Documentație proiect Sisteme Inteligente – Clasificarea legumelor

Iosif Adela, grupa 30233

1. Introducere

Scopul acestui proiect este dezvoltarea, antrenarea și evaluarea unei rețele neuronale convoluționale (CNN) pentru clasificarea imaginilor cu legume. Proiectul utilizează PyTorch pentru construcția și antrenarea rețelei, dar include și diverse funcționalități precum preprocesarea imaginilor, vizualizarea distribuției claselor și compararea performanțelor a doi algoritmi de optimizare: Adam și SGD.

Eu am implementat proiectul în Notebook-ul Jupyter (pornit din Anaconda Navigator) – acesta permite încărcarea pozelor direct cu path-ul local, nefiind nevoie să pun pozele în Drive. Cu Google Colab am așteptat câteva ore să se facă antrenarea, și nu s-a creat nicio epocă, de asta am folosit Jupyter.

2. Explicații implementare: cod, analiză performanțe, comparare

```
#Transforms
transformer=transforms.Compose([
    transforms.Resize((150,150)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),  #0-255 to 0-1, numpy to tensors
    transforms.Normalize([0.5,0.5,0.5], # 0-1 to [-1,1] , formula (x-mean)/std
                         [0.5,0.5,0.5])
])
```

Aici am definit un set de **transformări pentru preprocesarea imaginilor**. Transformările aplicate sunt (transforms.Compose permite compunerea mai multor transformări): redimensionarea imaginii la 150x150 pixeli, oglindire orizontală aleatoare a imaginilor (poate ajuta la îmbunătățirea generalizării modelului), transformarea imaginilor din numpy în tensori și normalizarea valorilor acestora între 0 și 1; apoi se normalizează valorile tensorilor imaginilor la intervalul [-1,1] (se folosește formula (x-mean)/std_dev, iar mean și std_dev sunt 0.5).

```
#Dataloader

# Paths for training, testing and validation directories

pred_path ='C:\\Users\\adeli\\OneDrive\\Documente\\Vegetable_Images\\validation'
train_path = 'C:\\Users\\adeli\\OneDrive\\Documente\\Vegetable_Images\\train'
test_path = 'C:\\Users\\adeli\\OneDrive\\Documente\\Vegetable_Images\\test'

train_loader=DataLoader(
    torchvision.datasets.ImageFolder(train_path,transform=transformer),
    batch_size=64, shuffle=True
)
test_loader=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=32, shuffle=True
)
```

Aici **se pregătesc datele pentru antrenare și testare**, organizându-le în batch-uri și aplicând transformările specifice asupra lor. Parametrul batch_size specifică câte imagini sunt încărcate într-un batch.

Modificarea valorii batch_size în DataLoader va afecta modul în care datele sunt încărcate și procesate în timpul antrenamentului sau testării modelelor (ex: modificare la 32 și 16). Reducerea valorii batch_size va reduce cantitatea de date procesate simultan în fiecare pas, deci va scădea timpul necesar pentru antrenament și testare. Totuși, reducerea dimensiunii

lotului poate duce la o mai mare variație a gradienților în timpul antrenamentului, ceea ce poate afecta stabilitatea procesului de antrenare (convergență mai lentă). Dimensiuni mai mari ale lotului duc la o convergență mai stabilă și la o generalizare mai bună, în timp ce dimensiuni mai mici ale lotului pot duce la o convergență mai rapidă, dar mai puțin stabilă și la o generalizare mai slabă.

```
# Visualization function
def visualize_data_distribution(loader):
    labels = []
    for _, label in loader.dataset:
        labels.append(label)
    plt.hist(labels, bins=len(set(labels)))
    plt.xlabel('Classes')
    plt.ylabel('Frequency')
    plt.title('Class Distribution')
    plt.show()  # afisare histograma
```

Pentru **vizualizarea distribuției claselor** într-un set de date am folosit funcția visualize_data_distribution. Astfel, se colectează etichetele tuturor imaginilor dintr-un set de date, apoi se construiește o histogramă care arată cum sunt distribuite diferitele clase în acel set de date. Această funcție este utilă pentru a vedea dacă setul de date este echilibrat sau dacă există supra/sub-reprezentare, ceea ce poate influența performanța unui model de învățare automată.

Setul meu de date de antrenare este unul echilibrat, fiecare clasă având 1000 de poze, după cum se observă în histogramă. Un set de date echilibrat este benefic, pentru că:

- Se evită bias-ul: dacă unele clase ar avea mult mai multe poze decât altele, modelul poate învăța să fie mai precis pe acele clase și să ignore celelalte clase. Asta ar duce la un model care nu generalizează bine și care are perfomanțe slabe pe clasele sub-reprezentate.
- Se evită overfitting-ul: un astfel de model ar putea avea performanță bună pe clasele mari în timpul antrenamentului, dar eșuează să generalizeze pe date noi, în special pe clase mici.

```
# Identify and sort categories
root = pathlib.Path(train_path)
classes = sorted([j.name.split('/')[-1] for j in root.iterdir()])
```

În continuare **am identificat și sortat categoriile** (clasele) în setul de date de antrenament. Astfel, se parcurg toate clasele din directorul de antrenament (train_path), se extrage numele claselor, se sortează alfabetic. Este importantă această operațiune, deoarece așa ne putem asigura că modelul de clasificare este antrenat pe toate clasele disponibile.

```
# CNN Network
class ConvNet(nn.Module):
```

Voi **defini o rețea neuronală convoluțională** (ConvNet) folosind modulul nn.Module din PyTorch, care va clasifica imagini în 15 clase.

Constructorul clasei inițializează toate straturile rețelei; metoda forward definește fluxul de date prin rețea (primește un tensor de intrare și îl propagă prin straturile rețelei, returnând

ieșirea finală). Deci, această rețea este concepută pentru clasificarea imaginilor, având un număr specificat de clase de ieșire (num_classes). Arhitectura rețelei este compusă din trei straturi de convoluție, intercalate cu straturi de normalizare și funcții de activare ReLu, urmate de un strat fully connected pentru clasificare.

```
# Helper function to train the model
def train_model(model, optimizer, num_epochs, train_loader, test_loader, train_count, test_count, optimizer_name):
```

Pentru **antrenarea și evaluarea modelului** de rețea neuronală am implementat funcția train_model. Așadar, se parcurge fiecare epocă de antrenament: se antrenează modelul pe setul de date de antrenament, se evaluează performanța modelului pe setul de date de testare (calculând acuratețea), se salvează modelul dacă se obține o acuratețe mai mare pe setul de testare decât cea înregistrată anterior.

```
# Initialize models and optimizers
model_adam = ConvNet(num_classes=15).to(device)
optimizer_adam = Adam(model_adam.parameters(), lr=0.001, weight_decay=0.0001)

model_sgd = ConvNet(num_classes=15).to(device)
optimizer_sgd = SGD(model_sgd.parameters(), lr=0.001, momentum=0.9)
```

Inițializarea modelelor și a celor doi optimizatori: se creează două instanțe separate ale rețelei CNN (model_adam și model_sgd), ambele configurate pentru a clasifica 15 clase diferite; se inițializează doi optimizatori diferiți (unul folosit pentru Adam și unul folosind algoritmul Stochastic Gradient Descent cu moment). Aceste modele și optimizatori vor fi folosiți ulterior pentru a antrena și evalua performanța rețelelor CNN utilizând cei doi optimizatori diferiți.

```
# Train and evaluate with Adam optimizer
train_losses_adam, val_accuracies_adam = train_model(model_adam, optimizer_adam, num_epochs, train_loader, test_loader, train_count, test_count, 'adam')
```

```
Epoch: 0 Train Loss: 7.058862209320068 Train Accuracy: 0.6235333333333334 Test Accuracy: 0.7983333333333333
Epoch: 1 Train Loss: 2.1469409465789795 Train Accuracy: 0.8284666666666667 Test Accuracy: 0.8333333333333334
Epoch: 2 Train Loss: 1.450282335281372 Train Accuracy: 0.8808 Test Accuracy: 0.8703333333333333
Epoch: 3 Train Loss: 0.976044774055481 Train Accuracy: 0.915 Test Accuracy: 0.908
Epoch: 4 Train Loss: 0.7952491641044617 Train Accuracy: 0.931 Test Accuracy: 0.8726666666666667
Epoch: 5 Train Loss: 0.6848713755607605 Train Accuracy: 0.9379333333333333 Test Accuracy: 0.884
Epoch: 6 Train Loss: 0.47119390964508057 Train Accuracy: 0.9529333333333333 Test Accuracy: 0.8363333333333334
Epoch: 7 Train Loss: 0.374555379152298 Train Accuracy: 0.9606 Test Accuracy: 0.8866666666666667
Epoch: 8 Train Loss: 0.39617228507995605 Train Accuracy: 0.9610666666666666 Test Accuracy: 0.8846666666666667
Epoch: 9 Train Loss: 0.34044963121414185 Train Accuracy: 0.9646 Test Accuracy: 0.9316666666666666
```

Antrenarea și evaluarea CNN-ului folosind optimizatorul Adam: modelul model_adam este antrenat folosind datele furnizate de train_loader; la sfârșitul fiecărei epoci, modelul este evaluat pe setul de testare folosind test_loader; dacă acuratețea pe setul de testare este mai bună decât cea mai bună acuratețe anterioară, modelul este salvat pe disc.

```
# Train and evaluate with SGD optimizer
train_losses_sgd, val_accuracies_sgd = train_model(model_sgd, optimizer_sgd, num_epochs, train_loader, test_loader, train_count, test_count, 'sgd')
```

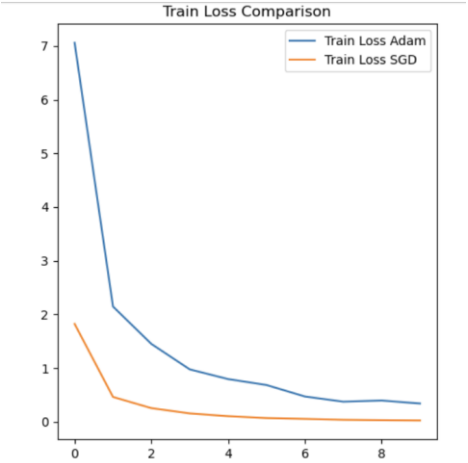
```
Epoch: 0 Train Loss: 1.8232213258743286 Train Accuracy: 0.6364 Test Accuracy: 0.8186666666666667
Epoch: 1 Train Loss: 0.4628056287765503 Train Accuracy: 0.8704 Test Accuracy: 0.875
Epoch: 2 Train Loss: 0.25421538949012756 Train Accuracy: 0.9232 Test Accuracy: 0.8906666666666667
Epoch: 3 Train Loss: 0.15607701241970062 Train Accuracy: 0.9508666666666666 Test Accuracy: 0.9086666666666666
Epoch: 4 Train Loss: 0.10434140264987946 Train Accuracy: 0.9688666666666667 Test Accuracy: 0.9433333333333334
Epoch: 5 Train Loss: 0.06976091116666794 Train Accuracy: 0.9810666666666666 Test Accuracy: 0.931
Epoch: 6 Train Loss: 0.053639572113752365 Train Accuracy: 0.9844 Test Accuracy: 0.947
Epoch: 7 Train Loss: 0.03767399862408638 Train Accuracy: 0.9916666666666667 Test Accuracy: 0.952
Epoch: 8 Train Loss: 0.030230415984988213 Train Accuracy: 0.9916 Test Accuracy: 0.9476666666666667
Epoch: 9 Train Loss: 0.02488499879837036 Train Accuracy: 0.9952 Test Accuracy: 0.9486666666666667
```

Antrenarea și evaluarea CNN-ului folosind optimizatorul SGD: se antrenează model_sgd folosind optimizatorul Stochastic Gradient Descent (SGD) pentru num_epochs epoci; evaluările sunt realizate pe setul de testare la sfârșitul fiecărei epoci.

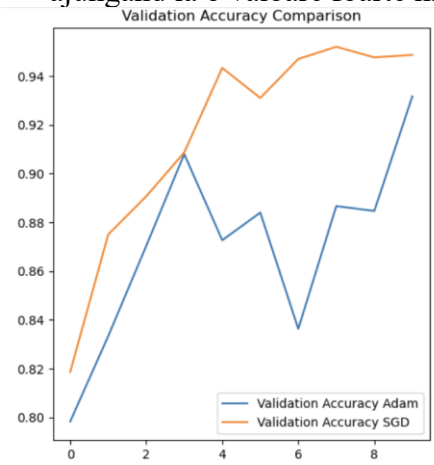
Compararea performanțelor – Adam și Stochastic Gradient Descent (SGD):

Optimizatorul Stochastic Gradient Descent (SGD) cu momentum a oferit o performanță superioară comparativ cu Adam în acest caz particular, fapt care ar putea fi datorat adaptabilității lui SGD la acest set de date și model specific, precum și efectului momentum-ului care ajută la traversarea mai rapidă a pierderilor.

- Acuratețe la antrenare și testare: SGD a avut o performanță mai bună în termeni de acuratețe atât la antrenare, cât și la testare față de Adam (Acuratețea la testare pentru SGD la finalul antrenamentului este de 94.87%, iar la Adam este 93.17%).
- Pierderi la antrenare: SGD a reușit să reducă pierderea mult mai rapid și la o valoare mult mai mică comparativ cu Adam (pierderea la finalul antrenamentului pentru SGD este de 0.025 comparativ cu 0.34 pentru Adam).
- Stabilitatea antrenamentului: SGD pare să fi avut un antrenament mai stabil și mai eficient, având în vedere valorile mai mici ale pierderilor și acuratețile mai mari obținute.



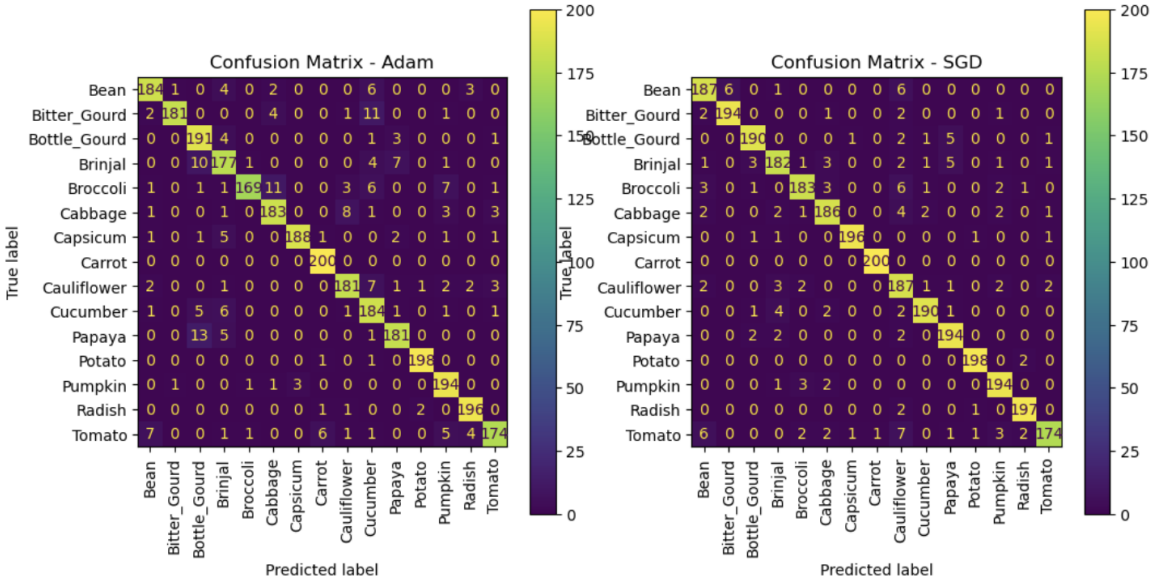
- Adam a început cu o pierdere foarte mare, dar a reușit să o reducă semnificativ până la sfârșitul antrenamentului. Cu toate acestea, pierderea finală este încă mai mare comparativ cu cea a SGD.
- SGD a avut o pierdere inițială mai mică și a reușit să o reducă mult mai eficient, ajungând la o valoare foarte mică până la sfârșitul antrenamentului.



- Adam a avut o acuratețe inițială relativ bună la testare și a reușit să o îmbunătățească constant pe parcursul antrenamentului, dar nu a atins nivelul la care a ajuns SGD.
- SGD a început cu o acuratețe mai mare la testare și a reușit să obțină o acuratețe finală superioară comparativ cu Adam, demonstrând o generalizare mai bună pe setul de testare.

```
#functia de predictie
def prediction(img_path, transformer, model, classes):
```

Realizarea de **predicții** folosind un model de învățare profundă pre-antrenat și apoi utilizarea acestei funcții pentru a prezice clasele imaginilor dintr-un set de date de validare: se încarcă un model deja antrenat, se pregătesc imaginile de validare, se realizează predicții pentru fiecare imagine și se afișează rezultatele.



Matricea de confuzie

- **Adam:** Există câteva cazuri în care modelul antrenat cu Adam a confundat clasele între ele (de exemplu, există câteva imagini clasificate ca fasole care au fost confundate cu castravete sau castraveți care au fost confundați cu fasole). Matricea de confuzie arată o distribuție relativ uniformă a greșelilor de clasificare pe toate clasele.
- **SGD:** Modelul antrenat cu SGD prezintă și el confuzii între clase, dar într-o măsură mai mică decât modelul antrenat cu Adam (există cazuri în care au existat confuzii între clasele de legume, cum ar fi între fasole și castravete). Distribuția greșelilor de clasificare pare să fie similară cu cea a modelului antrenat cu Adam.

Performanța modelului

```
Adam Optimizer - Accuracy: 0.927
Adam Optimizer - Recall: 0.927
```

```
SGD Optimizer - Accuracy: 0.9506666666666667
SGD Optimizer - Recall: 0.9506666666666667
```

- **Acuratețe:** Modelul antrenat cu SGD obține o acuratețe ușor mai mare decât cel antrenat cu Adam. Acest lucru sugerează că, în medie, modelul antrenat cu SGD a făcut mai puține greșeli de clasificare pe setul de date de testare.
- **Recall:** ambele modele au capacitatea de a identifica corect majoritatea claselor din setul de date.

Ambele modele au o performanță destul de bună, având în vedere că acuratețea și recall-ul sunt ambele peste 90%.

Este destul de neobișnuit ca acuratețea să fie exact egală cu recall-ul pentru modelele de clasificare, dar acest lucru se întâmplă mai des atunci când setul de date este echilibrat și modelele au o performanță uniformă pe toate clasele. Acestea ar explica valorile similare de la acuratețe și recall pentru ambii optimizatori.

Cross validation: tehnică utilizată în învățarea automată pentru a evalua performanța unui model și pentru a estima capacitatea acestuia de generalizare pe date noi. Ce aduce în plus cross validation:

- Reduce riscul de overfitting: se împarte setul de date în multiple fold-uri, deci se va face antrenarea și testarea modelului pe fiecare dintre acestea; cross-validation poate ajuta la identificarea modelelor care generalizează bine și evită supra-antrenarea.
- Estimează performanța generală a modelului: fiecare imagine din setul de date este utilizată atât pentru antrenare, cât și pentru testare în diverse combinații, permițând astfel o evaluare mai cuprinzătoare a performanței.

Fold 1:

```
Adam Optimizer - Confusion Matrix (Fold 1):
[[175   0   1   0   0   0   1   0   4  10   0   1   0   4   3]
 [  1 198   0   0   1   2   0   0   0   1   0   0   0   0   3]
 [  0   0 193   1   0   0   0   0   0   2   3   0   0   0   0]
 [  3   0  10 145   0   2   0   0   1  18   4   1   0   0   3]
 [  4   2   0   2 176   3   1   0   0   4   0   0   1   0   3]
 [  0   1   2   4   4 198   0   0   1   2   0   0   3   0   7]
 [  1   0   1   0   0   0 197   0   0   1   1   0   2   0   0]
 [  0   0   0   0   0   0   0 184   0   0   0   1   0   2   0]
 [  8   0   1   2   1   0   1   0 170   7   1   1   1   0   3]
 [  3   0   4   1   0   0   0   0   1 194   3   0   0   0   1]
 [  0   0   7   2   0   0   1   0   2   0 179   0   0   0   0]
 [  0   0   0   0   0   0   0   5   0   0   0 206   0   0   1]
 [  1   0   0   1   4   1   0   0   3   4   0   0 201   0   2]
 [  1   3   0   0   0   0   0   0   3   1   0   0   1 205   2]
 [  0   0   0   0   0   0   1   1   1   1   1   1   0   1 155]]
```

```
SGD Optimizer - Confusion Matrix (Fold 1):
[[177  10   0   0   0   1   1   0   2   3   0   0   1   0   4]
 [  0 197   0   1   1   2   0   0   0   3   0   0   0   0   2]
 [  0   0 192   5   0   0   0   0   0   0   2   0   0   0   0]
 [  2   0   3 166   0   0   0   0   1  14   0   1   0   0   0]
 [  0   1   0   1 188   1   0   0   4   1   0   0   0   0   0]
 [  0   3   0   3  10 199   0   0   2   1   0   0   1   0   3]
 [  0   0   0   0   0   0 201   0   0   1   1   0   0   0   0]
 [  0   0   0   0   0   0   0 185   0   0   0   0   0   0   2]
 [  2   1   0   3   1   0   0   0 184   0   0   0   4   0   1]
 [  1   1   0   3   3   2   0   0   1 194   0   0   0   1   1]
 [  0   0   2   3   0   0   2   0   2   0 180   0   0   0   2]
 [  0   0   0   0   1   0   0   1   0   0   0 207   0   1   2]
 [  0   2   0   0   4   1   1   0   3   2   0   1 203   0   0]
 [  1   5   0   0   0   1   0   1   3   0   0   3   0 201   1]
 [  1   1   0   1   0   2   1   0   0   0   0   1   0   0 155]]
```

```
Adam Optimizer - Accuracy (Fold 1): 0.9253333333333333
Adam Optimizer - Recall (Fold 1): 0.9253333333333333
```

```
SGD Optimizer - Accuracy (Fold 1): 0.943
SGD Optimizer - Recall (Fold 1): 0.943
```

Fold 2:

```
Adam Optimizer - Confusion Matrix (Fold 2):
[[191   8   0   2   1   3   5   0   9   0   0   0   0   0   1]
 [  3 194   0   1   0   1   0   0   1   0   0   0   0   0   0]
 [  0   0 160   2   0   0   8   0   0   0  11   0   0   0   0]
 [  5   0  11 166   1   4   2   0   1   2  14   0   2   0   1]
 [  1   2   0   2 166   3   0   0   6   1   0   0  13   0  10]
 [  0   3   1   4   0 159   1   0   3   1   0   0   5   0   0]
 [  0   0   0   0   0   0 178   0   0   0   0   0   0   0   1]
 [  0   0   0   0   0   0   0 215   0   0   0   0   0   1   3]
 [  0   3   0   1   0   6   0   0 197   1   0   0   5   2   3]
 [ 17   2   4  11   0   1   6   0  10 150   6   0   8   0   0]
 [  0   0   1   1   0   0   3   0   0   0 196   0   1   0   1]
 [  2   0   0   0   0   0   3   4   0   0   0 184   2   1   0]
 [  2   0   0   0   3   1   0   0   0   0   0   0 171   0   0]
 [  1   1   0   0   0   0   0   2   0   0   0   2   3 188   4]
 [  0   0   3   0   0   4   3   0   3   0   2   0   8   0 178]]
```

```
SGD Optimizer - Confusion Matrix (Fold 2):
[[201   5   0   1   0   1   1   0   6   2   0   0   0   0   3]
 [  1 195   0   1   0   0   0   0   1   1   0   0   0   0   1]
 [  0   0 170   3   0   0   0   0   0   0   7   0   0   0   1]
 [  2   0   4 176   1   7   0   0   3   6   9   0   0   0   1]
 [  0   2   0   0 183   3   0   0   3   7   0   0   5   1   0]
 [  1   0   0   1   1 162   0   0   1   3   0   0   5   0   3]
 [  0   0   0   0   0   0 179   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   1 217   0   0   0   1   0   0   0]
 [  0   0   0   1   0   4   0   0 201   5   0   0   2   2   3]
 [  6   1   3   2   0   0   0   0   0 199   4   0   0   0   0]
 [  0   0   0   1   0   0   2   0   0   0 200   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 195   0   0   1]
 [  0   0   0   1   4   1   0   0   2   1   0   1 166   0   1]
 [  2   0   1   0   0   1   0   0   2   0   0   3   0 192   0]
 [  1   2   1   1   1   2   1   0   0   0   0   1   3   3 185]]
```

```
Adam Optimizer - Accuracy (Fold 2): 0.8976666666666666
Adam Optimizer - Recall (Fold 2): 0.8976666666666666
```

```
SGD Optimizer - Accuracy (Fold 2): 0.9403333333333334
SGD Optimizer - Recall (Fold 2): 0.9403333333333334
```

Așadar, acuratețea pe setul de testare pare să fie destul de stabilă, ceea ce indică faptul că modelul a învățat să generalizeze bine pe setul de date de testare. Pierderea în timpul antrenamentului pare să scadă treptat în timpul epocilor, ceea ce sugerează că modelul se îmbunătățește pe măsură ce procesul de antrenament progresează.