In [ ]:
```
import torch
from torchvision import datasets, transforms
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
```

In [ ]:
```
trans = transforms.Compose([
    transforms.ToTensor(),  #it is like numpy array for pytorch
    transforms.Normalize((0.5,), (0.5,))  #this normalizes the data betwe
])
#download the data
train_data = datasets.MNIST(
    root="./data",
    train=True,
    transform=trans,
)
test_data = datasets.MNIST(
    root="./data",
    transform=trans,
    train=False,
)
#print the length of the data
print(len(train_data))
print(len(test_data))
```

```
60000
10000
```

In [ ]:
```
#split the data into train and validation using sklearn
train_data, val_data = train_test_split(train_data, test_size=0.2, random
print(len(train_data))
print(len(val_data))
```

```
48000
12000
```

In [ ]:
```
#load the data using dataloader
#what does data loder do?it loads the data in batches-it shuffles the dat
#what is the batch size?it is the number of samples that will be passed t
#what is shuffle?it shuffles the data so that the model will not learn th
batch_sz = 16
train_loader = DataLoader(train_data, batch_size=batch_sz, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_sz, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_sz, shuffle=True)
```

In [ ]:
```
print(len(train_loader))
#750 is the number of batches it came from 48000(the length of the train
print(len(val_loader))
#188 is the number of batches it came from 12000(the length of the valida
print(len(test_loader))
#157 is the number of batches it came from 10000(the length of the test d
```

```
3000
750
625
```

In [ ]:
```
import torch.nn as n
from torch import optim
```

In [ ]:
```
class neural_model(n.Module):
    def __init__(self, lr):
        super().__init__()
        self.layers = n.Sequential(
            n.Flatten(),
            n.Linear(28 * 28, 400),  #input layer_input=28*28=784 output
            n.LayerNorm(400),  # Layer Normalization to speed up the trai
            n.Dropout(0.5),  #droping out the unwanted overfiting feature
            n.ReLU(),
            n.Linear(400, 200),  #input h_layer_1=200 output h_layer_2=10
            n.ReLU(),
            n.Linear(200, 100),  #input h_layer_2=200 output h_layer_2=10
            n.ReLU(),
            n.Linear(100, 10),  #input output_layer=100 output output_lay
        )
        self.loss = n.CrossEntropyLoss()
        self.learning_rate = lr
        self.optimizer = optim.SGD(self.parameters(), lr=self.learning_ra

    def forward(self, x):
        x = self.layers(x)
        return x

    def fit(self, x, y):
        self.optimizer.zero_grad()  #this is to reset the gradients to ze
        y_output = self.forward(x)  #this is to get the predictions
        loss = self.loss(y_output, y)  #this is to get the loss
        loss.backward()  #this is to backpropagate the loss
        self.optimizer.step()  #this is to update the weights
        return loss.item()  #this is to return the loss

    def predict(self, x):
        with torch.no_grad():
            output = self.forward(x)
            _, predicted = torch.max(output, 1)
        return predicted
```

In [ ]:
```
model = neural_model(0.01)
```

In [ ]:
```
#train the model
epochs = 5
train_losses = []
val_losses = []
train_acc = []
val_acc = []
for epoch in range(epochs):
    train_loss = 0
    val_loss = 0
    train_accuracy = 0
    val_accuracy = 0
    model.eval()
    for x, y in train_loader:
        train_loss += model.fit(x, y)
        train_accuracy += torch.sum(model.predict(x) == y).item()

    train_losses.append(train_loss / len(train_loader))
    train_acc.append(train_accuracy / len(train_data))
    for x, y in val_loader:
        val_loss += model.fit(x, y)
```

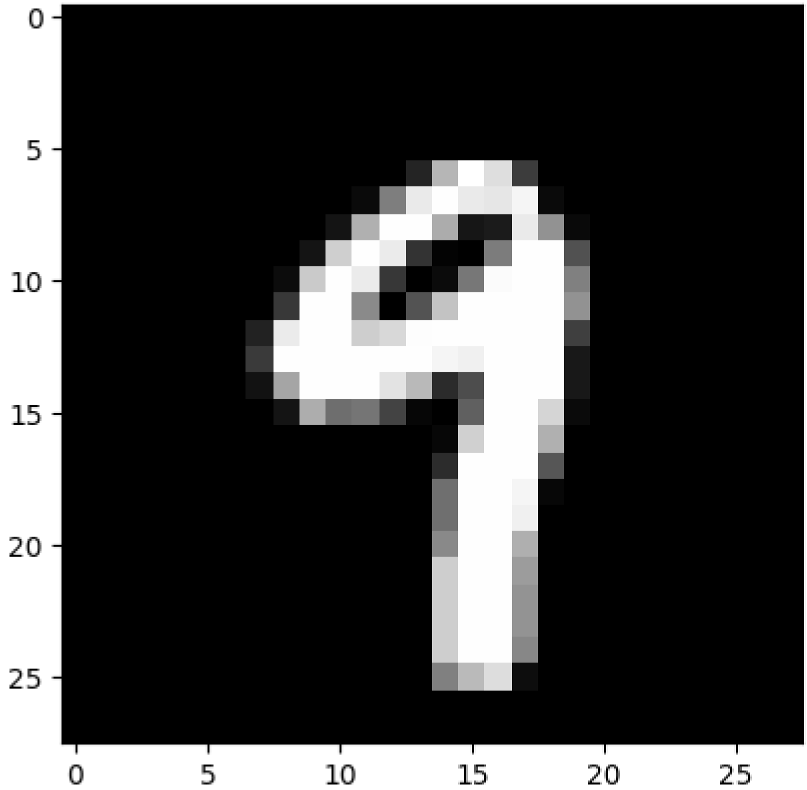
```
        val_accuracy += torch.sum(model.predict(x) == y).item()
    val_losses.append(val_loss / len(val_loader))
    val_acc.append(val_accuracy / len(val_data))
    print(
        f"Epoch {epoch + 1}/{epochs}.. "
        f"Train loss: {train_losses[-1]:.3f}.. "
        f"Val loss: {val_losses[-1]:.3f}.. "
        f"Train accuracy: {train_acc[-1]:.3f}.. "
        f"Val accuracy: {val_acc[-1]:.3f}"
    )
```

```
Epoch 1/5.. Train loss: 0.501.. Val loss: 0.208.. Train accuracy: 0.927..
Val accuracy: 0.988
Epoch 2/5.. Train loss: 0.165.. Val loss: 0.138.. Train accuracy: 0.993..
Val accuracy: 0.994
Epoch 3/5.. Train loss: 0.119.. Val loss: 0.103.. Train accuracy: 0.996..
Val accuracy: 0.996
Epoch 4/5.. Train loss: 0.095.. Val loss: 0.086.. Train accuracy: 0.997..
Val accuracy: 0.997
Epoch 5/5.. Train loss: 0.079.. Val loss: 0.071.. Train accuracy: 0.998..
Val accuracy: 0.999
```

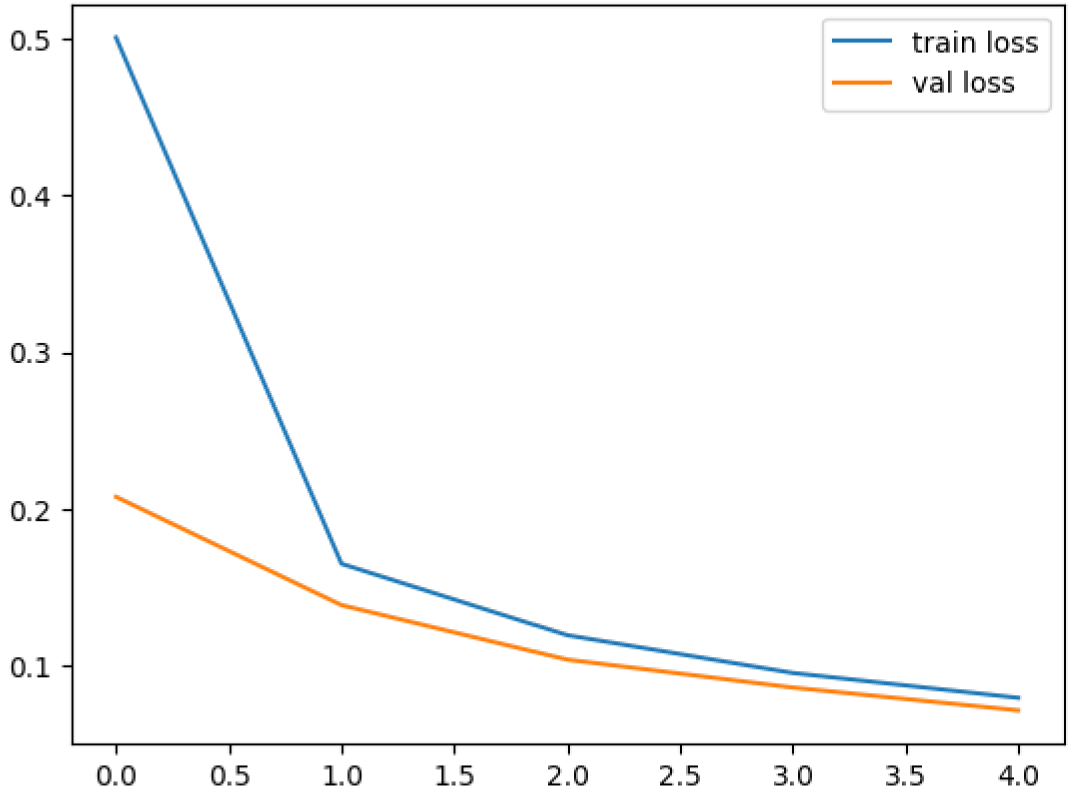
In [ ]:

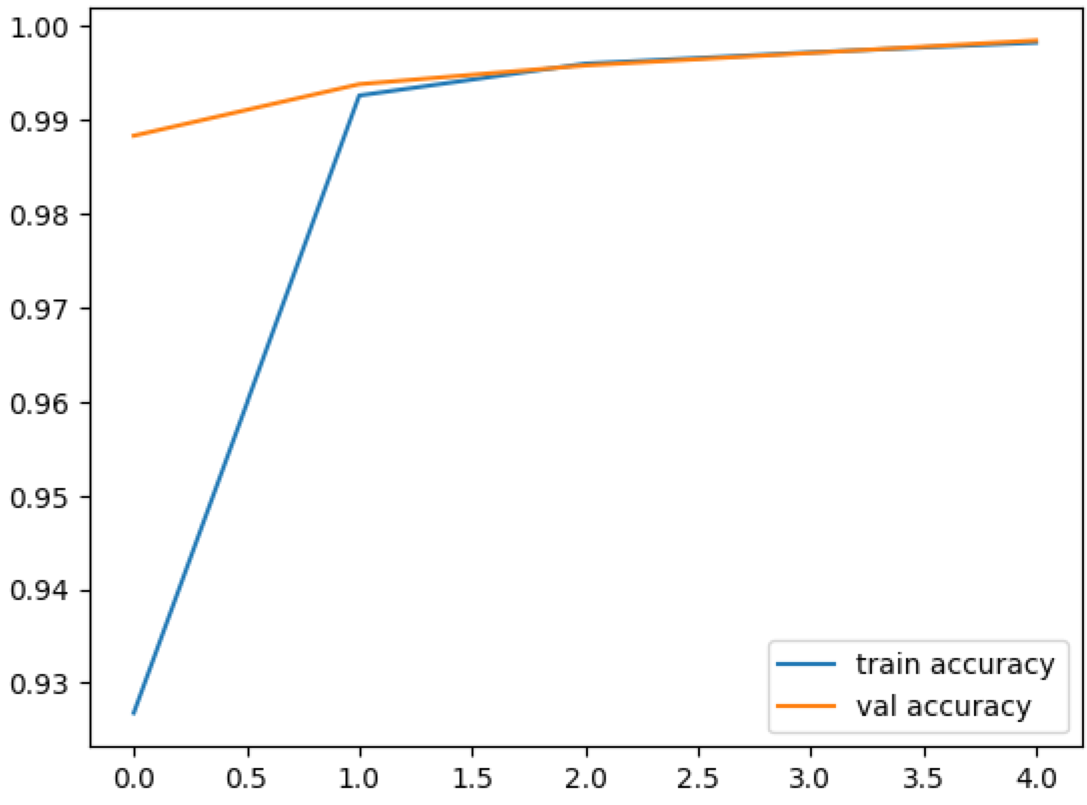
```
from matplotlib import pyplot as plt
import numpy as np
model.eval()

i = np.random.randint(0, len(test_data))

# input = input.type(torch.FloatTensor).to(device)
# input = input.unsqueeze(0)  # Adding batch dimension as models usually

with torch.no_grad():
    input = test_data[i][0].unsqueeze(0).to('cpu')

    output = model(input)

    output = (torch.max(torch.exp(output), 1)[1]).data.cpu().numpy()
#     output = model(input)

# label = torch.argmax(output, dim=1).cpu().numpy()
print("Predicted Label:", [output[0]])

# input_image = input.squeeze().permute(1, 2, 0).cpu().numpy()
plt.imshow(input.squeeze().cpu().numpy(), cmap='gray')  # Show the input
plt.show()
```

```
Predicted Label: [9]
```



In [ ]:
```
# #plot the losses - the loss should decrease
# #plot the accuracy - the accuracy should increase
import matplotlib.pyplot as plt

plt.plot(train_losses, label="train loss")
plt.plot(val_losses, label="val loss")
plt.legend()
plt.show()
plt.plot(train_acc, label="train accuracy")
plt.plot(val_acc, label="val accuracy")
plt.legend()
plt.show()
```

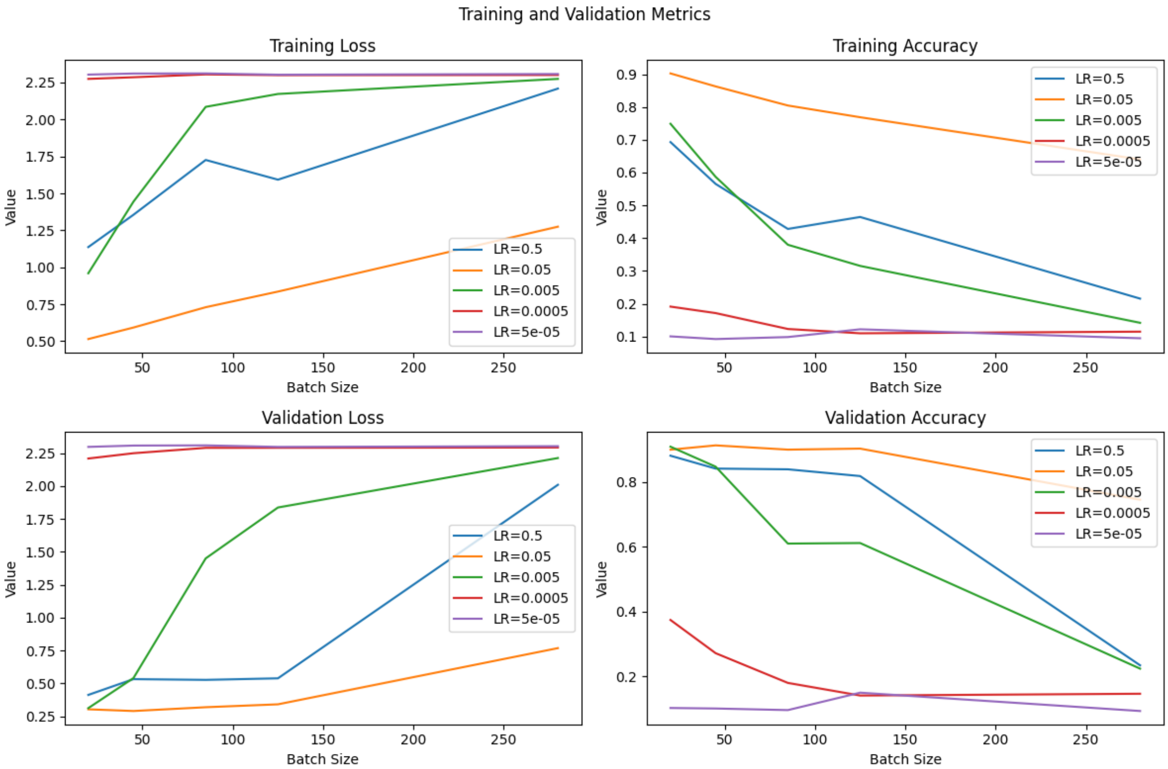




In [ ]:
```
import matplotlib.pyplot as plt

learning_rates = [0.5, 0.05, 0.005, 0.0005, 0.00005]
batch_sizes = [20, 45, 85, 125, 280]
results = []

train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []
```

```
for lr in learning_rates:
    for b in batch_sizes:
        # ... (previous code remains the same)
        model = neural_model(lr)
        train_loss = 0
        val_loss = 0
        train_correct = 0
        val_correct = 0

        train_loader = DataLoader(train_data, batch_size=b, shuffle=True)
        val_loader = DataLoader(val_data, batch_size=b, shuffle=False)

        for x, y in train_loader:
            train_loss += model.fit(x, y)
            y_pred = model.predict(x)
            train_correct += torch.sum(y_pred == y).item()

        train_loss /= len(train_loader)
        train_accuracy = train_correct / len(train_data)

        model.eval()
        with torch.no_grad():
            for x_val, y_val in val_loader:
                output_val = model.forward(x_val)
                val_loss += model.loss(output_val, y_val).item()
                y_pred_val = model.predict(x_val)
                val_correct += torch.sum(y_pred_val == y_val).item()

            val_loss = val_loss / (len(val_loader))
            val_accuracy = (val_correct / len(val_data))


        # Store values in lists
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)

        # append results to list of dictionaries for printing
        results.append({
            'learning_rate': lr,
            'batch_size': b,
            'train_loss': train_loss,
            'train_accuracy': train_accuracy,
            'val_loss': val_loss,
            'val_accuracy': val_accuracy
        })

# Reshape lists for plotting
train_losses = [train_losses[i:i + len(batch_sizes)] for i in range(0, le
train_accuracies = [train_accuracies[i:i + len(batch_sizes)] for i in ran
val_losses = [val_losses[i:i + len(batch_sizes)] for i in range(0, len(va
val_accuracies = [val_accuracies[i:i + len(batch_sizes)] for i in range(0

# Plotting
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
fig.suptitle('Training and Validation Metrics')

for i in range(len(learning_rates)):
```

```
    axs[0, 0].plot(batch_sizes, train_losses[i], label=f"LR={learning_rat
    axs[0, 0].set_title('Training Loss')
    axs[0, 1].plot(batch_sizes, train_accuracies[i], label=f"LR={learning
    axs[0, 1].set_title('Training Accuracy')
    axs[1, 0].plot(batch_sizes, val_losses[i], label=f"LR={learning_rates
    axs[1, 0].set_title('Validation Loss')
    axs[1, 1].plot(batch_sizes, val_accuracies[i], label=f"LR={learning_r
    axs[1, 1].set_title('Validation Accuracy')

for ax in axs.flat:
    ax.set(xlabel='Batch Size', ylabel='Value')
    ax.legend()

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



In [ ]:
```
#print statment of the results
for result in results:
     print(f"lr: {result['learning_rate']}, batch size: {result['batch_si
     print(f"lr: {result['learning_rate']}, batch size: {result['batch_si
     print("-------------------------------------------------")
```

```
lr: 0.5, batch size: 20, train loss: 1.1362699457040677, train accuracy:
0.692875
lr: 0.5, batch size: 20, val loss: 0.41168081014727553, val accuracy: 0.88
05833333333334
-------------------------------------------------
lr: 0.5, batch size: 45, train loss: 1.3551156631729373, train accuracy:
0.5653958333333333
lr: 0.5, batch size: 45, val loss: 0.5318885759802793, val accuracy: 0.840
8333333333333
-------------------------------------------------
lr: 0.5, batch size: 85, train loss: 1.7260847787941451, train accuracy:
0.428125
lr: 0.5, batch size: 85, val loss: 0.5262178303280347, val accuracy: 0.838
5833333333333
-------------------------------------------------
lr: 0.5, batch size: 125, train loss: 1.5926295218523592, train accuracy:
0.46460416666666665
lr: 0.5, batch size: 125, val loss: 0.5379975587129593, val accuracy: 0.81
775
-------------------------------------------------
lr: 0.5, batch size: 280, train loss: 2.209247794955276, train accuracy:
0.21572916666666667
lr: 0.5, batch size: 280, val loss: 2.009900459023409, val accuracy: 0.233
58333333333334
-------------------------------------------------
lr: 0.05, batch size: 20, train loss: 0.5137568657961674, train accuracy:
0.9022708333333334
lr: 0.05, batch size: 20, val loss: 0.3023469832601647, val accuracy: 0.89
91666666666667
-------------------------------------------------
lr: 0.05, batch size: 45, train loss: 0.5917083802087462, train accuracy:
0.8626666666666667
lr: 0.05, batch size: 45, val loss: 0.28931753400169063, val accuracy: 0.9
125
-------------------------------------------------
lr: 0.05, batch size: 85, train loss: 0.7293825796220155, train accuracy:
0.8044166666666667
lr: 0.05, batch size: 85, val loss: 0.31812720926819554, val accuracy: 0.8
993333333333333
-------------------------------------------------
lr: 0.05, batch size: 125, train loss: 0.835186971972386, train accuracy:
0.7687083333333333
lr: 0.05, batch size: 125, val loss: 0.34011754222835106, val accuracy: 0.
9025
-------------------------------------------------
lr: 0.05, batch size: 280, train loss: 1.2744344694669856, train accuracy:
0.6399166666666667
lr: 0.05, batch size: 280, val loss: 0.7676215074783148, val accuracy: 0.7
4475
-------------------------------------------------
lr: 0.005, batch size: 20, train loss: 0.9590186238335445, train accuracy:
0.7488333333333334
lr: 0.005, batch size: 20, val loss: 0.31111472891643643, val accuracy: 0.
9084166666666667
-------------------------------------------------
lr: 0.005, batch size: 45, train loss: 1.4438614030809411, train accuracy:
0.5869583333333334
lr: 0.005, batch size: 45, val loss: 0.5399252066897989, val accuracy: 0.8
4675
-------------------------------------------------
```

```
lr: 0.005, batch size: 85, train loss: 2.0856188046193758, train accuracy:
0.379875
lr: 0.005, batch size: 85, val loss: 1.4501339171973753, val accuracy: 0.6
093333333333333
--------------------------------------------------
lr: 0.005, batch size: 125, train loss: 2.1728346260885396, train accurac
y: 0.31547916666666664
lr: 0.005, batch size: 125, val loss: 1.8370666454235713, val accuracy: 0.
611
--------------------------------------------------
lr: 0.005, batch size: 280, train loss: 2.2747964512470156, train accurac
y: 0.1419375
lr: 0.005, batch size: 280, val loss: 2.2132574292116387, val accuracy: 0.
22333333333333333
--------------------------------------------------
lr: 0.0005, batch size: 20, train loss: 2.2748840550581613, train accurac
y: 0.1914375
lr: 0.0005, batch size: 20, val loss: 2.2096824280420937, val accuracy: 0.
3735
--------------------------------------------------
lr: 0.0005, batch size: 45, train loss: 2.285951790456584, train accuracy:
0.171625
lr: 0.0005, batch size: 45, val loss: 2.2498864859677434, val accuracy: 0.
271
--------------------------------------------------
lr: 0.0005, batch size: 85, train loss: 2.3051325447791444, train accurac
y: 0.12310416666666667
lr: 0.0005, batch size: 85, val loss: 2.2906570703210964, val accuracy: 0.
17925
--------------------------------------------------
lr: 0.0005, batch size: 125, train loss: 2.30012045117716, train accuracy:
0.10995833333333334
lr: 0.0005, batch size: 125, val loss: 2.2911691119273505, val accuracy:
0.14025
--------------------------------------------------
lr: 0.0005, batch size: 280, train loss: 2.301095573014991, train accurac
y: 0.1149375
lr: 0.0005, batch size: 280, val loss: 2.2935615029445913, val accuracy:
0.14575
--------------------------------------------------
lr: 5e-05, batch size: 20, train loss: 2.3038235036532084, train accuracy:
0.10060416666666666
lr: 5e-05, batch size: 20, val loss: 2.298100584745407, val accuracy: 0.10
183333333333333
--------------------------------------------------
lr: 5e-05, batch size: 45, train loss: 2.3111491225652863, train accuracy:
0.09241666666666666
lr: 5e-05, batch size: 45, val loss: 2.3081786266426914, val accuracy: 0.1
0025
--------------------------------------------------
lr: 5e-05, batch size: 85, train loss: 2.311866031072836, train accuracy:
0.09852083333333334
lr: 5e-05, batch size: 85, val loss: 2.3096442373705584, val accuracy: 0.0
9533333333333334
--------------------------------------------------
lr: 5e-05, batch size: 125, train loss: 2.303193747997284, train accuracy:
0.122125
lr: 5e-05, batch size: 125, val loss: 2.2980536594986916, val accuracy: 0.
14908333333333335
--------------------------------------------------
```

```
lr: 5e-05, batch size: 280, train loss: 2.3079203256340914, train accurac
y: 0.094875
lr: 5e-05, batch size: 280, val loss: 2.303438341894815, val accuracy: 0.0
925
-------------------------------------------------
```

insights

in this project working with neural networks we concluded the following: to prevent overfitting we need to work on finding the best learning rate "lr" which for us was 0.05 with validation accuracy of 91% to even give it a further boost we used a dropout layer set to "0.5" to drop the unnecarry features causing overfitting, new accuracy jumped to 93% unnecessary*

here what we used:

- used layer normalization to speed up the training process
- used relu activation function to speed up the training process
- used softmax activation function to get the probabilities of the output layer
- used cross entropy loss function to calculate the loss
- used stochastic gradient descent optimizer to update the weights
- used the dataloader to load the data in batches
- used the train_test_split to split the data into train and validation