Import necessary packages

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#import cnn_utils as U
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
torch.manual_seed(0)
```

Out[1]:

```
<torch._C.Generator at 0x269521ec6f0>
```

Implement model

In [2]:

```
# model train
def model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_type = "adam
        #Selecting the appropriate training device
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        net = U.Net().to(device)

        test_data_x = np.swapaxes(torch.Tensor(test_data_x), 1, 3)
        test_data_y = torch.Tensor(test_data_y.reshape(48))
        train_data_x = np.swapaxes(torch.Tensor(train_data_x), 1, 3)
        train_data_y = torch.Tensor(train_data_y.reshape(205))

        #Generating data loaders from the corresponding datasets
        batch_size = 5
        training_dataset = TensorDataset(train_data_x, train_data_y)
        train_loader = DataLoader(training_dataset, batch_size=batch_size)

        #Defining the model hyper parameters

        criterion = nn.CrossEntropyLoss()
        if optim_type == "adam":
                optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.01)
        elif optim_type == "SGD":
                optimizer = optim.SGD(net.parameters(), lr=0.01)
        elif optim_type == "Adagrad":
                optimizer = optim.Adagrad(net.parameters(), lr=0.01, weight_decay=0.01)
        else:
                return
        print("Start optimization, method: " + optim_type)
        #Training process begins
        train_loss_list = []
        test_accuracy_list = []
        for epoch in range(epochs_size):
                running_loss = 0

                for _, data in enumerate(train_loader):
                        #Extract data point
                        images = data[0].to(device)
                        labels = data[1].type(torch.LongTensor).to(device)

                        #Calculating loss
                        outputs = net(images)
                        loss = criterion(outputs, labels)
```

```
                #Clean the previous batch grad info
                optimizer.zero_grad()
                #Updating weights according to calculated loss
                loss.backward()
                optimizer.step()
                #Calculate running loss cumulation
                running_loss += loss.item()

            #Store the loss info
            train_loss_list.append(running_loss/len(train_loader))
            #Store test accuracy
            running_accuracy = model_test(test_data_x, test_data_y, net, epochs_size
            test_accuracy_list.append(running_accuracy)
            if printType == "loss":
                #Printing loss for each epoch
                print(f"Training loss at iteration {epoch+1} = {train_loss_list[
            # elif printType == "accurary":
            #       #Printing test accuracy for each epoch
            #       print(f"Testing accuracy at iteration {epoch+1} = {running_accur

        accuracy = model_test(test_data_x, test_data_y, net, epochs_size)
        print(f'Training finished. The final model accuracy on test dataset is: {accurac
        del net
        return train_loss_list, test_accuracy_list
```

Implement model test module with test dataset

In [3]:

```
# model test: can be called directly in model_train
def model_test(test_data_x, test_data_y, net, epoch_num):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        test_dataset = TensorDataset(test_data_x, test_data_y)
        test_loader = DataLoader(test_dataset, batch_size=5)
        correct = 0
        total = 0

        for data in test_loader:
                images, labels = data

                images = images.to(device)
                labels = labels.to(device)
                outputs = net(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        return 100 * correct / total
```

Part 1: Now run the CNN model

PA-II-1: run the Adam optimization

In [61]:

```
# load datasets
train_data_x, train_data_y, test_data_x, test_data_y = U.load_dataset()

# rescale data
train_data_x = train_data_x / 255.0
test_data_x = test_data_x / 255.0

# model train (model test function can be called directly in model_train)
methods = "adam"
result = model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_type =
```

```
Start optimization, method: adam
Training loss at iteration 1 = 0.6740498288375575
Training loss at iteration 2 = 0.6614793175604285
Training loss at iteration 3 = 0.6629482464092534
Training loss at iteration 4 = 0.6573547815404287
Training loss at iteration 5 = 0.6566850884658534
Training loss at iteration 6 = 0.6450089332534046
Training loss at iteration 7 = 0.6305457469893665
Training loss at iteration 8 = 0.6115487530464079
Training loss at iteration 9 = 0.5783464138100787
Training loss at iteration 10 = 0.5628379830499974
Training loss at iteration 11 = 0.5520378532933026
Training loss at iteration 12 = 0.5638543098438077
Training loss at iteration 13 = 0.5429950865303598
Training loss at iteration 14 = 0.543393579198093
Training loss at iteration 15 = 0.5394975034202018
Training loss at iteration 16 = 0.5327381208175566
Training loss at iteration 17 = 0.5321442673845989
Training loss at iteration 18 = 0.527504042881291
Training loss at iteration 19 = 0.5297199080630046
Training loss at iteration 20 = 0.5160502733253851
Training loss at iteration 21 = 0.5144291280246363
Training loss at iteration 22 = 0.5062428422090484
Training loss at iteration 23 = 0.5128082233231243
Training loss at iteration 24 = 0.5048790428696609
Training loss at iteration 25 = 0.5056996563585793
Training loss at iteration 26 = 0.49424124999744135
Training loss at iteration 27 = 0.49341932084502244
Training loss at iteration 28 = 0.490721022937356
Training loss at iteration 29 = 0.4956828086841397
Training loss at iteration 30 = 0.4882205673834173
Training loss at iteration 31 = 0.49554383682041636
Training loss at iteration 32 = 0.4920648547207437
Training loss at iteration 33 = 0.4853941209432555
Training loss at iteration 34 = 0.49319171105943077
Training loss at iteration 35 = 0.48803202771558996
Training loss at iteration 36 = 0.4817045163817522
Training loss at iteration 37 = 0.4783097781786105
Training loss at iteration 38 = 0.47875526329366175
Training loss at iteration 39 = 0.47972504903630514
Training loss at iteration 40 = 0.4838799608916771
Training loss at iteration 41 = 0.49733805438367334
Training loss at iteration 42 = 0.49801120380075964
Training loss at iteration 43 = 0.4929660057149282
Training loss at iteration 44 = 0.48785524179295797
Training loss at iteration 45 = 0.48689009939751976
Training loss at iteration 46 = 0.49357049712320655
Training loss at iteration 47 = 0.495388801504926
Training loss at iteration 48 = 0.4932039707172208
Training loss at iteration 49 = 0.48893441877714017
Training loss at iteration 50 = 0.48442735323091834
Training loss at iteration 51 = 0.4889696058703632
Training loss at iteration 52 = 0.4847147704624548
Training loss at iteration 53 = 0.4883134183360309
Training loss at iteration 54 = 0.49083509823171106
Training loss at iteration 55 = 0.49461484246137666
Training loss at iteration 56 = 0.4875804745569462
Training loss at iteration 57 = 0.48399915564350965
Training loss at iteration 58 = 0.4835314132818362
Training loss at iteration 59 = 0.48043634687981956
Training loss at iteration 60 = 0.47930195927619934
Training loss at iteration 61 = 0.48242360789601396
Training loss at iteration 62 = 0.48619821885736975
Training loss at iteration 63 = 0.4889675974845886
Training loss at iteration 64 = 0.4840547725921724
Training loss at iteration 65 = 0.4836712021653245
```

```
Training loss at iteration 66 = 0.48725534721118646
Training loss at iteration 67 = 0.48035556877531654
Training loss at iteration 68 = 0.4767027577249015
Training loss at iteration 69 = 0.47639762628369214
Training loss at iteration 70 = 0.475710226268303l
Training loss at iteration 71 = 0.47290488277993553
Training loss at iteration 72 = 0.48035552080084637
Training loss at iteration 73 = 0.4759207774953144
Training loss at iteration 74 = 0.4708551328356673
Training loss at iteration 75 = 0.4749298226542589
Training loss at iteration 76 = 0.4699070489988094
Training loss at iteration 77 = 0.46444663187352625
Training loss at iteration 78 = 0.46633086553434044
Training loss at iteration 79 = 0.4645599555678484
Training loss at iteration 80 = 0.46340596821249985
Training loss at iteration 81 = 0.46658640009600943
Training loss at iteration 82 = 0.46660903314264807
Training loss at iteration 83 = 0.4603375538093288
Training loss at iteration 84 = 0.4587710630602953
Training loss at iteration 85 = 0.4609460292792902
Training loss at iteration 86 = 0.4632845964373612
Training loss at iteration 87 = 0.4645038894036921
Training loss at iteration 88 = 0.4633774481168607
Training loss at iteration 89 = 0.47464147282809743
Training loss at iteration 90 = 0.46951082857643683
Training loss at iteration 91 = 0.4617460500903246
Training loss at iteration 92 = 0.4616162202707151
Training loss at iteration 93 = 0.4666838994840296
Training loss at iteration 94 = 0.46382376551628113
Training loss at iteration 95 = 0.45978818506729313
Training loss at iteration 96 = 0.45628924994933895
Training loss at iteration 97 = 0.4578580412922836
Training loss at iteration 98 = 0.4580627963310335
Training loss at iteration 99 = 0.44954506580422565
Training loss at iteration 100 = 0.45224104930714865
Training finished. The final model accuracy on test dataset is: 81.25
```

It shows that the model accuracy is 81.25 which is as expected

PA-II-2: Then I will compare SGD, Adagrad and Adam optimization methods with visualization

In [62]:

```python
methods = ["adam", "SGD", "Adagrad"]
results = []
for m in methods:
    result = model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_typ
    results.append(result[1])
results = pd.DataFrame(results)
plt.plot(results.T)
plt.legend(["adam", "SGD", "Adagrad"])
```

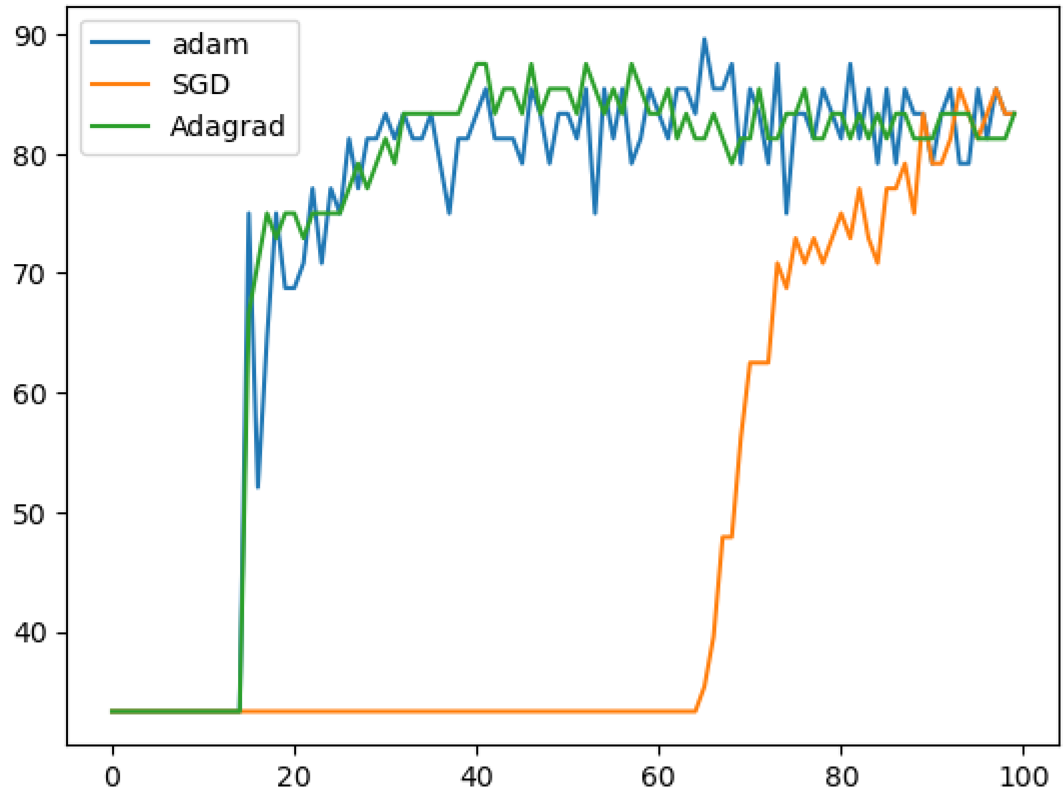
```
Start optimization, method: adam
Training finished. The final model accuracy on test dataset is: 79.16666666666667

Start optimization, method: SGD
Training finished. The final model accuracy on test dataset is: 81.25

Start optimization, method: Adagrad
Training finished. The final model accuracy on test dataset is: 83.33333333333333
```

Out[62]:

```
<matplotlib.legend.Legend at 0x201c3df2d60>
```



It shows that the 3 methods are comparable but Adagrad is the best one in this case

PA-II-3: Next I added the dropout p = 0.2 to the network and compared the 3 methods again. The dropout code is in cnn_utils.py with comments.

In [65]:

```
# dropout p = 0.2 has been added to the network, please check code in cnn_utils.py
methods = ["adam", "SGD", "Adagrad"]
results = []
for m in methods:
    result = model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_typ
    results.append(result[1])
results = pd.DataFrame(results)
plt.plot(results.T)
plt.legend(["adam", "SGD", "Adagrad"])
```

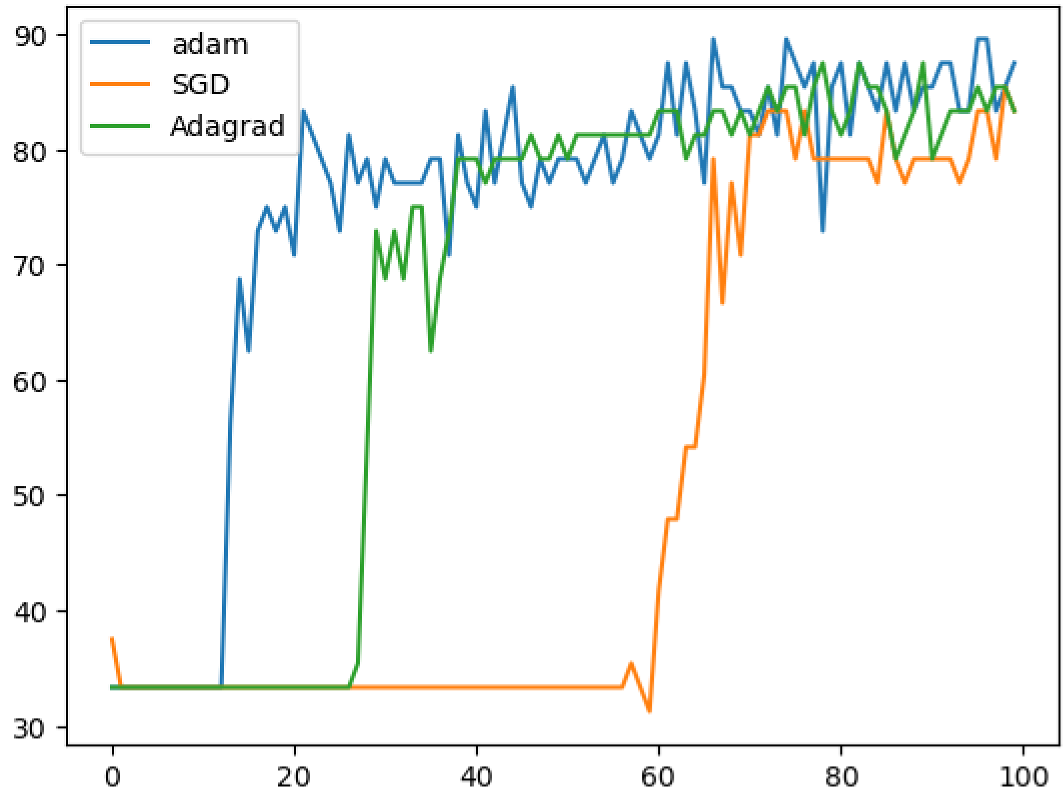
```
Start optimization, method: adam
Training finished. The final model accuracy on test dataset is: 87.5

Start optimization, method: SGD
Training finished. The final model accuracy on test dataset is: 83.33333333333333

Start optimization, method: Adagrad
Training finished. The final model accuracy on test dataset is: 81.25
```

Out[65]:

```
<matplotlib.legend.Legend at 0x2019c8d0a90>
```



It seems the dropout strategy increased the accuracy a little bit for Adam and SGD

Adam accuracy: 79.16% -> 87.50%

SGD accuracy: 81.25% -> 83.33%

Adagrad accuracy: 83.33% -> 81.25%

Part 2: Now run the RCNN model

In [17]:

```
del U
import rcnn_utils as U
```

In [20]:

```
# load datasets
train_data_x, train_data_y, test_data_x, test_data_y = U.load_dataset()

# rescale data
train_data_x = train_data_x / 255.0
test_data_x = test_data_x / 255.0

# model train (model test function can be called directly in model_train)
methods = "adam"
result = model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_type = 
```

```
Start optimization, method: adam
Training loss at iteration 1 = 0.675438584350958
Training loss at iteration 2 = 0.6607218755454551
Training loss at iteration 3 = 0.6629955884886951
Training loss at iteration 4 = 0.6578360512489225
Training loss at iteration 5 = 0.6554510455305983
Training loss at iteration 6 = 0.6540920458188871
Training loss at iteration 7 = 0.6487742240835981
Training loss at iteration 8 = 0.6420468237341904
Training loss at iteration 9 = 0.6303899339059504
Training loss at iteration 10 = 0.614539392110778
```

```
Training loss at iteration 11 = 0.5926502710435448
Training loss at iteration 12 = 0.5758761509162623
Training loss at iteration 13 = 0.568984378401826
Training loss at iteration 14 = 0.5622054403874932
Training loss at iteration 15 = 0.5603590971086083
Training loss at iteration 16 = 0.5559531399389592
Training loss at iteration 17 = 0.5540946242285938
Training loss at iteration 18 = 0.5507924760260233
Training loss at iteration 19 = 0.5437808552893196
Training loss at iteration 20 = 0.5421924707366199
Training loss at iteration 21 = 0.5420238630073827
Training loss at iteration 22 = 0.5376992901650871
Training loss at iteration 23 = 0.5376527978152763
Training loss at iteration 24 = 0.5343304683522481
Training loss at iteration 25 = 0.5409922701556508
Training loss at iteration 26 = 0.5484908041430683
Training loss at iteration 27 = 0.5434476318882733
Training loss at iteration 28 = 0.5307319629483107
Training loss at iteration 29 = 0.5423396396927718
Training loss at iteration 30 = 0.5379121456204391
Training loss at iteration 31 = 0.5336764854628865
Training loss at iteration 32 = 0.5565900831687741
Training loss at iteration 33 = 0.5392469436657138
Training loss at iteration 34 = 0.5393719564123851
Training loss at iteration 35 = 0.5539623615218372
Training loss at iteration 36 = 0.5486320700587296
Training loss at iteration 37 = 0.5444739141115328
Training loss at iteration 38 = 0.5353080855637062
Training loss at iteration 39 = 0.547271316371313
Training loss at iteration 40 = 0.5384172906235951
Training loss at iteration 41 = 0.5438574879634671
Training loss at iteration 42 = 0.5415150537723448
Training loss at iteration 43 = 0.539600926201518
Training loss at iteration 44 = 0.5399372119729112
Training loss at iteration 45 = 0.5310700245019866
Training loss at iteration 46 = 0.5254889821133962
Training loss at iteration 47 = 0.5246334955459688
Training loss at iteration 48 = 0.5275752282724148
Training loss at iteration 49 = 0.5334715726898938
Training loss at iteration 50 = 0.5280841639856013
Training loss at iteration 51 = 0.5405688896411802
Training loss at iteration 52 = 0.5085615406676036
Training loss at iteration 53 = 0.5193962731012484
Training loss at iteration 54 = 0.5318489830668379
Training loss at iteration 55 = 0.5077672324529509
Training loss at iteration 56 = 0.5165478100137013
Training loss at iteration 57 = 0.5278352295480123
Training loss at iteration 58 = 0.5242941677570343
Training loss at iteration 59 = 0.532749684118643
Training loss at iteration 60 = 0.5015693452300095
Training loss at iteration 61 = 0.5046120496784768
Training loss at iteration 62 = 0.5193155902187999
Training loss at iteration 63 = 0.5180948056825777
Training loss at iteration 64 = 0.5233434351479135
Training loss at iteration 65 = 0.4977739446046876
Training loss at iteration 66 = 0.5227233360453349
Training loss at iteration 67 = 0.5227103785770696
Training loss at iteration 68 = 0.49616839100674887
Training loss at iteration 69 = 0.49460986256599426
Training loss at iteration 70 = 0.5024129319481734
Training loss at iteration 71 = 0.5219485585282488
Training loss at iteration 72 = 0.4937431194433352
Training loss at iteration 73 = 0.49202029225302907
Training loss at iteration 74 = 0.4888085541201801
Training loss at iteration 75 = 0.492275026513309
Training loss at iteration 76 = 0.4932175794752633
```

```
Training loss at iteration 77 = 0.5098897255048519
Training loss at iteration 78 = 0.5259277260885006
Training loss at iteration 79 = 0.49071458127440476
Training loss at iteration 80 = 0.509453164368141
Training loss at iteration 81 = 0.4925251697621694
Training loss at iteration 82 = 0.49076258845445586
Training loss at iteration 83 = 0.49132505713439567
Training loss at iteration 84 = 0.500592316069254
Training loss at iteration 85 = 0.4857687426776421
Training loss at iteration 86 = 0.4994061247604649
Training loss at iteration 87 = 0.4896496221786592
Training loss at iteration 88 = 0.5066102786761958
Training loss at iteration 89 = 0.4966505431547398
Training loss at iteration 90 = 0.48355244572569683
Training loss at iteration 91 = 0.48455009809354455
Training loss at iteration 92 = 0.48301923129616714
Training loss at iteration 93 = 0.48077839758338
Training loss at iteration 94 = 0.4826788996777883
Training loss at iteration 95 = 0.4793362159554551
Training loss at iteration 96 = 0.48111272585101245
Training loss at iteration 97 = 0.47750502025208824
Training loss at iteration 98 = 0.4774546717725149
Training loss at iteration 99 = 0.47516378033451917
Training loss at iteration 100 = 0.47460968974159984
Training finished. The final model accuracy on test dataset is: 89.58333333333333
```

It shows that the model accuracy is 89.58 which is as expected

In [29]:

```
methods = ["adam", "SGD", "Adagrad"]
results = []
for m in methods:
    result = model_train(train_data_x, train_data_y, test_data_x, test_data_y, optim_typ
    results.append(result[1])
results = pd.DataFrame(results)
plt.plot(results.T)
plt.legend(["adam", "SGD", "Adagrad"])
```

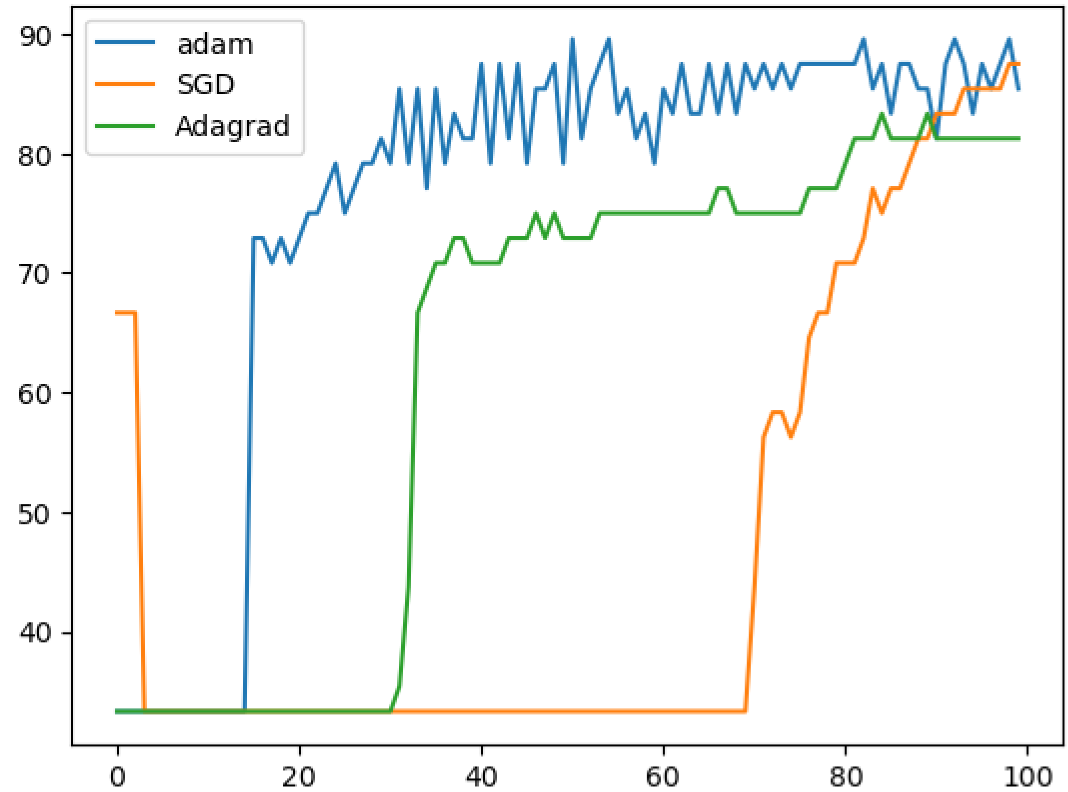
```
Start optimization, method: adam
Training finished. The final model accuracy on test dataset is: 85.41666666666667

Start optimization, method: SGD
Training finished. The final model accuracy on test dataset is: 87.5

Start optimization, method: Adagrad
Training finished. The final model accuracy on test dataset is: 81.25
```

Out[29]:

```
<matplotlib.legend.Legend at 0x2692bda3040>
```



The RCNN results are as expected