Model

This model is build to work with data from multiple datasets that have been processed to 112x112 size.

Imports and information

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from typing import Sequence
from functools import partial
from random import randint

import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn
import torchmetrics

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [4]:
```
# MLFW data paths
mlfw_X_fp = r'D:\data\face_mask\MLFW\MLFW_X.npy'
mlfw_y_fp = r'D:\data\face_mask\MLFW\MLFW_y.npy'

## WWMR data paths
wwmr_X_fp = r'D:\data\face_mask\WWMR cropped MediaPipe\WWMR_X_for_model.npy'
wwmr_y_fp = r'D:\data\face_mask\WWMR cropped MediaPipe\WWMR_y_for_model.npy'

# fmd_12k data paths
fmd12k_X_fp = r'D:\data\face_mask\FaceMaskDetection_12k\Cropped\images.npy'
fmd12k_y_fp = r'D:\data\face_mask\FaceMaskDetection_12k\Cropped\labels.npy'
```

Build data loader

In [5]:
```
class maskDataset(Dataset):
    def __init__(
        self,
        X_data,
        y_data,
        norm_0_1: bool = True,
        print_stats: bool = True,
    ):
        self.X_data = X_data

        # Norm
        if norm_0_1:
            self.X = self.X_data / 255
        else:
            self.X = self.X_data

        self.y = y_data

        self.length = len(self.y)

        # Print Stats
        if print_stats:
            print('# examples:    {}'.format(self.length))
            ratio = sum(self.y) / self.length
            print('class balence: {:.2f}'.format(ratio))

        # reshape?? see comment in __getitem__() ?????
        self.X = self.X.reshape((self.length, 3, 112, 112))

    def __len__(self):
        return self.length


    def __getitem__(self, index):
        image = self.X[index]

        # the input to a conv2d must be in [N, C, W, H] format
        #  n = number of examples, c is channels, w is width, and h is height
        #  This means we do not in fact need to transpose the data. it should
        #  be in the shape (3, 112, 112)
        #image = np.transpose(image)
        #image = np.rot90(image, k=3)

        return image.astype(np.float32), torch.tensor(self.y[index]).long()
```

In [6]:
```
# function to concat the various data sets and split into train val test

def merge_split(
    X_data_lists: list,
    y_data_lists: list,
    train=0.7,
    val=0.15,
    test=0.15
):
    if (train + val + test) != 1:
        print('splits must add to 1, added to {}'.format(train + val + test))
        return None
    if train < 0 or val < 0 or test < 0:
        print('splits must be positive')
        return None

    # Concat
    X = np.concatenate(X_data_lists)
    y = np.concatenate(y_data_lists)

    # split off test
    X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=test, random_state=42)

    # split off val
    val_percent_tv = val / (val + train)  # 15 percent of total data is equal to this
    X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=val_percent_tv, random_state=42)

    return [(X_train, y_train), (X_val, y_val), (X_test, y_test)]
```

Test data loader

In [7]:
```
# data file paths
X_fp_list = [mlfw_X_fp, wwmr_X_fp, fmd12k_X_fp]
y_fp_list = [mlfw_y_fp, wwmr_y_fp, fmd12k_y_fp]

X_data_list = [np.load(fp) for fp in X_fp_list]
y_data_list = [np.load(fp) for fp in y_fp_list]
```

In [8]:
```
# merge data and split into train val and test
(X_train, y_train), (X_val, y_val), (X_test, y_test) = merge_split(X_data_list, y_data_list)
```

In [9]:
```
ds = maskDataset(
    X_data=X_train,
    y_data=y_train,
    norm_0_1=True,
    print_stats=True,
)
```

```
# examples:    15078
class balence: 0.56
```

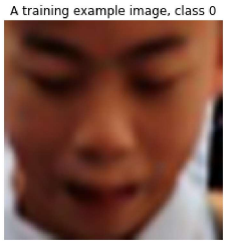
In [10]:
```
ds.X.shape
```

Out[10]:
```
(15078, 3, 112, 112)
```

In [11]:
```
ds[2][0].shape
```

Out[11]:
```
(3, 112, 112)
```

In [12]:
```
img_idx = randint(0, 8791)

image, label = ds[img_idx]

# un-normalize
image = (image * 255).astype(np.uint8)

# show image
plt.imshow(image.reshape(112, 112, 3))
plt.title('A training example image, class {}'.format(label) )
plt.axis('off')
plt.show()
```



Model

The following model is implemented in pytorch. It uses several convotutional layers followed by several linear layers.

In [275…
```
class CNN(nn.Module):
    def __init__(
        self,
        input_size: Sequence[int] = (3, 112, 112),
        num_classes: int = 2,
        channels: Sequence[int] = (8, 16, 32),
        kernel_sizes: Sequence[int] = (10, 10, 10),
        linear_units: Sequence[int] = (100, 10),
        lr: float = 0.001,
        epochs: int = 10
    ):
        super(CNN, self).__init__()

        self.input_size = input_size
        self.num_classes = num_classes
        self.channels = input_size[0:1] + channels
        self.kernel_sizes = kernel_sizes
        self.linear_units = linear_units
        self.lr = lr
        self.epochs = epochs

        self.flatten = nn.Flatten()
        self.pool = partial(nn.MaxPool2d, kernel_size=2, stride=1)  # first 2 is for 2x2 kernel, second arg is stride length
        self.pool2 = partial(nn.MaxPool2d, kernel_size=2, stride=2)  # first 2 is for 2x2 kernel, second arg is stride length
        self.dropout = nn.Dropout
        self.activation = nn.ReLU
        self.accuracy = torchmetrics.functional.accuracy
        self.conf_matrix = torchmetrics.functional.confusion_matrix

        # optional, define batch norm here

        # build the convolutional layers
        conv_layers = list()
        idx = 0
        for in_channels, out_channels, kernel_size in zip(
            self.channels[:-2], self.channels[1:-1], self.kernel_sizes[:-1]
        ):
            if idx == len(channels) - 2:
                conv_layers.append(
                    nn.Conv2d(
                        in_channels=in_channels,
                        out_channels=out_channels,
                        kernel_size=kernel_size,
                        stride=2,
                        #padding='same',
                    )
                )
            else:
                conv_layers.append(
                    nn.Conv2d(
                        in_channels=in_channels,
                        out_channels=out_channels,
                        kernel_size=kernel_size,
                        #stride=2,
                        #padding='same',
```

```
                    )
                )
            conv_layers.append(self.activation())
            conv_layers.append(self.pool())

            idx += 1

        # add final layer to convolutions
        conv_layers.append(
            nn.Conv2d(
                in_channels=self.channels[-2],
                out_channels=self.channels[-1],
                kernel_size=self.kernel_sizes[-1],
                stride=2,
                #padding='same',
            )
        )
        conv_layers.append(self.activation())
        conv_layers.append(self.pool2())


        # turn list into layers
        self.conv_net = nn.Sequential(*conv_layers)

        # calc lin layer input size
        temp = (self.input_size[2] - sum(kernel_sizes) )/ 4
        conv_out_size = int((temp + 1) / 2)
        lin_size = conv_out_size * conv_out_size * channels[-1]
        #print(lin_size)

        stride = [1] * len(self.kernel_sizes)
        stride[-1] = 2
        stride[-2] = 2
        #print(stride)
        curr = self.input_size[2]
        for i, k in enumerate(self.kernel_sizes):
            curr = curr - k
            curr = curr / stride[i]
            curr += 1
            #print(curr)
        curr = int(curr / 2)
        #print('new: {}'.format(curr))
        lin_size = curr * curr * channels[-1]

        # linear layers
        linear_layers = list()
        prev_linear_size = lin_size
        for dense_layer_size in self.linear_units:
            linear_layers.append(
                nn.Linear(
                    in_features=prev_linear_size,
                    out_features=dense_layer_size,
                )
            )
            linear_layers.append(self.activation())
            prev_linear_size=dense_layer_size

        self.penultimate_dense = nn.Sequential(*linear_layers)
        self.ultimate_dense = nn.Linear(
            in_features=self.linear_units[-1],
            out_features=self.num_classes
        )


    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.conv_net(x)
        #print(x.shape)
        x = self.flatten(x)
        # may need to expand dense entry since flatten
        x = self.penultimate_dense(x)
        x = self.ultimate_dense(x)
        return x


def train(dataloader, model, loss_fn, optimizer, verbose=False):
    #model = model.float()  # sometime fixes random obscure type error
    model.train()  # configures for training, grad on, dropout if there is dropout
    size = len(dataloader.dataset)

    for batch, (X, y) in enumerate(dataloader):
        optimizer.zero_grad()

        # compute prediction loss
        preds = model(X)
        loss = loss_fn(preds, y)

        # backprop
        loss.backward()
        optimizer.step()

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

# for evaluating on validation data too
def test(dataloader, model, loss_fn, verbose=False):
    model.eval()
    test_loss, correct = 0, 0
    size = len(dataloader.dataset)
    num_batches = len(dataloader)

    with torch.no_grad():
        for X, y in dataloader:

            pred = model(X.float())
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    if verbose:
        print(f"Results: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
    return correct, test_loss
```

The below 2 blocks show the structure of the model with the default parameters for the convolution sizes, convolution kernel sizes, and linear layer sizes.

In [276…

```
model_display = CNN(
```

```
    channels=(8, 16, 24, 32),
    kernel_sizes=(10, 10, 10, 10),
    linear_units=(1000, 500, 100, 10),
)
model_display
```

Out[276]:
```
CNN(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (conv_net): Sequential(
    (0): Conv2d(3, 8, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(8, 16, kernel_size=(10, 10), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(16, 24, kernel_size=(10, 10), stride=(2, 2))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(24, 32, kernel_size=(10, 10), stride=(2, 2))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (penultimate_dense): Sequential(
    (0): Linear(in_features=2048, out_features=1000, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1000, out_features=500, bias=True)
    (3): ReLU()
    (4): Linear(in_features=500, out_features=100, bias=True)
    (5): ReLU()
    (6): Linear(in_features=100, out_features=10, bias=True)
    (7): ReLU()
  )
  (ultimate_dense): Linear(in_features=10, out_features=2, bias=True)
)
```

In [277…
```
X, y = train_dataset[0:2]
X.shape
```

Out[277]:
```
(2, 3, 112, 112)
```

In [278…
```
model_display.conv_net(torch.Tensor(X)).shape
```

Out[278]:
```
torch.Size([2, 32, 8, 8])
```

In [279…
```
model_display.conv_net(torch.Tensor(X)).shape
```

Out[279]:
```
torch.Size([2, 32, 8, 8])
```

In [280…
```
8 * 8 * 32
```

Out[280]:
```
2048
```

In [281…
```
model_display.input_size[2] = 50
```

Out[281]:
```
62
```

In [282…
```
model_display.forward(torch.Tensor(X))
```

Out[282]:
```
tensor([[-0.1913,  0.2764],
        [-0.1912,  0.2763]], grad_fn=<AddmmBackward0>)
```

Running the model

Running the pytorch model involves several steps. First, the pytorch datasets must be set up. They take in the X and y data at construction to become an object that can serve up the data on command. Next, the pytorch data loaders are created. These data loaders are another pytorch object which takes in the dataset, whether to shuffle or not, and the batch size.

The model, loss function, and optimizer are created next. The training loop follows. This loop runs the training loop defined above with the model and then evaluates on the validation data.

In [230…
```
# Create datasets
train_dataset = maskDataset(
    X_data=X_train,
    y_data=y_train,
    norm_0_1=True,
    print_stats=False,
)

val_dataset = maskDataset(
    X_data=X_val,
    y_data=y_val,
    norm_0_1=True,
    print_stats=False,
)

test_dataset = maskDataset(
    X_data=X_test,
    y_data=y_test,
    norm_0_1=True,
    print_stats=False,
)
```

In [283…
```
# Create the dataLoaders
batch_size = 128

train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)

val_dataloader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False
)

test_dataloader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False
)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

# Create CNN
```

```
model = CNN(
    channels=(8, 16, 20, 24, 24),
    kernel_sizes=(10, 10, 10, 12, 12),
    linear_units=(1000, 250, 10),
)



# use cross entropy loss
loss_fn = nn.CrossEntropyLoss()

# SGD optimizer
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=0.005,
    momentum=0.9,
    #nesterov =True
    weight_decay =.0001
)
#optimizer = torch.optim.Adam(model.parameters(), lr=0.01, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)


# record results
train_loss = []
val_loss = []
val_accur = []

epochs = 15
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")

    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=True)
    train_loss.append(train_l)

    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=True)
    val_loss.append(val_l)
    val_accur.append(val_a)
```

```
Using cpu device
Epoch 1
-------------------------------
loss: 0.737606  [    0/15078]
loss: 0.744497  [  640/15078]
loss: 0.750687  [ 1280/15078]
loss: 0.727971  [ 1920/15078]
loss: 0.694906  [ 2560/15078]
loss: 0.713736  [ 3200/15078]
loss: 0.702511  [ 3840/15078]
loss: 0.699192  [ 4480/15078]
loss: 0.693599  [ 5120/15078]
loss: 0.689570  [ 5760/15078]
loss: 0.688264  [ 6400/15078]
loss: 0.677754  [ 7040/15078]
loss: 0.685393  [ 7680/15078]
loss: 0.683217  [ 8320/15078]
loss: 0.671568  [ 8960/15078]
loss: 0.698884  [ 9600/15078]
loss: 0.688622  [10240/15078]
loss: 0.688817  [10880/15078]
loss: 0.675889  [11520/15078]
loss: 0.683100  [12160/15078]
loss: 0.680901  [12800/15078]
loss: 0.674691  [13440/15078]
loss: 0.687334  [14080/15078]
loss: 0.674543  [14720/15078]
Results:
 Accuracy: 55.3%, Avg loss: 0.687457

Epoch 2
-------------------------------
loss: 0.685103  [    0/15078]
loss: 0.682854  [  640/15078]
loss: 0.682868  [ 1280/15078]
loss: 0.678288  [ 1920/15078]
loss: 0.683017  [ 2560/15078]
loss: 0.694738  [ 3200/15078]
loss: 0.718192  [ 3840/15078]
loss: 0.705647  [ 4480/15078]
loss: 0.687226  [ 5120/15078]
loss: 0.691650  [ 5760/15078]
loss: 0.712617  [ 6400/15078]
loss: 0.676448  [ 7040/15078]
loss: 0.664114  [ 7680/15078]
loss: 0.692985  [ 8320/15078]
loss: 0.678538  [ 8960/15078]
loss: 0.666383  [ 9600/15078]
loss: 0.690672  [10240/15078]
loss: 0.699628  [10880/15078]
loss: 0.677447  [11520/15078]
loss: 0.690391  [12160/15078]
loss: 0.690270  [12800/15078]
loss: 0.683413  [13440/15078]
loss: 0.695604  [14080/15078]
loss: 0.681383  [14720/15078]
Results:
 Accuracy: 55.3%, Avg loss: 0.685659

Epoch 3
-------------------------------
loss: 0.659585  [    0/15078]
loss: 0.676362  [  640/15078]
loss: 0.682638  [ 1280/15078]
loss: 0.674901  [ 1920/15078]
loss: 0.692717  [ 2560/15078]
loss: 0.665519  [ 3200/15078]
loss: 0.643234  [ 3840/15078]
loss: 0.625628  [ 4480/15078]
loss: 0.665546  [ 5120/15078]
loss: 0.701961  [ 5760/15078]
loss: 0.646444  [ 6400/15078]
loss: 0.681894  [ 7040/15078]
loss: 0.655971  [ 7680/15078]
loss: 0.638510  [ 8320/15078]
loss: 0.676354  [ 8960/15078]
loss: 0.610949  [ 9600/15078]
loss: 0.609477  [10240/15078]
loss: 0.558016  [10880/15078]
loss: 0.700724  [11520/15078]
loss: 0.695586  [12160/15078]
loss: 0.690591  [12800/15078]
loss: 0.680071  [13440/15078]
loss: 0.653812  [14080/15078]
loss: 0.677336  [14720/15078]
Results:
 Accuracy: 55.7%, Avg loss: 0.677452

Epoch 4
-------------------------------
loss: 0.685039  [    0/15078]
loss: 0.690661  [  640/15078]
loss: 0.666743  [ 1280/15078]
loss: 0.686046  [ 1920/15078]
loss: 0.685043  [ 2560/15078]
loss: 0.663619  [ 3200/15078]
loss: 0.660644  [ 3840/15078]
loss: 0.683730  [ 4480/15078]
loss: 0.660725  [ 5120/15078]
loss: 0.672558  [ 5760/15078]
loss: 0.661105  [ 6400/15078]
loss: 0.628046  [ 7040/15078]
loss: 0.645933  [ 7680/15078]
loss: 0.650137  [ 8320/15078]
loss: 0.638260  [ 8960/15078]
loss: 0.619083  [ 9600/15078]
loss: 0.575841  [10240/15078]
loss: 0.577204  [10880/15078]
loss: 0.611483  [11520/15078]
loss: 0.562393  [12160/15078]
loss: 0.646647  [12800/15078]
loss: 0.649900  [13440/15078]
loss: 0.602568  [14080/15078]
loss: 0.479614  [14720/15078]
Results:
 Accuracy: 75.3%, Avg loss: 0.565152

Epoch 5
-------------------------------
loss: 0.528919  [    0/15078]
```

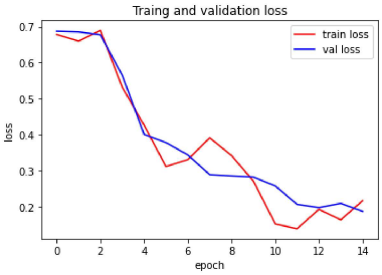
```
loss: 0.579774  [  640/15078]
loss: 0.552313  [ 1280/15078]
loss: 0.445595  [ 1920/15078]
loss: 0.416797  [ 2560/15078]
loss: 0.713425  [ 3200/15078]
loss: 0.441581  [ 3840/15078]
loss: 0.458463  [ 4480/15078]
loss: 0.488811  [ 5120/15078]
loss: 0.384381  [ 5760/15078]
loss: 0.461892  [ 6400/15078]
loss: 0.514252  [ 7040/15078]
loss: 0.566554  [ 7680/15078]
loss: 0.384214  [ 8320/15078]
loss: 0.457225  [ 8960/15078]
loss: 0.400152  [ 9600/15078]
loss: 0.415769  [10240/15078]
loss: 0.465008  [10880/15078]
loss: 0.432559  [11520/15078]
loss: 0.400775  [12160/15078]
loss: 0.397108  [12800/15078]
loss: 0.386401  [13440/15078]
loss: 0.444481  [14080/15078]
loss: 0.461054  [14720/15078]
Results:
 Accuracy: 85.2%, Avg loss: 0.400378

Epoch 6
-------------------------------
loss: 0.369146  [    0/15078]
loss: 0.353001  [  640/15078]
loss: 0.334424  [ 1280/15078]
loss: 0.350412  [ 1920/15078]
loss: 0.388124  [ 2560/15078]
loss: 0.368844  [ 3200/15078]
loss: 0.333448  [ 3840/15078]
loss: 0.441489  [ 4480/15078]
loss: 0.273292  [ 5120/15078]
loss: 0.352391  [ 5760/15078]
loss: 0.414047  [ 6400/15078]
loss: 0.282759  [ 7040/15078]
loss: 0.615680  [ 7680/15078]
loss: 0.403478  [ 8320/15078]
loss: 0.486642  [ 8960/15078]
loss: 0.383981  [ 9600/15078]
loss: 0.452083  [10240/15078]
loss: 0.343465  [10880/15078]
loss: 0.363207  [11520/15078]
loss: 0.451100  [12160/15078]
loss: 0.353805  [12800/15078]
loss: 0.456317  [13440/15078]
loss: 0.294094  [14080/15078]
loss: 0.482624  [14720/15078]
Results:
 Accuracy: 85.6%, Avg loss: 0.377752

Epoch 7
-------------------------------
loss: 0.265122  [    0/15078]
loss: 0.318746  [  640/15078]
loss: 0.264677  [ 1280/15078]
loss: 0.304751  [ 1920/15078]
loss: 0.380097  [ 2560/15078]
loss: 0.419340  [ 3200/15078]
loss: 0.285165  [ 3840/15078]
loss: 0.347367  [ 4480/15078]
loss: 0.295190  [ 5120/15078]
loss: 0.273959  [ 5760/15078]
loss: 0.293407  [ 6400/15078]
loss: 0.268339  [ 7040/15078]
loss: 0.346925  [ 7680/15078]
loss: 0.375767  [ 8320/15078]
loss: 0.330366  [ 8960/15078]
loss: 0.295518  [ 9600/15078]
loss: 0.247234  [10240/15078]
loss: 0.365530  [10880/15078]
loss: 0.318339  [11520/15078]
loss: 0.389294  [12160/15078]
loss: 0.406104  [12800/15078]
loss: 0.277492  [13440/15078]
loss: 0.272087  [14080/15078]
loss: 0.231786  [14720/15078]
Results:
 Accuracy: 85.1%, Avg loss: 0.343271

Epoch 8
-------------------------------
loss: 0.377668  [    0/15078]
loss: 0.362043  [  640/15078]
loss: 0.284892  [ 1280/15078]
loss: 0.270337  [ 1920/15078]
loss: 0.249997  [ 2560/15078]
loss: 0.513384  [ 3200/15078]
loss: 0.334363  [ 3840/15078]
loss: 0.295776  [ 4480/15078]
loss: 0.376370  [ 5120/15078]
loss: 0.280536  [ 5760/15078]
loss: 0.312855  [ 6400/15078]
loss: 0.376361  [ 7040/15078]
loss: 0.277280  [ 7680/15078]
loss: 0.234502  [ 8320/15078]
loss: 0.303375  [ 8960/15078]
loss: 0.379178  [ 9600/15078]
loss: 0.359067  [10240/15078]
loss: 0.426220  [10880/15078]
loss: 0.268612  [11520/15078]
loss: 0.339210  [12160/15078]
loss: 0.332665  [12800/15078]
loss: 0.307094  [13440/15078]
loss: 0.405919  [14080/15078]
loss: 0.260886  [14720/15078]
Results:
 Accuracy: 88.7%, Avg loss: 0.288205

Epoch 9
-------------------------------
loss: 0.314879  [    0/15078]
loss: 0.397098  [  640/15078]
loss: 0.240357  [ 1280/15078]
loss: 0.236184  [ 1920/15078]
loss: 0.324744  [ 2560/15078]
```

```
loss: 0.294322  [ 3200/15078]
loss: 0.302205  [ 3840/15078]
loss: 0.267808  [ 4480/15078]
loss: 0.397596  [ 5120/15078]
loss: 0.282512  [ 5760/15078]
loss: 0.321358  [ 6400/15078]
loss: 0.275850  [ 7040/15078]
loss: 0.217325  [ 7680/15078]
loss: 0.370935  [ 8320/15078]
loss: 0.248633  [ 8960/15078]
loss: 0.272027  [ 9600/15078]
loss: 0.294143  [10240/15078]
loss: 0.296216  [10880/15078]
loss: 0.427563  [11520/15078]
loss: 0.294912  [12160/15078]
loss: 0.351557  [12800/15078]
loss: 0.256641  [13440/15078]
loss: 0.270743  [14080/15078]
loss: 0.286957  [14720/15078]
Results:
 Accuracy: 88.0%, Avg loss: 0.284697

Epoch 10
-------------------------------
loss: 0.253547  [    0/15078]
loss: 0.336261  [  640/15078]
loss: 0.154033  [ 1280/15078]
loss: 0.391348  [ 1920/15078]
loss: 0.325755  [ 2560/15078]
loss: 0.298026  [ 3200/15078]
loss: 0.283209  [ 3840/15078]
loss: 0.359175  [ 4480/15078]
loss: 0.301513  [ 5120/15078]
loss: 0.232650  [ 5760/15078]
loss: 0.285993  [ 6400/15078]
loss: 0.162581  [ 7040/15078]
loss: 0.250664  [ 7680/15078]
loss: 0.270403  [ 8320/15078]
loss: 0.205876  [ 8960/15078]
loss: 0.194972  [ 9600/15078]
loss: 0.173291  [10240/15078]
loss: 0.312278  [10880/15078]
loss: 0.322124  [11520/15078]
loss: 0.264176  [12160/15078]
loss: 0.335821  [12800/15078]
loss: 0.249367  [13440/15078]
loss: 0.255549  [14080/15078]
loss: 0.272763  [14720/15078]
Results:
 Accuracy: 89.4%, Avg loss: 0.281641

Epoch 11
-------------------------------
loss: 0.394182  [    0/15078]
loss: 0.296740  [  640/15078]
loss: 0.264414  [ 1280/15078]
loss: 0.341650  [ 1920/15078]
loss: 0.257641  [ 2560/15078]
loss: 0.249503  [ 3200/15078]
loss: 0.205000  [ 3840/15078]
loss: 0.235511  [ 4480/15078]
loss: 0.162287  [ 5120/15078]
loss: 0.227625  [ 5760/15078]
loss: 0.233821  [ 6400/15078]
loss: 0.306066  [ 7040/15078]
loss: 0.334803  [ 7680/15078]
loss: 0.310557  [ 8320/15078]
loss: 0.256445  [ 8960/15078]
loss: 0.273013  [ 9600/15078]
loss: 0.326041  [10240/15078]
loss: 0.276011  [10880/15078]
loss: 0.201515  [11520/15078]
loss: 0.235076  [12160/15078]
loss: 0.283954  [12800/15078]
loss: 0.221669  [13440/15078]
loss: 0.169489  [14080/15078]
loss: 0.180030  [14720/15078]
Results:
 Accuracy: 90.4%, Avg loss: 0.257077

Epoch 12
-------------------------------
loss: 0.231052  [    0/15078]
loss: 0.238338  [  640/15078]
loss: 0.141971  [ 1280/15078]
loss: 0.238139  [ 1920/15078]
loss: 0.299030  [ 2560/15078]
loss: 0.222384  [ 3200/15078]
loss: 0.407773  [ 3840/15078]
loss: 0.304300  [ 4480/15078]
loss: 0.215079  [ 5120/15078]
loss: 0.172521  [ 5760/15078]
loss: 0.229010  [ 6400/15078]
loss: 0.271535  [ 7040/15078]
loss: 0.177634  [ 7680/15078]
loss: 0.173847  [ 8320/15078]
loss: 0.244967  [ 8960/15078]
loss: 0.262290  [ 9600/15078]
loss: 0.273834  [10240/15078]
loss: 0.239663  [10880/15078]
loss: 0.247852  [11520/15078]
loss: 0.207024  [12160/15078]
loss: 0.254134  [12800/15078]
loss: 0.251220  [13440/15078]
loss: 0.254038  [14080/15078]
loss: 0.226977  [14720/15078]
Results:
 Accuracy: 92.2%, Avg loss: 0.205852

Epoch 13
-------------------------------
loss: 0.273128  [    0/15078]
loss: 0.352053  [  640/15078]
loss: 0.188183  [ 1280/15078]
loss: 0.198786  [ 1920/15078]
loss: 0.265681  [ 2560/15078]
loss: 0.206594  [ 3200/15078]
loss: 0.270194  [ 3840/15078]
loss: 0.312147  [ 4480/15078]
loss: 0.334211  [ 5120/15078]
```

```
loss: 0.305709  [ 5760/15078]
loss: 0.265243  [ 6400/15078]
loss: 0.272193  [ 7040/15078]
loss: 0.323976  [ 7680/15078]
loss: 0.236783  [ 8320/15078]
loss: 0.272537  [ 8960/15078]
loss: 0.240322  [ 9600/15078]
loss: 0.210236  [10240/15078]
loss: 0.214519  [10880/15078]
loss: 0.238408  [11520/15078]
loss: 0.169403  [12160/15078]
loss: 0.234045  [12800/15078]
loss: 0.157109  [13440/15078]
loss: 0.190613  [14080/15078]
loss: 0.166466  [14720/15078]
Results:
 Accuracy: 92.6%, Avg loss: 0.196806

Epoch 14
-------------------------------
loss: 0.213288  [    0/15078]
loss: 0.302172  [  640/15078]
loss: 0.147474  [ 1280/15078]
loss: 0.259528  [ 1920/15078]
loss: 0.304372  [ 2560/15078]
loss: 0.224179  [ 3200/15078]
loss: 0.203741  [ 3840/15078]
loss: 0.171100  [ 4480/15078]
loss: 0.158566  [ 5120/15078]
loss: 0.220988  [ 5760/15078]
loss: 0.238889  [ 6400/15078]
loss: 0.127343  [ 7040/15078]
loss: 0.198742  [ 7680/15078]
loss: 0.159605  [ 8320/15078]
loss: 0.118203  [ 8960/15078]
loss: 0.263582  [ 9600/15078]
loss: 0.180435  [10240/15078]
loss: 0.167546  [10880/15078]
loss: 0.141859  [11520/15078]
loss: 0.196922  [12160/15078]
loss: 0.284080  [12800/15078]
loss: 0.186677  [13440/15078]
loss: 0.170035  [14080/15078]
loss: 0.223868  [14720/15078]
Results:
 Accuracy: 91.9%, Avg loss: 0.208578

Epoch 15
-------------------------------
loss: 0.204707  [    0/15078]
loss: 0.208794  [  640/15078]
loss: 0.242907  [ 1280/15078]
loss: 0.146547  [ 1920/15078]
loss: 0.173371  [ 2560/15078]
loss: 0.190307  [ 3200/15078]
loss: 0.141499  [ 3840/15078]
loss: 0.197997  [ 4480/15078]
loss: 0.192170  [ 5120/15078]
loss: 0.304215  [ 5760/15078]
loss: 0.263965  [ 6400/15078]
loss: 0.221300  [ 7040/15078]
loss: 0.246046  [ 7680/15078]
loss: 0.151570  [ 8320/15078]
loss: 0.197761  [ 8960/15078]
loss: 0.242839  [ 9600/15078]
loss: 0.204642  [10240/15078]
loss: 0.254564  [10880/15078]
loss: 0.196049  [11520/15078]
loss: 0.138466  [12160/15078]
loss: 0.181245  [12800/15078]
loss: 0.253298  [13440/15078]
loss: 0.180729  [14080/15078]
loss: 0.210717  [14720/15078]
Results:
 Accuracy: 92.7%, Avg loss: 0.186181
```

In [284…

```python
# plot training loss and validation loss
plt.plot(np.arange(len(train_loss)), [i.item() for i in train_loss], 'r', label='train loss') # train in red
plt.plot(np.arange(len(val_loss)), val_loss, 'b', label='val loss')# val in blue
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Traing and validation loss')
plt.show()
```



In [285…

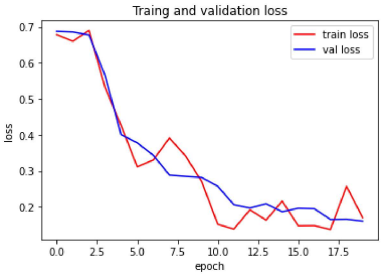
```python
# 5 more epochs

epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")

    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=True)
    train_loss.append(train_l)

    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=True)
    val_loss.append(val_l)
    val_accur.append(val_a)
```

```
Epoch 1
-------------------------------
loss: 0.215898  [    0/15078]
loss: 0.293785  [  640/15078]
loss: 0.201036  [ 1280/15078]
loss: 0.171557  [ 1920/15078]
loss: 0.145148  [ 2560/15078]
loss: 0.170801  [ 3200/15078]
loss: 0.262786  [ 3840/15078]
loss: 0.213447  [ 4480/15078]
loss: 0.179175  [ 5120/15078]
loss: 0.148568  [ 5760/15078]
loss: 0.202989  [ 6400/15078]
loss: 0.218207  [ 7040/15078]
loss: 0.203972  [ 7680/15078]
loss: 0.152598  [ 8320/15078]
loss: 0.186304  [ 8960/15078]
loss: 0.173801  [ 9600/15078]
loss: 0.123450  [10240/15078]
loss: 0.194701  [10880/15078]
loss: 0.264114  [11520/15078]
loss: 0.188916  [12160/15078]
loss: 0.260924  [12800/15078]
loss: 0.186690  [13440/15078]
loss: 0.147473  [14080/15078]
loss: 0.112222  [14720/15078]
Results:
 Accuracy: 92.1%, Avg loss: 0.196190

Epoch 2
-------------------------------
loss: 0.214472  [    0/15078]
loss: 0.162988  [  640/15078]
loss: 0.216132  [ 1280/15078]
loss: 0.135579  [ 1920/15078]
loss: 0.119976  [ 2560/15078]
loss: 0.193843  [ 3200/15078]
loss: 0.216987  [ 3840/15078]
loss: 0.144721  [ 4480/15078]
loss: 0.212627  [ 5120/15078]
loss: 0.272631  [ 5760/15078]
loss: 0.195819  [ 6400/15078]
loss: 0.152708  [ 7040/15078]
loss: 0.160433  [ 7680/15078]
loss: 0.146101  [ 8320/15078]
loss: 0.183569  [ 8960/15078]
loss: 0.220926  [ 9600/15078]
loss: 0.206687  [10240/15078]
loss: 0.210226  [10880/15078]
loss: 0.251408  [11520/15078]
loss: 0.265919  [12160/15078]
loss: 0.126998  [12800/15078]
loss: 0.137578  [13440/15078]
loss: 0.103647  [14080/15078]
loss: 0.197345  [14720/15078]
Results:
 Accuracy: 92.3%, Avg loss: 0.194825

Epoch 3
-------------------------------
loss: 0.250114  [    0/15078]
loss: 0.149061  [  640/15078]
loss: 0.168563  [ 1280/15078]
loss: 0.153256  [ 1920/15078]
loss: 0.123878  [ 2560/15078]
loss: 0.206976  [ 3200/15078]
loss: 0.194787  [ 3840/15078]
loss: 0.170303  [ 4480/15078]
loss: 0.162297  [ 5120/15078]
loss: 0.164027  [ 5760/15078]
loss: 0.126308  [ 6400/15078]
loss: 0.105619  [ 7040/15078]
loss: 0.162100  [ 7680/15078]
loss: 0.247633  [ 8320/15078]
loss: 0.327190  [ 8960/15078]
loss: 0.276610  [ 9600/15078]
loss: 0.176065  [10240/15078]
loss: 0.161013  [10880/15078]
loss: 0.263622  [11520/15078]
loss: 0.249051  [12160/15078]
loss: 0.285132  [12800/15078]
loss: 0.141110  [13440/15078]
loss: 0.165473  [14080/15078]
loss: 0.202096  [14720/15078]
Results:
 Accuracy: 93.4%, Avg loss: 0.164687

Epoch 4
-------------------------------
loss: 0.193732  [    0/15078]
loss: 0.151610  [  640/15078]
loss: 0.159758  [ 1280/15078]
loss: 0.201613  [ 1920/15078]
loss: 0.194667  [ 2560/15078]
loss: 0.271992  [ 3200/15078]
loss: 0.226739  [ 3840/15078]
loss: 0.223160  [ 4480/15078]
loss: 0.130273  [ 5120/15078]
loss: 0.206397  [ 5760/15078]
loss: 0.160588  [ 6400/15078]
loss: 0.135408  [ 7040/15078]
loss: 0.134991  [ 7680/15078]
loss: 0.191958  [ 8320/15078]
loss: 0.129193  [ 8960/15078]
loss: 0.192310  [ 9600/15078]
loss: 0.123743  [10240/15078]
loss: 0.215377  [10880/15078]
loss: 0.147592  [11520/15078]
loss: 0.103026  [12160/15078]
loss: 0.102311  [12800/15078]
loss: 0.162359  [13440/15078]
loss: 0.183303  [14080/15078]
loss: 0.238018  [14720/15078]
Results:
 Accuracy: 93.7%, Avg loss: 0.165469

Epoch 5
-------------------------------
loss: 0.172345  [    0/15078]
loss: 0.170895  [  640/15078]
```

```
loss: 0.146500  [ 1280/15078]
loss: 0.165464  [ 1920/15078]
loss: 0.205789  [ 2560/15078]
loss: 0.192033  [ 3200/15078]
loss: 0.179888  [ 3840/15078]
loss: 0.148833  [ 4480/15078]
loss: 0.203739  [ 5120/15078]
loss: 0.189191  [ 5760/15078]
loss: 0.207910  [ 6400/15078]
loss: 0.162195  [ 7040/15078]
loss: 0.186564  [ 7680/15078]
loss: 0.166259  [ 8320/15078]
loss: 0.171804  [ 8960/15078]
loss: 0.116966  [ 9600/15078]
loss: 0.206866  [10240/15078]
loss: 0.160577  [10880/15078]
loss: 0.194058  [11520/15078]
loss: 0.169982  [12160/15078]
loss: 0.159120  [12800/15078]
loss: 0.226216  [13440/15078]
loss: 0.169901  [14080/15078]
loss: 0.131895  [14720/15078]
Results:
 Accuracy: 93.7%, Avg loss: 0.160123
```

In [286…

```
# plot training loss and validation loss
plt.plot(np.arange(len(train_loss)), [i.item() for i in train_loss], 'r', label='train loss') # train in red
plt.plot(np.arange(len(val_loss)), val_loss, 'b', label='val loss')# val in blue
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Traing and validation loss')
plt.show()
```



In [ ]:

```
best_val_acc = 0
```

In [293…

```
epoch_counter = 27
```

In [294…

```
# more epochs, saving best val

weights_state_base_fp = r'./results/upgrade/full_model_upgrade_'
optimizer_state_base_fp = r'./results/upgrade/full_optimizer_upgrade_'


epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")

    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=True)
    train_loss.append(train_l)

    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=True)
    val_loss.append(val_l)
    val_accur.append(val_a)

    epoch_counter += 1

    if val_a > best_val_acc:
        best_val_acc = val_a
        out_fp = weights_state_base_fp + str(epoch_counter)
        torch.save(model.state_dict(), out_fp)
        optim_fp = optimizer_state_base_fp + str(epoch_counter)
        torch.save(optimizer.state_dict(), optim_fp)
```

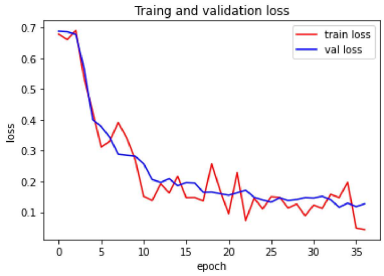
```
Epoch 1
-------------------------------
loss: 0.180649  [    0/15078]
loss: 0.098230  [  640/15078]
loss: 0.121553  [ 1280/15078]
loss: 0.086369  [ 1920/15078]
loss: 0.140690  [ 2560/15078]
loss: 0.167311  [ 3200/15078]
loss: 0.083937  [ 3840/15078]
loss: 0.123111  [ 4480/15078]
loss: 0.131189  [ 5120/15078]
loss: 0.103990  [ 5760/15078]
loss: 0.117956  [ 6400/15078]
loss: 0.199627  [ 7040/15078]
loss: 0.093161  [ 7680/15078]
loss: 0.116982  [ 8320/15078]
loss: 0.119668  [ 8960/15078]
loss: 0.148468  [ 9600/15078]
loss: 0.196880  [10240/15078]
loss: 0.228657  [10880/15078]
loss: 0.083997  [11520/15078]
loss: 0.134714  [12160/15078]
loss: 0.128567  [12800/15078]
loss: 0.161533  [13440/15078]
loss: 0.103664  [14080/15078]
loss: 0.075026  [14720/15078]
Results:
 Accuracy: 94.9%, Avg loss: 0.137986

Epoch 2
-------------------------------
loss: 0.121880  [    0/15078]
loss: 0.065191  [  640/15078]
loss: 0.142476  [ 1280/15078]
loss: 0.226767  [ 1920/15078]
loss: 0.147818  [ 2560/15078]
loss: 0.136746  [ 3200/15078]
loss: 0.086286  [ 3840/15078]
loss: 0.090283  [ 4480/15078]
loss: 0.098691  [ 5120/15078]
loss: 0.135880  [ 5760/15078]
loss: 0.042584  [ 6400/15078]
loss: 0.120148  [ 7040/15078]
loss: 0.182455  [ 7680/15078]
loss: 0.113923  [ 8320/15078]
loss: 0.076447  [ 8960/15078]
loss: 0.133307  [ 9600/15078]
loss: 0.125915  [10240/15078]
loss: 0.139545  [10880/15078]
loss: 0.126478  [11520/15078]
loss: 0.110621  [12160/15078]
loss: 0.138050  [12800/15078]
loss: 0.106898  [13440/15078]
loss: 0.167331  [14080/15078]
loss: 0.238931  [14720/15078]
Results:
 Accuracy: 94.5%, Avg loss: 0.141272

Epoch 3
-------------------------------
loss: 0.131424  [    0/15078]
loss: 0.118199  [  640/15078]
loss: 0.089611  [ 1280/15078]
loss: 0.127522  [ 1920/15078]
loss: 0.169141  [ 2560/15078]
loss: 0.094618  [ 3200/15078]
loss: 0.084821  [ 3840/15078]
loss: 0.100070  [ 4480/15078]
loss: 0.095901  [ 5120/15078]
loss: 0.089438  [ 5760/15078]
loss: 0.096197  [ 6400/15078]
loss: 0.069249  [ 7040/15078]
loss: 0.063950  [ 7680/15078]
loss: 0.110556  [ 8320/15078]
loss: 0.077628  [ 8960/15078]
loss: 0.141362  [ 9600/15078]
loss: 0.085373  [10240/15078]
loss: 0.230696  [10880/15078]
loss: 0.087790  [11520/15078]
loss: 0.100764  [12160/15078]
loss: 0.125782  [12800/15078]
loss: 0.114795  [13440/15078]
loss: 0.107793  [14080/15078]
loss: 0.103518  [14720/15078]
Results:
 Accuracy: 94.7%, Avg loss: 0.147353

Epoch 4
-------------------------------
loss: 0.121255  [    0/15078]
loss: 0.173690  [  640/15078]
loss: 0.039127  [ 1280/15078]
loss: 0.077344  [ 1920/15078]
loss: 0.092241  [ 2560/15078]
loss: 0.163412  [ 3200/15078]
loss: 0.165529  [ 3840/15078]
loss: 0.141219  [ 4480/15078]
loss: 0.099025  [ 5120/15078]
loss: 0.153880  [ 5760/15078]
loss: 0.101119  [ 6400/15078]
loss: 0.095273  [ 7040/15078]
loss: 0.142169  [ 7680/15078]
loss: 0.101347  [ 8320/15078]
loss: 0.104815  [ 8960/15078]
loss: 0.125322  [ 9600/15078]
loss: 0.090829  [10240/15078]
loss: 0.176826  [10880/15078]
loss: 0.151875  [11520/15078]
loss: 0.124423  [12160/15078]
loss: 0.073927  [12800/15078]
loss: 0.099312  [13440/15078]
loss: 0.114786  [14080/15078]
loss: 0.164877  [14720/15078]
Results:
 Accuracy: 94.7%, Avg loss: 0.146086

Epoch 5
-------------------------------
loss: 0.116828  [    0/15078]
loss: 0.049439  [  640/15078]
```

```
loss: 0.162675  [ 1280/15078]
loss: 0.166084  [ 1920/15078]
loss: 0.132756  [ 2560/15078]
loss: 0.117950  [ 3200/15078]
loss: 0.104137  [ 3840/15078]
loss: 0.124646  [ 4480/15078]
loss: 0.095677  [ 5120/15078]
loss: 0.139527  [ 5760/15078]
loss: 0.081084  [ 6400/15078]
loss: 0.069588  [ 7040/15078]
loss: 0.100656  [ 7680/15078]
loss: 0.104177  [ 8320/15078]
loss: 0.101513  [ 8960/15078]
loss: 0.101258  [ 9600/15078]
loss: 0.168712  [10240/15078]
loss: 0.158532  [10880/15078]
loss: 0.151317  [11520/15078]
loss: 0.118759  [12160/15078]
loss: 0.157467  [12800/15078]
loss: 0.176910  [13440/15078]
loss: 0.109307  [14080/15078]
loss: 0.123145  [14720/15078]
Results:
 Accuracy: 94.0%, Avg loss: 0.151628

Epoch 6
-------------------------------
loss: 0.172983  [    0/15078]
loss: 0.129490  [  640/15078]
loss: 0.089693  [ 1280/15078]
loss: 0.137315  [ 1920/15078]
loss: 0.077647  [ 2560/15078]
loss: 0.103054  [ 3200/15078]
loss: 0.152847  [ 3840/15078]
loss: 0.170138  [ 4480/15078]
loss: 0.123047  [ 5120/15078]
loss: 0.099369  [ 5760/15078]
loss: 0.102489  [ 6400/15078]
loss: 0.096290  [ 7040/15078]
loss: 0.100478  [ 7680/15078]
loss: 0.054635  [ 8320/15078]
loss: 0.079671  [ 8960/15078]
loss: 0.106327  [ 9600/15078]
loss: 0.194054  [10240/15078]
loss: 0.101296  [10880/15078]
loss: 0.096242  [11520/15078]
loss: 0.080119  [12160/15078]
loss: 0.114162  [12800/15078]
loss: 0.058704  [13440/15078]
loss: 0.056765  [14080/15078]
loss: 0.069221  [14720/15078]
Results:
 Accuracy: 94.8%, Avg loss: 0.140603

Epoch 7
-------------------------------
loss: 0.114662  [    0/15078]
loss: 0.101896  [  640/15078]
loss: 0.049298  [ 1280/15078]
loss: 0.097319  [ 1920/15078]
loss: 0.090794  [ 2560/15078]
loss: 0.103514  [ 3200/15078]
loss: 0.105806  [ 3840/15078]
loss: 0.069411  [ 4480/15078]
loss: 0.126983  [ 5120/15078]
loss: 0.175876  [ 5760/15078]
loss: 0.096569  [ 6400/15078]
loss: 0.193443  [ 7040/15078]
loss: 0.078657  [ 7680/15078]
loss: 0.152756  [ 8320/15078]
loss: 0.126600  [ 8960/15078]
loss: 0.149917  [ 9600/15078]
loss: 0.205565  [10240/15078]
loss: 0.090581  [10880/15078]
loss: 0.134973  [11520/15078]
loss: 0.090043  [12160/15078]
loss: 0.107263  [12800/15078]
loss: 0.078955  [13440/15078]
loss: 0.083958  [14080/15078]
loss: 0.081450  [14720/15078]
Results:
 Accuracy: 96.0%, Avg loss: 0.115663

Epoch 8
-------------------------------
loss: 0.040010  [    0/15078]
loss: 0.072523  [  640/15078]
loss: 0.074546  [ 1280/15078]
loss: 0.091808  [ 1920/15078]
loss: 0.044569  [ 2560/15078]
loss: 0.096456  [ 3200/15078]
loss: 0.145287  [ 3840/15078]
loss: 0.234806  [ 4480/15078]
loss: 0.127469  [ 5120/15078]
loss: 0.104550  [ 5760/15078]
loss: 0.138099  [ 6400/15078]
loss: 0.100416  [ 7040/15078]
loss: 0.078979  [ 7680/15078]
loss: 0.073516  [ 8320/15078]
loss: 0.098078  [ 8960/15078]
loss: 0.050798  [ 9600/15078]
loss: 0.102734  [10240/15078]
loss: 0.085547  [10880/15078]
loss: 0.120930  [11520/15078]
loss: 0.084859  [12160/15078]
loss: 0.056501  [12800/15078]
loss: 0.079270  [13440/15078]
loss: 0.050432  [14080/15078]
loss: 0.116560  [14720/15078]
Results:
 Accuracy: 95.2%, Avg loss: 0.129575

Epoch 9
-------------------------------
loss: 0.113674  [    0/15078]
loss: 0.106667  [  640/15078]
loss: 0.064100  [ 1280/15078]
loss: 0.114960  [ 1920/15078]
loss: 0.180117  [ 2560/15078]
loss: 0.094108  [ 3200/15078]
```

```
loss: 0.156685  [ 3840/15078]
loss: 0.082430  [ 4480/15078]
loss: 0.104597  [ 5120/15078]
loss: 0.155683  [ 5760/15078]
loss: 0.130276  [ 6400/15078]
loss: 0.108323  [ 7040/15078]
loss: 0.088232  [ 7680/15078]
loss: 0.054648  [ 8320/15078]
loss: 0.086295  [ 8960/15078]
loss: 0.118343  [ 9600/15078]
loss: 0.054712  [10240/15078]
loss: 0.049820  [10880/15078]
loss: 0.050276  [11520/15078]
loss: 0.114396  [12160/15078]
loss: 0.088853  [12800/15078]
loss: 0.110971  [13440/15078]
loss: 0.121849  [14080/15078]
loss: 0.088440  [14720/15078]
Results:
 Accuracy: 95.9%, Avg loss: 0.117858

Epoch 10
-------------------------------
loss: 0.115144  [    0/15078]
loss: 0.099973  [  640/15078]
loss: 0.058744  [ 1280/15078]
loss: 0.062313  [ 1920/15078]
loss: 0.117635  [ 2560/15078]
loss: 0.054909  [ 3200/15078]
loss: 0.060918  [ 3840/15078]
loss: 0.076504  [ 4480/15078]
loss: 0.101324  [ 5120/15078]
loss: 0.065509  [ 5760/15078]
loss: 0.146410  [ 6400/15078]
loss: 0.064414  [ 7040/15078]
loss: 0.119174  [ 7680/15078]
loss: 0.123521  [ 8320/15078]
loss: 0.180212  [ 8960/15078]
loss: 0.197509  [ 9600/15078]
loss: 0.062888  [10240/15078]
loss: 0.072620  [10880/15078]
loss: 0.207759  [11520/15078]
loss: 0.073850  [12160/15078]
loss: 0.085429  [12800/15078]
loss: 0.090996  [13440/15078]
loss: 0.123145  [14080/15078]
loss: 0.071042  [14720/15078]
Results:
 Accuracy: 95.8%, Avg loss: 0.127133
```

In [295…

```python
# plot training loss and validation loss
plt.plot(np.arange(len(train_loss)), [i.item() for i in train_loss], 'r', label='train loss') # train in red
plt.plot(np.arange(len(val_loss)), val_loss, 'b', label='val loss')# val in blue
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Traing and validation loss')
plt.show()
```



Since the validation loss continues to improve, the training will continue.

In [296…

```python
# more epochs, saving best val

weights_state_base_fp = r'./results/upgrade/full_model_upgrade_'
optimizer_state_base_fp = r'./results/upgrade/full_optimizer_upgrade_'


epochs = 15
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")

    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=True)
    train_loss.append(train_l)

    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=True)
    val_loss.append(val_l)
    val_accur.append(val_a)

    epoch_counter += 1

    if val_a > best_val_acc:
        best_val_acc = val_a
        print('new best val acc of {} at {}'.format(best_val_acc, epoch_counter))
        out_fp = weights_state_base_fp + str(epoch_counter)
        torch.save(model.state_dict(), out_fp)
        optim_fp = optimizer_state_base_fp + str(epoch_counter)
        torch.save(optimizer.state_dict(), optim_fp)
```

```
Epoch 1
-------------------------------
loss: 0.048315  [    0/15078]
loss: 0.101188  [  640/15078]
loss: 0.087509  [ 1280/15078]
loss: 0.064532  [ 1920/15078]
loss: 0.055311  [ 2560/15078]
loss: 0.105524  [ 3200/15078]
loss: 0.058189  [ 3840/15078]
loss: 0.068133  [ 4480/15078]
loss: 0.129250  [ 5120/15078]
loss: 0.151189  [ 5760/15078]
loss: 0.080917  [ 6400/15078]
loss: 0.091961  [ 7040/15078]
loss: 0.087784  [ 7680/15078]
loss: 0.050792  [ 8320/15078]
loss: 0.048568  [ 8960/15078]
loss: 0.090291  [ 9600/15078]
loss: 0.116269  [10240/15078]
loss: 0.045887  [10880/15078]
loss: 0.066923  [11520/15078]
loss: 0.158926  [12160/15078]
loss: 0.126868  [12800/15078]
loss: 0.114654  [13440/15078]
loss: 0.139294  [14080/15078]
loss: 0.206911  [14720/15078]
Results:
 Accuracy: 94.6%, Avg loss: 0.154291

Epoch 2
-------------------------------
loss: 0.089915  [    0/15078]
loss: 0.062476  [  640/15078]
loss: 0.075853  [ 1280/15078]
loss: 0.084240  [ 1920/15078]
loss: 0.088036  [ 2560/15078]
loss: 0.055492  [ 3200/15078]
loss: 0.069664  [ 3840/15078]
loss: 0.112405  [ 4480/15078]
loss: 0.061097  [ 5120/15078]
loss: 0.092229  [ 5760/15078]
loss: 0.083542  [ 6400/15078]
loss: 0.132202  [ 7040/15078]
loss: 0.073000  [ 7680/15078]
loss: 0.113988  [ 8320/15078]
loss: 0.095810  [ 8960/15078]
loss: 0.063416  [ 9600/15078]
loss: 0.058423  [10240/15078]
loss: 0.114373  [10880/15078]
loss: 0.053636  [11520/15078]
loss: 0.124803  [12160/15078]
loss: 0.093123  [12800/15078]
loss: 0.098614  [13440/15078]
loss: 0.069072  [14080/15078]
loss: 0.128360  [14720/15078]
Results:
 Accuracy: 95.6%, Avg loss: 0.116227

Epoch 3
-------------------------------
loss: 0.063796  [    0/15078]
loss: 0.044874  [  640/15078]
loss: 0.060079  [ 1280/15078]
loss: 0.121586  [ 1920/15078]
loss: 0.074280  [ 2560/15078]
loss: 0.024888  [ 3200/15078]
loss: 0.042138  [ 3840/15078]
loss: 0.037539  [ 4480/15078]
loss: 0.067741  [ 5120/15078]
loss: 0.043757  [ 5760/15078]
loss: 0.047721  [ 6400/15078]
loss: 0.039706  [ 7040/15078]
loss: 0.043824  [ 7680/15078]
loss: 0.157402  [ 8320/15078]
loss: 0.055218  [ 8960/15078]
loss: 0.080359  [ 9600/15078]
loss: 0.030794  [10240/15078]
loss: 0.031361  [10880/15078]
loss: 0.158109  [11520/15078]
loss: 0.130090  [12160/15078]
loss: 0.062557  [12800/15078]
loss: 0.073987  [13440/15078]
loss: 0.144648  [14080/15078]
loss: 0.061928  [14720/15078]
Results:
 Accuracy: 95.1%, Avg loss: 0.141743

Epoch 4
-------------------------------
loss: 0.058048  [    0/15078]
loss: 0.049713  [  640/15078]
loss: 0.033328  [ 1280/15078]
loss: 0.038529  [ 1920/15078]
loss: 0.061960  [ 2560/15078]
loss: 0.081416  [ 3200/15078]
loss: 0.062308  [ 3840/15078]
loss: 0.086045  [ 4480/15078]
loss: 0.038257  [ 5120/15078]
loss: 0.061664  [ 5760/15078]
loss: 0.050502  [ 6400/15078]
loss: 0.042503  [ 7040/15078]
loss: 0.027753  [ 7680/15078]
loss: 0.102937  [ 8320/15078]
loss: 0.085449  [ 8960/15078]
loss: 0.115439  [ 9600/15078]
loss: 0.058115  [10240/15078]
loss: 0.047526  [10880/15078]
loss: 0.062909  [11520/15078]
loss: 0.140801  [12160/15078]
loss: 0.128469  [12800/15078]
loss: 0.100216  [13440/15078]
loss: 0.130129  [14080/15078]
loss: 0.124900  [14720/15078]
Results:
 Accuracy: 95.5%, Avg loss: 0.120330

Epoch 5
-------------------------------
loss: 0.069639  [    0/15078]
loss: 0.056030  [  640/15078]
```

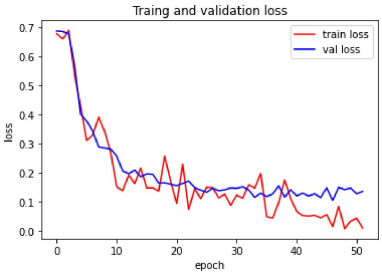
```
loss: 0.105341  [ 1280/15078]
loss: 0.087126  [ 1920/15078]
loss: 0.080636  [ 2560/15078]
loss: 0.083822  [ 3200/15078]
loss: 0.057141  [ 3840/15078]
loss: 0.107152  [ 4480/15078]
loss: 0.058339  [ 5120/15078]
loss: 0.100396  [ 5760/15078]
loss: 0.077925  [ 6400/15078]
loss: 0.057808  [ 7040/15078]
loss: 0.040496  [ 7680/15078]
loss: 0.124829  [ 8320/15078]
loss: 0.125073  [ 8960/15078]
loss: 0.081084  [ 9600/15078]
loss: 0.075319  [10240/15078]
loss: 0.086264  [10880/15078]
loss: 0.055595  [11520/15078]
loss: 0.097870  [12160/15078]
loss: 0.078582  [12800/15078]
loss: 0.089530  [13440/15078]
loss: 0.044427  [14080/15078]
loss: 0.116320  [14720/15078]
Results:
 Accuracy: 95.3%, Avg loss: 0.129567

Epoch 6
-------------------------------
loss: 0.038006  [    0/15078]
loss: 0.070799  [  640/15078]
loss: 0.053380  [ 1280/15078]
loss: 0.078688  [ 1920/15078]
loss: 0.124719  [ 2560/15078]
loss: 0.107938  [ 3200/15078]
loss: 0.094257  [ 3840/15078]
loss: 0.080192  [ 4480/15078]
loss: 0.085211  [ 5120/15078]
loss: 0.104200  [ 5760/15078]
loss: 0.081883  [ 6400/15078]
loss: 0.095795  [ 7040/15078]
loss: 0.092027  [ 7680/15078]
loss: 0.037855  [ 8320/15078]
loss: 0.058924  [ 8960/15078]
loss: 0.048283  [ 9600/15078]
loss: 0.120248  [10240/15078]
loss: 0.088377  [10880/15078]
loss: 0.076080  [11520/15078]
loss: 0.073045  [12160/15078]
loss: 0.059690  [12800/15078]
loss: 0.088788  [13440/15078]
loss: 0.027118  [14080/15078]
loss: 0.072097  [14720/15078]
Results:
 Accuracy: 95.8%, Avg loss: 0.119949

Epoch 7
-------------------------------
loss: 0.025944  [    0/15078]
loss: 0.086741  [  640/15078]
loss: 0.032111  [ 1280/15078]
loss: 0.130480  [ 1920/15078]
loss: 0.081630  [ 2560/15078]
loss: 0.028396  [ 3200/15078]
loss: 0.056995  [ 3840/15078]
loss: 0.063259  [ 4480/15078]
loss: 0.063301  [ 5120/15078]
loss: 0.057689  [ 5760/15078]
loss: 0.062788  [ 6400/15078]
loss: 0.039845  [ 7040/15078]
loss: 0.095671  [ 7680/15078]
loss: 0.046999  [ 8320/15078]
loss: 0.114953  [ 8960/15078]
loss: 0.121759  [ 9600/15078]
loss: 0.134922  [10240/15078]
loss: 0.018699  [10880/15078]
loss: 0.060158  [11520/15078]
loss: 0.065887  [12160/15078]
loss: 0.030172  [12800/15078]
loss: 0.045885  [13440/15078]
loss: 0.074546  [14080/15078]
loss: 0.165402  [14720/15078]
Results:
 Accuracy: 96.2%, Avg loss: 0.127809

new best val acc of 0.9616336633663366 at 44
Epoch 8
-------------------------------
loss: 0.029061  [    0/15078]
loss: 0.061649  [  640/15078]
loss: 0.076969  [ 1280/15078]
loss: 0.044876  [ 1920/15078]
loss: 0.145294  [ 2560/15078]
loss: 0.064219  [ 3200/15078]
loss: 0.049637  [ 3840/15078]
loss: 0.092491  [ 4480/15078]
loss: 0.124808  [ 5120/15078]
loss: 0.165505  [ 5760/15078]
loss: 0.051942  [ 6400/15078]
loss: 0.159914  [ 7040/15078]
loss: 0.083303  [ 7680/15078]
loss: 0.030631  [ 8320/15078]
loss: 0.063622  [ 8960/15078]
loss: 0.049304  [ 9600/15078]
loss: 0.046660  [10240/15078]
loss: 0.059022  [10880/15078]
loss: 0.063747  [11520/15078]
loss: 0.071197  [12160/15078]
loss: 0.080583  [12800/15078]
loss: 0.024927  [13440/15078]
loss: 0.043437  [14080/15078]
loss: 0.027700  [14720/15078]
Results:
 Accuracy: 96.8%, Avg loss: 0.114353

new best val acc of 0.9675123762376238 at 45
Epoch 9
-------------------------------
loss: 0.048102  [    0/15078]
loss: 0.027999  [  640/15078]
loss: 0.052347  [ 1280/15078]
loss: 0.070285  [ 1920/15078]
```

```
loss: 0.090946  [ 2560/15078]
loss: 0.041395  [ 3200/15078]
loss: 0.067101  [ 3840/15078]
loss: 0.070836  [ 4480/15078]
loss: 0.033499  [ 5120/15078]
loss: 0.060179  [ 5760/15078]
loss: 0.080537  [ 6400/15078]
loss: 0.049866  [ 7040/15078]
loss: 0.041908  [ 7680/15078]
loss: 0.033390  [ 8320/15078]
loss: 0.032740  [ 8960/15078]
loss: 0.108137  [ 9600/15078]
loss: 0.079860  [10240/15078]
loss: 0.060009  [10880/15078]
loss: 0.063121  [11520/15078]
loss: 0.121325  [12160/15078]
loss: 0.049285  [12800/15078]
loss: 0.040614  [13440/15078]
loss: 0.108943  [14080/15078]
loss: 0.074676  [14720/15078]
Results:
 Accuracy: 95.4%, Avg loss: 0.147673

Epoch 10
-------------------------------
loss: 0.042979  [    0/15078]
loss: 0.097515  [  640/15078]
loss: 0.034844  [ 1280/15078]
loss: 0.054227  [ 1920/15078]
loss: 0.048413  [ 2560/15078]
loss: 0.073148  [ 3200/15078]
loss: 0.019744  [ 3840/15078]
loss: 0.028945  [ 4480/15078]
loss: 0.068514  [ 5120/15078]
loss: 0.066426  [ 5760/15078]
loss: 0.077415  [ 6400/15078]
loss: 0.026774  [ 7040/15078]
loss: 0.060074  [ 7680/15078]
loss: 0.098643  [ 8320/15078]
loss: 0.014447  [ 8960/15078]
loss: 0.072266  [ 9600/15078]
loss: 0.283453  [10240/15078]
loss: 0.102702  [10880/15078]
loss: 0.060799  [11520/15078]
loss: 0.057245  [12160/15078]
loss: 0.034214  [12800/15078]
loss: 0.056850  [13440/15078]
loss: 0.025153  [14080/15078]
loss: 0.043490  [14720/15078]
Results:
 Accuracy: 97.1%, Avg loss: 0.105066

new best val acc of 0.9706064356435643 at 47
Epoch 11
-------------------------------
loss: 0.026685  [    0/15078]
loss: 0.015421  [  640/15078]
loss: 0.029668  [ 1280/15078]
loss: 0.015718  [ 1920/15078]
loss: 0.031593  [ 2560/15078]
loss: 0.066105  [ 3200/15078]
loss: 0.053149  [ 3840/15078]
loss: 0.061841  [ 4480/15078]
loss: 0.067080  [ 5120/15078]
loss: 0.050388  [ 5760/15078]
loss: 0.031266  [ 6400/15078]
loss: 0.013416  [ 7040/15078]
loss: 0.032130  [ 7680/15078]
loss: 0.030170  [ 8320/15078]
loss: 0.022381  [ 8960/15078]
loss: 0.053913  [ 9600/15078]
loss: 0.031240  [10240/15078]
loss: 0.030134  [10880/15078]
loss: 0.024967  [11520/15078]
loss: 0.031484  [12160/15078]
loss: 0.010217  [12800/15078]
loss: 0.037803  [13440/15078]
loss: 0.068215  [14080/15078]
loss: 0.028727  [14720/15078]
Results:
 Accuracy: 94.9%, Avg loss: 0.149412

Epoch 12
-------------------------------
loss: 0.079077  [    0/15078]
loss: 0.041711  [  640/15078]
loss: 0.082131  [ 1280/15078]
loss: 0.013174  [ 1920/15078]
loss: 0.021233  [ 2560/15078]
loss: 0.006386  [ 3200/15078]
loss: 0.032977  [ 3840/15078]
loss: 0.045894  [ 4480/15078]
loss: 0.049782  [ 5120/15078]
loss: 0.032868  [ 5760/15078]
loss: 0.056738  [ 6400/15078]
loss: 0.039937  [ 7040/15078]
loss: 0.027667  [ 7680/15078]
loss: 0.077451  [ 8320/15078]
loss: 0.076166  [ 8960/15078]
loss: 0.107705  [ 9600/15078]
loss: 0.047204  [10240/15078]
loss: 0.043933  [10880/15078]
loss: 0.035443  [11520/15078]
loss: 0.044808  [12160/15078]
loss: 0.032745  [12800/15078]
loss: 0.007295  [13440/15078]
loss: 0.024925  [14080/15078]
loss: 0.039272  [14720/15078]
Results:
 Accuracy: 96.1%, Avg loss: 0.141784

Epoch 13
-------------------------------
loss: 0.050432  [    0/15078]
loss: 0.054637  [  640/15078]
loss: 0.028124  [ 1280/15078]
loss: 0.058941  [ 1920/15078]
loss: 0.059802  [ 2560/15078]
loss: 0.109020  [ 3200/15078]
loss: 0.012396  [ 3840/15078]
```

```
loss: 0.054739  [ 4480/15078]
loss: 0.043103  [ 5120/15078]
loss: 0.028448  [ 5760/15078]
loss: 0.047524  [ 6400/15078]
loss: 0.084778  [ 7040/15078]
loss: 0.041916  [ 7680/15078]
loss: 0.095343  [ 8320/15078]
loss: 0.038379  [ 8960/15078]
loss: 0.044348  [ 9600/15078]
loss: 0.042953  [10240/15078]
loss: 0.009743  [10880/15078]
loss: 0.040310  [11520/15078]
loss: 0.028937  [12160/15078]
loss: 0.068115  [12800/15078]
loss: 0.015486  [13440/15078]
loss: 0.012303  [14080/15078]
loss: 0.068129  [14720/15078]
Results:
 Accuracy: 96.5%, Avg loss: 0.147016

Epoch 14
-------------------------------
loss: 0.035996  [    0/15078]
loss: 0.007922  [  640/15078]
loss: 0.083442  [ 1280/15078]
loss: 0.030388  [ 1920/15078]
loss: 0.019921  [ 2560/15078]
loss: 0.024716  [ 3200/15078]
loss: 0.050805  [ 3840/15078]
loss: 0.055838  [ 4480/15078]
loss: 0.011829  [ 5120/15078]
loss: 0.044141  [ 5760/15078]
loss: 0.034400  [ 6400/15078]
loss: 0.062354  [ 7040/15078]
loss: 0.068149  [ 7680/15078]
loss: 0.028177  [ 8320/15078]
loss: 0.016519  [ 8960/15078]
loss: 0.013345  [ 9600/15078]
loss: 0.072115  [10240/15078]
loss: 0.113281  [10880/15078]
loss: 0.170508  [11520/15078]
loss: 0.102226  [12160/15078]
loss: 0.081075  [12800/15078]
loss: 0.082210  [13440/15078]
loss: 0.044553  [14080/15078]
loss: 0.076758  [14720/15078]
Results:
 Accuracy: 95.6%, Avg loss: 0.127591

Epoch 15
-------------------------------
loss: 0.085346  [    0/15078]
loss: 0.021051  [  640/15078]
loss: 0.064314  [ 1280/15078]
loss: 0.032986  [ 1920/15078]
loss: 0.024962  [ 2560/15078]
loss: 0.042023  [ 3200/15078]
loss: 0.052230  [ 3840/15078]
loss: 0.028574  [ 4480/15078]
loss: 0.017218  [ 5120/15078]
loss: 0.030924  [ 5760/15078]
loss: 0.031214  [ 6400/15078]
loss: 0.021400  [ 7040/15078]
loss: 0.039092  [ 7680/15078]
loss: 0.041992  [ 8320/15078]
loss: 0.049546  [ 8960/15078]
loss: 0.023910  [ 9600/15078]
loss: 0.025057  [10240/15078]
loss: 0.084954  [10880/15078]
loss: 0.040836  [11520/15078]
loss: 0.054152  [12160/15078]
loss: 0.060119  [12800/15078]
loss: 0.061740  [13440/15078]
loss: 0.051553  [14080/15078]
loss: 0.024696  [14720/15078]
Results:
 Accuracy: 96.0%, Avg loss: 0.135697
```

In [297…

```
# plot training loss and validation loss
plt.plot(np.arange(len(train_loss)), [i.item() for i in train_loss], 'r', label='train loss') # train in red
plt.plot(np.arange(len(val_loss)), val_loss, 'b', label='val loss')# val in blue
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Traing and validation loss')
plt.show()
```



We see validation loss plateau, and training loss diverge from validation loss around epoch 40. We get our best validation loss and accuracy at epoch 47. Shortly afterwards, validation loss starts to actually climb, a definite sign of overfitting.

In [307…

```
best_model = CNN(
    channels=(8, 16, 20, 24, 24),
    kernel_sizes=(10, 10, 10, 12, 12),
    linear_units=(1000, 250, 10),
)

best_weights_fp = r'C:\Users\Andrew\Documents\2022 Summer\Data Mining\Project\results\upgrade\full_model_upgrade_47'
best_model.load_state_dict(torch.load(best_weights_fp))
```

Out[307]:

```
<All keys matched successfully>
```

In [308…
```
# get val predictions and true labels for a classification report
preds = []
y_true = []

best_model.eval()
with torch.no_grad():
    for X, y in val_dataloader:
        pred = best_model(X.float())
        preds.append(pred)
        y_true.append(y)

y_pred = np.concatenate(preds).argmax(1)
y_true = np.concatenate(y_true)

report = classification_report(y_true=y_true, y_pred=y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1444
           1       0.98      0.97      0.97      1788

    accuracy                           0.97      3232
   macro avg       0.97      0.97      0.97      3232
weighted avg       0.97      0.97      0.97      3232
```

In [309…
```
# Save FULL model
# different than just weights
# https://pytorch.org/tutorials/beginner/saving_loading_models.html

# model out path
#model_out_path = r'C:\Users\Andrew\Documents\2022 Summer\Data Mining\Project\results\upgrade\full_model_best'
#torch.save(best_model, model_out_path)
```

With the current parameters, we get an accuracy of 97% on the validation data and an f1 score of 0.97.

Optimizer param sweep

This is only viable for really quickly trained models. The current model took over a day to train and scored high accuracy, so this parameter sweep does not meet a cost/benefit threshold.

In [298…
```
"""
# Create the dataloaders
batch_size = 128

train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)

val_dataloader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False
)

test_dataloader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False
)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

lrs = [.005, .01]
decays = [.0001, .0005]
channels = [(16, 24, 32), (8, 16, 32), (8, 16, 24, 32)]
linear_units = [ (100, 10), (100, 50, 10)]

lr_list = []
deacy_list = []
max_val_acc_list = []
at_epoch_list = []
channel_list = []
linear_units_list = []

counter = 0
num_runs = len(lrs) * len(decays) * len(channels) * len(linear_units)

for lr in lrs:
    for decay in decays:
        for channel in channels:
            for linear_unit in linear_units:

                max_val_acc = 0
                best_epoch = 0

                # Create CNN
                model = CNN(
                    channels=channel,
                    #kernel_sizes=kernel_size,
                    linear_units=linear_unit,
                )

                # use cross entropy loss
                loss_fn = nn.CrossEntropyLoss()

                # SGD optimizer
                optimizer = torch.optim.SGD(
                    model.parameters(),
                    lr=lr,
                    momentum=0.9,
                    #nesterov =True
                    weight_decay=decay
                )
                #optimizer = torch.optim.Adam(model.parameters(), lr=0.003, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)


                # record results
                train_loss = []
                val_loss = []
                val_accur = []

                epochs = 25
                for t in range(epochs):
                    #print(f"Epoch {t+1}\n-------------------------------")
```

```
                    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=False)
                    train_loss.append(train_l)

                    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=False)
                    val_loss.append(val_l)
                    val_accur.append(val_a)

                    if val_a > max_val_acc:
                        max_val_acc = val_a
                        best_epoch = t


                lr_list.append(lr)
                deacy_list.append(decay)
                max_val_acc_list.append(max_val_acc)
                at_epoch_list.append(best_epoch)
                channel_list.append(channel)
                linear_units_list.append(linear_unit)

                counter += 1
                print('finished {} of {}'.format(counter, num_runs))
            print('finished linear unit {}'.format(linear_unit))
        print('finished channels set')
    print('finished lr {}'.format(lr))

"""
print()
```

In [299…
```
"""
results = pd.DataFrame(
    data={
        'lr': lr_list,
        'decay': deacy_list,
        'val acc': max_val_acc_list,
        'at epoch': at_epoch_list,
        'channels': channel_list,
        'linear units': linear_units_list
    }
)
"""
print()
```

In [300…
```
#results
```

In [301…
```
#results.sort_values(by='val acc', ascending=False)
```

In [302…
```
#results.to_pickle('./results/default_model_full_hyperparam_results.pickle')
```

In [303…
```
# load results
#results_from_file = pd.read_pickle('./results/default_model_full_hyperparam_results.pickle')
#sorted_results = results_from_file.sort_values(by='val acc', ascending=False)
#sorted_results.head(7)
```

This sweep tells us that a lr of .010 and a decay of .0001 gives the best validation accuracy of 98% at epoch 23. The channels are 8, 16, 32 and the linear units are 100, 10.

Training an optimized model

In [304…
```
## Since no parameter sweep, the current model is the best model so far.

"""
# Create the dataloaders
batch_size = 128

train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)

val_dataloader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False
)

test_dataloader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False
)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")



max_val_acc = 0
best_epoch = 0

# Create CNN Model
model = CNN(
    channels = (8, 16, 32),
    #kernel_sizes = (10, 10, 10, 10),
    linear_units = (100, 10),
)

# use cross entropy loss
loss_fn = nn.CrossEntropyLoss()

# SGD optimizer
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=.010,
    momentum=0.9,
    #nesterov =True
    weight_decay=.0001
)
#optimizer = torch.optim.Adam(model.parameters(), lr=0.003, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)


# record results
train_loss = []
val_loss = []
```

```
val_accur = []


epochs = 23
for t in range(epochs):
    #print(f"Epoch {t+1}\n-------------------------------")

    train_l = train(train_dataloader, model, loss_fn, optimizer, verbose=False)
    train_loss.append(train_l)

    val_a, val_l = test(val_dataloader, model, loss_fn, verbose=False)
    val_loss.append(val_l)
    val_accur.append(val_a)

    if val_a > max_val_acc:
        max_val_acc = val_a
        best_epoch = t
"""
print()
```

In [305…
```
# plot training loss and validation loss

#plt.plot(np.arange(len(train_loss)), [i.item() for i in train_loss], 'r', label='train loss') # train in red
#plt.plot(np.arange(len(val_loss)), val_loss, 'b', label='val loss')# val in blue
#plt.xlabel('epoch')
#plt.ylabel('loss')
#plt.legend()
#plt.title('Traing and validation loss for 1 hidden layer size 64')
#plt.show()
```

Performance on Test Data

In [310…
```
# get val predictions and true labels for a classification report on Test Data

preds = []
y_true = []

best_model.eval()
with torch.no_grad():
    for X, y in test_dataloader:
        pred = best_model(X.float())
        preds.append(pred)
        y_true.append(y)

y_pred = np.concatenate(preds).argmax(1)
y_true = np.concatenate(y_true)

report = classification_report(y_true=y_true, y_pred=y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      1404
           1       0.96      0.95      0.96      1828

    accuracy                           0.95      3232
   macro avg       0.95      0.95      0.95      3232
weighted avg       0.95      0.95      0.95      3232
```

In [311…
```
val_a, val_l = test(test_dataloader, best_model, loss_fn, verbose=False)
print('Test accuracy: {:.2%}'.format(val_a))
```

```
Test accuracy: 95.17%
```

The variance in this value is due to the stochastic nature of pytorch in how I've implemented the model. This result deviated from results in the sweep by just a little, which is no cause for alarm.

Saving the model weights

To reuse the model, we will save the weights. Pytorch offers a very easy way to save model weights. The model itself will be placed into a python file so it can be imported.

In [40]:
```
#  CHANGE NAME TO NOT OVERWRITE
#weights_fp = './results/torch_full_model_weights2'
#torch.save(model.state_dict(), weights_fp)
```

In [ ]: