Exercise Sheet 6 - Regularization 2

In this exercise we will take a look at simple but effective methods used to combat overfitting when learning neural networks. We will use a very small subset of the FashionMNIST dataset to artificially induce overfitting, train and evaluate our model. Finally we will look at how to incorporate early stopping and how adding noise to our data makes our model more robust.

In [142…

```
from typing import Callable
from functools import partial

import torch
from torch import nn
from torch.optim import Optimizer, SGD

from torchvision.transforms import v2

#from solution import *
from utils import *

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
DEVICE
```

Out[142…

```
'cuda'
```

Training configuration

Throughout this notebook we will use the following training configuration:

In [144…

```
TRAIN_SET_SIZE = 200
VAL_SET_SIZE = 1000
EPOCHS = 500
HIDDEN_DIMS = [64, 32]
LR = 0.05
BATCH_SIZE = 32
```

Dataset (FashionMNIST)

As mentioned before, we will use the FashionMNIST dataset. This dataset behaves exactly the same as the standard MNIST dataset (grayscale images with height and width of 28 pixels, 10 classes (0-9), train set with 60k samples and test set with 10k samples) with the only difference being the depicted images. While MNIST shows handwritten digits, FashionMNIST shows 10 different types of clothing. Example images are shown after execution of the following cell.

In [146…

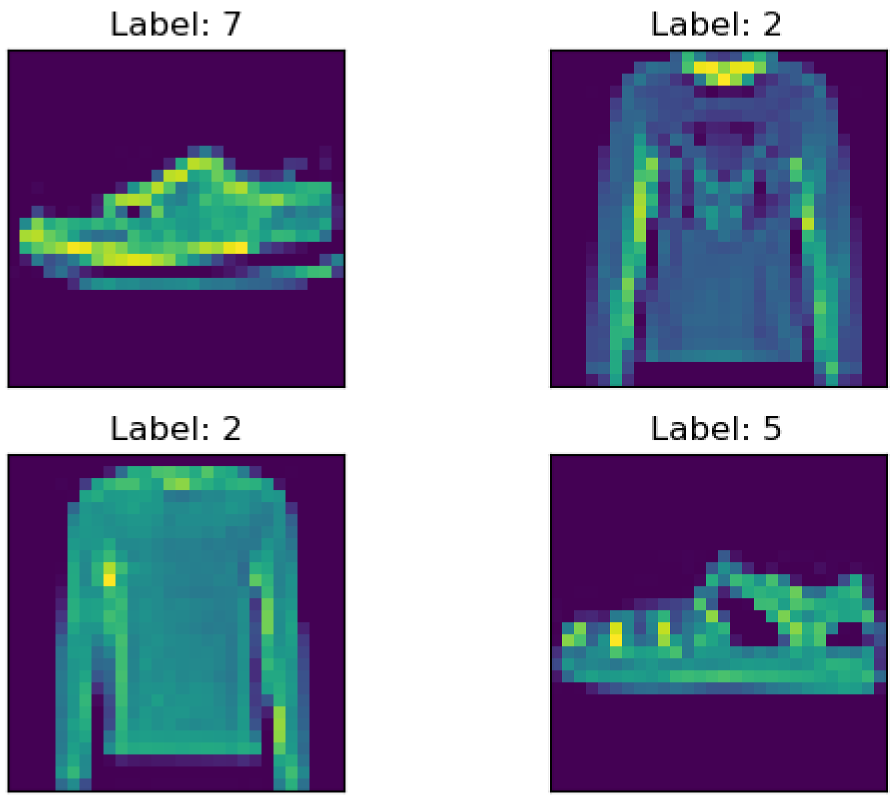
```
train_set, val_set, test_set = get_fashion_mnist_subset(TRAIN_SET_SIZE, VAL_SET_

train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)
```

```
val_loader = DataLoader(val_set, batch_size=BATCH_SIZE)
test_loader = DataLoader(test_set, batch_size=BATCH_SIZE)

visualize_first_4(train_loader)
```

```
Shape of images is torch.Size([32, 1, 28, 28])
```



Create model

First, we need a function to create a model that is able to classify FashionMNIST data. The model takes in inputs of the shape (batch_size x 1 x 28 x 28) and outputs a 10-dimensional vector. It should first flatten the input images, then apply a given number of linear layers to it, and finally map to a 10-dimensional vector which will be used to predict which type of clothing it is.

Task 1a) (20 P)

Complete the missing code in the following function. The function takes a list of hidden dimensions which correspond to the dimensionality of the linear layers. The input dimension of 28 x 28 and the output dimension of 10 should be hardcoded into the model. There should be as many layers as hidden dimensions, plus a final output layer. As an activation function we will use the ReLU activation.

In [149…

```
def create_model(hidden_dims: list[int]):  # TODO: explain in docstring that inp
    """Create a model that works for classifying the FahsionMNIST dataset."""
    layers = []
    input_dim = 28 * 28
    for dim in hidden_dims:
        layers.append(nn.Linear(input_dim, dim))
        layers.append(nn.ReLU())
        input_dim = dim
    layers.append(nn.Linear(input_dim, 10))
    model = nn.Sequential(*layers)
    model.to(DEVICE)
    return model
```

Train, evaluate and save models

Task 2a) (20 P)

Write a function that trains a model for one epoch / one iteration through the train dataset. Make sure to zero the gradients before each step and to apply the optimizers functions to train the model. This is a repetition from last exercise, so check there in case you are unsure.

In [152…

```
def train_one_epoch(model, dataloader, optimizer):
    model.train()
    total_loss = 0
    for inputs, labels in dataloader:
        inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)
        optimizer.zero_grad()
        outputs = model(inputs.view(inputs.size(0), -1))
        loss = nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        avg_loss = total_loss / len(dataloader)
    return avg_loss
```

Task 2b) (20 P)

Write a function that iterates through a dataloader, and outputs the average loss and accuracy. Make sure that no gradient computation is triggered, e.g. use torch.no_grad(). Furthermore, make sure that your model in evaluation mode and to switch back afterwards. This will be important for the last task where we are adding Dropout layers to our neural network. For more information see this StackOverflow question.

In [154…

```
def validate(model: nn.Module, dataloader: DataLoader) -> tuple[float, float]:
    model.eval()
    total_loss = 0
    correct = 0
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)
            outputs = model(inputs.view(inputs.size(0), -1))
            loss = nn.CrossEntropyLoss()(outputs, labels)
            total_loss += loss.item()
            preds = outputs.argmax(dim=1)
            correct += (preds == labels).sum().item()
    avg_loss = total_loss / len(dataloader)
    avg_accuracy = correct / len(dataloader.dataset)
    return avg_loss, avg_accuracy
```

Here we define our complete training function. It simply iterates for `n_epochs` epochs through the training dataset, evaluates after each epoch on the validation dataset, and finally returns an array with the train and validation losses for each epoch. An important feature of this training function is that it can take a function as an argument (that's the `callback` argument) which gets the model, the current epoch, the array of train losses up until this point, the array of validation losses until this point and the last validation accuracy. The follwing tasks will partly consist of writing functions that we will pass into the training function.

In [156…

```
def train(
        model: nn.Module,
        train_loader: DataLoader,
        val_loader: DataLoader,
        optimizer: Optimizer,
        n_epochs: int,
        callback: Callable[[nn.Module, int, list[float], list[float], float], No
    ) -> tuple[list[float], list[float]]:
    train_losses = []
    val_losses = []
    for epoch in range(n_epochs):
        train_loss = train_one_epoch(model, train_loader, optimizer)
        val_loss, val_acc = validate(model, val_loader)

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

        callback(model, epoch, train_losses, val_losses, val_acc)

    return train_losses, val_losses
```

Task 2c) (10 P)

In this task you should simply write a function that we can pass into the training function above that prints the current stats every `n` epochs. This function shouldn't return anything.

In [158…

```
def print_loss_every_n_epochs(
        model: nn.Module,
        epoch: int,
        train_losses: list[float],
        val_losses: list[float],
        val_acc,
        n: int) -> None:
    if epoch % n == 0:
        print(f"[EPOCH {epoch}] Train Loss: {train_losses[-1]:.4f}, "
              f"Validation Loss: {val_losses[-1]:.4f}, Validation Accuracy: {val
```

Next we will train a model. We will use the above defined hyperparameters, and a simple SGD optimizer. Furthermore we will print the training stats every epoch.

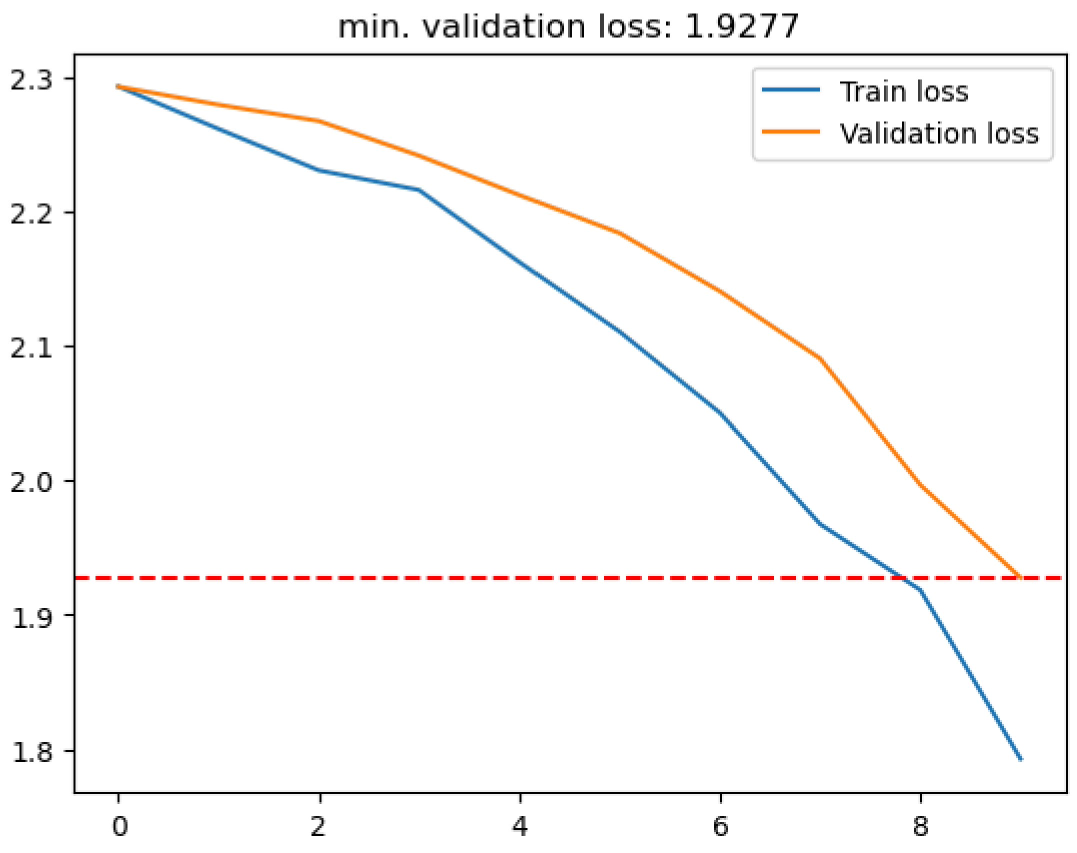
In [160…

```
model = create_model(HIDDEN_DIMS)
optimizer = SGD(model.parameters(), lr=LR)

train_losses , val_losses = train(model,
                                   train_loader,
                                   val_loader,
                                   optimizer,
                                   n_epochs=10,
                                   callback=partial(print_loss_every_n_epochs, n

plot_train_and_val_loss(train_losses, val_losses, title=f"min. validation loss:
```

```
[EPOCH 0] Train Loss: 2.2926, Validation Loss: 2.2925, Validation Accuracy: 0.134
0
[EPOCH 1] Train Loss: 2.2610, Validation Loss: 2.2791, Validation Accuracy: 0.164
0
[EPOCH 2] Train Loss: 2.2303, Validation Loss: 2.2670, Validation Accuracy: 0.189
0
[EPOCH 3] Train Loss: 2.2156, Validation Loss: 2.2412, Validation Accuracy: 0.148
0
[EPOCH 4] Train Loss: 2.1621, Validation Loss: 2.2119, Validation Accuracy: 0.107
0
[EPOCH 5] Train Loss: 2.1101, Validation Loss: 2.1835, Validation Accuracy: 0.117
0
[EPOCH 6] Train Loss: 2.0499, Validation Loss: 2.1403, Validation Accuracy: 0.179
0
[EPOCH 7] Train Loss: 1.9674, Validation Loss: 2.0903, Validation Accuracy: 0.167
0
[EPOCH 8] Train Loss: 1.9185, Validation Loss: 1.9966, Validation Accuracy: 0.287
0
[EPOCH 9] Train Loss: 1.7928, Validation Loss: 1.9277, Validation Accuracy: 0.393
0
```



Early stopping

Early stopping is the most simple thing to prevent your final model to overfit: you simply track train and test errors and use a model checkpoint before your model started to overfit.

Task 3a) (10 P)

Implementing early stopping is pretty straight forward. Simply save your model each time you've reached a new best validation loss, and otherwise don't. To reduce clutter in your filesystem you can simply override the saved model each time. Saving and loading PyTorch models is described in this guide. The function to load models is given.

In [163…

```
def save_model_if_improved(model, epoch, train_losses, val_losses, val_acc, file
    min_loss = min(val_losses)
    if val_losses[-1] <= min_loss:
        print(f"Found new best model at epoch {epoch} with a validation loss of
        torch.save(model.state_dict(), filename)

def load_model(model, model_file_path: str) -> nn.Module:
    model.load_state_dict(torch.load(model_file_path, weights_only=True))
    return model
```

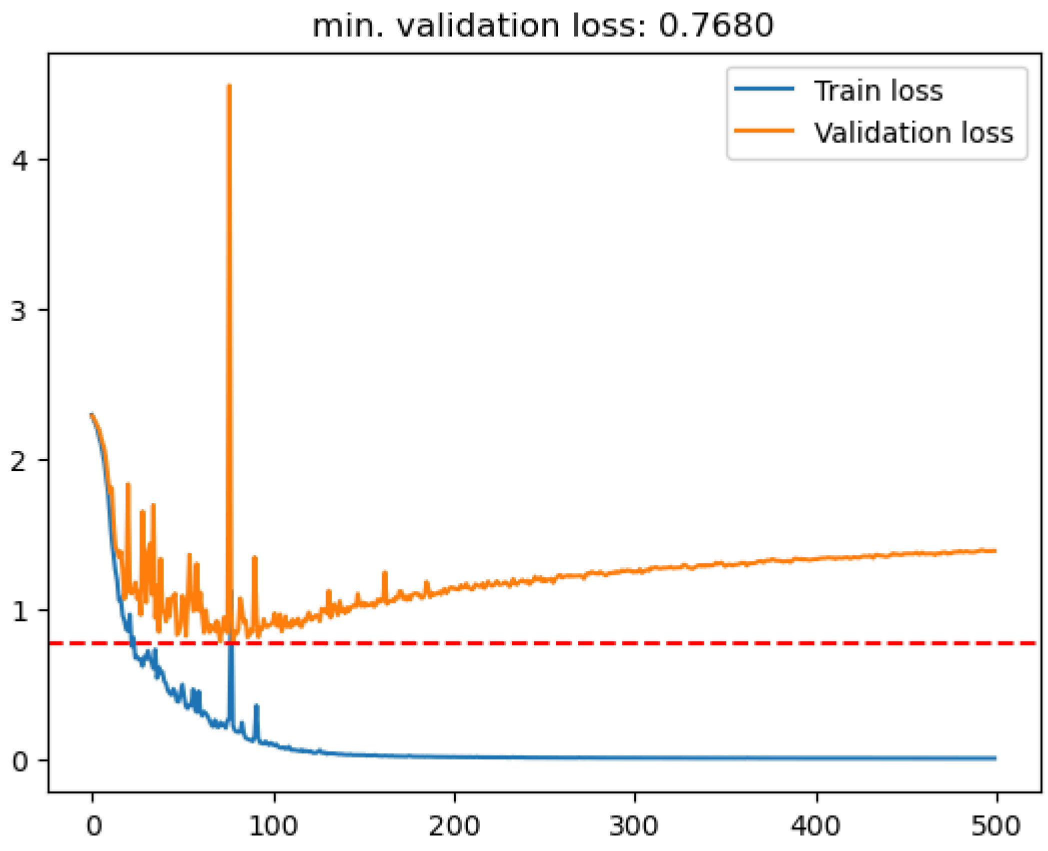
This time we will train our model for more epochs, to show how it overfits. Your implemented function should stop saving the model once the model starts to overfit.

In [165…

```
model = create_model(HIDDEN_DIMS)
optimizer = SGD(model.parameters(), lr=LR)
train_losses, val_losses = train(model,
                                 train_loader,
                                 val_loader,
                                 optimizer,
                                 n_epochs=EPOCHS,
                                 callback=partial(save_model_if_improved, filenam

plot_train_and_val_loss(train_losses, val_losses, title=f"min. validation loss:
```

```
Found new best model at epoch 0 with a validation loss of 2.2836
Found new best model at epoch 1 with a validation loss of 2.2664
Found new best model at epoch 2 with a validation loss of 2.2447
Found new best model at epoch 3 with a validation loss of 2.2162
Found new best model at epoch 4 with a validation loss of 2.1935
Found new best model at epoch 5 with a validation loss of 2.1425
Found new best model at epoch 6 with a validation loss of 2.0980
Found new best model at epoch 7 with a validation loss of 2.0568
Found new best model at epoch 8 with a validation loss of 1.9706
Found new best model at epoch 9 with a validation loss of 1.8576
Found new best model at epoch 10 with a validation loss of 1.7702
Found new best model at epoch 12 with a validation loss of 1.5655
Found new best model at epoch 13 with a validation loss of 1.4130
Found new best model at epoch 14 with a validation loss of 1.3849
Found new best model at epoch 15 with a validation loss of 1.3422
Found new best model at epoch 17 with a validation loss of 1.2389
Found new best model at epoch 18 with a validation loss of 1.0645
Found new best model at epoch 27 with a validation loss of 0.9585
Found new best model at epoch 35 with a validation loss of 0.9378
Found new best model at epoch 37 with a validation loss of 0.8455
Found new best model at epoch 47 with a validation loss of 0.8259
Found new best model at epoch 52 with a validation loss of 0.8178
Found new best model at epoch 71 with a validation loss of 0.7795
Found new best model at epoch 78 with a validation loss of 0.7680
```



We now evaluate our model on the test set, to get a better estimate of the generalization error. As you can see we've named the model

In [167…

```python
pretrained_model = create_model(HIDDEN_DIMS)
load_model(pretrained_model, model_file_path="early_stopping.pth")
test_loss, test_accuracy = validate(pretrained_model, test_loader)

print(f"Early stopping model achieved test loss of {test_loss:.4f} and accuarcy
Early stopping model achieved test loss of 0.8319 and accuarcy 0.7292
```

Dropout

Dropout is a technique where neurons are randoml set to zero, which introduces noise into the network, and helps to generalize better. Details are described in this blogpost.

Task 4a) (20 P)

Implement a function similar to the one above where we build a standard feed forward network. But now, after each activation layer, add a nn.Dropout() layer with the dropout probability specified in the parameter `p`.

In [170…

```python
def create_model_with_dropout(hidden_dims: list[int], p: float):
    layers = []
    input_dim = 28 * 28
    for dim in hidden_dims:
        layers.append(nn.Linear(input_dim, dim))
        layers.append(nn.ReLU())
        layers.append(nn.Dropout(p))
        input_dim = dim
    layers.append(nn.Linear(input_dim, 10))
    model = nn.Sequential(*layers)
    model.to(DEVICE)
    return model
```

Next we will train a neural network with the exact same parameters as above, only that we now have added dropout layers.

In [172…

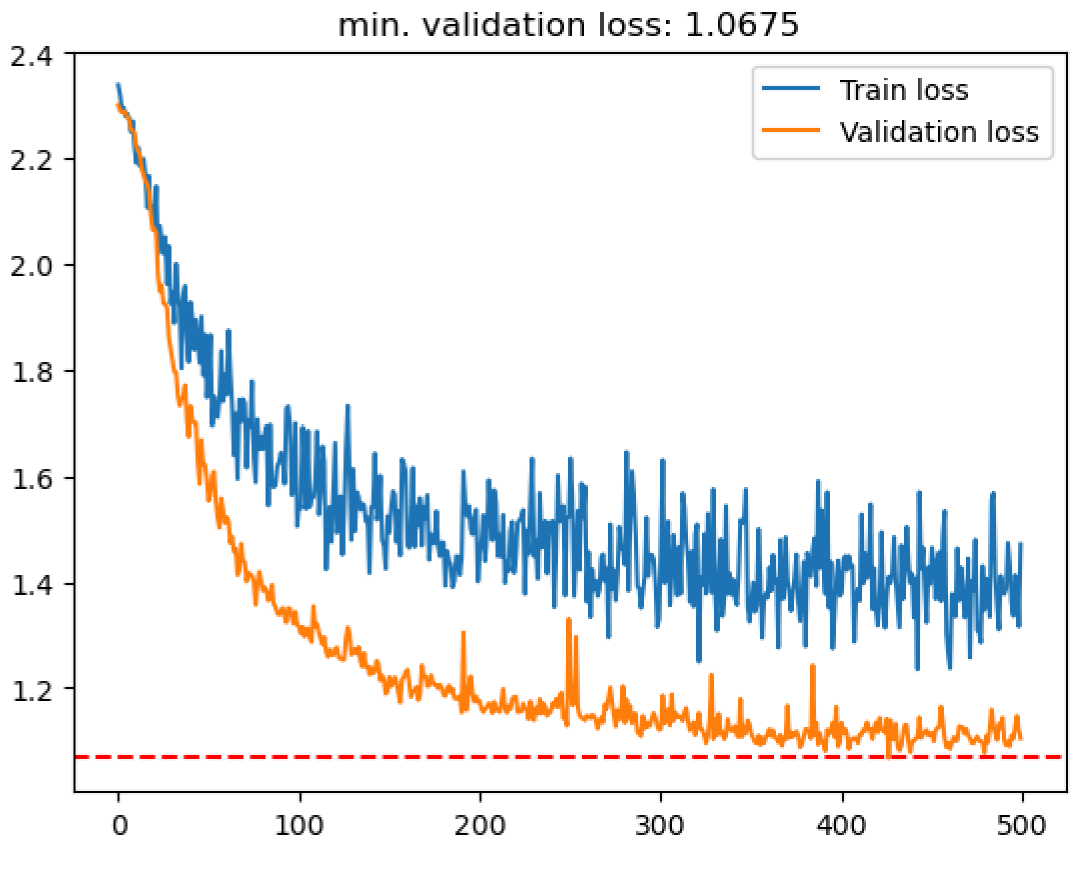
```python
DROPOUT = 0.8

model = create_model_with_dropout(HIDDEN_DIMS, p=DROPOUT)
optimizer = SGD(model.parameters(), lr=LR)

train_losses, val_losses = train(model,
                                 train_loader,
                                 val_loader,
                                 optimizer,
                                 n_epochs=EPOCHS,
                                 callback=partial(save_model_if_improved, filenam

plot_train_and_val_loss(train_losses, val_losses, title=f"min. validation loss:
```

```
Found new best model at epoch 0 with a validation loss of 2.3010
Found new best model at epoch 1 with a validation loss of 2.2905
Found new best model at epoch 2 with a validation loss of 2.2873
Found new best model at epoch 4 with a validation loss of 2.2864
Found new best model at epoch 5 with a validation loss of 2.2789
Found new best model at epoch 6 with a validation loss of 2.2731
Found new best model at epoch 7 with a validation loss of 2.2548
Found new best model at epoch 8 with a validation loss of 2.2527
Found new best model at epoch 9 with a validation loss of 2.2489
Found new best model at epoch 10 with a validation loss of 2.2216
Found new best model at epoch 11 with a validation loss of 2.2145
Found new best model at epoch 12 with a validation loss of 2.2057
Found new best model at epoch 13 with a validation loss of 2.1794
Found new best model at epoch 14 with a validation loss of 2.1693
Found new best model at epoch 15 with a validation loss of 2.1565
Found new best model at epoch 16 with a validation loss of 2.1520
Found new best model at epoch 17 with a validation loss of 2.1384
Found new best model at epoch 18 with a validation loss of 2.0951
Found new best model at epoch 19 with a validation loss of 2.0654
Found new best model at epoch 21 with a validation loss of 2.0594
Found new best model at epoch 22 with a validation loss of 1.9780
Found new best model at epoch 23 with a validation loss of 1.9488
Found new best model at epoch 25 with a validation loss of 1.9258
Found new best model at epoch 26 with a validation loss of 1.9244
Found new best model at epoch 27 with a validation loss of 1.9164
Found new best model at epoch 28 with a validation loss of 1.8605
Found new best model at epoch 29 with a validation loss of 1.8394
Found new best model at epoch 30 with a validation loss of 1.8206
Found new best model at epoch 31 with a validation loss of 1.7964
Found new best model at epoch 32 with a validation loss of 1.7949
Found new best model at epoch 33 with a validation loss of 1.7504
Found new best model at epoch 34 with a validation loss of 1.7322
Found new best model at epoch 38 with a validation loss of 1.7045
Found new best model at epoch 39 with a validation loss of 1.6750
Found new best model at epoch 44 with a validation loss of 1.6281
Found new best model at epoch 45 with a validation loss of 1.5860
Found new best model at epoch 50 with a validation loss of 1.5538
Found new best model at epoch 55 with a validation loss of 1.5217
Found new best model at epoch 56 with a validation loss of 1.5033
Found new best model at epoch 62 with a validation loss of 1.4746
Found new best model at epoch 64 with a validation loss of 1.4581
Found new best model at epoch 66 with a validation loss of 1.4138
Found new best model at epoch 71 with a validation loss of 1.4015
Found new best model at epoch 75 with a validation loss of 1.4009
Found new best model at epoch 76 with a validation loss of 1.3573
Found new best model at epoch 87 with a validation loss of 1.3528
Found new best model at epoch 88 with a validation loss of 1.3417
Found new best model at epoch 89 with a validation loss of 1.3385
Found new best model at epoch 92 with a validation loss of 1.3357
Found new best model at epoch 93 with a validation loss of 1.3240
Found new best model at epoch 97 with a validation loss of 1.3214
Found new best model at epoch 99 with a validation loss of 1.3193
Found new best model at epoch 100 with a validation loss of 1.3155
Found new best model at epoch 101 with a validation loss of 1.3033
Found new best model at epoch 103 with a validation loss of 1.2972
Found new best model at epoch 106 with a validation loss of 1.2943
Found new best model at epoch 107 with a validation loss of 1.2868
Found new best model at epoch 113 with a validation loss of 1.2824
Found new best model at epoch 115 with a validation loss of 1.2674
Found new best model at epoch 116 with a validation loss of 1.2592
Found new best model at epoch 122 with a validation loss of 1.2570
Found new best model at epoch 123 with a validation loss of 1.2561
Found new best model at epoch 124 with a validation loss of 1.2533
Found new best model at epoch 134 with a validation loss of 1.2504
Found new best model at epoch 135 with a validation loss of 1.2403
Found new best model at epoch 137 with a validation loss of 1.2402
Found new best model at epoch 139 with a validation loss of 1.2253
Found new best model at epoch 145 with a validation loss of 1.2141
Found new best model at epoch 148 with a validation loss of 1.1888
Found new best model at epoch 156 with a validation loss of 1.1725
Found new best model at epoch 190 with a validation loss of 1.1535
Found new best model at epoch 207 with a validation loss of 1.1522
Found new best model at epoch 227 with a validation loss of 1.1408
Found new best model at epoch 230 with a validation loss of 1.1389
Found new best model at epoch 248 with a validation loss of 1.1287
Found new best model at epoch 265 with a validation loss of 1.1231
Found new best model at epoch 287 with a validation loss of 1.1154
Found new best model at epoch 289 with a validation loss of 1.1090
Found new best model at epoch 325 with a validation loss of 1.0953
Found new best model at epoch 353 with a validation loss of 1.0952
Found new best model at epoch 355 with a validation loss of 1.0921
Found new best model at epoch 367 with a validation loss of 1.0903
Found new best model at epoch 377 with a validation loss of 1.0880
Found new best model at epoch 391 with a validation loss of 1.0808
Found new best model at epoch 426 with a validation loss of 1.0675
```



In [173…

```python
pretrained_model = create_model_with_dropout(HIDDEN_DIMS, DROPOUT)
load_model(pretrained_model, "dropout.pth")
test_loss, test_accuracy = validate(pretrained_model, test_loader)

print(f"Dropout model achieved test loss of {test_loss:.4f} and accuracy of {tes
```

```
Dropout model achieved test loss of 1.0988 and accuracy of 0.5811
```