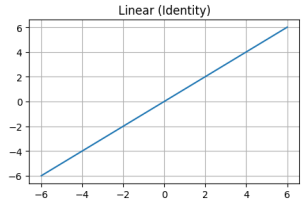
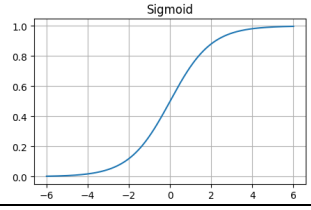
Лабораторна робота № 4. Оптимізація нейронних мереж: дослідження функцій активації та стратегій оптимізації

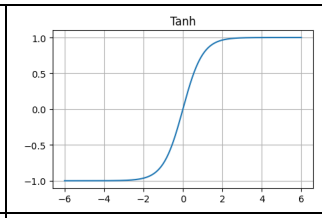
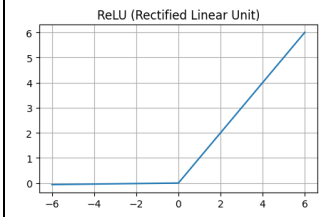
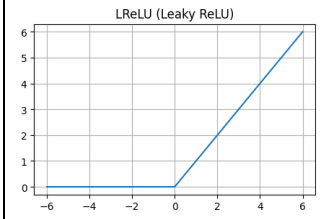
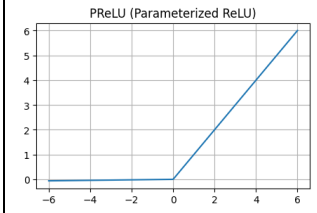
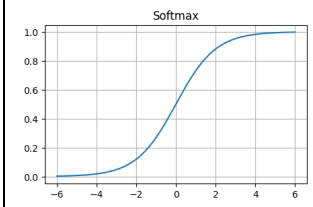
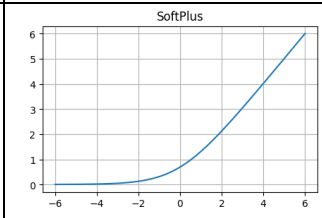
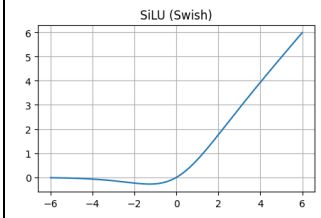
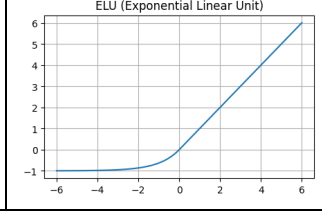
Мета: сформулювати практичні навички у виборі та застосуванні різноманітних функцій активації та оптимізаторів для ефективного навчання нейронних мереж. Лабораторна робота включає в себе аналітику та порівняння впливів різних комбінацій функцій активації та оптимізаторів на процес навчання та загальну продуктивність моделі на різних датасетах.

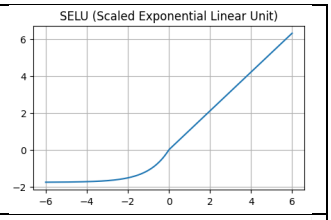
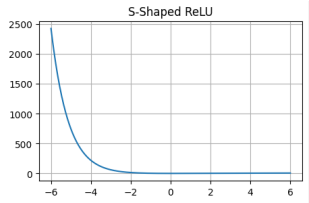
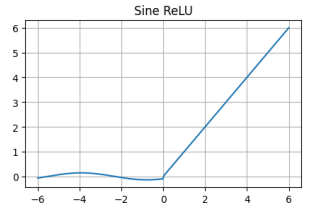
Теоретичні відомості

Оптимізація нейронних мереж є ключовим елементом у глибокому навчанні, що дозволяє моделям ефективно навчатися, адаптуючись до складних даних та вирішуючи різноманітні задачі. Центральними концепціями в оптимізації є функції активації та алгоритми оптимізації.

Функції активації відіграють критичну роль у нейронних мережах, додаючи нелінійність до моделі, що дозволяє їй виявляти складніші патерни в даних. Вони визначають, як нейрон буде активований, залежно від суми вхідних сигналів. Популярні функції активації включають ReLU, Sigmoid, Tanh, Leaky ReLU, та інші, кожна з яких має свої переваги та сценарії застосування.

№	Функція активації	Опис	Використання	
1	Linear (або Identity)	Повертає вхід без змін. $f(x) = x$	Застосовується в випадках, коли потрібно зберегти розподіл вхідних даних, наприклад, у вихідному шарі для задач регресії.	
2	Sigmoid	Приводить вихідні значення до діапазону між 0 та 1. $\frac{1}{1+e^{-x}}$	Часто використовується в вихідному шарі для бінарної класифікації.	

3	Tanh (гіперболічний тангенс)	Схожа на сигмоїд, але виводить значення між -1 та 1. $f(x) = \tanh(x)$	Добре підходить для прихованих шарів завдяки своїй центрованості близько нуля.	
4	ReLU (Rectified Linear Unit)	Виводить вхід, якщо він позитивний; інакше повертає нуль. $f(x) = \max(0, x)$	Найпопулярніша функція для прихованих шарів у глибоких нейронних мережах завдяки простоті та ефективності.	
5	LReLU (Leaky ReLU)	Схожа на ReLU, але пропускає маленьке значення для негативних вхідних даних. $f(x) = \max(0.01x, x)$	Допомагає уникнути проблеми «мертвих нейронів», яка може виникати при використанні ReLU.	
6	PReLU (Parameterized ReLU)	Загальний випадок Leaky ReLU, де коефіцієнт негативних значень є параметром, який може навчатися.	Дозволяє мережі самостійно визначати оптимальний коефіцієнт для негативних значень.	
7	Softmax	Перетворює вектор чисел на вектор ймовірностей, що сумуються в 1.	Застосовується у вихідному шарі для багатокласової класифікації.	
8	SoftPlus	Гладка версія ReLU. $f(x) = \ln(1 + e^x)$	Може використовуватися як альтернатива ReLU, надаючи гладку апроксимацію.	
9	SiLU (Swish)	Самоштовхуюча функція активації. $f(x) = x \cdot \sigma(x)$	Демонструє хороші результати в деяких задачах, може використовуватися як альтернатива ReLU.	
10	ELU (Exponential Linear Unit)	Подібна до ReLU, але з експоненціальним нахилом для негативних вхідних даних.	Допомагає зменшити зсув активацій, може покращити швидкість навчання.	

11	SELU (Scaled Exponential Linear Unit)	Самонормалізуюча версія ELU.	Забезпечує самонормалізацію вхідних даних, ефективна для глибоких мереж.	
12	S-Shaped ReLU	Варіація ReLU з S-подібною формою для покращення нелінійності.	Експериментальна функція, яка може покращити навчання в певних задачах.	
13	Sine ReLU	Використовує синусоїду для активації негативних значень замість лінійного або експоненційного нахилу.	Експериментальна функція для введення періодичності в активації, може бути корисною в специфічних задачах.	

Оптимізатори керують процесом оновлення ваг моделі в процесі навчання, з метою мінімізації функції втрат. Вибір оптимізатора може значно вплинути на швидкість навчання та здатність моделі досягати глобального мінімуму. Оптимізатори, такі як SGD, Adam, RMSprop, та інші, різняться за своїми стратегіями оновлення ваг та адаптивністю до змін у градієнтах.

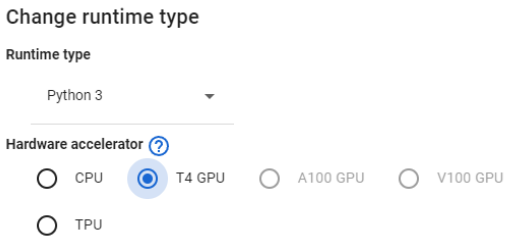
№	Оптимізатор	Опис	Використання
1	SGD	Класичний метод оптимізації, який оновлює ваги моделі на основі градієнта втрати по відношенню до кожного ваги.	Використовується для навчання різноманітних типів нейромереж завдяки своїй простоті та ефективності на великих датасетах.
2	Adam	Алгоритм оптимізації, який адаптивно коригує швидкість навчання для кожного параметра.	Дуже популярний в глибокому навчанні завдяки здатності швидко збігатися та ефективно працювати на різноманітних задачах.
3	AdamW	Варіант оптимізатора Adam, який включає в себе вагову регуляризацію в процес оновлення параметрів, що дозволяє краще контролювати перенавчання.	Використовується в задачах, де важливо зменшити вплив перенавчання та покращити загальну узагальнювану здатність моделі.
4	Adagrad	Алгоритм, що адаптує швидкість навчання для кожного параметра на основі частоти його оновлення, зменшуючи швидкість для часто оновлюваних параметрів.	Корисний при роботі з рідкісними даними, але може бути менш ефективний для тривалого навчання через необоротне зменшення швидкості навчання.
5	RMSprop	Алгоритм оптимізації, який налаштовує швидкість навчання, використовуючи квадратичний середній корінь останніх градієнтів.	Ефективний на невеликих датасетах і в задачах, де потрібна адаптивність швидкості навчання.

6	Adadelta	Вдосконалення Adagrad, яке намагається уникнути його агресивного, монотонного зменшення швидкості навчання.	Використовується для задач, де необхідно більш стабільне та адаптивне оновлення швидкості навчання.
7	Nadam	Комбінація Adam та Nesterov accelerated gradient (NAG), яка використовує передбачувальний градієнт для покращення збіжності.	Ефективний для широкого спектра задач глибокого навчання, особливо коли потрібна швидка збіжність.

Важливим аспектом ефективної оптимізації є розуміння, як вибір функції активації та оптимізатора впливає на здатність моделі до навчання, її збіжність, та уникнення проблем, таких як перенавчання або «мертві нейрони». Експериментування з різними комбінаціями може допомогти визначити оптимальні параметри для конкретної задачі.

Рекомендації до лабораторної роботи

При виконанні лабораторної роботи в Google Colab використовуйте «T4 GPU». Для цього оберіть «Runtime» → «Change runtime type» → «T4 GPU».



Приклад виконання роботи

Завдання

1. Створіть модель для класифікації зображень з датасету відповідно до вашого варіанту.
2. Застосуйте три функції активації відповідно до вашого варіанту у прихованих шарах вашої моделі та порівняйте їх вплив на точність класифікації та швидкість збіжності.
3. Використайте кожен з трьох оптимізаторів відповідно до вашого варіанту для тренування вашої моделі та проаналізуйте їх ефективність.
4. Зібрані результати тренування та тестування представте у вигляді порівняльної таблиці. Проаналізуйте, як комбінація функції активації та оптимізатора впливає на загальну продуктивність моделі на датасеті. Зробіть висновки про оптимальні стратегії навчання для цього конкретного датасету.

Варіант	Датасет	Функція активації №1	Функція активації №2	Функція активації №3	Опти-мізатор №1	Опти-мізатор №2	Опти-мізатор №3
T	MNIST	SiLU	SoftPlus	PReLU	Nadam	AdamW	SGD

Контрольне запитання №1

Вкажіть показник Test Accuracy для комбінації другої функції активації та другого оптимізатора.
Відповідь: _____

Контрольне запитання №2

Вкажіть показник Test Loss для комбінації третьої функції активації та першого оптимізатора.
Відповідь: _____

Контрольне запитання №3

Вкажіть найбільш ефективну комбінацію функції активації та оптимізатора за показником точності.
Відповідь: _____

Контрольне запитання №4

Як правильно налаштувати трансформації для зображень з 3 кольоровими каналами (RGB) в Pytorch, використовуючи transforms.Compose?
Відповідь: _____

Хід роботи

1. Встановлення необхідних бібліотек та модулів, завантаження та підготовка набору даних MNIST

```
# Завдання №1. Створення моделі для класифікації зображень з MNIST
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim

transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))]) # 2

train_data = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

Опишемо архітектуру нейронної мережі «Net», що містить 3 повністю з’єднанні шари для класифікації зображень розміром 28x28 пікселів у 10 класів.

```
class Net(nn.Module):
    def __init__(self, activation_func=F.relu):
        super(Net, self).__init__()
        self.activation_func = activation_func
        self.fc1 = nn.Linear(28*28, 512) # 1, 2
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 10) # 3

    def forward(self, x):
        x = x.view(-1, 28*28) # 1, 2
        x = self.fc1(x)
        if self.activation_func is not None:
            x = self.activation_func(x)
        x = self.fc2(x)
        if self.activation_func is not None:
            x = self.activation_func(x, dim=1) if self.activation_func ==
F.softmax else self.activation_func(x)
```

```
        x = self.fc3(x)
        return x

class SShapedReLU(nn.Module):
    def __init__(self):
        super(SShapedReLU, self).__init__()

    def forward(self, x):
        return x * (torch.sigmoid(x) - torch.sigmoid(-x))

class SineReLU(nn.Module):
    def __init__(self, beta=0.1):
        super(SineReLU, self).__init__()
        self.beta = beta

    def forward(self, x):
        return torch.where(x > 0, x, self.beta * (torch.sin(x) -
torch.cos(x)))
```

Опишемо процес тренування та тестування нейронної мережі, обчислюючи втрати та точність на навчальному та тестовому наборах.

```
def train(model, train_loader, optimizer, epoch):
    model.train()
    total_loss = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

        if batch_idx % 100 == 0:
            print(f'Train Epoch: {epoch} [{batch_idx *
len(data)}/{len(train_loader.dataset)} ({100. * batch_idx /
len(train_loader):.0f}%)]\tLoss: {loss.item():.6f}')

    average_loss = total_loss / len(train_loader)
    return average_loss

def test(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
```

```
            test_loss += F.cross_entropy(output, target,
reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    test_accuracy = correct / len(test_loader.dataset)

    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy:
{correct}/{len(test_loader.dataset)} ({100. * correct /
len(test_loader.dataset):.0f}%)\n')
    return test_loss, test_accuracy
```

Опишемо доступний список функцій активації та список оптимізаторів.

```
from ipywidgets import interact, Dropdown, widgets

# Список функцій активації
activation_functions = {
    'Linear': None,
    'ReLU': F.relu,
    'Sigmoid': torch.sigmoid,
    'Tanh': torch.tanh,
    'Leaky ReLU': nn.LeakyReLU(),
    'Parameterized ReLU': nn.PReLU(),
    'Softmax': F.softmax,
    'SoftPlus': F.softplus,
    'SiLU (Swish)': F.silu,
    'ELU': F.elu,
    'SELU': F.selu,
    'S-Shaped ReLU': SShapedReLU(),
    'Sine ReLU': SineReLU(beta=0.1)
}

# Список оптимізаторів
optimizers_dict = {
    'SGD': optim.SGD,
    'Adam': optim.Adam,
    'AdamW': optim.AdamW,
    'RMSprop': optim.RMSprop,
    'Adagrad': optim.Adagrad,
    'Adadelta': optim.Adadelta,
    'Nadam': optim.NAdam
}
```

2. Ініціалізуємо та виконаємо експеримент з навчання нейронної мережі, використовуючи функції активації SiLU, SoftPlus, PReLU та оптимізатори Nadam, AdamW, SGD.

```
# Завдання №2. Застосуємо функції активації SiLU, Soft, Plus, PReLU
# Завдання №3. Використаємо оптимізатори Nadam, AdamW, SGD
results = []

def run_experiment(activation='ReLU', optimizer='SGD'):
    optimizer_label = optimizer
    if activation == 'Softmax':
        # Логіка для Softmax
        activation_func = lambda x: F.softmax(x, dim=1)
    else:
        activation_func = activation_functions[activation] if activation
!= 'Linear' else lambda x: x
    model = Net(activation_func=activation_func)
    optimizer = optimizers_dict[optimizer](model.parameters(), lr=0.01)

    epoch_losses = []

    for epoch in range(1, 2):
        train_loss = train(model, train_loader, optimizer, epoch)
        epoch_losses.append(train_loss)

    test_loss, test_accuracy = test(model, test_loader)

    results.append({
        'Activation': activation,
        'Optimizer': optimizer_label,
        'Train Loss': sum(epoch_losses) / len(epoch_losses),
        'Test Loss': test_loss,
        'Test Accuracy': test_accuracy
    })

activation_dropdown =
widgets.Dropdown(options=list(activation_functions.keys()),
description='Activation:')
optimizer_dropdown =
widgets.Dropdown(options=list(optimizers_dict.keys()),
description='Optimizer:')

start_button = widgets.Button(description="Start Experiment")

output = widgets.Output()
```

```
def on_start_button_clicked(b):
    with output:
        output.clear_output()
        run_experiment(activation=activation_dropdown.value,
optimizer=optimizer_dropdown.value)

start_button.on_click(on_start_button_clicked)

display(activation_dropdown, optimizer_dropdown, start_button, output)
```

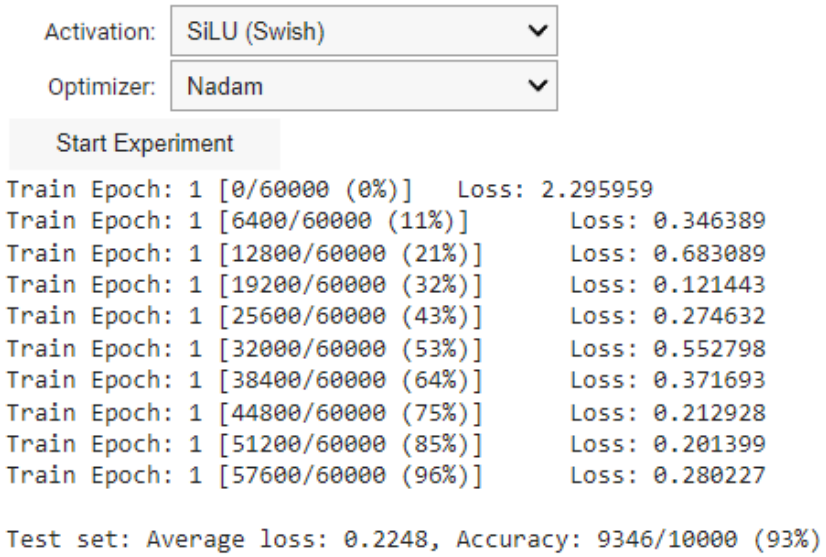


Рисунок 3.1. Результат класифікації з використанням функції активації SiLU та оптимізатора Nadam

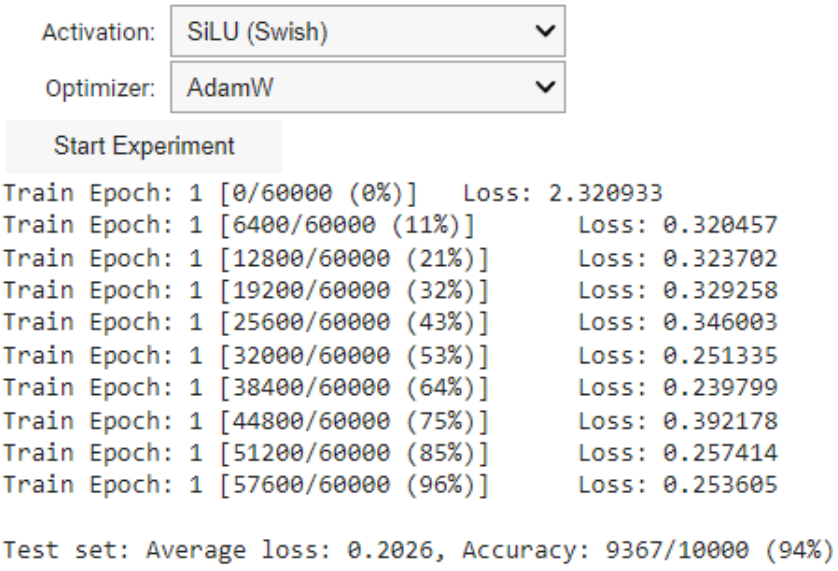


Рисунок 3.2. Результат класифікації з використанням функції активації SiLU та оптимізатора AdamW

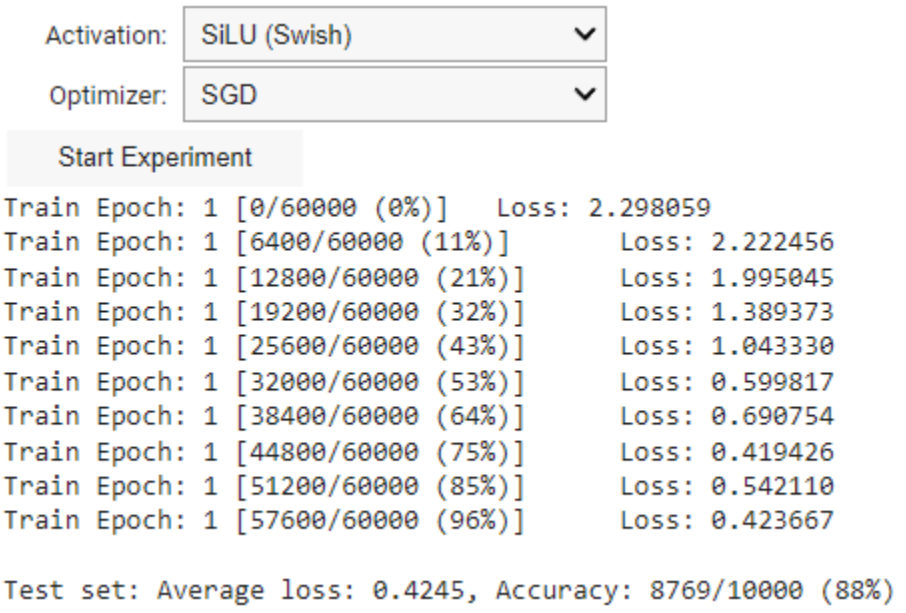


Рисунок 3.3. Результат класифікації з використанням функції активації SiLU та оптимізатора SGD

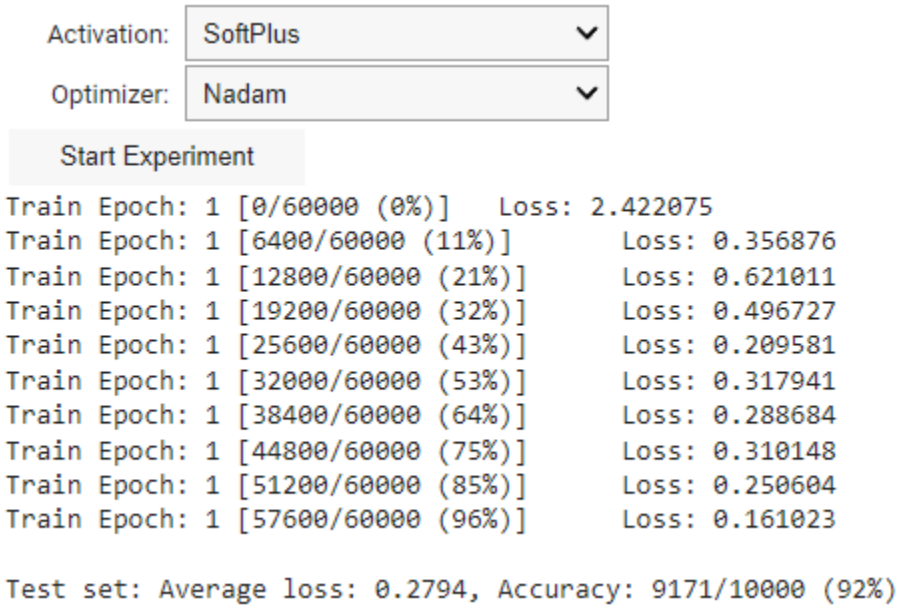
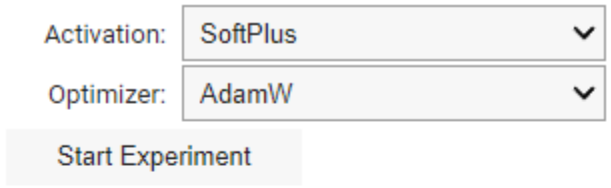


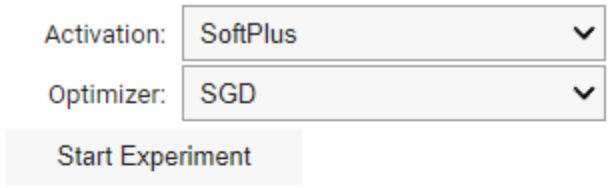
Рисунок 3.4. Результат класифікації з використанням функції активації SoftPlus та оптимізатора Nadam




```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.376677
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.398250
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.464113
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.483108
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.466702
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.338021
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.290466
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.352154
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.258126
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.547350

Test set: Average loss: 0.3387, Accuracy: 8931/10000 (89%)
```

Рисунок 3.5. Результат класифікації з використанням функції активації SoftPlus та оптимізатора AdamW




```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.437082
Train Epoch: 1 [6400/60000 (11%)]       Loss: 2.234930
Train Epoch: 1 [12800/60000 (21%)]      Loss: 2.190727
Train Epoch: 1 [19200/60000 (32%)]      Loss: 1.976690
Train Epoch: 1 [25600/60000 (43%)]      Loss: 1.326463
Train Epoch: 1 [32000/60000 (53%)]      Loss: 1.060935
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.861820
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.560046
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.518664
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.703482

Test set: Average loss: 0.6155, Accuracy: 7834/10000 (78%)
```

Рисунок 3.6. Результат класифікації з використанням функції активації SoftPlus та оптимізатора SGD

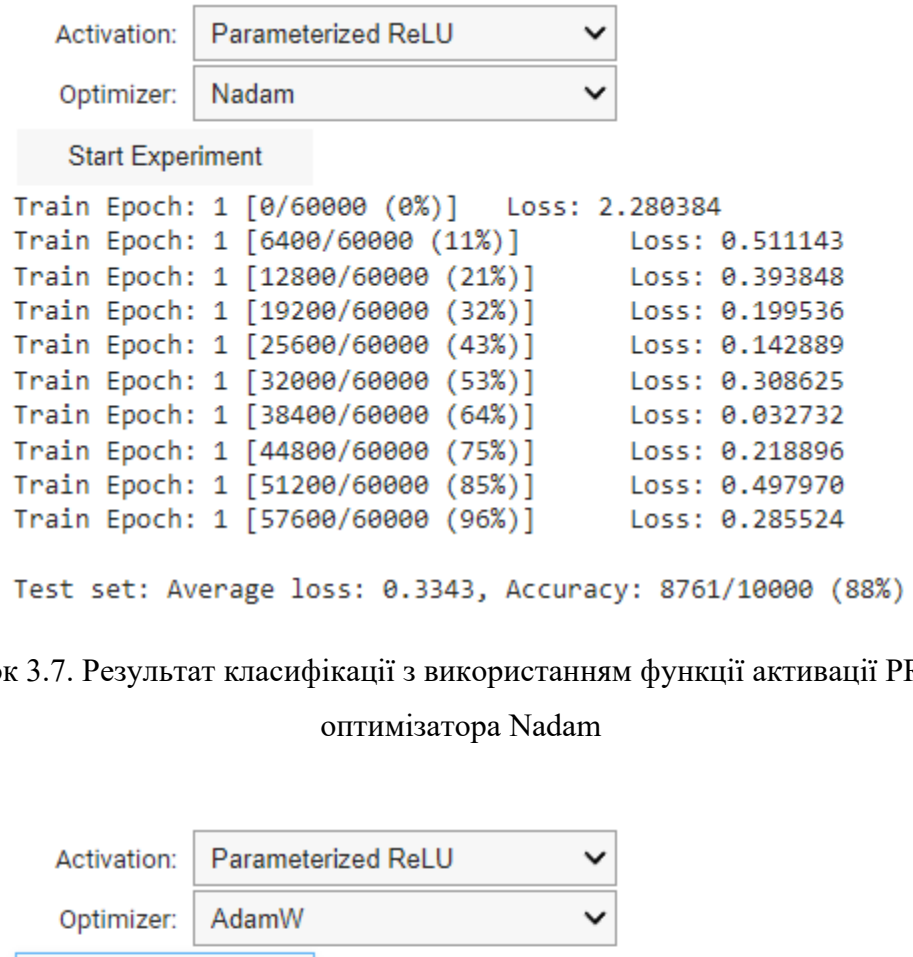


Рисунок 3.7. Результат класифікації з використанням функції активації PReLU та оптимізатора Nadam

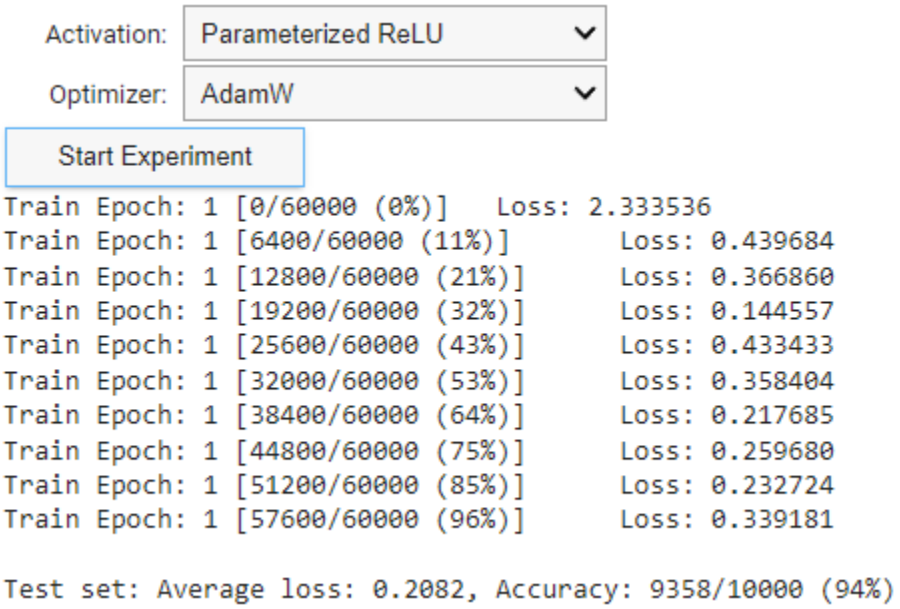


Рисунок 3.8. Результат класифікації з використанням функції активації PReLU та оптимізатора AdamW

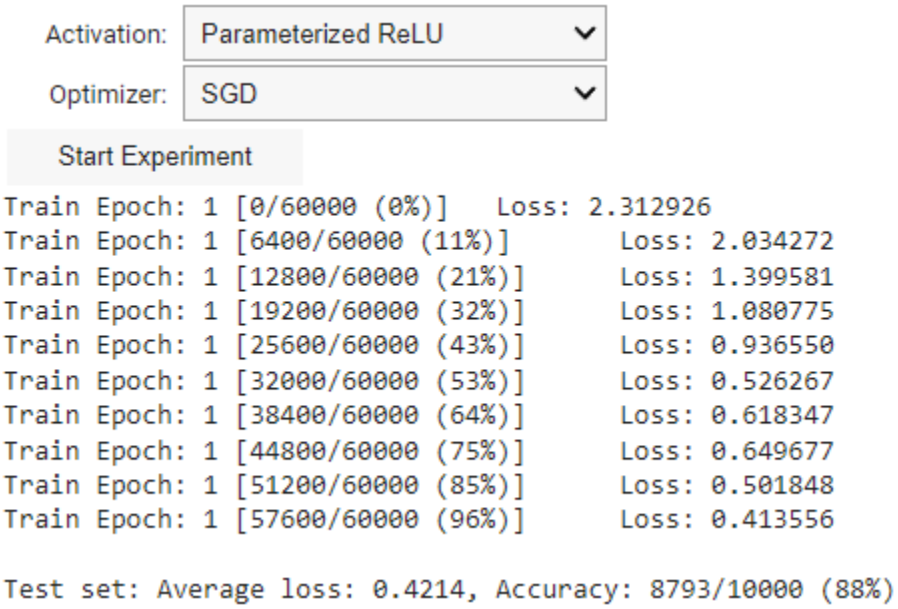


Рисунок 3.9. Результат класифікації з використанням функції активації PReLU та оптимізатора SGD

3. Створимо порівняльну таблицю та проаналізуємо різні комбінації функцій активації та оптимізаторів.

```
# Завдання №4. Створимо порівняльну таблицю результатів
import pandas as pd
results_df = pd.DataFrame(results)

styled_df = results_df.style.set_table_styles(
    [{'selector': 'th', 'props': [('background-color', 'lightblue'),
('color', 'black')]},
     {'selector': 'td', 'props': [('text-align', 'center')]}]
    ).set_properties(**{'background-color': 'lavender', 'color': 'black',
'border-color': 'darkgray'}
                     ).hide_index().format({'Train Loss': '{:.4f}', 'Test
Loss': '{:.4f}', 'Test Accuracy': '{:.2f}%'})

display(styled_df)
```

Activation	Optimizer	Train Loss	Test Loss	Test Accuracy
SiLU (Swish)	Nadam	0.2410	0.1805	95.12%
SiLU (Swish)	AdamW	0.2534	0.1872	95.39%
SiLU (Swish)	SGD	0.4829	0.2699	92.20%
SoftPlus	Nadam	0.2555	0.2135	94.37%
SoftPlus	AdamW	0.3794	0.2879	91.75%
SoftPlus	SGD	0.5491	0.3865	88.67%
Parameterized ReLU	Nadam	3.4909	0.3114	91.98%
Parameterized ReLU	AdamW	0.2871	0.2104	94.59%
Parameterized ReLU	SGD	0.4239	0.2033	93.82%

Рисунок 3.10. Таблиця результатів тестування з використанням різних функцій активації та оптимізації

Побудуємо графік для візуалізації отриманих результатів

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.barplot(x='Optimizer', y='Test Accuracy', hue='Activation',
data=results_df)
plt.title('Test Accuracy for Different Activation Functions and
Optimizers')
plt.show()
```

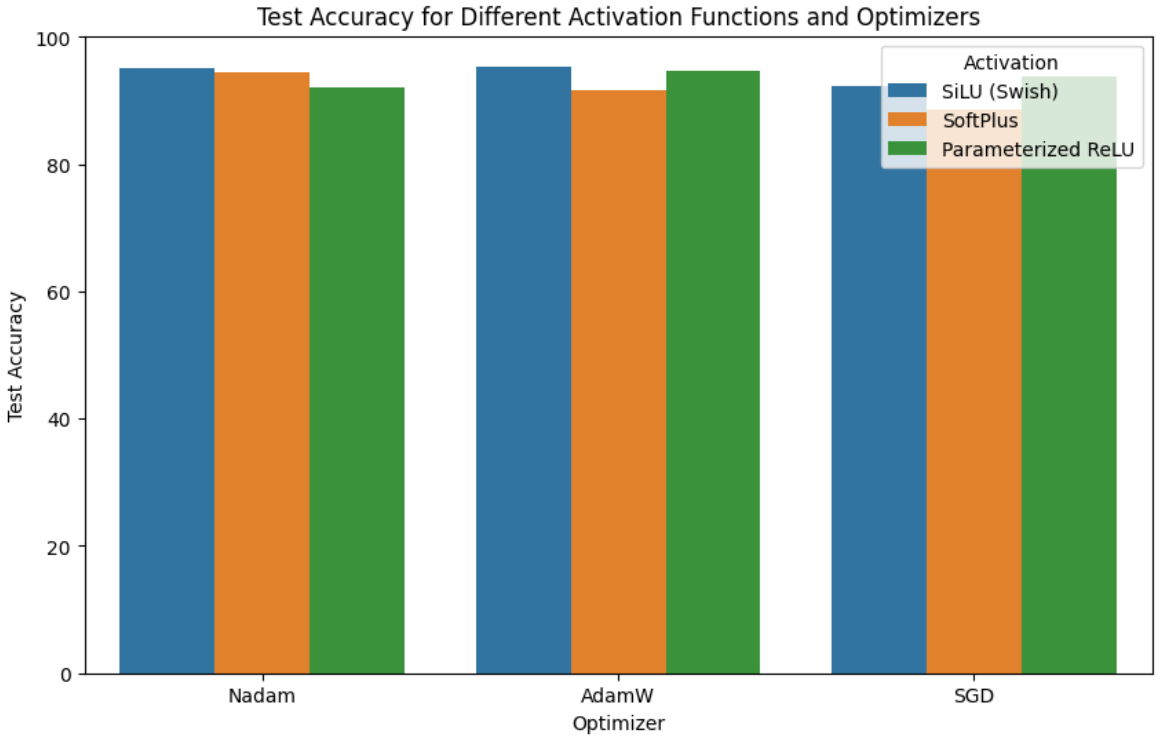


Рисунок 3.11. Графік точності на тестовому наборі за умови використання різних функцій активації та оптимізаторів

Таким чином, аналізуючи отримані дані, можна зазначити, що оптимізатори Nadam і AdamW забезпечують кращі результати для всіх трьох функцій активації порівняно з SGD, що відображається в менших втратах та вищій точності, що свідчить про їх більшу ефективність у оптимізації ваг моделі для даного набору даних.

Комбінація SiLU з AdamW демонструє найкращі результати, а саме Test Loss (0.1872), Test Accuracy (95.39%), що робить її найефективнішою для даної задачі.

Комбінація PReLU з AdamW також продемонструвала високу ефективність, маючи схожі показники Test Loss (0.2104), Test Accuracy (94.59%).

Комбінація SoftPlus з SGD продемонструвала найвищий показник Test Loss (0.3865) та найнижчий результат Test Accuracy (88.67%), що вказує на недостатню адаптацію цієї комбінації до умов даної задачі.

Контрольне запитання №1

Вкажіть показник Test Accuracy для комбінації другої функції активації та другого оптимізатора.

Відповідь: **91.75%**

Контрольне запитання №2

Вкажіть показник Test Loss для комбінації третьої функції активації та першого оптимізатора.
Відповідь: **0.3114**

Контрольне запитання №3

Вкажіть найбільш ефективну комбінацію функції активації та оптимізатора за показником точності.
Відповідь: **SiLU (Swish) та AdamW**

Контрольне запитання №4

Як правильно налаштувати трансформації для зображень з 3 кольоровими каналами (RGB) в Pytorch, використовуючи transforms.Compose?
Відповідь: **…**

Висновки: …

Завдання для виконання лабораторної роботи

1. Створіть модель для класифікації зображень з датасету відповідно до вашого варіанту.
2. Застосуйте три функції активації відповідно до вашого варіанту у прихованих шарах вашої моделі та порівняйте їх вплив на точність класифікації та швидкість збіжності.
3. Використайте кожен з трьох оптимізаторів відповідно до вашого варіанту для тренування вашої моделі та проаналізуйте їх ефективність.
4. Зібрані результати тренування та тестування представте у вигляді порівняльної таблиці. Проаналізуйте, як комбінація функції активації та оптимізатора впливає на загальну продуктивність моделі на датасеті. Зробіть висновки про оптимальні стратегії навчання для цього конкретного датасету.

Варіант	**Датасет**	**Функція активації №1**	**Функція активації №2**	**Функція активації №3**	**Опти-мізатор №1**	**Опти-мізатор №2**	**Опти-мізатор №3**
1	MNIST	ReLU	Sigmoid	Tanh	SGD	Adam	RMSprop
2	Fashion MNIST	Leaky ReLU	ELU	ReLU	Adam	Adagrad	Adadelta
3	MNIST	ELU	Softmax	Sigmoid	RMSprop	Adam	SGD
4	MNIST	Tanh	Softmax	PReLU	Adadelta	Adagrad	AdamW
5	Fashion MNIST	Sigmoid	ReLU	ELU	Adagrad	SGD	RMSprop
6	CIFAR-10	Softmax	Tanh	Sigmoid	AdamW	Adadelta	Adagrad
7	MNIST	ELU	Softplus	ReLU	RMSprop	Adadelta	SGD
8	Fashion MNIST	ReLU	Sigmoid	Tanh	Adagrad	Nadam	RMSprop
9	CIFAR-10	Tanh	SELU	Leaky ReLU	SGD	RMSprop	Adadelta
10	MNIST	Softmax	ELU	Sigmoid	Adadelta	Adagrad	AdamW
11	Fashion MNIST	Sine ReLU	ELU	Sigmoid	Adam	SGD	Adagrad
12	CIFAR-10	Softmax	Tanh	Leaky ReLU	RMSprop	Adadelta	Adam
13	MNIST	ELU	Sigmoid	Softmax	Adadelta	Adagrad	SGD
14	Fashion MNIST	Tanh	S-Shaped ReLU	ELU	SGD	RMSprop	Adam
15	CIFAR-10	Softmax	Sigmoid	Linear	Adagrad	AdamW	RMSprop
16	MNIST	ELU	Softmax	Tanh	Adadelta	SGD	Adagrad
17	Fashion MNIST	Sigmoid	SiLU	ReLU	RMSprop	Adagrad	Nadam

18	CIFAR-10	Leaky ReLU	Softmax	Linear	AdamW	Adadelta	SGD
19	MNIST	Tanh	Sigmoid	ELU	Adagrad	RMSprop	Adadelta
20	Fashion MNIST	ReLU	Tanh	Softmax	SGD	Nadam	RMSprop
21	CIFAR-10	Softmax	PReLU	Leaky ReLU	Adadelta	RMSprop	Adagrad
22	MNIST	ELU	Tanh	Sigmoid	RMSprop	SGD	Adam
23	Fashion MNIST	Leaky ReLU	RReLU	Softmax	AdamW	Adagrad	SGD
24	CIFAR-10	Sigmoid	Softmax	PReLU	Adadelta	RMSprop	Adagrad
25	MNIST	Linear	Leaky ReLU	Softmax	SGD	Adagrad	RMSprop
26	Fashion MNIST	Softmax	Sigmoid	ReLU	Adam	SGD	Adadelta
27	CIFAR-10	ReLU	Softplus	Tanh	RMSprop	Adagrad	Nadam
28	MNIST	ELU	Softplus	Sigmoid	Adadelta	AdamW	SGD
29	Fashion MNIST	Sigmoid	Softmax	ELU	Adagrad	RMSprop	Adam
30	CIFAR-10	Tanh	ELU	Leaky ReLU	SGD	Adadelta	RMSprop

Контрольне запитання №1

Вкажіть показник Test Accuracy для комбінації другої функції активації та другого оптимізатора.

Відповідь: _____

Контрольне запитання №2

Вкажіть показник Test Loss для комбінації третьої функції активації та першого оптимізатора.

Відповідь: _____

Контрольне запитання №3

Вкажіть найбільш ефективну комбінацію функції активації та оптимізатора за показником точності.

Відповідь: _____

Контрольне запитання №4

Як правильно налаштувати трансформації для зображень з 3 кольоровими каналами (RGB) в Pytorch, використовуючи transforms.Compose?

Відповідь: _____

Welcome to Google Colab

https://colab.research.google.com/drive/1KbxoeoObNgr-e00QkB6UrRLNxZYTzLrY?usp=sharing

Головне про PyTorch: **PyTorch** – це відкрита бібліотека машинного навчання, яка широко використовується для наукових досліджень та розробки в галузі штучного інтелекту. PyTorch надає потужні інструменти для створення та тренування нейронних мереж, дозволяючи легко працювати з глибоким навчанням. Особливість PyTorch полягає в його динамічних обчислювальних графах, що робить експериментування та ітерації більш гнучкими та інтуїтивно зрозумілими. Бібліотека підтримує роботу з GPU для прискорення обчислень, має велику кількість попередньо навчених моделей через torchvision та інтегрується з багатьма іншими інструментами для наукових обчислень та візуалізації даних. PyTorch Datasets – це компонент PyTorch, який забезпечує зручний доступ до стандартних наборів даних, що дозволяє легко завантажувати та препроцесингувати дані для тренування моделей, забезпечуючи широкий спектр наборів даних для різних задач машинного навчання та комп'ютерного зору.

За додатковою інформацією щодо PyTorch звертайтеся до документації https://pytorch.org/vision/stable/datasets.html