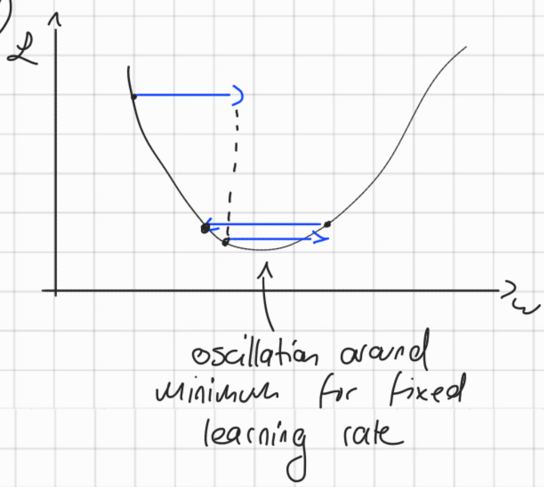
Exercise 1: Optimization and Training Tricks

1)

- Batch Gradient Descent (BGD): The Loss gradient is computed on the whole Training set before an update is made. While this is ideal for convex problems because the gradient is noiseless, pointing to the true minimum, the computational cost of just doing one update per epoch is very high. Thus BGD is rather slow and not apt for very large data sets. Additionally, BGD might get stuck in a local minimum due to the absence of noise.
- Stochastic Gradient Descent (SGD): The loss gradient is computed and the update to the model's parameters is computed for each instance. This reduces computational cost making it ideal for large datasets. The thus computed gradient is very noise, which helps to escape local minima, but can also cause oscillation around a minima.
- Minibatch SGD: The Loss gradient is computed and the update is performed on a (small) subset of the training set. This is a compromise between BGD and SGD. Consequently it benefits from the advantages of both, ie. reduced noise and computational cost, while mitigating the disadvantages, ie. no noise or too much noise and high computational cost. Minibatch is still able to escape local minima, due to the noise introduced by subsampling the training set.

		BGD	SGD	Minibatch SGD
Computational cost		high	low	medium
Speed of convergence	#epochs	few	many	moderately many
	time	slow (esp. for large Datasets)	fast	often fastest (=> allows vectorized operations)

2) a)



Getting closer to a Loss minimum while maintaining the same large learning rate makes it likely to "jump" over that minimum and eventually oscillate around it.

Thus it is common practice to reduce the learning rate going further into training to be able to go deeper into a minimum, while still approaching these minima fast by the initially large learning rate and jumping over sub-optimal local minima.

b)

Exponential decay: $\tau(t) = \tau_0 \exp(-\lambda t)$

with decay rate λ and training step t.

Reduces the learning rate exponentially which yields the above mentioned advantage.

3)

a)

The training set is used to train model.

The validation set is used to judge the model performance and optimise its hyperparameters accordingly

The test set is not involved in training or hyperparameter optimisation, but merely serves to measure the models generalisation capability.

b) This would implicitly involve the test set in the training process, which contradicts the principle of generalisation to unseen data and thus undermines the purpose of the test set.

c) In grid search the hypermeter space is scanned over and for each point the model is trained and test using the validation set. That way the best hyperparameters are determined.

4)

a)

i) RMSProp: The gradient is normalised using the root mean square of the previous gradients:

$$\Delta\theta_{t-1}^{RMS} = \sqrt{\frac{1}{t-1}\sum_{\tau=1}^{t-1}\Delta\theta_{\tau}^{2}} \qquad (10)$$

where this RMS is updated using a decay $\beta < 1$:

$$\Delta\theta_t^{RMS} = \beta\,\Delta\theta_{t-1}^{RMS} + (1-\beta)\,\Delta\theta_t$$

Leading to a training step of the form:

$$\theta_t = \theta_{t-1} + \tau\frac{\Delta\theta_t}{\sqrt{\Delta\theta_{t-1}^{2}+\varepsilon}}$$ with $\varepsilon \ll 1$ preventing division by zero.

This leads to a reduced learning relatively fast because the gradients commonly decrease faster than the accumulated RMS if $\beta \gg 0$.

However, this method is able to adapt too large or too small learning rates through the RMS.

ii) Momentum: The gradient is combined with the gained "momentum" of past gradients via:

$$p_t = \gamma \, p_{t-1} + (1-\gamma)\Delta\Theta_t$$

with momentum p_i and momentum decay γ.
That way even if the current gradient would be zero, there would still be some momentum left, "pushing" the model out of e.g. a local minimum.

b) Adam: $$\omega_t = \omega_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta)g_t \, , \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$m_{t-1} = 0{,}5$, $v_{t-1} = 0{,}2$, $g_t = 2$, $\beta_1 = 0.9$, $\beta_2 = 0{,}99$, $\alpha = 0.01$, $\varepsilon = 10^{-8}$

i) $m_t = 0{,}65$, $v_t = 0{,}238$

ii) $\Delta w = -0{,}0133$

iii)
- $v_t' \gg v_t$: the history factors in much more than the current gradient with $\beta_2 = 0{,}99$
- $|\Delta w_t'| \ll |\Delta w_t|$: the effective learning rate decreases accounting for the previous very large gradients
- Adam seeks to keep the learning rate on an appropriate level since the history of second moments counters the development of too large/small training steps.

Machine Learning Essentials SS25 - Exercise Sheet 5

Instructions

- `TODO` 's indicate where you need to complete the implementations.
- You may use external resources, but **write your own solutions**.
- Provide concise, but comprehensible comments to explain what your code does.
- Code that's unnecessarily extensive and/or not well commented will not be scored.

Exercise 1

Exercise 2

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
# TODO: Import the stuff you need from torch and torchvision
import torch
from torchvision import transforms
from torchvision.datasets import FashionMNIST
from torch.utils.data import random_split, DataLoader
from torch import nn, optim

""""
If you stay in ML-related fields, you will likely be working on a server
always remember to set the number of CPU (or even GPU) threads you're usi
might sometimes use all available threads by default, which will lead to
also want to use some of the threads.
"""
# Example of limiting CPU threads:
# import os
# os.environ["OMP_NUM_THREADS"] = "15"
# os.environ["MKL_NUM_THREADS"] = "15"
# torch.set_num_threads(15) # If you only want to use PyTorch threads
```

Out[1]:
```
'"\nIf you stay in ML-related fields, you will likely be working on a se
rver or cluster. If you do so,\nalways remember to set the number of CPU
(or even GPU) threads you\'re using, as Jupyter notebooks or Python scri
pts \nmight sometimes use all available threads by default, which will l
ead to unhappy colleagues or classmates that\nalso want to use some of t
he threads.\n'
```

In [2]:
```
num_workers = 8
```

2.1

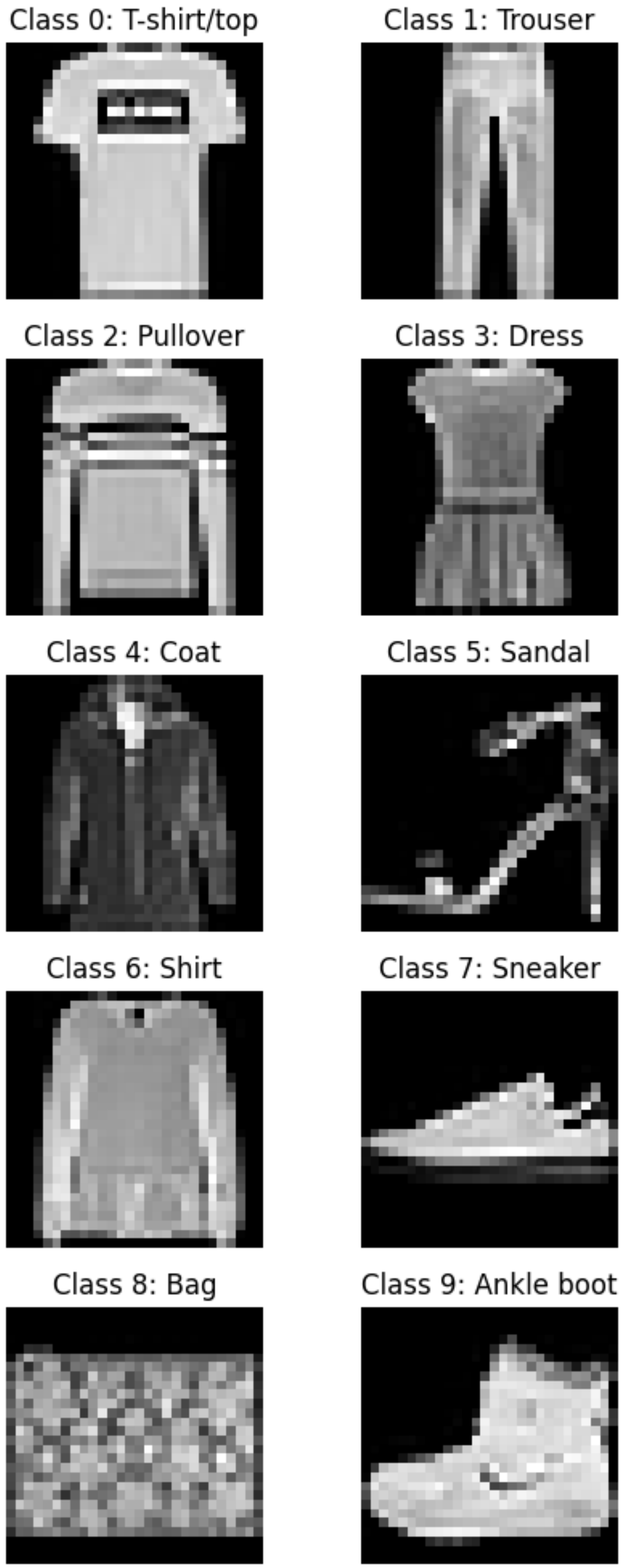
In [3]:
```
# TODO: Define transformations
# Given statistics of training set:
mu_train = 0.286
```

```python
std_train = 0.353
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((mu_train,), (std_train,))
])

# TODO: Load FashionMNIST train/testsets
train_dataset_full = FashionMNIST(
    root='./data',
    train=True,
    download=True,
    transform=transform
)
test_dataset = FashionMNIST(
    root='./data',
    train=False,
    download=True,
    transform=transform
)

print(f"Full training dataset size: {len(train_dataset_full)}")
print(f"Test dataset size: {len(test_dataset)}")
```

```
Full training dataset size: 60000
Test dataset size: 10000
```

In [4]:

```python
# TODO: Create 5x2 subplot grid w/ example image for each class
fig, axes = plt.subplots(5, 2, figsize=(5, 10))
for i in range(10):
    img = np.where(train_dataset_full.targets == i)[0][0]
    img_tensor = train_dataset_full[img][0]
    img_array = img_tensor.numpy().squeeze()
    ax = axes[i // 2, i % 2]
    ax.imshow(img_array, cmap='gray')
    ax.set_title(f'Class {i}: {train_dataset_full.classes[i]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Class 0: T-shirt/top
Class 1: Trouser
Class 2: Pullover
Class 3: Dress
Class 4: Coat
Class 5: Sandal
Class 6: Shirt
Class 7: Sneaker
Class 8: Bag
Class 9: Ankle boot

In [5]:
```
# TODO: Create a validation set from the training set
validation_size = 10000
train_size = len(train_dataset_full) - validation_size
train_dataset, validation_dataset = random_split(
    train_dataset_full, [train_size, validation_size]
)
```

2.2

In [6]:
```
# TODO: Define your model architecture: A class called MLP that inherits
class MLP(nn.Module):
    def __init__(self, input_size=28*28, k=10, hidden_size=128, output_si
        super(MLP, self).__init__()
        layers = []
        layers.append(nn.Linear(input_size, hidden_size))
        layers.append(nn.ReLU())
        for _ in range(k):
            layers.append(nn.Linear(hidden_size, hidden_size))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_size, output_size))
        self.model = nn.Sequential(*layers)
    def forward(self, x):
        x = x.view(x.size(0), -1)
        return nn.functional.softmax(self.model(x), dim=1)


# TODO: Define appropriate loss
criterion = nn.CrossEntropyLoss()
```

2.3

In [7]:
```
BATCH_SIZE_DEFAULT = 2048 # TODO: Set your default batch size. The capita

#TODO: Define DataLoaders for training, validation, and test sets
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE_DEFAULT,
    shuffle=True,
    num_workers=num_workers
)
validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE_DEFAULT,
    shuffle=False,
    num_workers=num_workers
)
test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE_DEFAULT,
    shuffle=False,
    num_workers=num_workers
)
```

In [14]:
```
def calculate_accuracy(outputs, labels):
    """
    Calculate accuracy given model outputs and true labels.
    """
```

```
    _, predicted = torch.max(outputs.data, 1) # Prediction = class with h
    total = labels.size(0)
    correct = (predicted == labels).sum().item() #.item() converts a sing
    return correct / total

def train_model(model, criterion, optimizer, train_loader, val_loader, nu
    # Device configuration: if available use GPU (needs CUDA installed an
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    print(f"Training on device: {device}")

    # TODO: Define training loop that for each epoch iterates over all mi
    # Record and return the training&validation loss and accuracy for eac
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(num_epochs):

        # mini batches
        model.train()
        running_loss = 0.0
        running_accuracy = 0.0

        for _ , (images, labels) in enumerate(train_loader):

            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            running_accuracy += calculate_accuracy(outputs, labels)

        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)

        train_accuracy = running_accuracy / len(train_loader)
        train_accuracies.append(train_accuracy)

        model.eval()

        with torch.no_grad():
            running_val_acc = 0.0
            running_val_loss = 0.0
            for _, (images, labels) in enumerate(val_loader):
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)

                running_val_acc += calculate_accuracy(outputs, labels)
                loss = criterion(outputs, labels)
                running_val_loss += loss.item()

            val_accuracy = running_val_acc / len(val_loader)
            val_accuracies.append(val_accuracy)
```

```
            val_loss = running_val_loss / len(val_loader)
            val_losses.append(val_loss)

        print(f'Epoch [{epoch+1}/{num_epochs}], '
              f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accu
              f'Val Accuracy: {val_accuracy:.4f}')

    return train_losses, val_losses, train_accuracies, val_accuracies
```

2.4

In [16]:
```
# TODO: Define hyperparameter grid for tuning
k = [2, 5, 10]
hidden_size = [32, 64, 128, 256]
batch_size = [256, 512, 1024, BATCH_SIZE_DEFAULT]

# fixed hybperparameters
num_epochs = 100
input_size = 28 * 28
output_size = 10

default_k = 2
default_hidden_size = 64


# TODO: For each hyperparameter setting, instantiate model&optimizer,
# train the model, and store the results for evaluation later
results_k_adam = {}
results_k_sgd = {}
for k_value in k:
    model_adam = MLP(input_size=input_size, k=k_value, hidden_size=defaul
    model_sgd = MLP(input_size=input_size, k=k_value, hidden_size=default_
    optimizer_adam = optim.Adam(model_adam.parameters())
    optimizer_sgd = optim.SGD(model_sgd.parameters(), lr=0.01, momentum=0

    print(f"Training model with k={k_value} using Adam optimizer")
    train_losses_adam, val_losses_adam, train_accuracies_adam, val_accura
        model_adam, criterion, optimizer_adam, train_loader, validation_l
    )
    results_k_adam[k_value] = {
        'train_losses': train_losses_adam,
        'val_losses': val_losses_adam,
        'train_accuracies': train_accuracies_adam,
        'val_accuracies': val_accuracies_adam
    }
    print(f"Training model with k={k_value} using SGD optimizer")
    train_losses_sgd, val_losses_sgd, train_accuracies_sgd, val_accuracie
        model_sgd, criterion, optimizer_sgd, train_loader, validation_loa
    )
    results_k_sgd[k_value] = {
        'train_losses': train_losses_sgd,
        'val_losses': val_losses_sgd,
        'train_accuracies': train_accuracies_sgd,
        'val_accuracies': val_accuracies_sgd
    }
```

```
Epoch [93/100], Train Loss: 1.6367, Train Accuracy: 0.8293, Val Accuracy:
0.8144
Epoch [94/100], Train Loss: 1.6365, Train Accuracy: 0.8291, Val Accuracy:
0.8146
Epoch [95/100], Train Loss: 1.6359, Train Accuracy: 0.8299, Val Accuracy:
0.8154
Epoch [96/100], Train Loss: 1.6353, Train Accuracy: 0.8307, Val Accuracy:
0.8148
Epoch [97/100], Train Loss: 1.6354, Train Accuracy: 0.8303, Val Accuracy:
0.8146
Epoch [98/100], Train Loss: 1.6347, Train Accuracy: 0.8310, Val Accuracy:
0.8158
Epoch [99/100], Train Loss: 1.6352, Train Accuracy: 0.8303, Val Accuracy:
0.8139
Epoch [100/100], Train Loss: 1.6346, Train Accuracy: 0.8311, Val Accuracy:
0.8146
```

In [ ]:
```
results_batch_size_adam = {}
results_batch_size_sgd = {}

for batch_size_value in batch_size:
    train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size_value,
        shuffle=True,
        num_workers=num_workers
    )
    validation_loader = DataLoader(
        validation_dataset,
        batch_size=batch_size_value,
        shuffle=False,
        num_workers=num_workers
    )

    model_adam = MLP(input_size=input_size, k=default_k, hidden_size=defa
    model_sgd = MLP(input_size=input_size, k=default_k, hidden_size=defau
    optimizer_adam = optim.Adam(model_adam.parameters())
    optimizer_sgd = optim.SGD(model_sgd.parameters(), lr=0.01, momentum=0

    print(f"Training model with batch size={batch_size_value} using Adam 
    train_losses_adam, val_losses_adam, train_accuracies_adam, val_accura
        model_adam, criterion, optimizer_adam, train_loader, validation_l
    )
    results_batch_size_adam[batch_size_value] = {
        'train_losses': train_losses_adam,
        'val_losses': val_losses_adam,
        'train_accuracies': train_accuracies_adam,
        'val_accuracies': val_accuracies_adam
    }

    print(f"Training model with batch size={batch_size_value} using SGD o
    train_losses_sgd, val_losses_sgd, train_accuracies_sgd, val_accuracie
        model_sgd, criterion, optimizer_sgd, train_loader, validation_loa
    )
    results_batch_size_sgd[batch_size_value] = {
        'train_losses': train_losses_sgd,
        'val_losses': val_losses_sgd,
        'train_accuracies': train_accuracies_sgd,
        'val_accuracies': val_accuracies_sgd
    }
```

In [26]:
```
# TODO: Plot evolution of validation accuracy for each hyperparameter set

def plot_results(results, title):
    plt.figure(figsize=(12, 8))
    for key, value in results.items():
        plt.plot(value['val_accuracies'], label=f'k={key}' if 'k' in titl
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Validation Accuracy')
    plt.legend()
    plt.grid()
    plt.show()
```

In [27]:
```
print(results_hidden_size_adam.values())
```

dict_values([{'train_losses': [2.2086666584014893, 1.9124801301956176, 1.7
935973978042603, 1.72051860332489, 1.694562406539917, 1.682826418876648, 1
.6749101448059083, 1.6701427459716798, 1.6660133695602417, 1.6606020832061
767, 1.6582284259796143, 1.654193000793457, 1.6505397081375122, 1.64865959
16748046, 1.6456811761856078, 1.6433111333847046, 1.6431246423721313, 1.64
10329580307006, 1.639166374206543, 1.6387004518508912, 1.636634693145752,
1.6357011032104491, 1.6344984722137452, 1.6333588504791259, 1.633451471328
7354, 1.6324427127838135, 1.6314708805084228, 1.6311833572387695, 1.631136
1360549927, 1.6294871520996095, 1.6280263662338257, 1.628728461265564, 1.6
28010950088501, 1.6270203542709352, 1.626003670692444, 1.6255670881271362,
1.6252462577819824, 1.6239779281616211, 1.6247189807891846, 1.623516893386
841, 1.6220717096328736, 1.6221473979949952, 1.6217905855178834, 1.6206709
09881592, 1.621665015220642, 1.620404977798462, 1.6200374460220337, 1.6192
26861000061, 1.6188181447982788, 1.6207627058029175, 1.6185651588439942, 1
.6174710083007813, 1.6185905027389527, 1.6174226474761964, 1.6162547159194
947, 1.6161060428619385, 1.616579875946045, 1.6157733583450318, 1.61605782
5088501, 1.614568901062117, 1.6145924568176269, 1.6161954832077026, 1.613
6467695236205, 1.6136060762405395, 1.61245361328125, 1.6124424171447753, 1
.6127238750457764, 1.6125553417205811, 1.5989297389984132, 1.5895776271820
068, 1.5872467470169067, 1.5843375873565675, 1.5835417604446411, 1.5823227
977752685, 1.5816703748703003, 1.5788419914245606, 1.577428879737854, 1.57
79587650299072, 1.575684952735901, 1.5744591999053954, 1.574035406112671,
1.575721492767334, 1.5733262538909911, 1.5704761028289795, 1.5713236427307
13, 1.570775146484375, 1.5690933561325073, 1.5688835525512694, 1.568105053
9016723, 1.5691445922851563, 1.567627387046814, 1.5675666999816895, 1.5689
304780960083, 1.5655420160293578, 1.5661112785339355, 1.5647973251342773,
1.5636361598968507, 1.565658574104309, 1.5642124795913697, 1.5649220705032
35], 'val_losses': [2.042501163482666, 1.8264740705490112, 1.7542208433151
245, 1.7088290691375732, 1.6928551912307739, 1.6849909782409669, 1.6776272
535324097, 1.6743304491043092, 1.6712412118911744, 1.6643535852432252, 1.6
61929988861084, 1.6596421957015992, 1.6569008827209473, 1.6552968263626098
, 1.6532380819320678, 1.6524292707443238, 1.6528838872909546, 1.6504288673
40088, 1.6493853330612183, 1.6483260631561278, 1.6475474119186402, 1.64641
55197143555, 1.6466788053512573, 1.6463876724243165, 1.64597647190094, 1.6
441298961639403, 1.6441519737243653, 1.6478918313980102, 1.644204187393188
5, 1.6439706325531005, 1.6434713363647462, 1.640738320350647, 1.6407545328
140258, 1.6428239822387696, 1.6413987874984741, 1.640782403945923, 1.64034
28077697755, 1.6401729583740234, 1.6410790920257567, 1.6383211612701416, 1
.6387939453125, 1.6380725860595704, 1.637694787979126, 1.6388476848602296,
1.6379369497299194, 1.6392048835754394, 1.638838768005371, 1.6370249032974
242, 1.6387346744537354, 1.6372810363769532, 1.6382737398147582, 1.6382448
91166687, 1.6364982366561889, 1.6359021425247193, 1.6355789184570313, 1.63
6524748802185, 1.6367555856704712, 1.6367313861846924, 1.6350367546081543,
1.635895037651062, 1.636530613899231, 1.6349081516265869, 1.63478548526763
91, 1.6347235918045044, 1.6350157737731934, 1.6345439672470092, 1.63439512
25280762, 1.6264461278915405, 1.613888192176819, 1.6134798049926757, 1.608
5321187973023, 1.6052253007888795, 1.6037676572799682, 1.6024924993515015,
1.6042128086090088, 1.6016602277755738, 1.6005648374557495, 1.599534845352
173, 1.5989341497421266, 1.5986332654953004, 1.6011247396469117, 1.5982038
974761963, 1.5968698263168335, 1.5972932815551757, 1.597111463546753, 1.59
81909036636353, 1.5965644359588622, 1.5957716464996339, 1.5960610389709473
, 1.5958025455474854, 1.594939661026001, 1.598764419555664, 1.595660853385
9253, 1.5944181680679321, 1.5943343877792358, 1.594592523574829, 1.5951914
548873902, 1.596961998939514, 1.595246434211731, 1.5945375442504883], 'tra
in_accuracies': [0.347799233490566, 0.5920327240566038, 0.6811265477594339
, 0.7601923643867924, 0.7769136939858491, 0.7859559257075471, 0.7929105247
641509, 0.796893794221698, 0.8004057340801887, 0.8055081810141509, 0.80682
74616745283, 0.811510539504717, 0.8145688384433963, 0.8168720518867925, 0.
8199705188679245, 0.8218256191037736, 0.8217891362028301, 0.82375663325471
69, 0.8256360554245283, 0.8258118366745283, 0.8277760170990567, 0.82920216

68632076, 0.8298887087264152, 0.8312621609669811, 0.8311431308962264, 0.83
15941922169812, 0.832904628537736, 0.8328629864386792, 0.832468676297169 9,
0.8342493366745284, 0.835806308962264, 0.8348087411556604, 0.8358361586084
905, 0.8368462558962264, 0.8379385318396227, 0.8378699882075472, 0.8384419
221698113, 0.8398798643867924, 0.839289873231132, 0.8402778596698113, 0.84
16767393867924, 0.8412529481132076, 0.8420795253537736, 0.843268720518867 9
, 0.8417213295990567, 0.8431747494103773, 0.843899248231132, 0.84409087558
96227, 0.8445190890330189, 0.8424624115566037, 0.8446167452830189, 0.84598
17216981133, 0.8447685731132075, 0.8459551886792452, 0.8470091391509433, 0
.847149174528302, 0.8463026975235849, 0.847624189268868, 0.84752505896226 4
, 0.8487083579009433, 0.8483943838443396, 0.8465761350235849, 0.8494044811
320756, 0.8494199587264152, 0.850610627948113 2, 0.8508744840801887, 0.8503
965212264152, 0.849942143278302, 0.8651676739386792, 0.8749738354952831, 0
.8766667895047169, 0.8792264887971699, 0.8803873083726416, 0.8811803508254
716, 0.8812271521226416, 0.8848677034198112, 0.8860134139150944, 0.8853526
680424528, 0.8878364534198113, 0.8889386792452831, 0.8894932930424528, 0.8
872619398584906, 0.8903611438679245, 0.8932432930424529, 0.891987028301886
8, 0.892746904481132, 0.8944767099056604, 0.8946189563679244, 0.8953202387
971699, 0.8944265919811321, 0.896047685731132, 0.8957952535377359, 0.89445
23879716981, 0.8980332399764152, 0.897708210495283, 0.8989173054245283, 0.
900358564268868, 0.8980564563679245, 0.8989007222877359, 0.898552476415094
4], 'val_accuracies': [0.40941043279867256, 0.6526877074115044, 0.73010145
87942478, 0.7650909153761062, 0.7760275511615043, 0.7838901756084071, 0.78
87142215154868, 0.7912722967367257, 0.792619607300885, 0.8014337320243363,
0.8033151272123893, 0.8057954231194691, 0.8083993017146017, 0.809082895464
6018, 0.8110878733407081, 0.8122078954646017, 0.8118492464048673, 0.813216
4339048673, 0.8132293971238939, 0.81455078125, 0.815547220685840 7, 0.81647
79798119468, 0.8140564504977876, 0.8159438951880531, 0.8169792242809735, 0
.8182029521570797, 0.8189453125, 0.8152473382190266, 0.8177864007190265, 0
.8169532978429203, 0.8203643528761061, 0.821405731471239, 0.8216987002212 3
9, 0.8190948216261061, 0.8205008987831859, 0.8206763343473451, 0.821842159
8451326, 0.8216727737831858, 0.8214644980641592, 0.8240554134402656, 0.823
6388620022124, 0.8247390071902656, 0.8243872718473451, 0.8228187223451326,
0.8235671321902656, 0.82099609375, 0.8229034153761061, 0.8236388620022124,
0.8236777516592919, 0.8246543141592919, 0.822479950221239, 0.8223304410951
326, 0.8253508711283185, 0.825897918971239, 0.8253837112831859, 0.82512963
21902656, 0.824849626659292, 0.8245765348451327, 0.8255721100663717, 0.825
3508711283185, 0.824140106471239, 0.826509782909292, 0.8266402793141593, 0
.8263533600663717, 0.8260085384402656, 0.8265296598451327, 0.8270239905973
451, 0.8350629148230089, 0.8488963979535399, 0.8474514311393806, 0.8534983
407079647, 0.8571574253318583, 0.8578081789269911, 0.8580881844579646, 0.8
572611310840708, 0.8596766108960177, 0.861583932522124, 0.862111034292035
, 0.8629381568030974, 0.8639596584623893, 0.8609720685840708, 0.8622675262
72124, 0.8637194068030973, 0.8631593957411503, 0.8648325152101769, 0.86304
18625553098, 0.8651643736172566, 0.8652101769911503, 0.8659974764933628, 0
.8649431346792035, 0.8653596861172567, 0.8627428443030973, 0.8662187154314
159, 0.86651859789823, 0.8668374930862832, 0.8670198423672566, 0.866459831
3053098, 0.8647020188053098, 0.8665704507743364, 0.8664468680862832]}, {'t
rain_losses': [2.145920763015747, 1.7990608310699463, 1.7131030988693237,
1.6869616460800172, 1.6741858673095704, 1.664165472984314, 1.6577818250656
13, 1.6533356809616089, 1.6485765027999877, 1.6452419757843018, 1.64342959
88082885, 1.6397058248519898, 1.638141679763794, 1.6361491727828978, 1.634
9687671661377, 1.6321216964721679, 1.6303195333480835, 1.6297683143615722,
1.6285014152526855, 1.6263620567321777, 1.625418734550476, 1.6237761163711
548, 1.623421583175659, 1.6235025882720948, 1.622590379714966, 1.622690725
3265381, 1.6216603851318359, 1.6198959970474243, 1.618521466255188, 1.6199
506521224976, 1.6179446649551392, 1.6171039867401122, 1.6170352029800414,
1.6161783647537231, 1.6150891304016113, 1.6144938278198242, 1.614603199958
8013, 1.6129531478881836, 1.6148677110671996, 1.6140899419784547, 1.612811
0599517822, 1.6112449407577514, 1.61107479095459, 1.610211334228515 6, 1.61

13668584823608, 1.6123454570770264, 1.6019066286087036, 1.5881720495223999
, 1.581423087120056, 1.577786660194397, 1.573950128555298, 1.5749277114868
163, 1.569568657875061, 1.5688910293579101, 1.567005352973938, 1.565790200
2334594, 1.565465545654297, 1.5651554441452027, 1.56364830493927, 1.563018
0025100708, 1.5622372007369996, 1.5598554182052613, 1.5616003561019898, 1.
5595099782943727, 1.5610424518585204, 1.5594987297058105, 1.55783263206481
93, 1.5588799285888673, 1.5562115335464477, 1.5543180513381958, 1.55267440
79589845, 1.5529496622085572, 1.552830457687378, 1.5514457082748414, 1.551
2262630462645, 1.5517416763305665, 1.5498287343978883, 1.5506195783615113,
1.548193621635437, 1.5479455614089965, 1.5476676225662231, 1.5488037872314
453, 1.5454917240142823, 1.545870032310486, 1.5441962814331054, 1.54317887
30621339, 1.542298755645752, 1.5433381748199464, 1.5427030563354491, 1.541
5703535079956, 1.5424179887771607, 1.5408151054382324, 1.5409051847457886,
1.5412213897705078, 1.5386480808258056, 1.539801516532898, 1.5384561204910
279, 1.5370307874679565, 1.537038722038269, 1.5375845432281494], 'val_loss
es': [1.9093505620956421, 1.7403450727462768, 1.70059015750885, 1.68672780
9906006, 1.675393295288086, 1.668319010734558, 1.666588592529297, 1.656855
1301956176, 1.6551606893539428, 1.653127956390381, 1.650002670288086, 1.64
82117891311645, 1.6488466024398805, 1.6455583810806274, 1.6453126192092895
, 1.6444809436798096, 1.6455478906631469, 1.6451456785202025, 1.6395103216
171265, 1.6387269258499146, 1.6388546228408813, 1.6374081373214722, 1.6386
996984481812, 1.642271137237549, 1.636710023880005, 1.6374837160110474, 1.
6404514551162719, 1.6354127645492553, 1.637186551094055, 1.637076330184936
5, 1.6365205287933349, 1.6350101709365845, 1.6345837831497192, 1.633335018
157959, 1.63598473072052, 1.6334725379943849, 1.6322363376617433, 1.633541
3932800293, 1.634214949607849, 1.6327777624130249, 1.6329921245574952, 1.6
334795713424684, 1.6315420627593995, 1.6322375774383544, 1.632891082763671
8, 1.6316339492797851, 1.6138067722320557, 1.6042073488235473, 1.602189421
6537476, 1.5996078729629517, 1.598812198638916, 1.5941715002059937, 1.5924
615383148193, 1.5936115503311157, 1.5914504051208496, 1.5905061960220337,
1.5967690467834472, 1.5889180898666382, 1.5888860940933227, 1.591835832595
8253, 1.58718683719635, 1.5892498254776002, 1.5877869367599486, 1.59186186
7904663, 1.5883119106292725, 1.5869086027145385, 1.590118670463562, 1.5858
268022537232, 1.5864718675613403, 1.5831793785095214, 1.5839133977890014,
1.5851529121398926, 1.5840908527374267, 1.5845986366271974, 1.583620166778
5645, 1.5855608463287354, 1.5817489862442016, 1.580184531211853, 1.5824403
047561646, 1.5818200588226319, 1.5825438976287842, 1.5836559057235717, 1.5
80185842514038, 1.5821757555007934, 1.5800763368606567, 1.5817991495132446
, 1.5798296689987184, 1.5826917171478272, 1.584291696548462, 1.57951028347
01538, 1.5806933879852294, 1.5795471668243408, 1.5794405698776246, 1.57991
09935760498, 1.579488182067871, 1.5784868955612184, 1.580325198173523, 1.5
795660257339477, 1.5797866344451905, 1.5792770862579346], 'train_accuracie
s': [0.41769752358490564, 0.6893410966981132, 0.7580774616745284, 0.780305
1297169812, 0.7926160819575472, 0.8020673643867924, 0.8081191037735849, 0.
8109673496462264, 0.8162791126179245, 0.8193609964622641, 0.82080888856132
07, 0.8236468160377359, 0.8254779628537736, 0.8274675707547169, 0.82862728
4787736, 0.8310591096698112, 0.8330804097877359, 0.8337334168632076, 0.834
7670990566037, 0.8367390182783019, 0.8379079451650944, 0.8394376474056604,
0.8394000589622641, 0.8394652859669811, 0.8404013119103773, 0.840197892099
0567, 0.8412264150943396, 0.8430951503537736, 0.8445566774764152, 0.843005
9699292453, 0.8452421137971698, 0.8458085200471699, 0.8459120725235849, 0.
8466188826650943, 0.8476798349056603, 0.8484459758254717, 0.84851488797169
8, 0.8494767099056604, 0.8478235554245284, 0.8481076798349056, 0.849837485
2594339, 0.8514670548349056, 0.8516461527122641, 0.8525563826650944, 0.851
289799528302, 0.8498301149764151, 0.8610627948113208, 0.874524985259434, 0
.88154296875, 0.8852012087264152, 0.8888697670990566, 0.8882303950471698,
0.8943532576650943, 0.8944048496462265, 0.8964283608490566, 0.897165757665
0943, 0.8981651680424528, 0.8981191037735848, 0.8991564711084905, 0.900125
6633254716, 0.9008291568396227, 0.9034109669811321, 0.9013959316037736, 0.
9032771963443396, 0.9016737912735848, 0.9036287588443396, 0.9047520636792

45, 0.9042294369103773, 0.9067725530660377, 0.9091347287735849, 0.91048275
35377359, 0.910510392099567, 0.9103438237028301, 0.9124812057783019, 0.91
18510465801887, 0.9114184109669812, 0.9133940153301887, 0.9126488797169812
, 0.9148315890330189, 0.9153847287735849, 0.915964033018868, 0.91417194870
28301, 0.9179838590801888, 0.9171373820754717, 0.9190042747641509, 0.92015
18278301887, 0.9211737175707547, 0.920156987028302, 0.9203939416273584, 0.
9215842423349057, 0.9205000737028302, 0.9225866008254716, 0.92223319575471
7, 0.9214442069575471, 0.9250294811320755, 0.9233192069575472, 0.924886129
1273585, 0.9260373673349056, 0.9259411851415095, 0.9254915978773585], 'val
_accuracies': [0.605295907079646, 0.7319690265486726, 0.7692114905973451,
0.7799718266039823, 0.7910121681415929, 0.7979060080199115, 0.798348485896
0177, 0.808262755807522, 0.8101311877765486, 0.8106782356194691, 0.8122467
851216815, 0.8154824045907081, 0.8131844579646017, 0.8173179964048674, 0.8
180594925331859, 0.8171028069690266, 0.8159767353429203, 0.817421702157079
8, 0.8218620367809735, 0.823651825221239, 0.8237754079092919, 0.8240294870
022125, 0.822870575221239, 0.8193808766592919, 0.824628387721239, 0.822968
231471239, 0.8217704300331858, 0.8263663232853983, 0.8238410882190266, 0.8
248306139380531, 0.8245048050331858, 0.8257026064712389, 0.827115597345132
7, 0.8275710384402656, 0.8250708655973451, 0.8281370990044248, 0.828645257
1902656, 0.8292502074115043, 0.8265227461283186, 0.8285735273783186, 0.827
2262168141593, 0.826288543971239, 0.8297454023783185, 0.829139587942478, 0
.8289900788163717, 0.8290618086283186, 0.8472889587942477, 0.8578868224557
523, 0.8582316440818584, 0.8636278000553098, 0.8620065334623893, 0.8662774
820243364, 0.8689142007743363, 0.8674882466814159, 0.8696176714601769, 0.8
712052336836283, 0.8643960868362832, 0.871518079369469, 0.8723969856194691
, 0.8688554341814159, 0.873880842090708, 0.8714791897123894, 0.87330786780
97345, 0.8695779175884957, 0.8732102115597344, 0.8744270257190265, 0.87042
39836836283, 0.8758918694690265, 0.8736786158738938, 0.8793098382190265, 0
.8771484375, 0.8763741012168141, 0.8752998824668141, 0.8761468127765486, 0
.8772659706858408, 0.8749351839048674, 0.8799018252212389, 0.8811065403761
061, 0.8783073492809734, 0.8795241634402655, 0.8783332757190265, 0.8771095
478429203, 0.8809890071902655, 0.879315887721239, 0.8817054410951327, 0.87
97263896570797, 0.8828833655973451, 0.8777603014380532, 0.8764449668141593
, 0.8807288785951327, 0.8798300954092919, 0.8817313675331858, 0.8818679134
402656, 0.8809362900995575, 0.8815559319690266, 0.8822127350663717, 0.8802
993639380532, 0.881099626659292, 0.8816008711283185, 0.881978532909292]},
{'train_losses': [2.0271785402297975, 1.7282268285751343, 1.68396348476409
92, 1.6666369247436523, 1.6583967685699463, 1.6524731254577636, 1.64838691
23458863, 1.646834020614624, 1.6439767503738403, 1.6389569187164306, 1.635
3382396697997, 1.623781270980835, 1.609028582572937, 1.6012655353546144, 1
.5984177589416504, 1.5948091316223145, 1.591699767112732, 1.59049969673156
74, 1.5870997095108033, 1.5829159736633, 1.5812674522399903, 1.5776179742
81311, 1.578244252204895, 1.5765232467651367, 1.5752345132827759, 1.572385
5686187744, 1.5714051818847656, 1.5721747589111328, 1.5696622848510742, 1.
5671147727966308, 1.563820824623108, 1.5640416765213012, 1.56152889728546l
5, 1.5608096885681153, 1.559441146850586, 1.5599852561950684, 1.5568905115
127563, 1.554735598564148, 1.5551681041717529, 1.5546569013595581, 1.55268
3072090149, 1.5525076055526734, 1.552058081626892, 1.5506078386306763, 1.5
485750913619996, 1.550602240562439, 1.5480125188827514, 1.5471388244628905
, 1.5464976406097413, 1.545755271911621, 1.544031205177307, 1.548004555702
2095, 1.5448163366317749, 1.5433929967880249, 1.5422344255447387, 1.542384
9439620971, 1.5407220792770386, 1.5422044372558594, 1.5387277507781982, 1.
5372378969192504, 1.5377240705490112, 1.5374323749542236, 1.53765862941741
95, 1.5374916076660157, 1.537065954208374, 1.534888472557068, 1.5387653303
146362, 1.5374104261398316, 1.535936574935913, 1.5342303657531737, 1.53479
57801818848, 1.5347988176345826, 1.5347228097915648, 1.5308429384231568, 1
.5337612342834472, 1.5332383251190185, 1.5319320917129517, 1.5325584840774
535, 1.5329518413543701, 1.5299924564361573, 1.5294185638427735, 1.5292125
177383422, 1.529498176574707, 1.5298959159851073, 1.528254280090332, 1.527
5247240066527, 1.5277625131607055, 1.5291568326950074, 1.52674455642700 2,

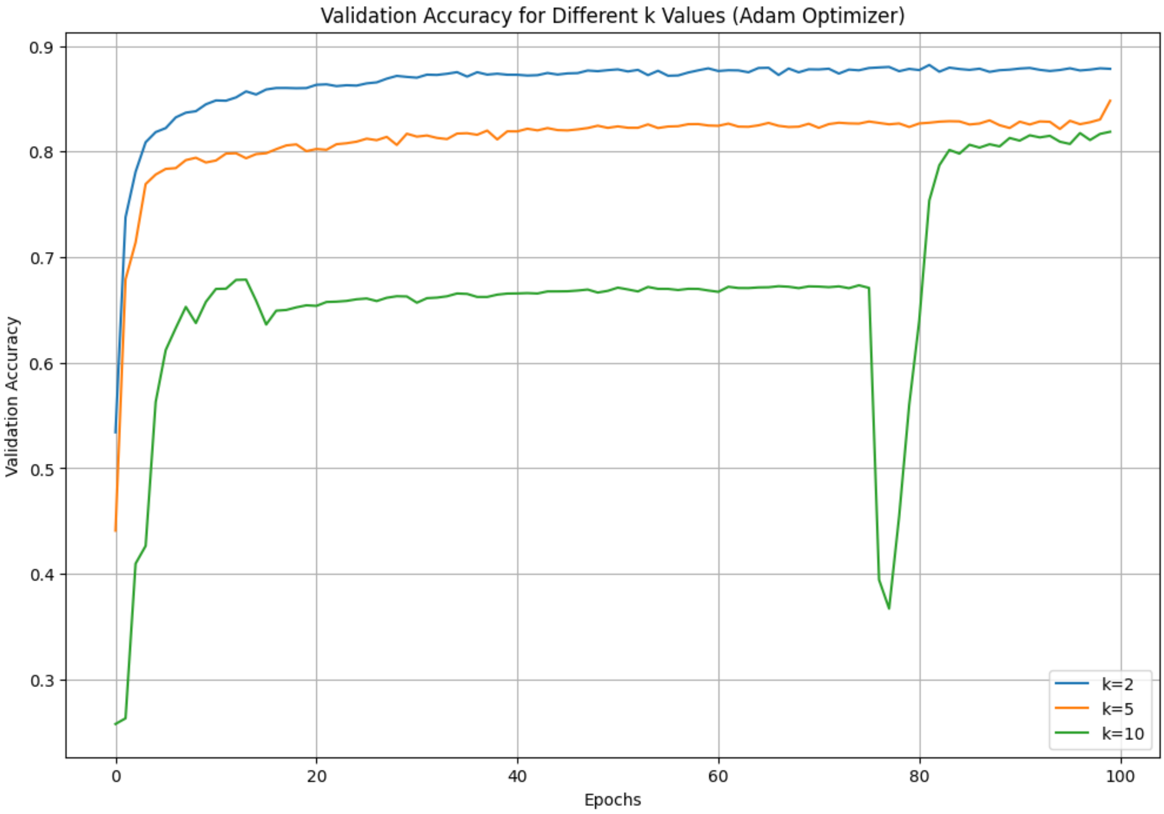
1.5274211168289185, 1.5296026420593263, 1.531154041290283, 1.5270272254943
849, 1.5260605478286744, 1.52358163356781, 1.5261203908920289, 1.524618105
8883668, 1.5246931838989257, 1.52309139251709, 1.5230295753479004], 'val_l
osses': [1.7896323204040527, 1.7004621028900146, 1.6771831274032594, 1.667
2874212265014, 1.6599085569381713, 1.6588497400283813, 1.6514943361282348,
1.6521634578704834, 1.6482295274734498, 1.6453479528427124, 1.642649364471
4355, 1.6252258539199829, 1.6127204895019531, 1.6115930318832397, 1.607823
1573104858, 1.6052505254745484, 1.6029546499252318, 1.6025701999664306, 1.
5998642921447754, 1.5999191522598266, 1.597684669494629, 1.593801879882812
4, 1.5934484958648683, 1.5951380252838134, 1.594032883644104, 1.5884480714
797973, 1.5895514965057373, 1.5888717889785766, 1.5884185552597045, 1.5845
432519912719, 1.583852505683899, 1.5829604625701905, 1.5825628519058228, 1
.5811726331710816, 1.5825895547866822, 1.5833954572677613, 1.5846493005752
564, 1.584132432937622, 1.5788427591323853, 1.5821452379226684, 1.58113434
31472777, 1.582375454902649, 1.5805487394332887, 1.5778645515441894, 1.580
3476333618165, 1.583198070526123, 1.5805450677871704, 1.5851022243499755,
1.5790602684020996, 1.580253577232361, 1.5808779001235962, 1.5771541833877
563, 1.5791536092758178, 1.5786488771438598, 1.576088261604309, 1.57776768
20755004, 1.578601861000061, 1.576300859451294, 1.5759633302688598, 1.5782
87672996521, 1.576562190055847, 1.5788485288619996, 1.5761998414993286, 1.
5783742189407348, 1.5797568798065185, 1.5784961700439453, 1.5804172992706
298, 1.5778483152389526, 1.5789556503295898, 1.578775119781494, 1.578008127
2125244, 1.5751897811889648, 1.576167917251587, 1.5765167474746704, 1.5807
71255493164, 1.5766636610031128, 1.5758847713470459, 1.5800289869308473, 1
.5786994457244874, 1.5778679609298707, 1.5773868083953857, 1.5771833658218
384, 1.5770013809204102, 1.578310489654541, 1.5761276006698608, 1.57649388
31329345, 1.5751750707626342, 1.5771316289901733, 1.5761592626571654, 1.57
64260292053223, 1.578830599784851, 1.5773207426071167, 1.5772318840026855,
1.5751943826675414, 1.576586699485779, 1.5765919923782348, 1.5769528627395
63, 1.5763001680374145, 1.5748233079910279, 1.5766592264175414], 'train_ac
curacies': [0.5295913178066037, 0.7446333284198112, 0.7817747641509434, 0.
7987208873820755, 0.8060871167452831, 0.8114766362028302, 0.81541826356132
07, 0.8160318396226416, 0.8190481279481132, 0.8236177034198113, 0.82831515
33018868, 0.8396440153301887, 0.8540761350235848, 0.862661777712264, 0.864
3086674528301, 0.8681014150943396, 0.8717482311320756, 0.8722855247641509,
0.8763513413915095, 0.8799620430424528, 0.8819483343160377, 0.885601415094
3397, 0.8844026385613207, 0.8860546875, 0.887818764740566, 0.8905708284198
113, 0.8909728773584905, 0.8912315742924528, 0.8929794369103773, 0.8954694
87028302, 0.8994748673349057, 0.8986626621462264, 0.9015650795990566, 0.90
21849204009433, 0.9037798496462264, 0.9028283461084905, 0.9061040683962264
, 0.9088856132075471, 0.9082900943396227, 0.9080085495283019, 0.9099421432
783019, 0.9102107900943396, 0.9113273879716981, 0.912221403301887, 0.9145
010318396227, 0.9121314858490567, 0.9148839180424528, 0.9159397110849057,
0.9162426297169811, 0.9166999557783019, 0.9190028007075471, 0.914560731132
0755, 0.9178261350235849, 0.918874189268868, 0.9201617777122642, 0.9200608
048349056, 0.9220268278301887, 0.9201142393867925, 0.9238904038915094, 0.9
251875737028301, 0.9251776238207547, 0.925325766509434, 0.9249922612028301
, 0.9252723319575472, 0.9256043632075471, 0.9278239239386792, 0.9235760613
207548, 0.9247575176886792, 0.926484375, 0.9280697228773584, 0.92737654775
9434, 0.9271569133254717, 0.9274789946933963, 0.93169921875, 0.92893241450
47169, 0.9289276238207548, 0.9301289799528302, 0.929753095518868, 0.928948
6291273584, 0.9322085053066037, 0.9326875737028302, 0.9330004422169812, 0.
932610922759434, 0.932396079009434, 0.9339261497641509, 0.9347166126179245
, 0.9345445165094339, 0.9329849646226416, 0.9356364239386792, 0.9351415094
339623, 0.9324675707547169, 0.9310709021226415, 0.9349694133254717, 0.9360
642688679245, 0.9386423938679245, 0.9359360259433963, 0.9374819428066037,
0.9373828125, 0.939010170990566, 0.9390672906839623], 'val_accuracies':
[0.6812897538716814, 0.7644142353429204, 0.7870549294800885, 0.79767785536
50442, 0.8046892284292035, 0.8051317063053098, 0.8113549156526549, 0.81040
4279590708, 0.8123703678097345, 0.81630859375, 0.8192512444690265, 0.83534

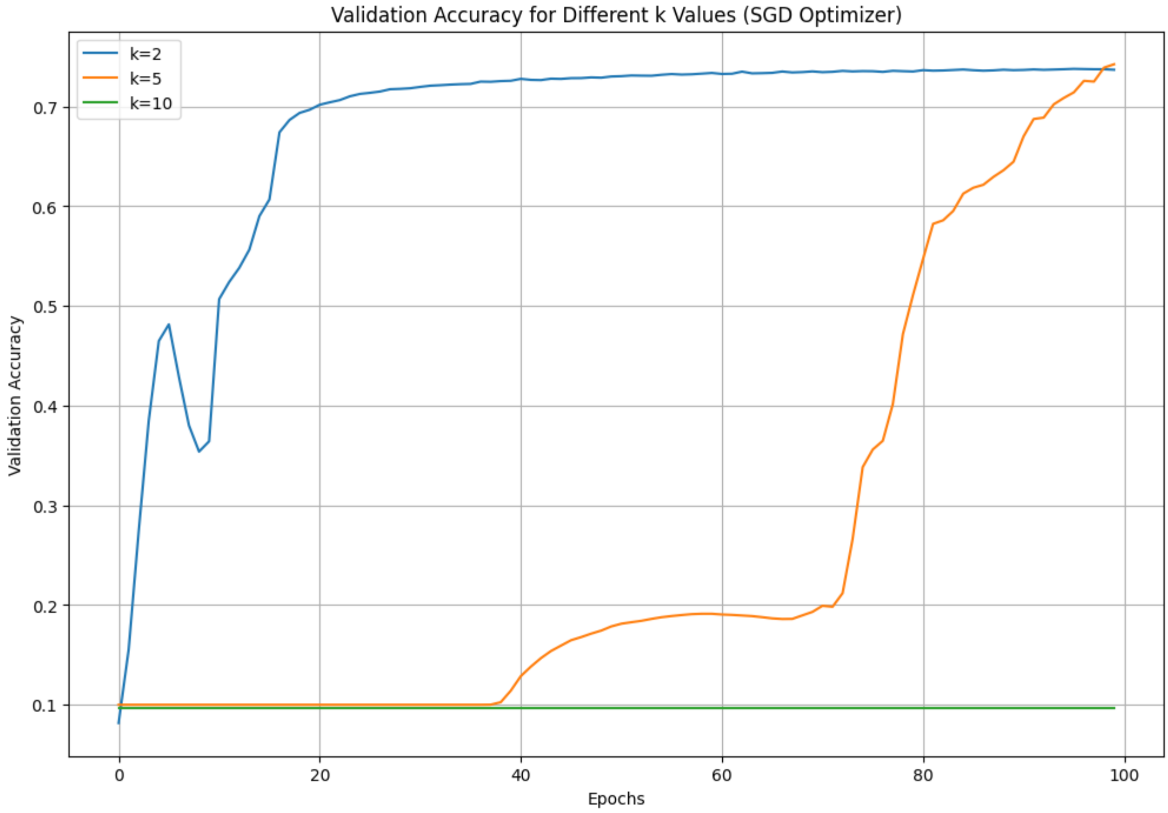
89698561948, 0.8495143113938053, 0.8508616219579647, 0.854240701050885, 0.
8574624930862832, 0.8581011476769911, 0.8591874654314159, 0.86140071902654
86, 0.8611146639933628, 0.8625017284292035, 0.8662835315265486, 0.86876987
69358407, 0.8655541344026549, 0.8660234029314159, 0.8721031526548673, 0.87
16407978429203, 0.8732162610619468, 0.8726303235619468, 0.8777594372234514
, 0.877851043971239, 0.878515625, 0.8793029245022124, 0.880213806692478, 0
.8779746266592919, 0.879218231471239, 0.8749274059734514, 0.87708967090707
97, 0.8825774336283185, 0.8783064850663717, 0.8799407148783185, 0.87909464
87831859, 0.880025407909292, 0.8827468196902656, 0.8807608545353982, 0.877
479431692478, 0.8797842920353982, 0.8755332204092919, 0.8817961836283186,
0.8794653968473451, 0.8792113177544248, 0.8839437569137167, 0.880734928097
3451, 0.8820692754424779, 0.8849531595685841, 0.882850525442478, 0.8822576
742256636, 0.8843024059734514, 0.8849073561946902, 0.882752869192478, 0.88
35341191924779, 0.881073700221239, 0.8841917865044249, 0.8821150788163716,
0.8816717367256637, 0.8814314850663717, 0.8796278691924779, 0.882023472068
5841, 0.8808317201327434, 0.8820493985066372, 0.8827848451327434, 0.886000
5876659292, 0.8848943929756636, 0.8843214186946902, 0.879849108130531, 0.8
84829576880531, 0.8847578470685841, 0.8804938122234514, 0.8819975456305309
, 0.8827978083517699, 0.8839895602876107, 0.8829870713495576, 0.8849592090
707965, 0.8826292865044248, 0.8849989629424779, 0.8840353636615044, 0.8862
097275995575, 0.8830717643805309, 0.8844190749446902, 0.8842047497234514,
0.8821928581305309, 0.8831167035398231, 0.883683628318584, 0.8850240251659
292, 0.8837423949115044, 0.88455562085177, 0.8840682038163716, 0.883357819
4137168, 0.8859098451327434, 0.8846213011615044]}, {'train_losses': [1.939
1490411758423, 1.7017345190048219, 1.6569927883148194, 1.6379427528381347,
1.6253878164291382, 1.6162162733078003, 1.6077009010314942, 1.603294730186
4624, 1.598596272468567, 1.593658037185669, 1.589891686439514, 1.589056267
7383422, 1.5825969552993775, 1.5806571340560913, 1.5779737377166747, 1.576
5964221954345, 1.5730879926681518, 1.5711377382278442, 1.5685000658035277,
1.568650860786438, 1.566154398918152, 1.5619098854064941, 1.56164135456085
2, 1.5602398204803467, 1.55835608959198, 1.5563276386260987, 1.55828306198
12011, 1.5588959312438966, 1.5549491357803344, 1.5541894721984864, 1.55102
3621559143, 1.5491585302352906, 1.5476258325576782, 1.5486376953125, 1.547
893214225769, 1.5483766603469848, 1.5483001852035523, 1.546991400718689, 1
.5422148275375367, 1.5419144964218139, 1.541050500869751, 1.53893047332763
68, 1.5398930501937866, 1.5387225198745726, 1.5422870302200318, 1.53987004
75692748, 1.54247887134552, 1.5377320289611816, 1.5350881671905519, 1.5356
61873817444, 1.5351251745224, 1.5341546154022216, 1.5329350900650025, 1.53
36716747283936, 1.5304371786117554, 1.5277919101715087, 1.5283949232101441
, 1.530180687904358, 1.5287980604171754, 1.5271943235397338, 1.52693953514
09911, 1.5259891033172608, 1.526298213005066, 1.527925238609314, 1.5282400
08354187, 1.5275344896316527, 1.530422806739807l, 1.5266441345214843, 1.52
52146530151367, 1.5251950168609618, 1.5254203796386718, 1.5260533857345582
, 1.527951865196228, 1.5233890295028687, 1.5247842168807983, 1.52327167034
14917, 1.5203036403656005, 1.520896143913269, 1.520479621887207, 1.5189000
272750854, 1.5195200395584108, 1.523554720878601, 1.5197964811325073, 1.51
95857334136962, 1.5203759670257568, 1.5186953830718994, 1.5200030136108398
, 1.519166307449341, 1.5167597389221192, 1.5170713233947755, 1.51929068565
36865, 1.5170779085159303, 1.5139438438415527, 1.5140044736862182, 1.51770
161151886, 1.5162876176834106, 1.5182000589370728, 1.5159592533111572, 1.5
15744342803955, 1.515320372581482], 'val_losses': [1.7318199157714844, 1.6
771112203598022, 1.652111792564392, 1.639249587059021, 1.627892804145813,
1.6214645147323608, 1.611996054649353, 1.6130372047424317, 1.6045491456985
475, 1.6056326389312745, 1.6079801082611085, 1.5979606866836549, 1.5941951
990127563, 1.5920346260070801, 1.598395085334778, 1.5888273954391479, 1.58
93430233001709, 1.586949610710144, 1.588267159461975, 1.588915467262268, 1
.5858299016952515, 1.5865006685256957, 1.584887409210205, 1.58214316368103
04, 1.5811862468719482, 1.5831445932388306, 1.5985884428024293, 1.57945001
1253357, 1.5835878372192382, 1.5782142162322998, 1.584055519104004, 1.5796
173334121704, 1.5783264875411986, 1.580205464363098, 1.581164264678955, 1.

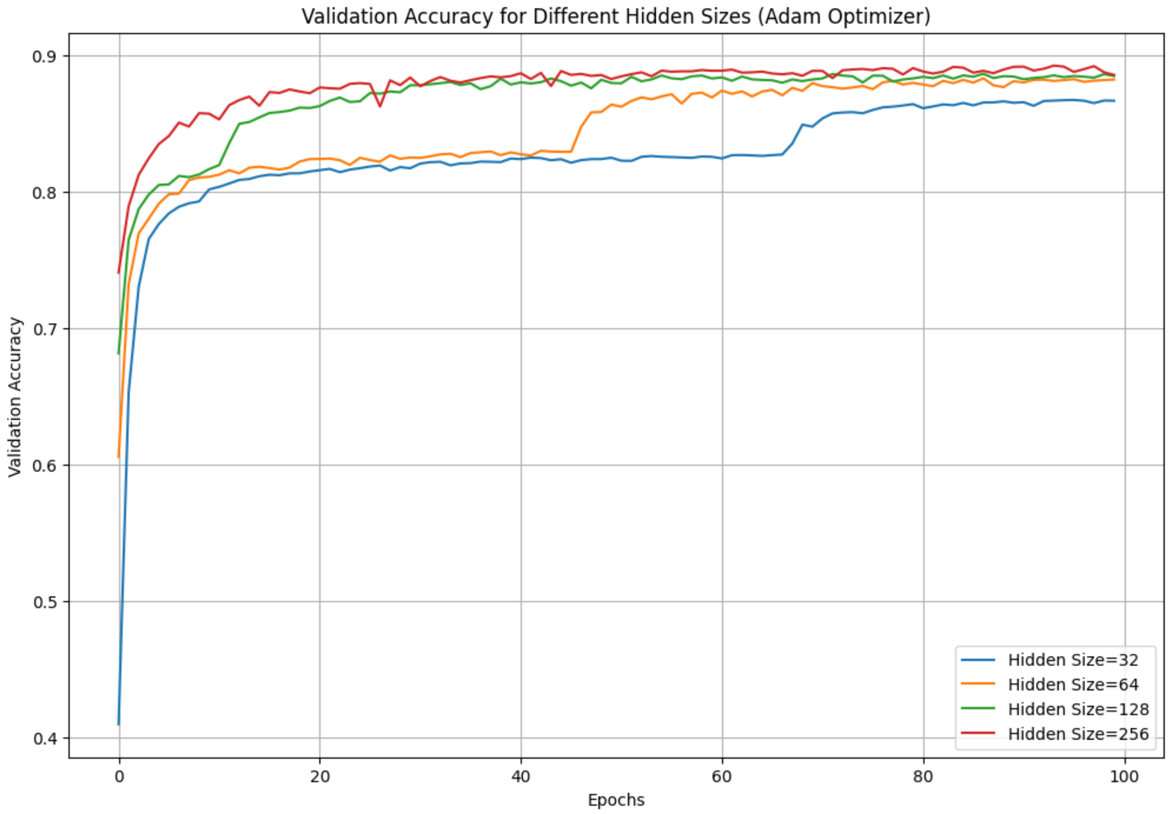
5789693593978882, 1.5770629167556762, 1.5769945621490478, 1.57775278091430
66, 1.5768295764923095, 1.5744629144668578, 1.5782834529876708, 1.57364473
34289552, 1.5833574056625366, 1.5735422372817993, 1.5753206729888916, 1.57
48462438583375, 1.5757315158843994, 1.576244330406189, 1.5780821800231934,
1.576840376853943, 1.574353861808777, 1.5738393783569335, 1.57525734901428
23, 1.572511887550354, 1.572832465171814, 1.5734018802642822, 1.5724486827
850341, 1.5717255115509032, 1.573135757446289, 1.5721918106079102, 1.57223
1650352478, 1.5734341382980346, 1.5740158319473267, 1.572614312171936, 1.5
751601696014403, 1.5753623247146606, 1.5735590696334838, 1.575989723205566
4, 1.5725719213485718, 1.572435235977173, 1.5784170150756835, 1.5721742153
167724, 1.5710911273956298, 1.5709704875946044, 1.5713726043701173, 1.5705
903768539429, 1.5707157611846925, 1.5746825695037843, 1.57058265209198, 1.
5723994255065918, 1.5738993883132935, 1.5726303339004517, 1.56978595256805
42, 1.5695710897445678, 1.572905683517456, 1.5725059270858766, 1.573926043
510437, 1.5715892314910889, 1.5701830863952637, 1.5699885606765747, 1.5717
318296432494, 1.5708453416824342, 1.568836259841919, 1.56981041431427, 1.5
73228907585144, 1.5715716361999512, 1.5693573236465455, 1.5733681678771974
, 1.5747947216033935], 'train_accuracies': [0.5732830925707547, 0.76278117
62971699, 0.8083995430424529, 0.8256552181603773, 0.8381776975235848, 0.84
66947965801888, 0.8553110259433963, 0.8593849498820755, 0.8643344634433963
, 0.8689501031839623, 0.872773068985849, 0.8735793779481132, 0.88021300117
92453, 0.8824329304245283, 0.8851330336084905, 0.8854613797169811, 0.88904
59168632076, 0.8914302034198113, 0.894283608490566, 0.8939740566037736, 0.
8959743514150944, 0.9003552476415095, 0.900579672759434, 0.902173496462264
1, 0.9042213295990567, 0.9062625294811321, 0.9038682930424529, 0.903141951
6509434, 0.9069520194575472, 0.9081028891509433, 0.9113708726415095, 0.913
6479215801887, 0.914616376768868, 0.9135506338443397, 0.9139980100235848,
0.9138141214622642, 0.9139298349056604, 0.9148817069575472, 0.919735038325
4717, 0.9204153154481132, 0.9211903007075471, 0.9233044663915095, 0.922098
3195754716, 0.9230424528301887, 0.9194019015330188, 0.9218691037735849, 0.
9187813237028302, 0.9243886350235848, 0.9268510465801887, 0.92618698408018
87, 0.9271727594339623, 0.9277395341981132, 0.9288719781839623, 0.92825361
14386792, 0.9315400206367924, 0.933795327240566, 0.9337728478773585, 0.931
7563384433963, 0.932974277712264, 0.9344063237028302, 0.9348470666273585,
0.9359323408018868, 0.9355726709905661, 0.9336541863207547, 0.933541789504
7169, 0.9344206957547169, 0.9315142246462265, 0.9349657281839623, 0.936391
5094339623, 0.9367047464622641, 0.9362776385613207, 0.9357926739386793, 0.
9335336821933963, 0.9385366303066037, 0.9369159050707547, 0.93805645636792
45, 0.9411715064858491, 0.9407683520047169, 0.9410694280660377, 0.94288878
24292452, 0.94181640625, 0.9382502948113207, 0.9419284345518868, 0.9420327
240566038, 0.9411866155660377, 0.9429182635613208, 0.9415967718160377, 0.9
422475678066037, 0.9448142688679245, 0.9443263561320755, 0.942250884433962
3, 0.944517983490566, 0.9477387971698112, 0.9476182930424528, 0.9438653449
292452, 0.945071491745283, 0.9430501916273585, 0.9458947523584905, 0.94571
08637971698, 0.9459861438679245], 'val_accuracies': [0.7405238868915929, 0
.7890979327986726, 0.8121353014380531, 0.824223071073009, 0.83468525304203
53, 0.8405956167035399, 0.8504321073008849, 0.8474894565818584, 0.85731298
3960177, 0.8569681623340708, 0.8527421529314159, 0.863217298119469, 0.8669
144081858408, 0.8694534706858408, 0.8627419800884957, 0.872772918971239, 0
.8721619192477876, 0.874726043971239, 0.8732352737831859, 0.87192080337389
39, 0.8761580475663717, 0.875604950221239, 0.875214325221239, 0.8787947663
163717, 0.8793867533185841, 0.878729950221239, 0.8622338219026549, 0.88133
98783185841, 0.8778769704092919, 0.883436462942478, 0.8770239905973451, 0.
880897400442478, 0.8837881982853982, 0.881073700221239, 0.8798102184734514
, 0.8814963011615043, 0.8830588011615044, 0.8842954922566373, 0.8835997995
022125, 0.8844709278207965, 0.8866003525995575, 0.882362244192478, 0.88683
45547566372, 0.8772262168141592, 0.8881948285398231, 0.8853506982853983, 0
.886125034568584, 0.8846403138827433, 0.885291067477876, 0.882350145188053
1, 0.8843214186946902, 0.8859297220685841, 0.8872839463495575, 0.884400062
2234514, 0.8885076742256637, 0.8877203747234514, 0.8879934665376107, 0.888

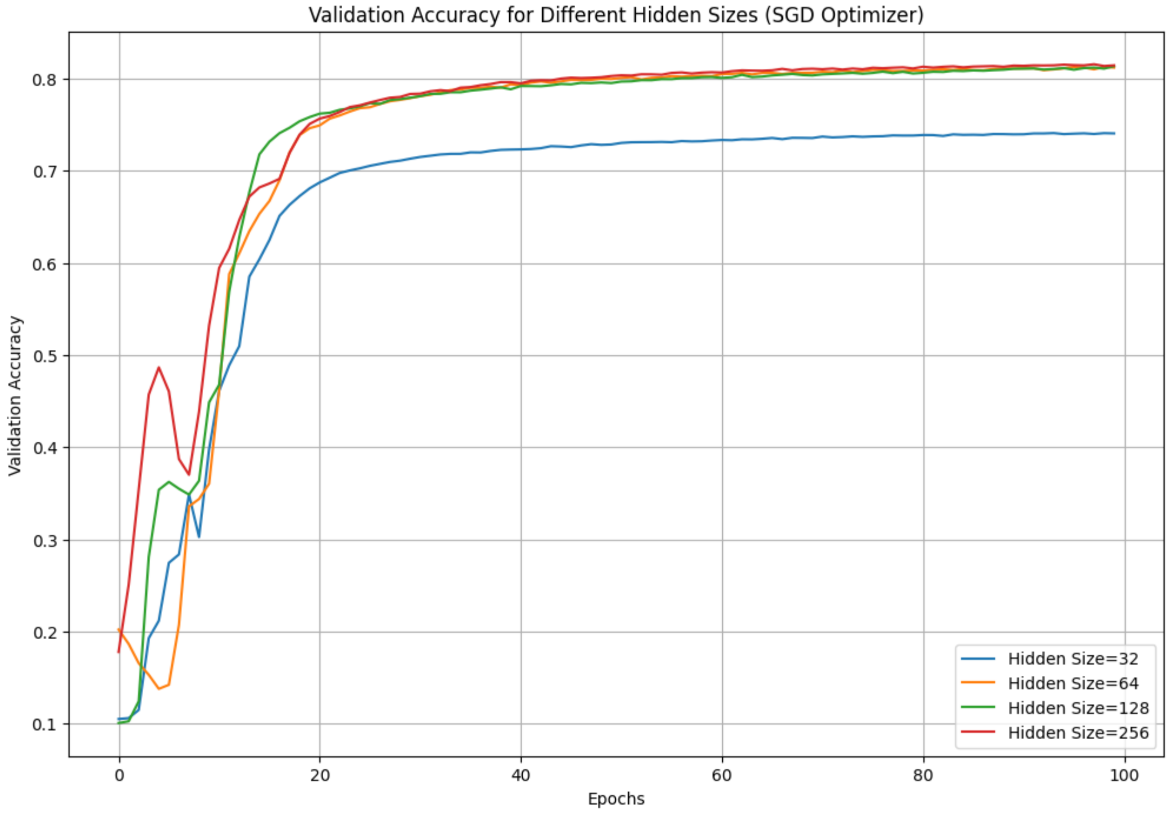
0124792588496, 0.8889302751659292, 0.8884100179756637, 0.8884549571349558, 0.8892431208517699, 0.8870298672566372, 0.8872511061946902, 0.887778277101 7699, 0.88641108960177, 0.8858191025995575, 0.8866262790376107, 0.88474488 38495575, 0.88836421460177, 0.8882734720685841, 0.8830458379424779, 0.8887 678028207965, 0.88934077710177, 0.8897244883849558, 0.8888982992256637, 0. 890271536227876, 0.8898679480088496, 0.885682556692478, 0.8903562292588496 , 0.8878629701327434, 0.8863851631637167, 0.8877134610066373, 0.8912420492 256636, 0.8906880876659292, 0.8870169040376107, 0.8883702641039823, 0.8866 392422566373, 0.8892889242256636, 0.8912878525995576, 0.8914174847898231, 0.8885595271017699, 0.8898938744469026, 0.892198734789823, 0.8914304480088 496, 0.887589878318584, 0.8895749792588497, 0.89178218335177, 0.8873556761 615043, 0.8853766247234514]}])

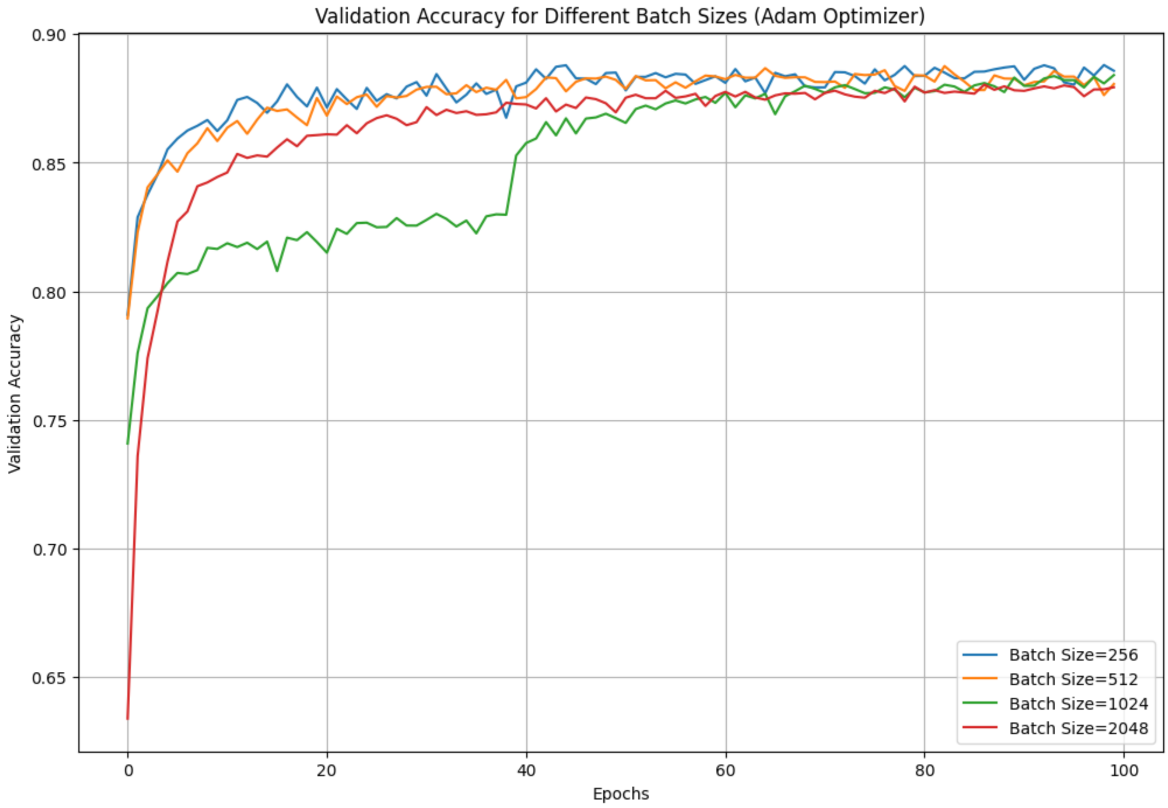
In [28]:
```
plot_results(results_k_adam, 'Validation Accuracy for Different k Values
plot_results(results_k_sgd, 'Validation Accuracy for Different k Values (
plot_results(results_hidden_size_adam, 'Validation Accuracy for Different
plot_results(results_hidden_size_sgd, 'Validation Accuracy for Different
plot_results(results_batch_size_adam, 'Validation Accuracy for Different
plot_results(results_batch_size_sgd, 'Validation Accuracy for Different B
```

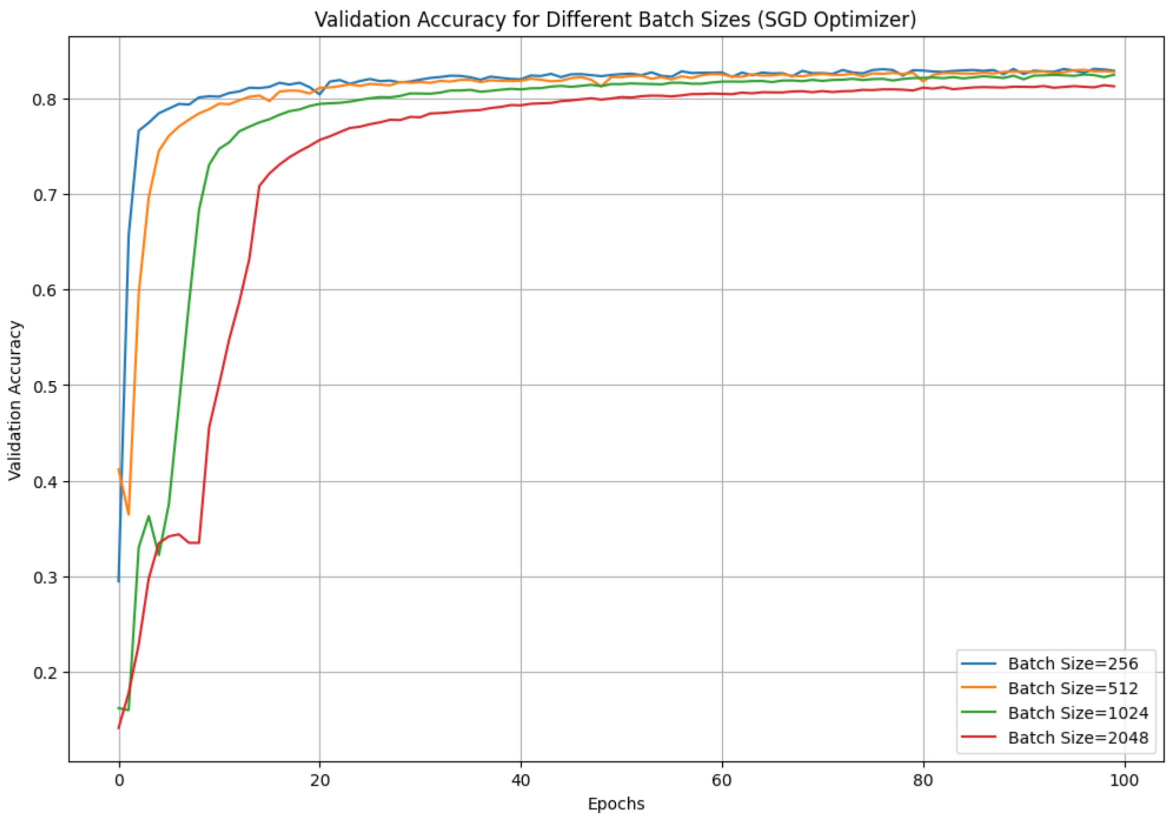



Validation Accuracy for Different k Values (SGD Optimizer)
k=2
k=5
k=10
Validation Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100


Validation Accuracy for Different Hidden Sizes (Adam Optimizer)
Hidden Size=32
Hidden Size=64
Hidden Size=128
Hidden Size=256
Validation Accuracy
Epochs
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100


Validation Accuracy for Different Hidden Sizes (SGD Optimizer)
Validation Accuracy
Epochs
Hidden Size=32
Hidden Size=64
Hidden Size=128
Hidden Size=256


Validation Accuracy for Different Batch Sizes (Adam Optimizer)
Validation Accuracy
Epochs
Batch Size=256
Batch Size=512
Batch Size=1024
Batch Size=2048



TODO: Justify which batch size and learning rate combination you will go with.

For the Batch Size it seems to not make a large difference for the convergence. Smaller Batch size seems to converge faster but you make much more mini opitmization steps, which results in this behavior, so we choose 1024, because 2048 performed slightly worse, if we choose the batch size too small the training takes longer because the CPU and GPU communication takes longer.

For the hidden size 64 performed equally well compared to the others, but we will choose 128 because, the learning seems to not go beyond the 0.75 accurracy mark, and a larger network maybe helps to achieve better accuracies and learns the underlying distribution better.

For the hidden layers k, we choose 5. It seems sufficient to use 2, but we choose more for the same reason as before.

But we train for more epochs than before

2.4

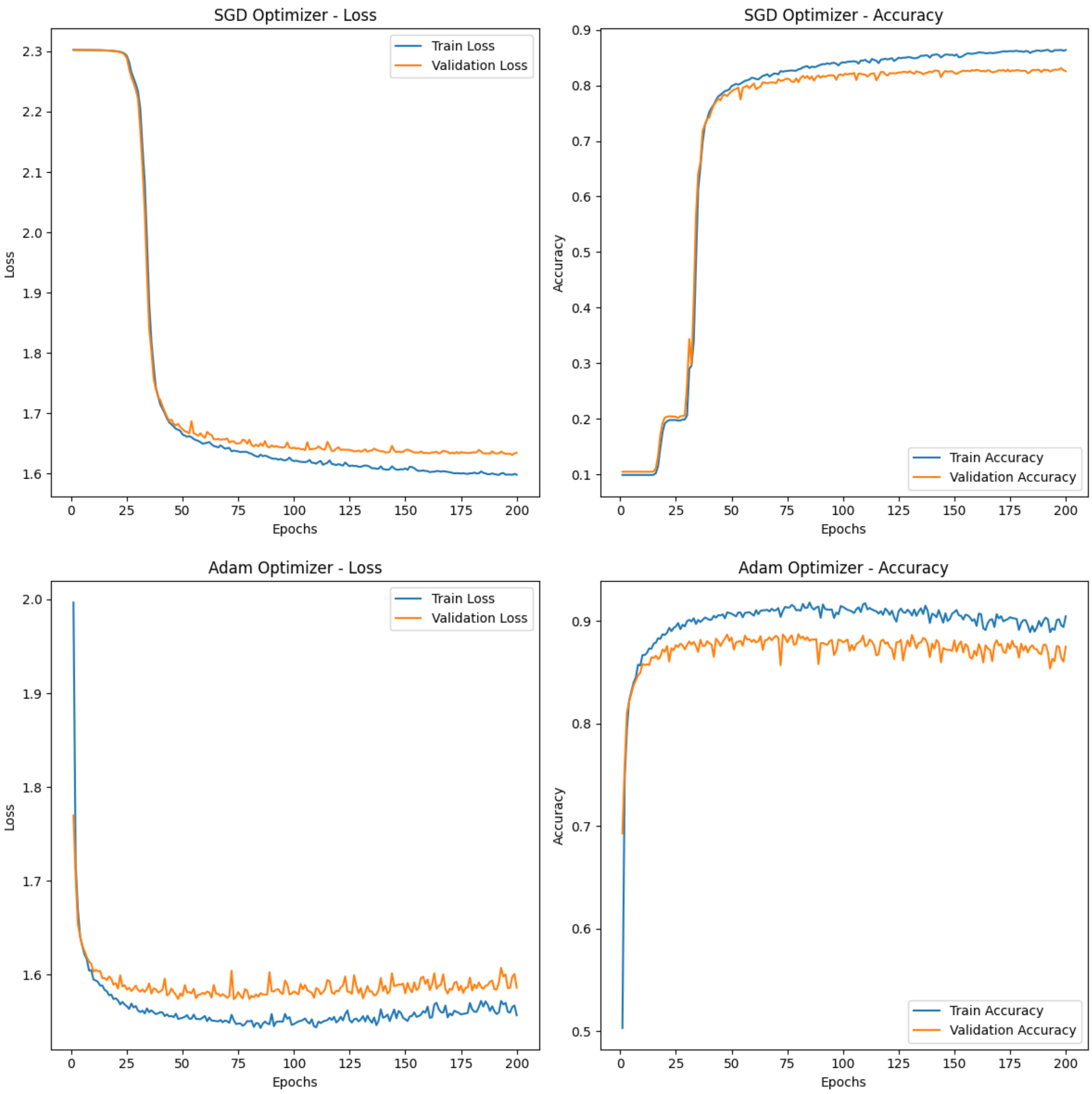
In [32]:
```
# TODO: Train model w/ best hyperparameters for SGD and compare to defaul
best_k = 5
best_hidden_size = 128
best_batch_size = 1024

train_loader = DataLoader(
    train_dataset,
    batch_size=best_batch_size,
    shuffle=True,
    num_workers=12
```

```
)
validation_loader = DataLoader(
    validation_dataset,
    batch_size=best_batch_size,
    shuffle=False,
    num_workers=12
)


model_best_sgd = MLP(input_size=input_size, k=best_k, hidden_size=best_hi
optimizer_best_sgd = optim.SGD(model_best_sgd.parameters(), lr=0.01, mome

print("Training model with best hyperparameters using SGD optimizer")
train_losses_best_sgd, val_losses_best_sgd, train_accuracies_best_sgd, va
    model_best_sgd, criterion, optimizer_best_sgd, train_loader, validati

model_best_adam = MLP(input_size=input_size, k=best_k, hidden_size=best_h
optimizer_best_adam = optim.Adam(model_best_adam.parameters())

print("Training model with best hyperparameters using Adam optimizer")
train_losses_best_adam, val_losses_best_adam, train_accuracies_best_adam,
    model_best_adam, criterion, optimizer_best_adam, train_loader, valida
```

```
Epoch [189/200], Train Loss: 1.5663, Train Accuracy: 0.8948, Val Accuracy:
0.8675
Epoch [190/200], Train Loss: 1.5610, Train Accuracy: 0.9001, Val Accuracy:
0.8766
Epoch [191/200], Train Loss: 1.5580, Train Accuracy: 0.9032, Val Accuracy:
0.8769
Epoch [192/200], Train Loss: 1.5597, Train Accuracy: 0.9014, Val Accuracy:
0.8694
Epoch [193/200], Train Loss: 1.5721, Train Accuracy: 0.8889, Val Accuracy:
0.8535
Epoch [194/200], Train Loss: 1.5682, Train Accuracy: 0.8928, Val Accuracy:
0.8631
Epoch [195/200], Train Loss: 1.5699, Train Accuracy: 0.8912, Val Accuracy:
0.8608
Epoch [196/200], Train Loss: 1.5605, Train Accuracy: 0.9007, Val Accuracy:
0.8755
Epoch [197/200], Train Loss: 1.5596, Train Accuracy: 0.9016, Val Accuracy:
0.8749
Epoch [198/200], Train Loss: 1.5655, Train Accuracy: 0.8957, Val Accuracy:
0.8636
Epoch [199/200], Train Loss: 1.5672, Train Accuracy: 0.8940, Val Accuracy:
0.8604
Epoch [200/200], Train Loss: 1.5567, Train Accuracy: 0.9044, Val Accuracy:
0.8749
```

In [33]:

```python
# TODO: Plot "learning curves" of the best SGD model and the Adam model
def plot_learning_curves(train_losses, val_losses, train_accuracies, val_
    epochs = range(1, len(train_losses) + 1)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_losses, label='Train Loss')
    plt.plot(epochs, val_losses, label='Validation Loss')
    plt.title(f'{title} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_accuracies, label='Train Accuracy')
    plt.plot(epochs, val_accuracies, label='Validation Accuracy')
    plt.title(f'{title} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

plot_learning_curves(train_losses_best_sgd, val_losses_best_sgd, train_ac
plot_learning_curves(train_losses_best_adam, val_losses_best_adam, train_
```



TODO: Based on plots, compare (mini-batch) SGD and Adam, select overall best Model

Select Adam because overall performance is better

In [34]:
```
#TODO: Evaluate the best model on the test set, print the test/train/vali

def evaluate_model(model, test_loader, train_loader, validation_loader):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    model.eval()

    test_accuracy = 0.0
    train_accuracy = 0.0
    validation_accuracy = 0.0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)

            test_accuracy += calculate_accuracy(outputs, labels)

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
```

```
            train_accuracy += calculate_accuracy(outputs, labels)
        for images, labels in validation_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            validation_accuracy += calculate_accuracy(outputs, labels)

    test_accuracy /= len(test_loader)
    train_accuracy /= len(train_loader)
    validation_accuracy /= len(validation_loader)

    return test_accuracy, train_accuracy, validation_accuracy

test_acc_best, train_acc_best, val_acc_best = evaluate_model(
    model_best_adam, test_loader, train_loader, validation_loader
)

print(f"Adam best model test accuracy: {test_acc_best:.4f}")
print(f"Adam best model train accuracy: {train_acc_best:.4f}")
print(f"Adam best model validation accuracy: {val_acc_best:.4f}")
```

```
Adam best model test accuracy: 0.8669
Adam best model train accuracy: 0.9016
Adam best model validation accuracy: 0.8749
```

TODO: Briefly discuss your results.

Adam converged faster than SGD and also had a steeper learning curve. SGD took more epochs to achieve higher accuracies compared to Adam. With more epochs, the results maybe would have been better, but this network is not designed for image classification, odher models would achieve better results.