            val_one_epoch(val_dataloader, model)
```

[ ]:

```
def simple_train():
    model = create_model_on_device(SatAlexNet)
    train_set = DatasetResisc45("dataset_resisc45.csv",␣
  ↪dataset_type=DatasetTypes.train, val_fold=5)
    val_set = DatasetResisc45("dataset_resisc45.csv", dataset_type=DatasetTypes.
  ↪val, val_fold=5)
    train_dataloader = DataLoader(train_set, batch_size=64, shuffle=True)
    val_dataloader = DataLoader(val_set, batch_size=64, shuffle=False)

    epochs = 10
    for t in range(epochs):
        print(f"Epoch {t+1}\n-------------------------------")
        train_one_epoch(train_dataloader, model)
        val_one_epoch(val_dataloader, model)
```

[ ]:

```
simple_train()
```

```
device: mps
SatAlexNet(
  (conv1): Conv2d(3, 50, kernel_size=(3, 3), stride=(1, 1))
  (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()
  (maxpool4): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (conv5): Conv2d(200, 100, kernel_size=(3, 3), stride=(1, 1))
  (relu5): ReLU()
  (maxpool5): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=0,
dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=2500, out_features=45, bias=True)
)
Epoch 1
```

```
-------------------------------
Started train.
loss: 3.806273  [   64/25200]
loss: 3.474065  [ 6464/25200]
loss: 2.936102  [12864/25200]
loss: 2.446790  [19264/25200]
Started validation.
Test Error:
 Accuracy: 32.9%, Avg loss: 2.364211

Epoch 2
-------------------------------
Started train.
loss: 2.398020  [   64/25200]
loss: 1.911188  [ 6464/25200]
loss: 2.067113  [12864/25200]
loss: 2.186711  [19264/25200]
Started validation.
Test Error:
 Accuracy: 43.1%, Avg loss: 1.932437

Epoch 3
-------------------------------
Started train.
loss: 1.844561  [   64/25200]
loss: 2.025104  [ 6464/25200]
loss: 1.833625  [12864/25200]
loss: 1.731433  [19264/25200]
Started validation.
Test Error:
 Accuracy: 48.9%, Avg loss: 1.700356

Epoch 4
-------------------------------
Started train.
loss: 1.305466  [   64/25200]
loss: 1.348948  [ 6464/25200]
loss: 1.635152  [12864/25200]
loss: 1.632333  [19264/25200]
Started validation.
Test Error:
 Accuracy: 52.2%, Avg loss: 1.583759

Epoch 5
-------------------------------
Started train.
loss: 1.613610  [   64/25200]
loss: 1.260794  [ 6464/25200]
```

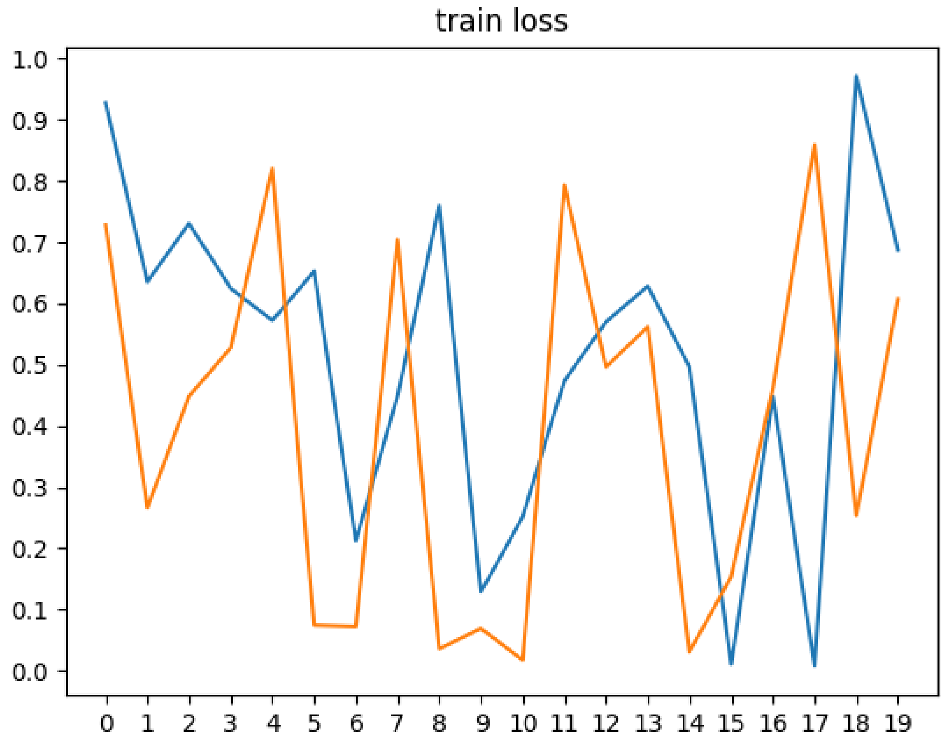
```
loss: 1.493123  [12864/25200]
loss: 1.378588  [19264/25200]
Started validation.
Test Error:
 Accuracy: 52.8%, Avg loss: 1.607404

Epoch 6
-------------------------------
Started train.
loss: 1.569013  [   64/25200]
loss: 1.201636  [ 6464/25200]
loss: 1.124559  [12864/25200]
loss: 1.296306  [19264/25200]
Started validation.
Test Error:
 Accuracy: 57.2%, Avg loss: 1.433341

Epoch 7
-------------------------------
Started train.
loss: 1.261605  [   64/25200]
loss: 1.517849  [ 6464/25200]
loss: 1.233839  [12864/25200]
loss: 1.598638  [19264/25200]
Started validation.
Test Error:
 Accuracy: 61.7%, Avg loss: 1.299462

Epoch 8
-------------------------------
Started train.
loss: 1.280254  [   64/25200]
loss: 0.906131  [ 6464/25200]
loss: 1.044051  [12864/25200]
loss: 0.985112  [19264/25200]
Started validation.
Test Error:
 Accuracy: 61.4%, Avg loss: 1.303278

Epoch 9
-------------------------------
Started train.
loss: 0.748255  [   64/25200]
loss: 1.170680  [ 6464/25200]
loss: 1.078530  [12864/25200]
loss: 0.927029  [19264/25200]
Started validation.
Test Error:
 Accuracy: 63.5%, Avg loss: 1.240218

Epoch 10
-------------------------------
Started train.
loss: 1.012814  [   64/25200]
loss: 1.044277  [ 6464/25200]
loss: 1.018787  [12864/25200]
loss: 1.054679  [19264/25200]
Started validation.
Test Error:
 Accuracy: 63.7%, Avg loss: 1.231658
```

3.3 Bias-variance trade

- work in progress..

[ ]:

```
%matplotlib ipympl

epochs = np.arange(0, 20, 1)
train_losses = []
val_losses = []
for epoc in epochs:
    train_losses.append(random.random())
    val_losses.append(random.random())

x = epochs
y = train_losses
y2 = val_losses

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 loss")
plt.show()
```


train loss
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

```
[ ]:
```