Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Course No. : CSE 4238
Course Name : Soft Computing Lab

Assignment No. : 02

Submitted By:

Name : Emdadul Haque
ID No. : 17 01 04 028
Session : Fall - 2020
Section : A (A2)

Table of Contents

Figure of Content

1. Dataset Creation

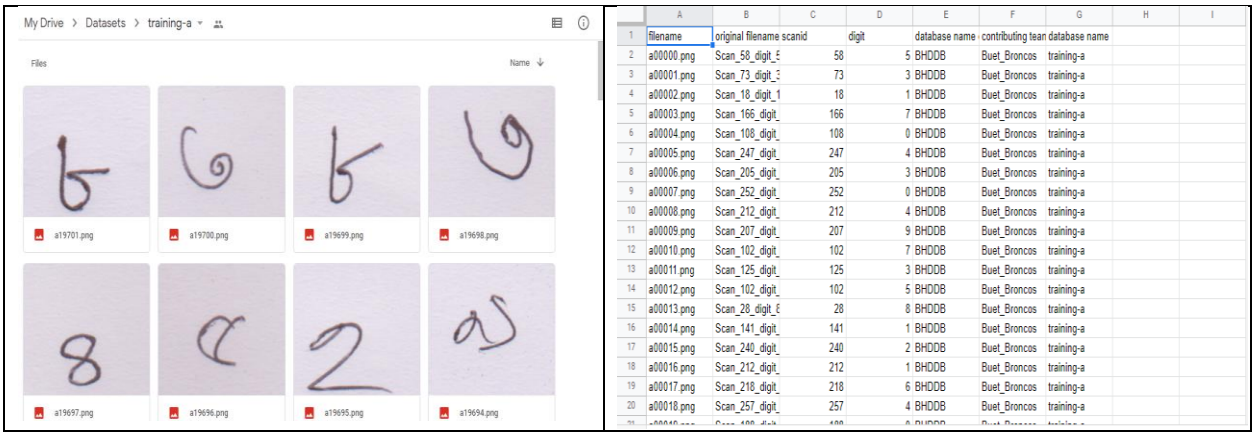
In this section, we know about the creation process of the dataset.

For the data creation process, we divide it into two different sub-processes.

1. Data Download
2. Creating Dataset Proper Format

1.1. Data Downloading:

In the data downloading process, first, we need to download the files based on the ID. My is 170104028. So, it is even ID. That's why I need to download the Dataset A. In dataset we saw that this is Bangla Hand Written digit. After extracting the dataset, we can saw that there are two files. One is train training-a folder training-a.csv. In folder we see all the image. **training-a.csv** file have the information of the image. The two files content is-



	filename	original filename	scanid	digit	database name	contributing team	database name
2	a00000.png	Scan_58_digit_5	58	5	BHDDB	Buet_Broncos	training-a
3	a00001.png	Scan_73_digit_3	73	3	BHDDB	Buet_Broncos	training-a
4	a00002.png	Scan_18_digit_1	18	1	BHDDB	Buet_Broncos	training-a
5	a00003.png	Scan_166_digit_	166	7	BHDDB	Buet_Broncos	training-a
6	a00004.png	Scan_108_digit_	108	0	BHDDB	Buet_Broncos	training-a
7	a00005.png	Scan_247_digit_	247	4	BHDDB	Buet_Broncos	training-a
8	a00006.png	Scan_205_digit_	205	3	BHDDB	Buet_Broncos	training-a
9	a00007.png	Scan_252_digit_	252	0	BHDDB	Buet_Broncos	training-a
10	a00008.png	Scan_212_digit_	212	4	BHDDB	Buet_Broncos	training-a
11	a00009.png	Scan_207_digit_	207	9	BHDDB	Buet_Broncos	training-a
12	a00010.png	Scan_102_digit_	102	7	BHDDB	Buet_Broncos	training-a
13	a00011.png	Scan_125_digit_	125	3	BHDDB	Buet_Broncos	training-a
14	a00012.png	Scan_102_digit_	102	5	BHDDB	Buet_Broncos	training-a
15	a00013.png	Scan_28_digit_8	28	8	BHDDB	Buet_Broncos	training-a
16	a00014.png	Scan_141_digit_	141	1	BHDDB	Buet_Broncos	training-a
17	a00015.png	Scan_240_digit_	240	2	BHDDB	Buet_Broncos	training-a
18	a00016.png	Scan_212_digit_	212	1	BHDDB	Buet_Broncos	training-a
19	a00017.png	Scan_218_digit_	218	6	BHDDB	Buet_Broncos	training-a
20	a00018.png	Scan_257_digit_	257	4	BHDDB	Buet_Broncos	training-a

Figure 1: Raw Dataset

Form this, dataset we saw that this is not properly formatted. That's why we need to format the dataset.

For download the dataset I used gdown package. Because this is google drive link. For downloading I used given code-

```
!gdown --id 1txyKhs1Zt5AKswGGK9VI_jE0JNHuQT85

Downloading...
From: https://drive.google.com/uc?id=1txyKhs1Zt5AKswGGK9VI_jE0JNHuQT85
To: /content/Dataset A.zip
815MB [00:08, 90.7MB/s]
```

Figure 2: Data Download Code

1.2. Creating Dataset Proper Format

After Collecting the raw data. We need to generate our dataset. In to a good format. Because, next time when we train, we don't need to change anything our dataset. Also, future, we can train our dataset easily. After formatted dataset the dataset looks like-

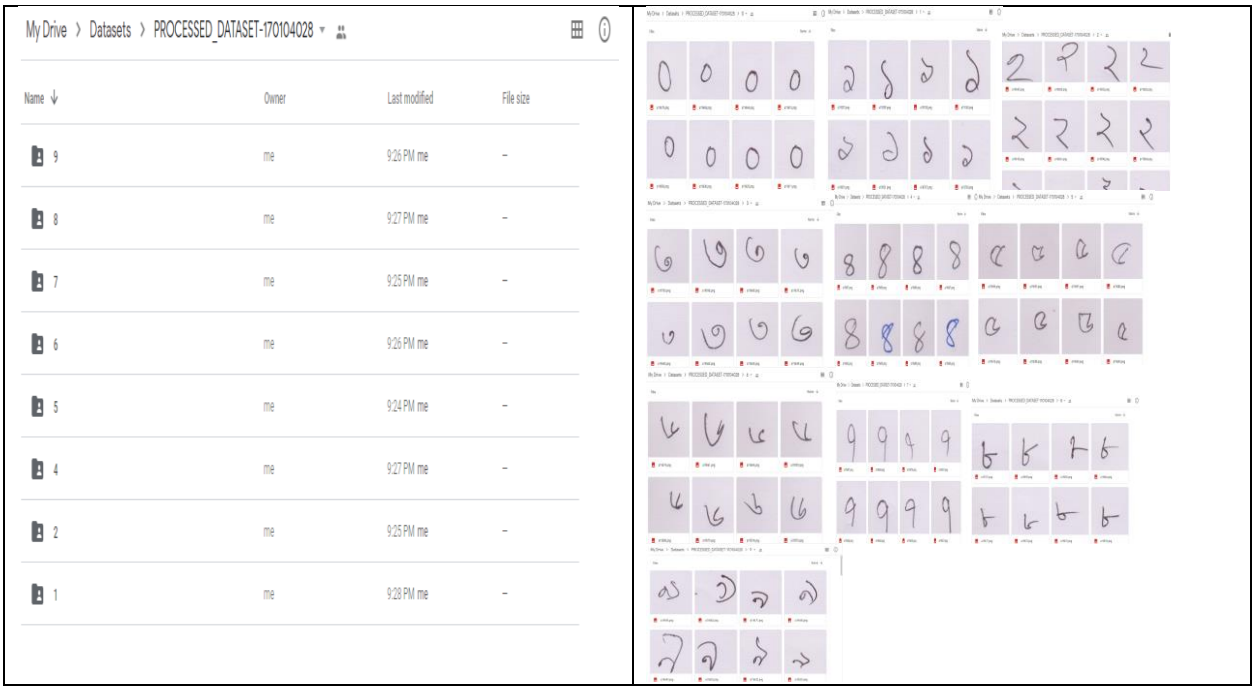


Figure 3: After format data

To format the dataset, we used the given code -

```
[ ]  traning_csv = '/content/training-a.csv'
     read_df = pd.read_csv(traning_csv)
```

```
number_of_digit_class = read_df['digit'].nunique()

DATASET_ROOT_DIR = './PROCESSED_DATASET_170104028/'   <-- Saved Path
path = Path(DATASET_ROOT_DIR)
path.mkdir(parents=True, exist_ok=True)
DATASET_ROOT_DIR = os.path.abspath(path)

for i in range(0,number_of_digit_class):
  select_digit = read_df[read_df['digit'] == i]
  for index,val in select_digit.iterrows():
    file_relative_path_from= f"./{val['database name']}/{val['filename']}"
    file_relative_dir_to = f"{DATASET_ROOT_DIR}/{val['digit']}"
    Path(file_relative_dir_to).mkdir(parents=True, exist_ok=True)   <-- Folder creation based on the digit
    shutil.copy(file_relative_path_from, file_relative_dir_to)   <-- Copy file
    print(file_relative_path_from , file_relative_dir_to , "OK")
```

```
All images are correct or not using pytorch ImageFolder

img_path: " /content/PROCESSED_DATASET-170104028

Show code
```

```
ZIP_FILE_NAME = 'PROCESSED_DATASET-170104028.zip'
!zip -r $ZIP_FILE_NAME 'PROCESSED_DATASET-170104028/'   <-- Zip Create
```

Figure 4: Data format code

2. Training Process

For training process, we need to follow two parts. First one is Experiment 1 and second one is Experiment 2.

2.1. Experiment 1

For experiment 1, must follow some rules. The rules are, we must use 6 hidden layer, 200 nodes, 0.01 learning rate etc. For model creation I use this code -

```
class LIN_MODEL(torch.nn.Module):
    def __init__(self,outDim):
        super(LIN_MODEL, self).__init__()

        self.fc_1 = torch.nn.Linear(784, 90)
        self.fc_2 = torch.nn.Linear(90, 50)
        self.fc_3 = torch.nn.Linear(50, 30)
        self.fc_4 = torch.nn.Linear(30, 18)
        self.fc_5 = torch.nn.Linear(18, 12)
        self.fc_6 = torch.nn.Linear(12, outDim)

        self.linear = torch.nn.Linear(784, outDim)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.nn.functional.relu(self.fc_1(x))
        x = torch.nn.functional.softmax(self.fc_2(x),dim=1)
        x = torch.nn.functional.relu(self.fc_3(x))
        x = torch.nn.functional.softmax(self.fc_4(x),dim=1)
        x = torch.nn.functional.relu(self.fc_5(x))
        x = self.fc_6(x)
        # x = torch.nn.functional.softmax(self.linear(x),dim=1)
        return x


model_1 = LIN_MODEL(OUTPUT_DIM).to(device)

summary( model_1, input_size=(1, 28, 28))
# select CPU or GPU as a device
print(model_1)
```

model forword pass

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [-1, 90]          70,650
            Linear-2                   [-1, 50]           4,550
            Linear-3                   [-1, 30]           1,530
            Linear-4                   [-1, 18]             558
            Linear-5                   [-1, 12]             228
            Linear-6                   [-1, 10]             130
================================================================
Total params: 77,646
Trainable params: 77,646
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.30
Estimated Total Size (MB): 0.30
----------------------------------------------------------------
```

Figure 5: Experiment 1, Model 1

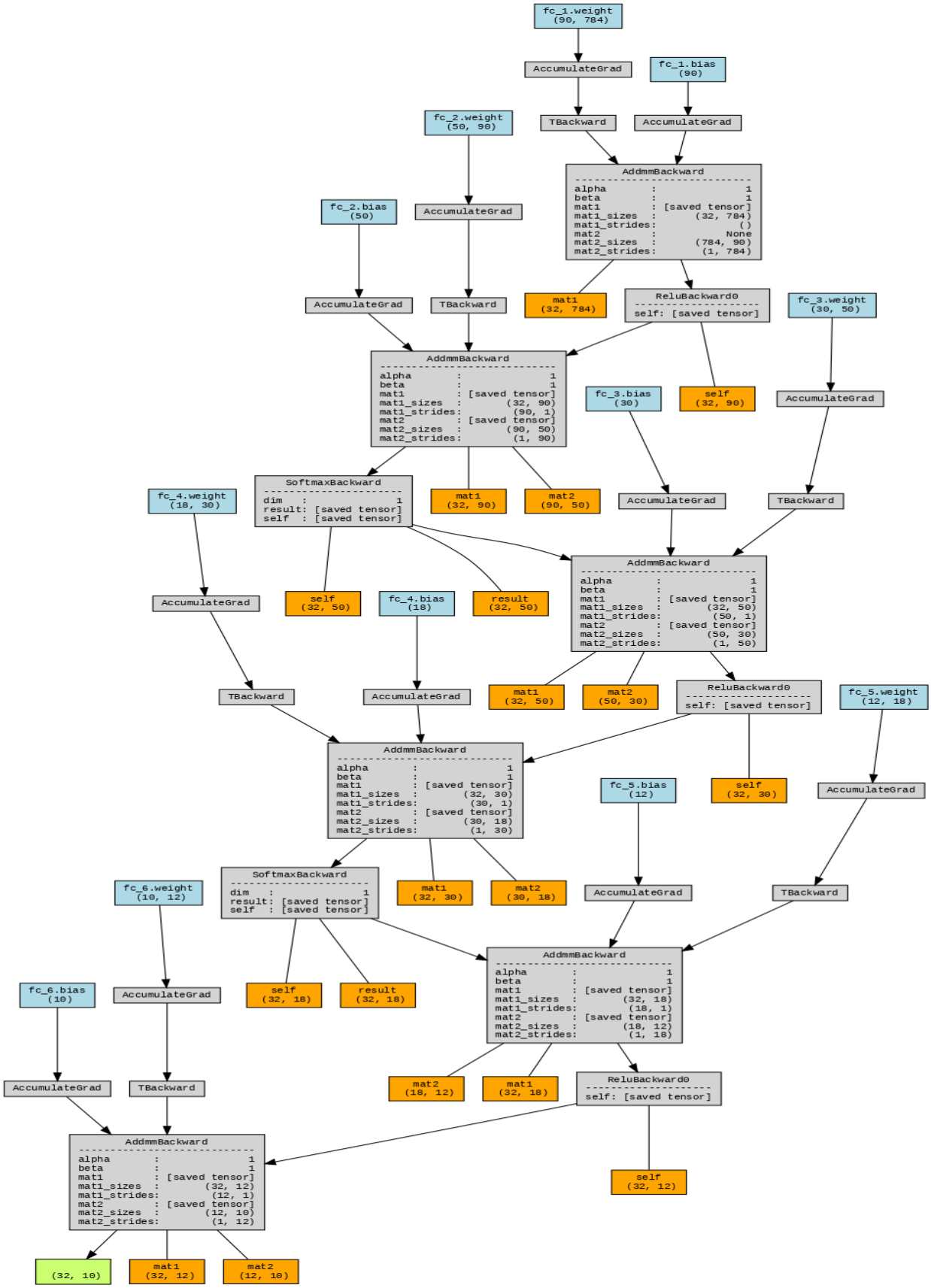


Figure 6: Model 1 summary

2.2. Experiment 2

For experiment 2, must follow the 85% accuracy of dataset.

2.2.1. Model-1 V2

```
class LIN_MODEL_2(torch.nn.Module):
    def __init__(self,outDim):
        super(LIN_MODEL_2, self).__init__()

        self.fc_1 = torch.nn.Linear(784, 742)
        self.fc_2 = torch.nn.Linear(742, 621)
        self.fc_3 = torch.nn.Linear(621, 510)
        self.fc_6 = torch.nn.Linear(510, outDim)

    def forward(self, x):

        x = x.view(-1, 28 * 28)
        x = torch.nn.functional.relu(self.fc_1(x))
        x = torch.nn.functional.relu(self.fc_2(x))
        x = torch.nn.functional.relu(self.fc_3(x))
        x = torch.nn.functional.relu(self.fc_6(x))

        return x

model_1_2 = LIN_MODEL_2(OUTPUT_DIM).to(device)

summary( model_1_2, input_size=(1, 28, 28))
# select CPU or GPU as a device
print(model_1_2)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 742]         582,470
            Linear-2                  [-1, 621]         461,403
            Linear-3                  [-1, 510]         317,220
            Linear-4                   [-1, 10]           5,110
================================================================
Total params: 1,366,203
Trainable params: 1,366,203
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 5.21
Estimated Total Size (MB): 5.23
----------------------------------------------------------------
```

Figure 7: Experiment 2, Model 1 V2, This is based on linear layer

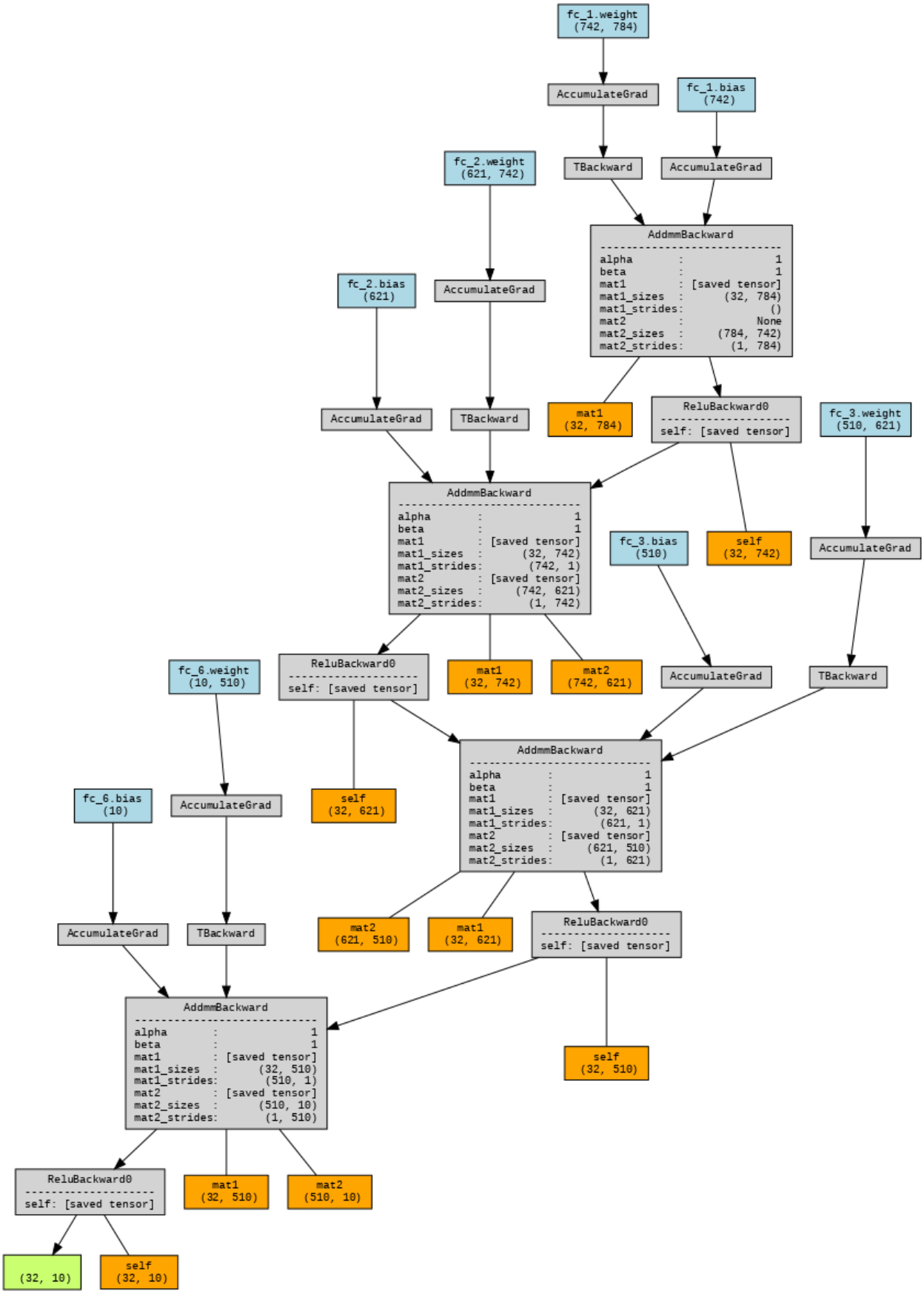


Figure 8: Model V2 flow

2.2.1. Model-1 V3

```
class LIN_MODEL_3(torch.nn.Module):
    def __init__(self,outDim):
        super(LIN_MODEL_3, self).__init__()

        self.fc_1 = torch.nn.Linear(784, 512)
        self.fc_2 = torch.nn.Linear(512, 256)
        self.fc_6 = torch.nn.Linear(256, outDim)

    def forward(self, x):

        x = x.view(-1, 28 * 28)
        x = torch.nn.functional.relu(self.fc_1(x))
        x = torch.nn.functional.relu(self.fc_2(x))
        x = torch.nn.functional.relu(self.fc_6(x))

        return x

model_1_3 = LIN_MODEL_3(OUTPUT_DIM).to(device)

summary( model_1_3, input_size=(1, 28, 28))
print(model_1_3)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]         401,920
            Linear-2                  [-1, 256]         131,328
            Linear-3                   [-1, 10]           2,570
================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 2.04
Estimated Total Size (MB): 2.05
----------------------------------------------------------------
```

Figure 9:Model 1 V3.

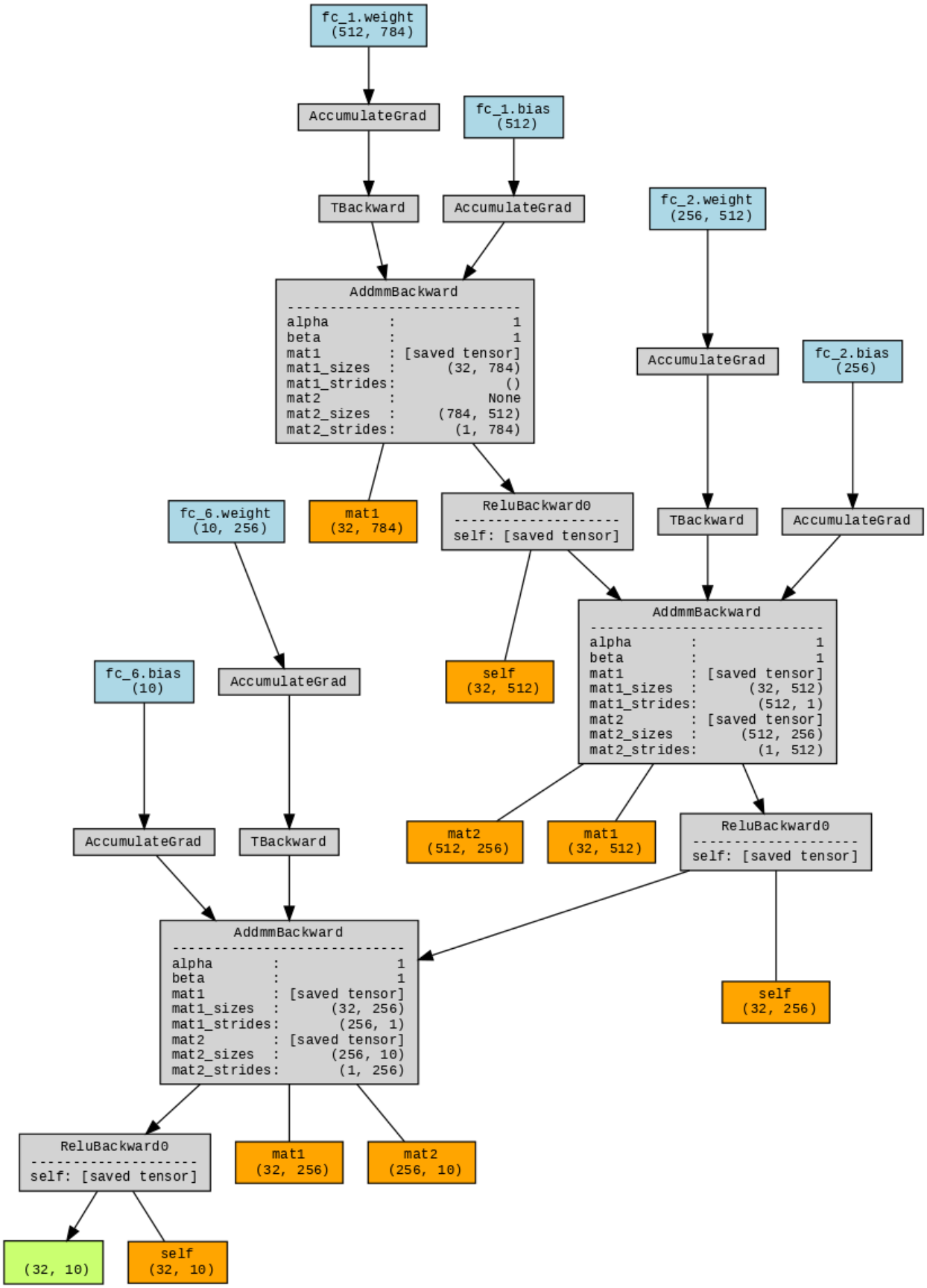


Figure 10: Model 1 V3 Flow summary

2.2.2. Model-2-CNN

```
class CNN(torch.nn.Module):
    def __init__(self ,outDim):
        super(CNN, self).__init__()

        #initializing convolution layer
        self.conv1 = torch.nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)
        self.conv2 = torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3)

        #initializing dropout
        self.dropout = torch.nn.Dropout(0.2)

        #initializing dropout
        self.pool= torch.nn.MaxPool2d(2,2)

        #initializing linear
        self.fc1 = torch.nn.Linear(32* 30* 30, 512)
        self.fc2 = torch.nn.Linear(512,64)
        self.fc3 = torch.nn.Linear(64,10)


    def forward(self, x):
        x = self.pool(torch.nn.functional.relu(self.conv1(x)))
        x = self.dropout(x)
        x = self.pool(torch.nn.functional.relu(self.conv2(x)))
        x = self.dropout(x)
        x = x.view(-1, 32* 30* 30)
        # print(x.shape)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model_2 = CNN(OUTPUT_DIM).to(device)

summary( model_2, input_size=(3, 128, 128))
# select CPU or GPU as a device
print(model_2)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 126, 126]             448
         MaxPool2d-2           [-1, 16, 63, 63]               0
           Dropout-3           [-1, 16, 63, 63]               0
            Conv2d-4           [-1, 32, 61, 61]           4,640
         MaxPool2d-5           [-1, 32, 30, 30]               0
           Dropout-6           [-1, 32, 30, 30]               0
            Linear-7                 [-1, 512]      14,746,112
            Linear-8                  [-1, 64]          32,832
            Linear-9                  [-1, 10]             650
================================================================
Total params: 14,784,682
Trainable params: 14,784,682
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 4.26
Params size (MB): 56.40
Estimated Total Size (MB): 60.85
----------------------------------------------------------------
```

Figure 11: Model 3, Based on the CNN

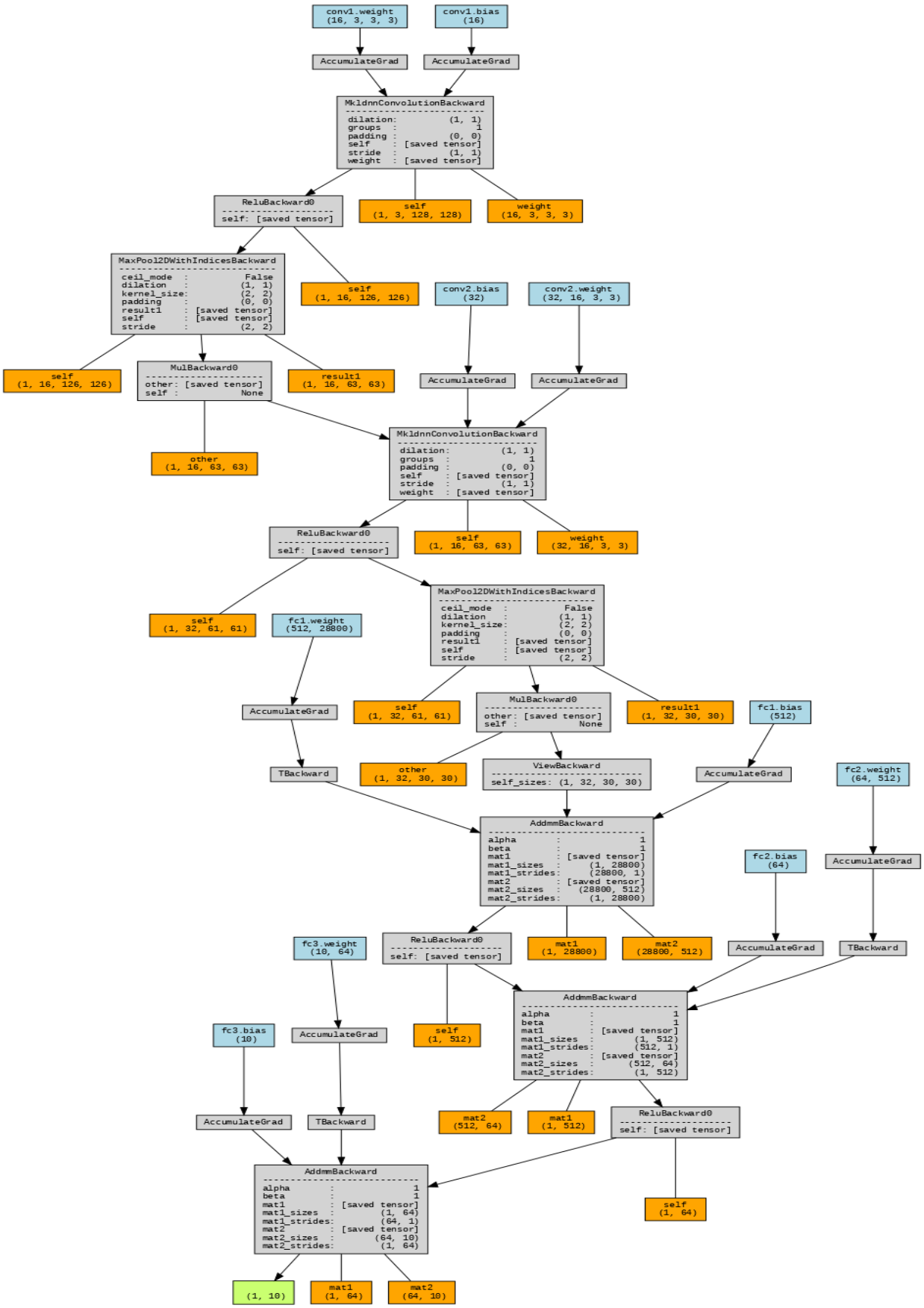


Figure 12: Model 2 flow summary

2.2.3. Model-3-RSNET152

```
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False


num_classes = OUTPUT_DIM


model_3 = models.resnet152(pretrained=True)
set_parameter_requires_grad(model_3, True)
num_ftrs = model_3.fc.in_features
model_3.fc = torch.nn.Linear(num_ftrs, num_classes)

model_3 = model_3.to(device)
```

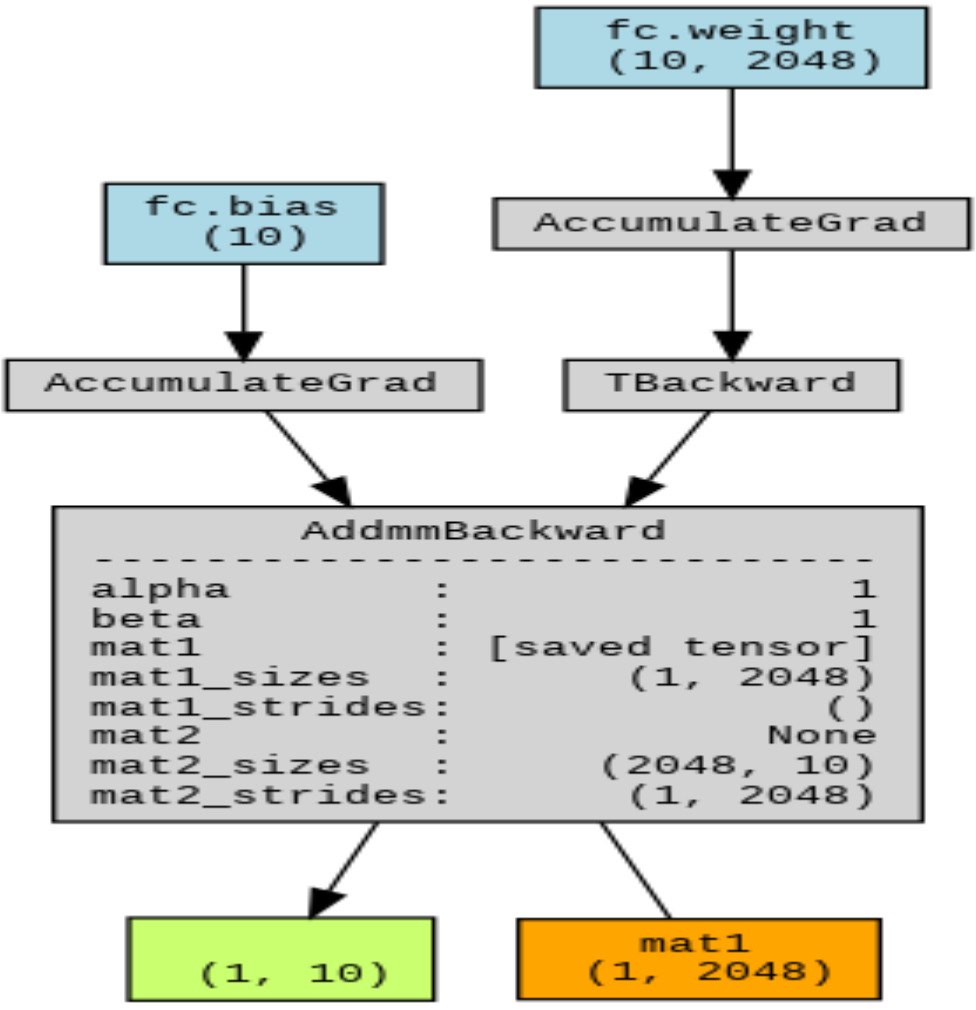


Figure 13:Model 3, Based on the pretrained and flow of summary

2.3. Training Flow

For training the mode. I use 4 basic flows.

1. Hyper parameter and variables
2. Load dataset
3. Model Creation
4. Model train and save the model

2.3.1. Hyper parameter

For model hyper parameter and variables. I used this code.

```
base_dir = '/content/PROCESSED_DATASET-170104028'
BATCH_SIZE = 32

# 28(model-1), 128(model-2), 224(model-3),
IMAGE_SIZE = 128
LEARNING_RATE = 0.01
TEST_SIZE = 0.2
OUTPUT_DIM=10
```

```
from pathlib import Path

START=0
model_save_path = './EXP-2-GRAY-OWN/' # must give `/` for the folder directory

model_used= 'EXPERIMENT_MODEL_2_GRAY'
# model_ft = model_1
# model_ft = model_1_2
# model_ft = model_1_3
model_ft = model_2
# model_ft = model_3
optimizer_ft = optimizer
all_tranning_loss = [];
all_validation_loss = [];
all_tranning_accuracy = [];
all_validation_accuracy = [];

load_saved_model='/content/EXP-1-2-OWN/EXPERIMENT_MODEL_1_2-checkpoint-epoch-187.pt'
p = Path(load_saved_model)
if len(load_saved_model) > 1 and p.exists():
  loadedModel = torch.load(load_saved_model, map_location=device)
  model_used = loadedModel['MODEL_USED']
  model_ft =  loadedModel['model_full']
  model_ft.load_state_dict(loadedModel['model_state'])
  optimizer_ft = loadedModel['optimizer_full']
  optimizer_ft.load_state_dict(loadedModel['optimizer_state'] )
  START = loadedModel['epoch'] + 1
  all_tranning_loss = loadedModel['tranning_loss']
  all_validation_loss = loadedModel['validation_loss']
  all_tranning_accuracy = loadedModel['tranning_acc']
  all_validation_accuracy = loadedModel['validation_acc']

END=100
trainloader= trainloader
testloader = validationloader
```

Figure 14: Training Parameter's

2.3.2. Load Dataset

For loading the dataset. I use ImageFolder pytorch function. And I use transforms of the dataset based on the model parameters. Mainly I used three shape of image there are 1*28*28, 1*128*128, 3*128*128, 3*224*224, 1*224*224. The main code is-

```
transform = transforms.Compose([
                                # transforms.ToPILImage(),

                                ## this is only for when model is 1
                                # transforms.Grayscale(),

                                # transforms.RandomRotation(20,expand=True), ## adding random rotation 20deg
                                # torchvision.transforms.ColorJitter(hue=.05, saturation=.05), ## adding color filter
                                # transforms.RandomVerticalFlip(), ## adding vertical flip
                                # transforms.RandomHorizontalFlip(), ## adding horizontal flip
                                transforms.Resize(IMAGE_SIZE),  ## image resize
                                transforms.CenterCrop(IMAGE_SIZE), ## image center crop
                                transforms.ToTensor(), ## array converted into torch tensor and then divided by 255 (1.0/255)
                                # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                               ])

#load training dataset
dataset = torchvision.datasets.ImageFolder(base_dir, transform=transform)
n = len(dataset)
n_test = int(TEST_SIZE * n) # 10% validation
trainDataset, validDataSet = torch.utils.data.random_split(dataset,[n - n_test,n_test]) #random split dataset
trainloader = torch.utils.data.DataLoader(trainDataset, batch_size=BATCH_SIZE, shuffle=True, pin_memory=True,)
validationloader = torch.utils.data.DataLoader(validDataSet, batch_size=BATCH_SIZE, shuffle=True, pin_memory=True,)
print("Length of the trainloader:", len(trainloader ) * BATCH_SIZE)
print("Length of the validationloader:", len(validationloader ) * BATCH_SIZE)

Length of the trainloader: 15776
Length of the validationloader: 3968
```

Based on the channels

Based on the Image Size

Figure 15: Dataset Transform

The basic visualization of dataset-

```
{0: 1975, 1: 1963, 2: 1984, 3: 1976, 4: 1982, 5: 1965, 6: 1983, 7: 1960, 8: 1950, 9: 1964}
```

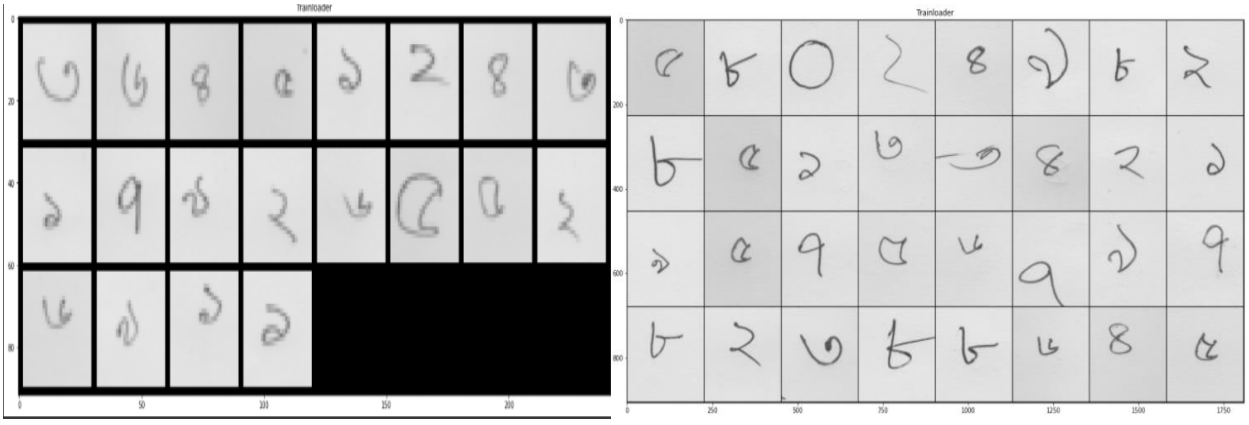


Figure 16: Basic Visualization

2.3.3. Model Creation

In model creation I divide into 2 parts first one is the model class creation and other one optimizer & criterion. In Section2.1 and Section2.2 we saw all the model information. And the optimizer & criterion is given bellow-

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_2.parameters(),lr=LEARNING_RATE, )
```

Based on Learning Rate Value

2.3.4. Model train and save the model

For training the model I use two functions first one is Model save and second one is model training function. For model saving the I use this function-

```
def save_model(
                MODEL_USED,
                SAVEPATH,
                epoch,
                batch_size,
                model,
                optimizer,
                image_size,
                tranning_loss=[],
                tranning_acc=[],
                validation_loss=[],
                validation_acc=[],
                learning_rate=0.001,
                meta_data=None):
  SAVEPATH += f"{MODEL_USED}-checkpoint-epoch-{epoch}.pt"
  save_obj = {
        'MODEL_USED':MODEL_USED,
        'batch_size':batch_size,
        'epoch': epoch,
        'model_full': model,
        'optimizer_full': optimizer,
        'model_state': model.state_dict(),
        'optimizer_state': optimizer.state_dict(),
        'image_size': image_size,
        'tranning_loss': tranning_loss,
        'tranning_acc': tranning_acc,
        'validation_loss': validation_loss,
        'validation_acc': validation_acc,
        'learning_rate':learning_rate,
        'meta_data':meta_data
        }
  torch.save(save_obj, SAVEPATH)
```

Figure 17: Model saved code

And the model train function is –

```
train_model(START, END,model_used, model_save_path, model_ft, criterion, optimizer_ft, trainloader, testloader,
            all_tranning_loss,
            all_validation_loss,
            all_tranning_accuracy,
            all_validation_accuracy)
```

Figure 18: Model Train Function

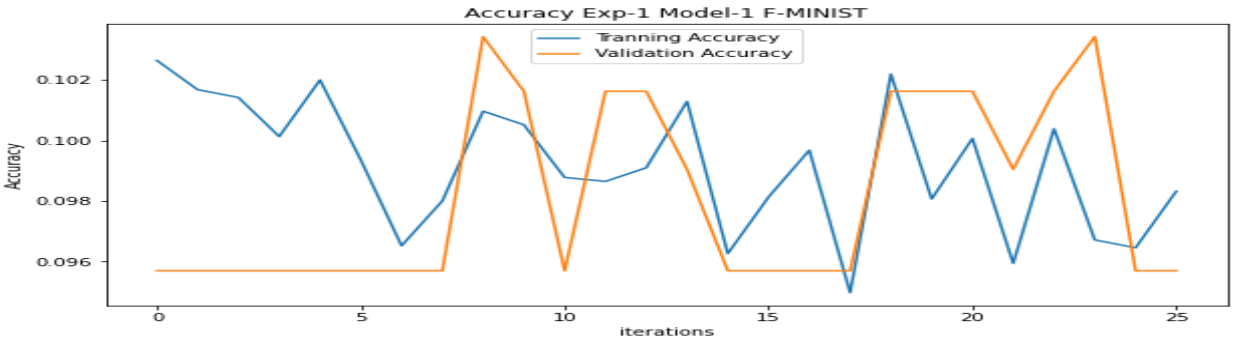
3. Training Results

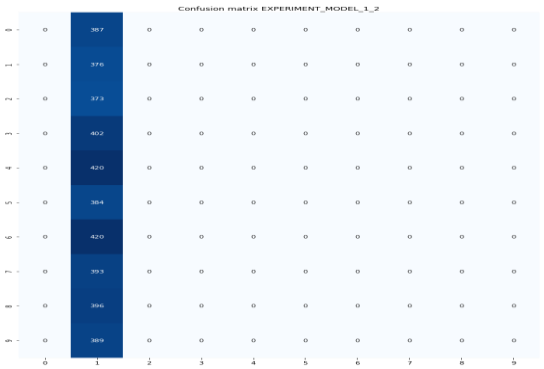
Here we can saw all the training result of model. Also you can see he parameters of the model training process.

3.1. Experiment 1

3.1.1. Own(training-a)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	20
Iteration	20490
Training and Testing Split	80% and 20%
Image Shape	1 * 28 * 28
Optimizer	SGD





	precision	recall	f1-score	support
0	0.00	0.00	0.00	387
1	0.10	1.00	0.17	376
2	0.00	0.00	0.00	373
3	0.00	0.00	0.00	402
4	0.00	0.00	0.00	420
5	0.00	0.00	0.00	384
6	0.00	0.00	0.00	420
7	0.00	0.00	0.00	393
8	0.00	0.00	0.00	396
9	0.00	0.00	0.00	389
accuracy			0.10	3940
macro avg	0.01	0.10	0.02	3940
weighted avg	0.01	0.10	0.02	3940

3.1.2. Fashion-MNIST

Name	Values
Total dataset	60000
Learning rate	0.01
Batch Size	20
Iteration	65000
Image Shape	1 * 28 * 28
Optimizer	SGD

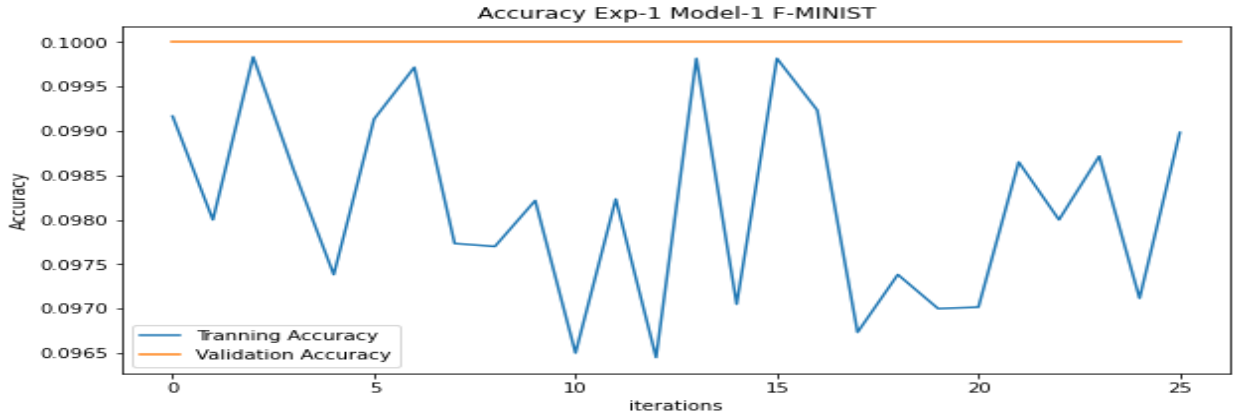


Figure 19: Accuracy Graph Exp-1

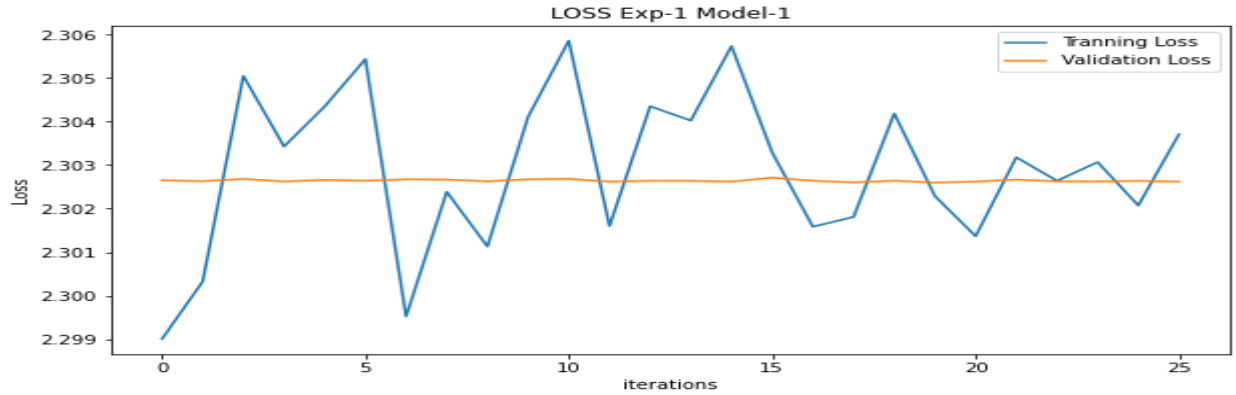
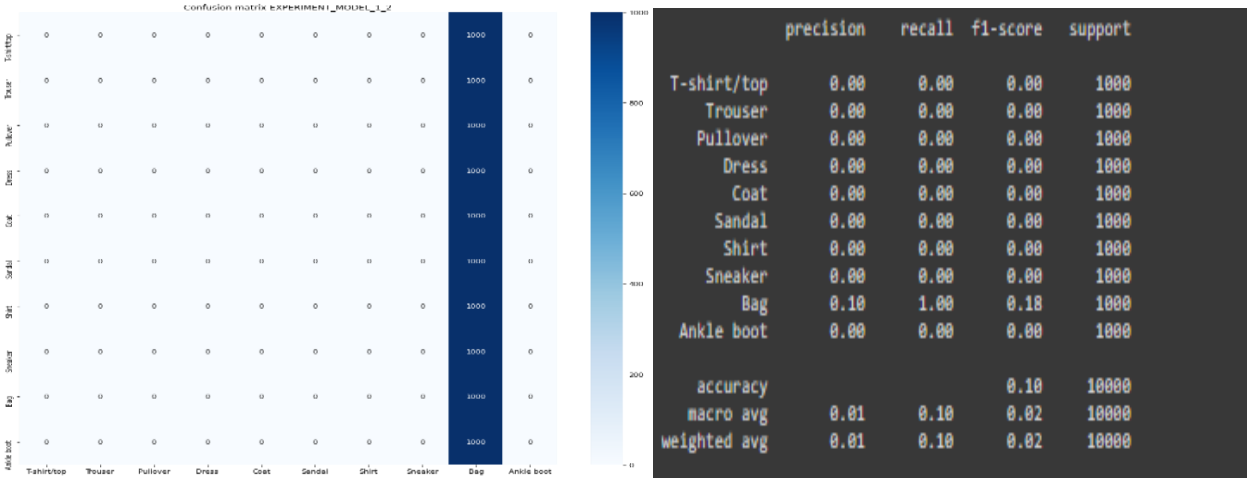


Figure 20: Loss Graph Exp-1



	precision	recall	f1-score	support
T-shirt/top	0.00	0.00	0.00	1000
Trouser	0.00	0.00	0.00	1000
Pullover	0.00	0.00	0.00	1000
Dress	0.00	0.00	0.00	1000
Coat	0.00	0.00	0.00	1000
Sandal	0.00	0.00	0.00	1000
Shirt	0.00	0.00	0.00	1000
Sneaker	0.00	0.00	0.00	1000
Bag	0.10	1.00	0.18	1000
Ankle boot	0.00	0.00	0.00	1000
accuracy			0.10	10000
macro avg	0.01	0.10	0.02	10000
weighted avg	0.01	0.10	0.02	10000

Figure 21: Performance Matrice of exp-1

3.2. Experiment 2 – Model 1-V2

3.2.1. Own(training-a)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	32
Iteration	125715
Training and Testing Split	80% and 20%
Image Shape	1 * 28 * 28
Epochs	255
Optimizer	SGD

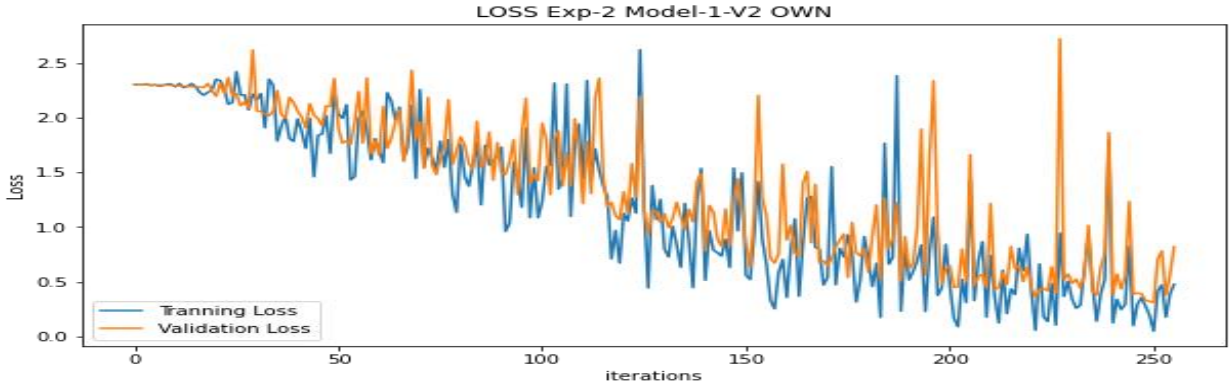


Figure 22:Loss graph exp-2

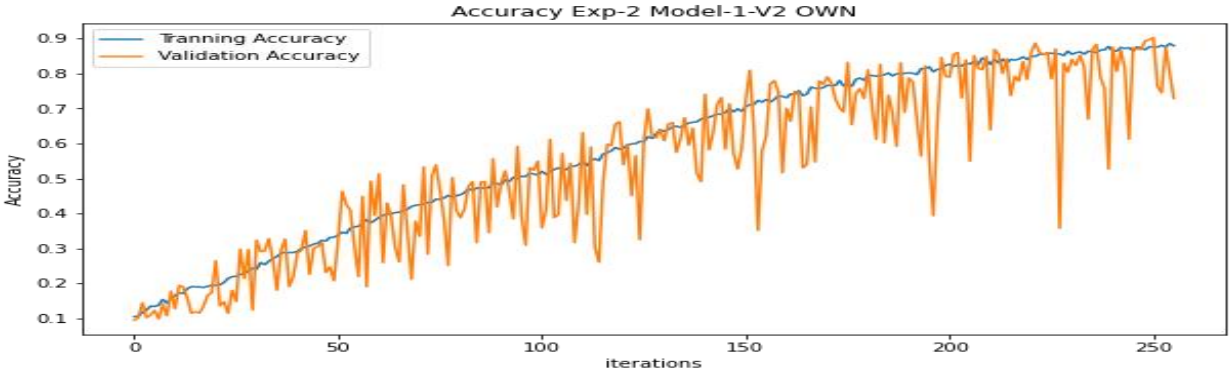
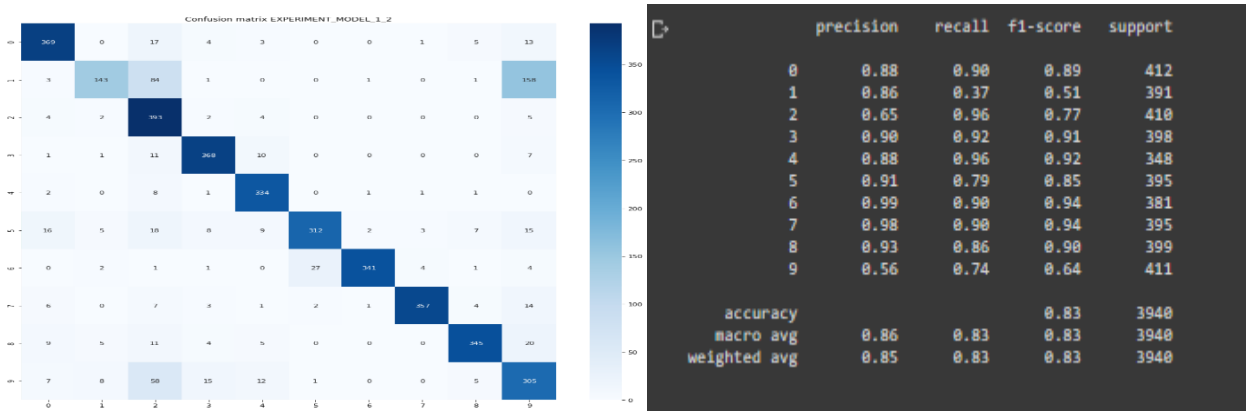


Figure 23: Accuracy graph exp-2



	precision	recall	f1-score	support
0	0.88	0.90	0.89	412
1	0.86	0.37	0.51	391
2	0.65	0.96	0.77	410
3	0.90	0.92	0.91	398
4	0.88	0.96	0.92	348
5	0.91	0.79	0.85	395
6	0.99	0.90	0.94	381
7	0.98	0.90	0.94	395
8	0.93	0.86	0.90	399
9	0.56	0.74	0.64	411
accuracy			0.83	3940
macro avg	0.86	0.83	0.83	3940
weighted avg	0.85	0.83	0.83	3940

Figure 24: Performance matrices of exp-2 model v2

3.2.2. Fashion-MNIST

Name	Values
Total dataset	60000
Learning rate	0.01
Batch Size	32
Iteration	478125
Training and Testing Split	80% and 20%
Image Shape	1 * 28 * 28
Epochs	255
Optimizer	SGD

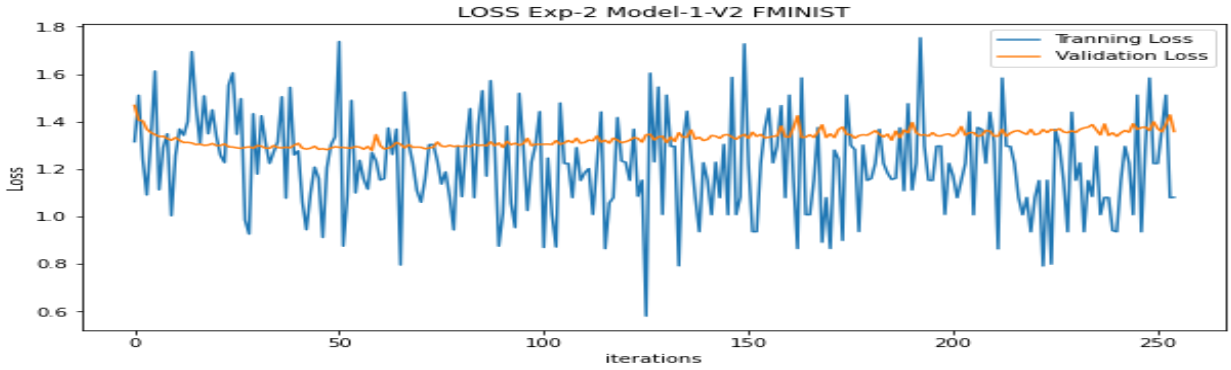


Figure 25: F-MINIST Exp-2 Loss

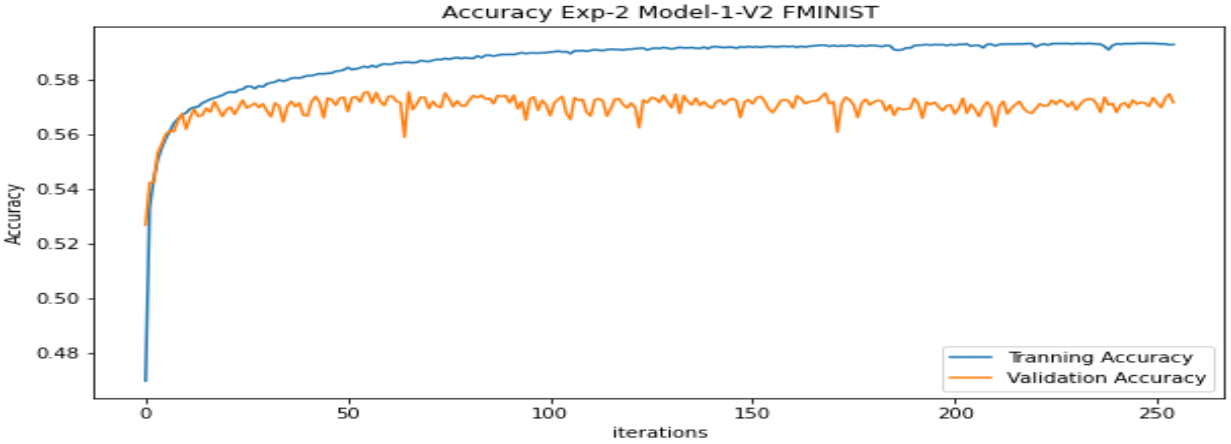
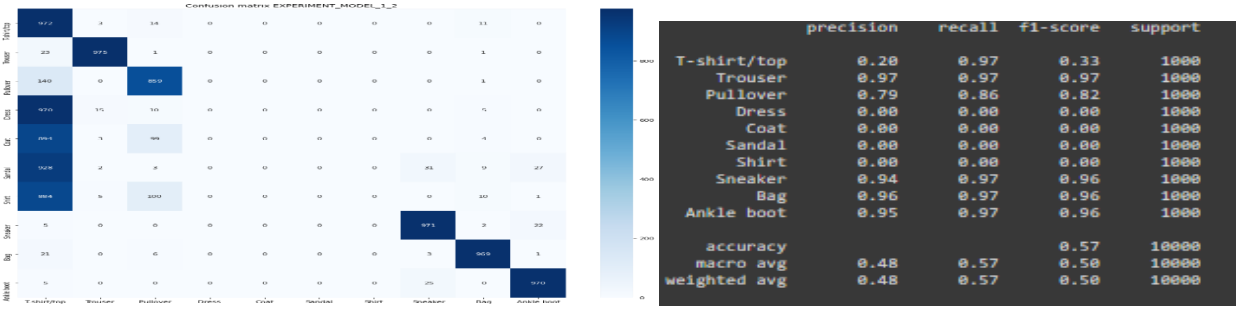


Figure 26:F-MINIST Exp-2 Accuracy



	precision	recall	f1-score	support
T-shirt/top	0.20	0.97	0.33	1000
Trouser	0.97	0.97	0.97	1000
Pullover	0.79	0.86	0.82	1000
Dress	0.00	0.00	0.00	1000
Coat	0.00	0.00	0.00	1000
Sandal	0.00	0.00	0.00	1000
Shirt	0.00	0.00	0.00	1000
Sneaker	0.94	0.97	0.96	1000
Bag	0.96	0.97	0.96	1000
Ankle boot	0.95	0.97	0.96	1000
accuracy			0.57	10000
macro avg	0.48	0.57	0.50	10000
weighted avg	0.48	0.57	0.50	10000

Figure 27: Performance matrices F- MINIST

3.3. Experiment 2 – Model 1-V3

3.3.1. Own(training-a)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	32
Iteration	125715
Training and Testing Split	80% and 20%
Epochs	299
Image Shape	1 * 28 * 28
Optimizer	SGD

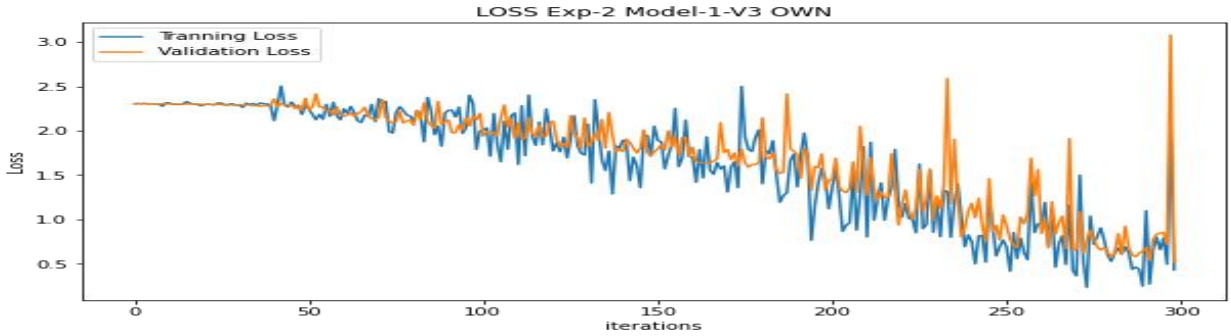


Figure 28: Loss graph exp-2

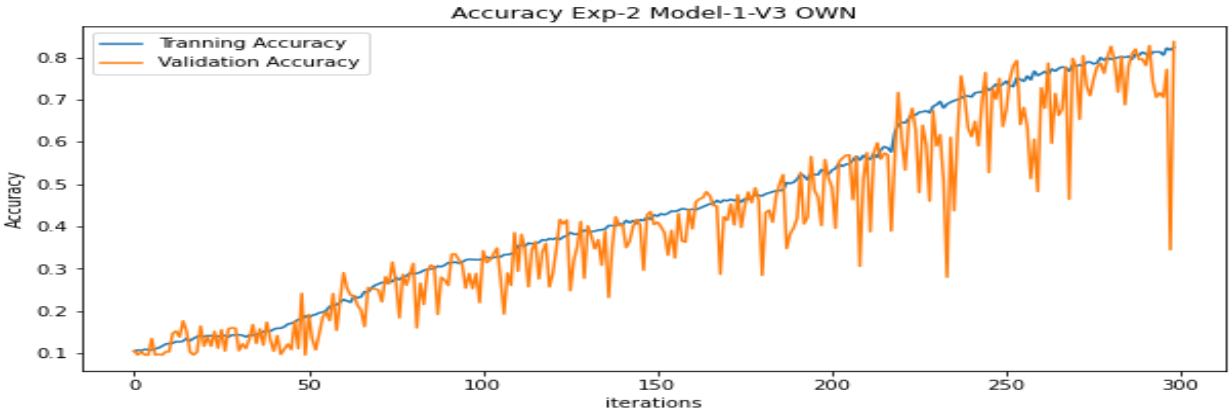
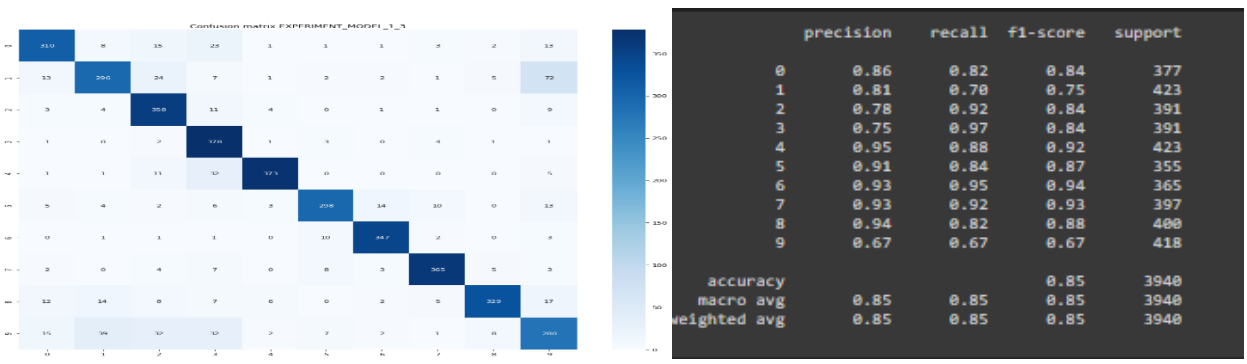


Figure 29:Accuracy graph exp-2

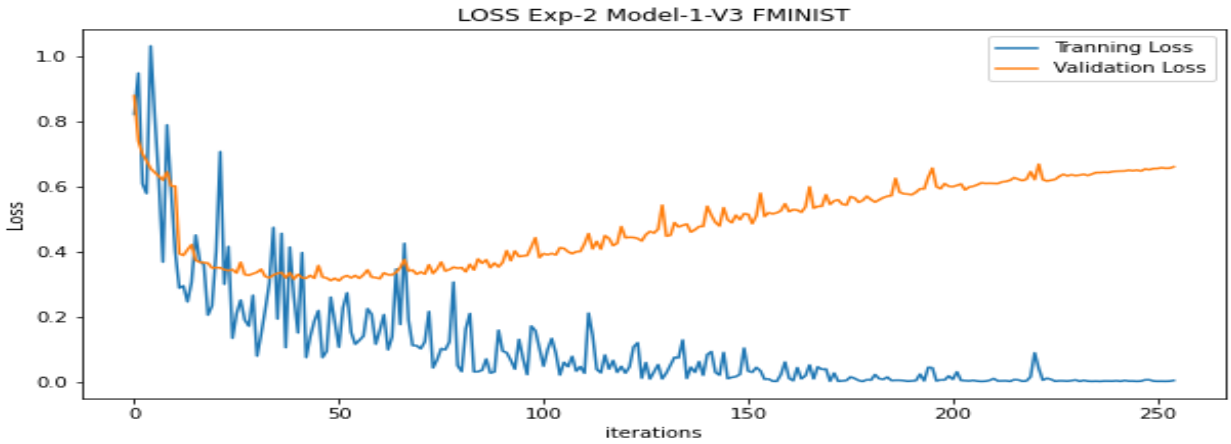


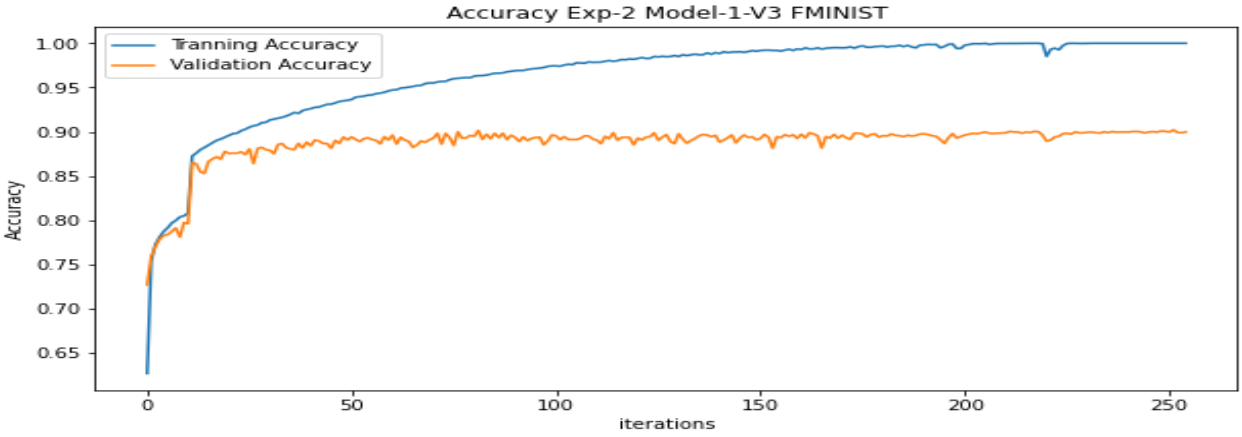
	precision	recall	f1-score	support
0	0.86	0.82	0.84	377
1	0.81	0.70	0.75	423
2	0.78	0.92	0.84	391
3	0.75	0.97	0.84	391
4	0.95	0.88	0.92	423
5	0.91	0.84	0.87	355
6	0.93	0.95	0.94	365
7	0.93	0.92	0.93	397
8	0.94	0.82	0.88	400
9	0.67	0.67	0.67	418
accuracy			0.85	3940
macro avg	0.85	0.85	0.85	3940
weighted avg	0.85	0.85	0.85	3940

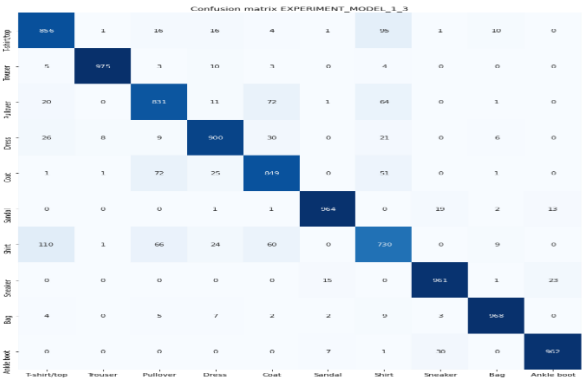
Figure 30: Performance Matrices

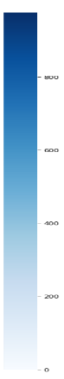
3.3.2. Fashion-MNIST

Name	Values
Total dataset	60000
Learning rate	0.01
Batch Size	32
Iteration	478125
Training and Testing Split	80% and 20%
Image Shape	1 * 28 * 28
Epochs	255
Optimizer	SGD









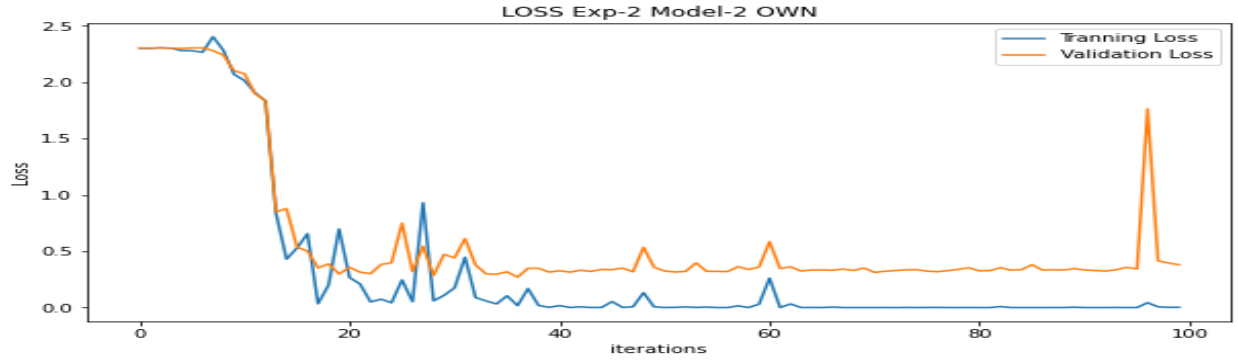
	precision	recall	f1-score	support
T-shirt/top	0.84	0.86	0.85	1000
Trouser	0.99	0.97	0.98	1000
Pullover	0.83	0.83	0.83	1000
Dress	0.91	0.90	0.90	1000
Coat	0.83	0.85	0.84	1000
Sandal	0.97	0.96	0.97	1000
Shirt	0.75	0.73	0.74	1000
Sneaker	0.95	0.96	0.95	1000
Bag	0.97	0.97	0.97	1000
Ankle boot	0.96	0.96	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

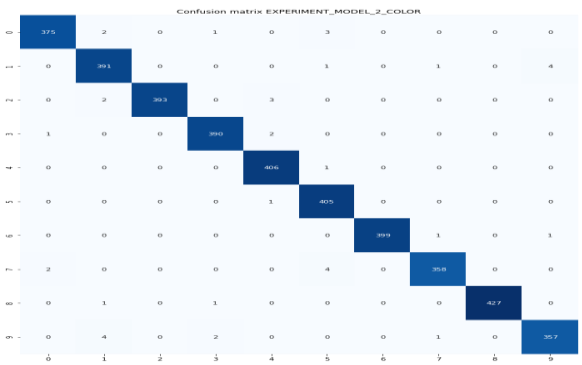
3.4. Experiment 2 – Model 2

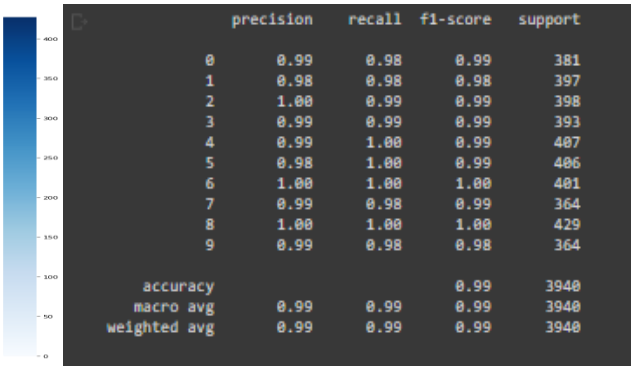
3.4.1. Own(training-a)

3.4.1.1. Input Channel 3(RGB-Image)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	32
Iteration	49300
Training and Testing Split	80% and 20%
Epochs	100
Image Shape	3 * 128 * 128
Optimizer	SGD



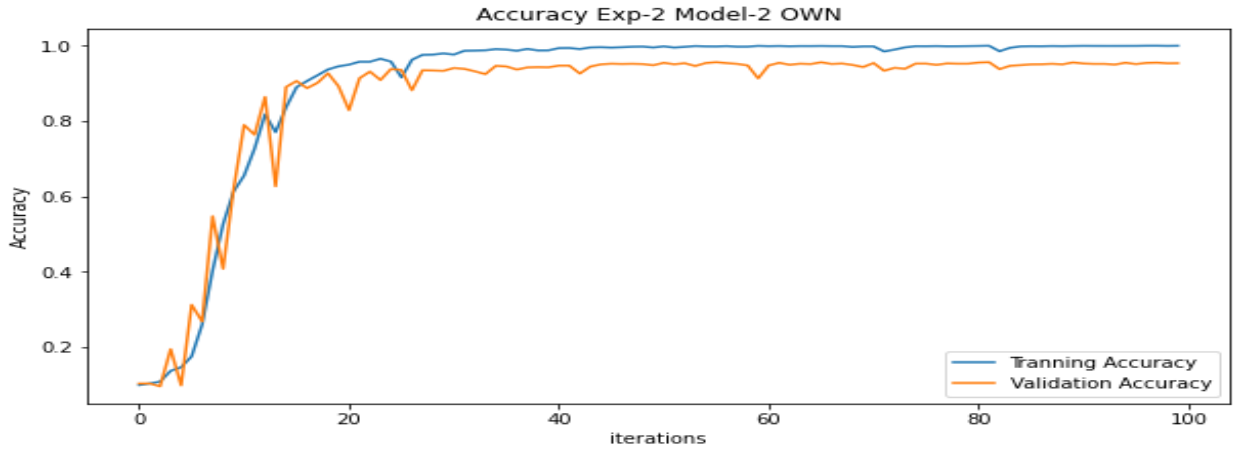


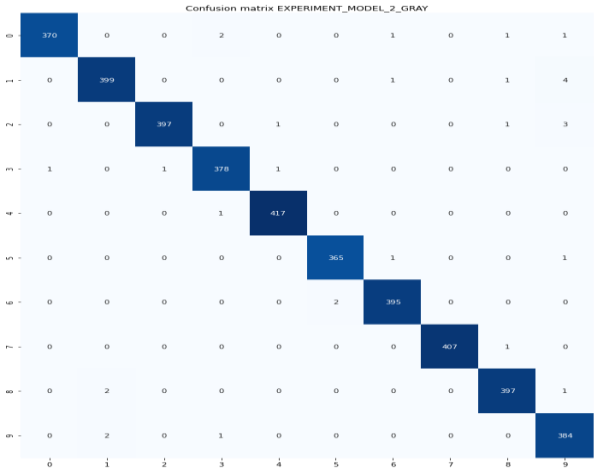


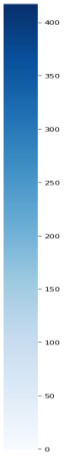
	precision	recall	f1-score	support
0	0.99	0.98	0.99	381
1	0.98	0.98	0.98	397
2	1.00	0.99	0.99	398
3	0.99	0.99	0.99	393
4	0.99	1.00	0.99	407
5	0.98	1.00	0.99	406
6	1.00	1.00	1.00	401
7	0.99	0.98	0.99	364
8	1.00	1.00	1.00	429
9	0.99	0.98	0.98	364
accuracy			0.99	3940
macro avg	0.99	0.99	0.99	3940
weighted avg	0.99	0.99	0.99	3940

3.4.1.2. Input Channel 1(GRAY-Image)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	32
Iteration	49300
Training and Testing Split	80% and 20%
Epochs	100
Image Shape	1 * 128 * 128
Optimizer	SGD



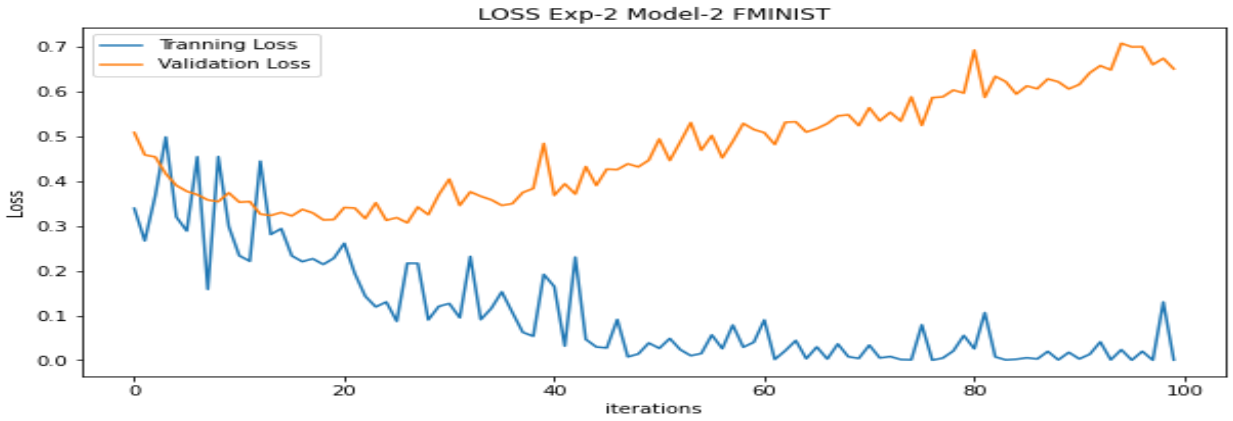


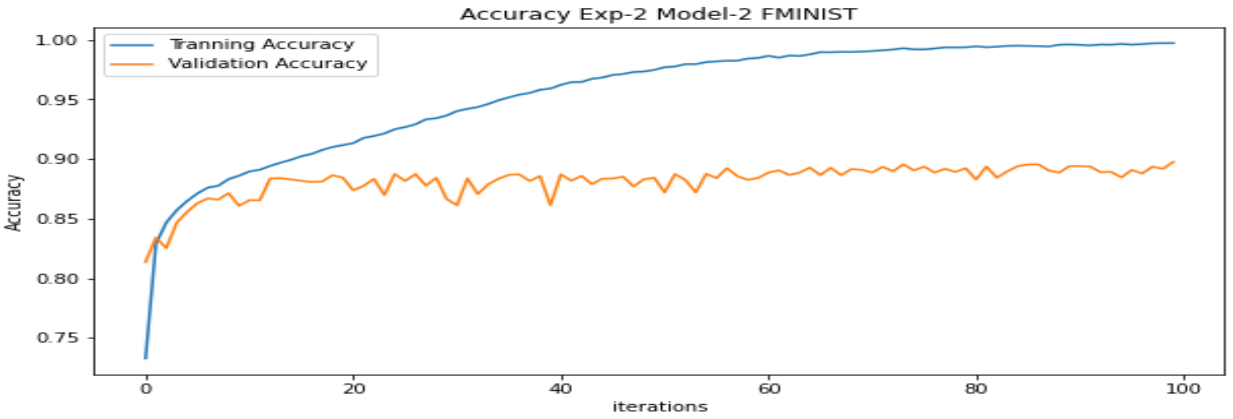


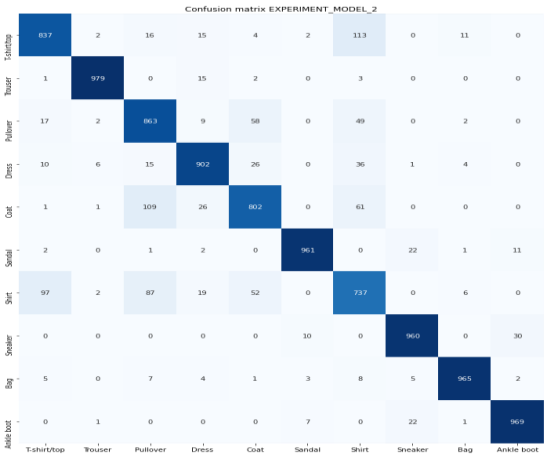
	precision	recall	f1-score	support
0	0.99	0.98	0.99	381
1	0.98	0.98	0.98	397
2	1.00	0.99	0.99	398
3	0.99	0.99	0.99	393
4	0.99	1.00	0.99	407
5	0.98	1.00	0.99	406
6	1.00	1.00	1.00	401
7	0.99	0.98	0.99	364
8	1.00	1.00	1.00	429
9	0.99	0.98	0.98	364
accuracy			0.99	3940
macro avg	0.99	0.99	0.99	3940
weighted avg	0.99	0.99	0.99	3940

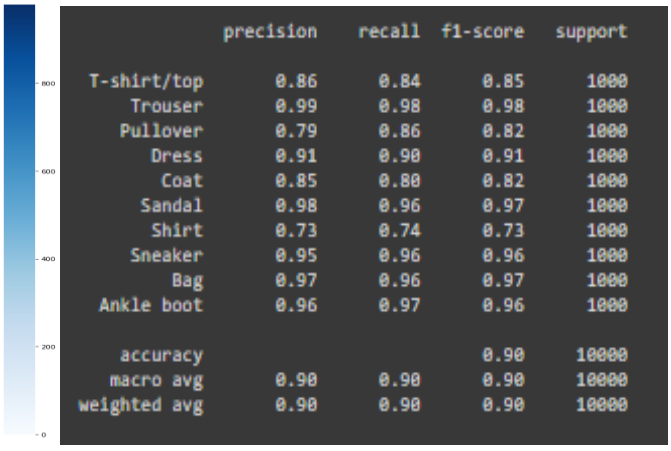
3.4.2. Fashion-MNIST

Name	Values
Total dataset	60000
Learning rate	0.01
Batch Size	32
Iteration	187500
Training and Testing Split	80% and 20%
Image Shape	1 * 128 * 128
Epochs	100
Optimizer	SGD









	precision	recall	f1-score	support
T-shirt/top	0.86	0.84	0.85	1000
Trouser	0.99	0.98	0.98	1000
Pullover	0.79	0.86	0.82	1000
Dress	0.91	0.90	0.91	1000
Coat	0.85	0.80	0.82	1000
Sandal	0.98	0.96	0.97	1000
Shirt	0.73	0.74	0.73	1000
Sneaker	0.95	0.96	0.96	1000
Bag	0.97	0.96	0.97	1000
Ankle boot	0.96	0.97	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

3.5. Experiment 2 – Model 3

3.5.1. Own(training-a)

3.5.1.1. Input Channel 3(RGB-Image)

Name	Values
Total dataset	19702
Learning rate	0.001
Batch Size	32
Iteration	12325
Training and Testing Split	80% and 20%
Epochs	25
Image Shape	3 * 224 * 224
Optimizer	SGD

Figure 31: Loss Graph of Model-3

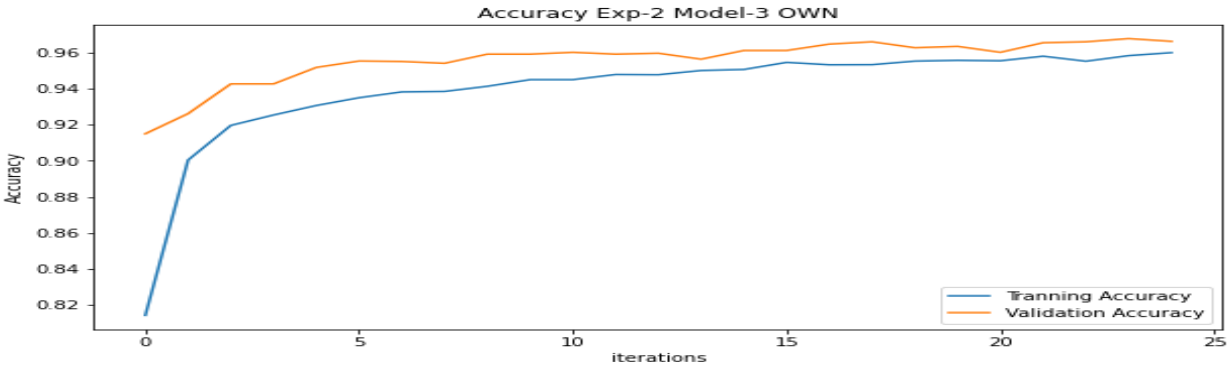
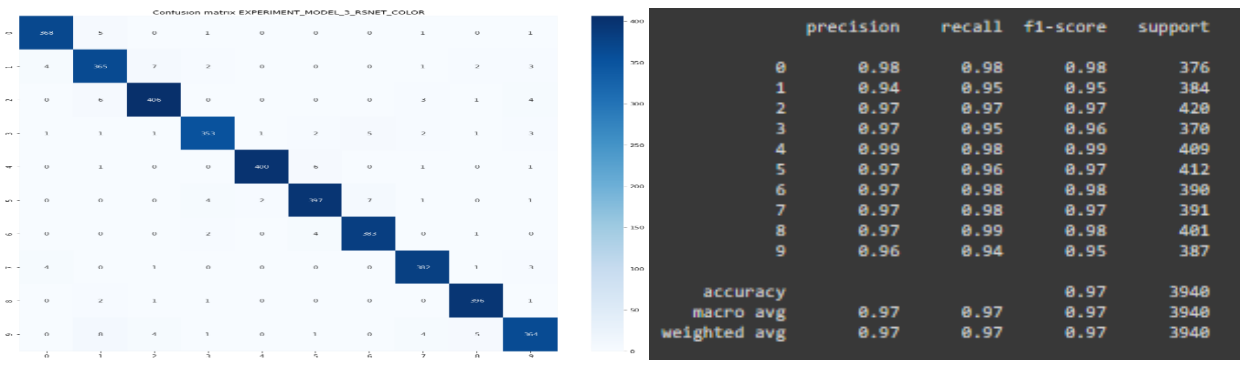


Figure 32:Accuracy Graph of Model 3

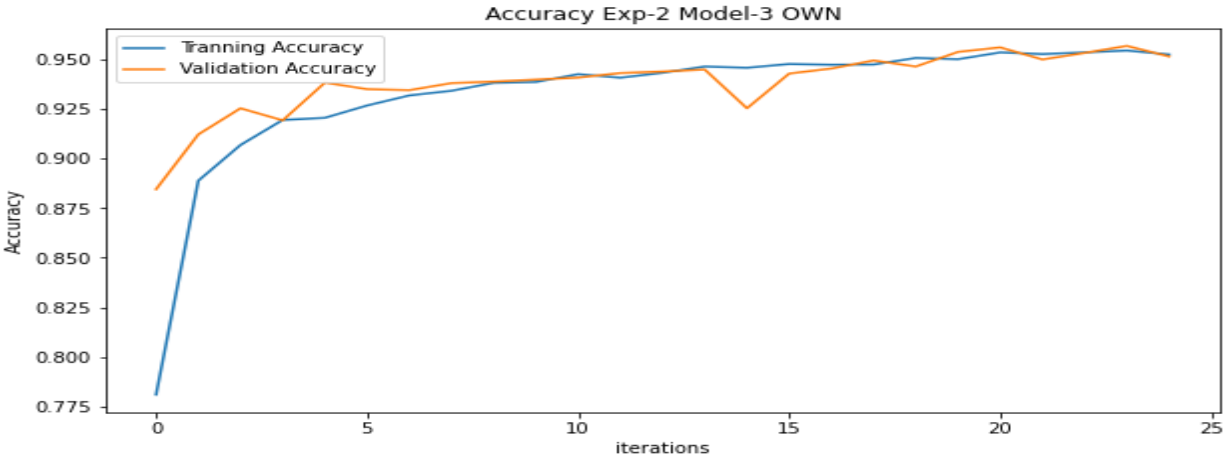


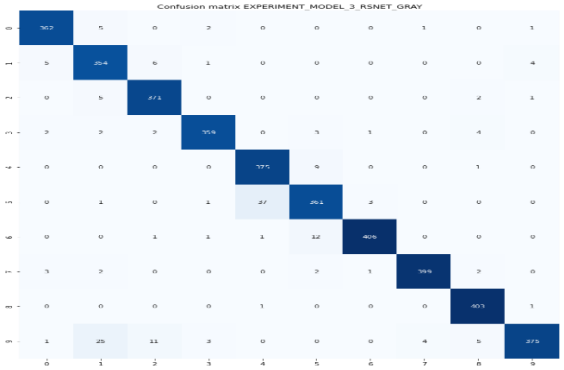
	precision	recall	f1-score	support
0	0.98	0.98	0.98	376
1	0.94	0.95	0.95	384
2	0.97	0.97	0.97	420
3	0.97	0.95	0.96	370
4	0.99	0.98	0.99	409
5	0.97	0.96	0.97	412
6	0.97	0.98	0.98	390
7	0.97	0.98	0.97	391
8	0.97	0.99	0.98	401
9	0.96	0.94	0.95	387
accuracy			0.97	3940
macro avg	0.97	0.97	0.97	3940
weighted avg	0.97	0.97	0.97	3940

Figure 33: Performance Matrices

3.5.1.2. Input Channel 1(GRAY-Image)

Name	Values
Total dataset	19702
Learning rate	0.01
Batch Size	32
Iteration	49300
Training and Testing Split	80% and 20%
Epochs	100
Image Shape	1 * 244 * 244
Optimizer	SGD





	precision	recall	f1-score	support
0	0.97	0.98	0.97	371
1	0.90	0.96	0.93	370
2	0.95	0.98	0.96	379
3	0.98	0.96	0.97	373
4	0.91	0.97	0.94	385
5	0.93	0.90	0.91	403
6	0.99	0.96	0.98	421
7	0.99	0.98	0.98	409
8	0.97	1.00	0.98	405
9	0.98	0.88	0.93	424
accuracy			0.96	3940
macro avg	0.96	0.96	0.96	3940
weighted avg	0.96	0.96	0.96	3940

3.5.2. Fashion-MNIST

Name	Values
Total dataset	60000
Learning rate	0.01
Batch Size	32
Iteration	45000
Training and Testing Split	80% and 20%
Image Shape	1 * 224 * 224
Epochs	24
Optimizer	SGD

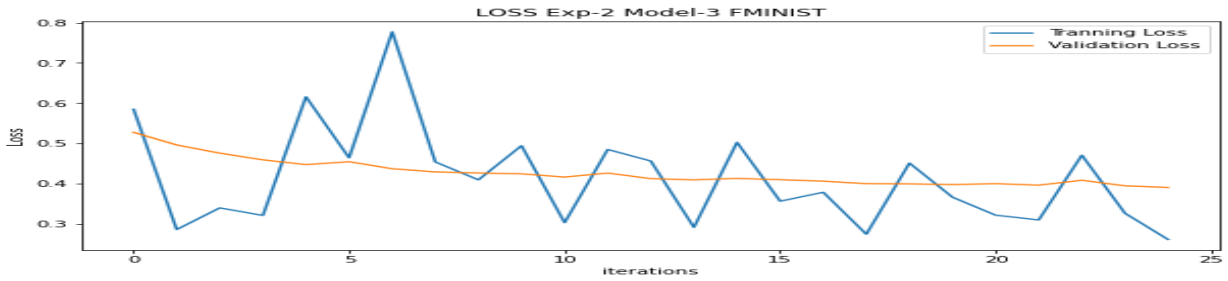


Figure 34:FMINIST loss in mode3

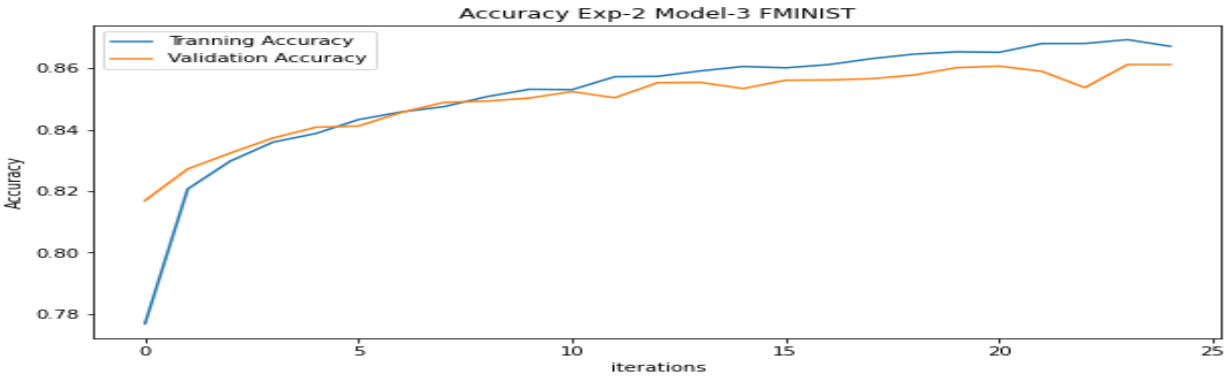
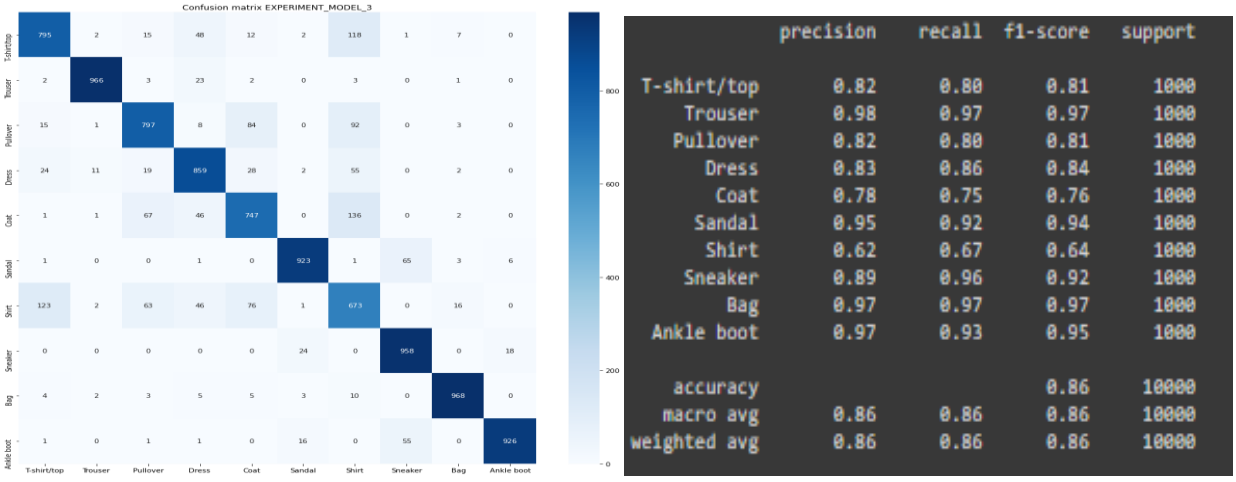


Figure 35:Accuracy Model3



	precision	recall	f1-score	support
T-shirt/top	0.82	0.80	0.81	1000
Trouser	0.98	0.97	0.97	1000
Pullover	0.82	0.80	0.81	1000
Dress	0.83	0.86	0.84	1000
Coat	0.78	0.75	0.76	1000
Sandal	0.95	0.92	0.94	1000
Shirt	0.62	0.67	0.64	1000
Sneaker	0.89	0.96	0.92	1000
Bag	0.97	0.97	0.97	1000
Ankle boot	0.97	0.93	0.95	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Figure 36: Performance Matrices

4. Reference

Source code

https://colab.research.google.com/drive/1mAg2xBR7uTMOr0k6Qjcl9jwMDJJWoyOx?usp=sharing

https://colab.research.google.com/drive/1NI0o0kO7aOmjh4enWf-JDu6XxzZnwdYo?usp=sharing

https://colab.research.google.com/drive/1FFhvZVv8sMc4jYovDvXp7GbN9pLwuJ4j?usp=sharing

https://colab.research.google.com/drive/1pSmlRV21bNGXygJSJgsO4uP7qfid87H5?usp=sharing

Given Dataset Trained Model

https://drive.google.com/file/d/1-7a44W76z-joINkLC_0ffcdBkPkRY_y8/view?usp=sharing

https://drive.google.com/file/d/1-8AKgpmaqLDT5K6CGmfZ-SrgC5vtxqYG/view?usp=sharing

https://drive.google.com/file/d/1-8x2xQQ0NOxpkF1MMe0k2Sz-PjnAS8Lc/view?usp=sharing

https://drive.google.com/file/d/1IxM0bieCLAaWWe_r67sd_Jz-oS8qXVfN/view?usp=sharing

https://drive.google.com/file/d/1LwOS_LP56Pm1xcbjXRzf_EYClpz167PK/view?usp=sharing

https://drive.google.com/file/d/1VLPrGSLzQBbL_G1rTufcs7EmMZ_YJGH-/view?usp=sharing

https://drive.google.com/file/d/1qpsII4q2Xk1mdSrS3KW4K8Pe7ugY8rs3/view?usp=sharing

F-MNIST Trained Model

https://drive.google.com/file/d/1-92tqY9DwsAHgUikt5B_9dMPc7Oq1sWH/view?usp=sharing

https://drive.google.com/file/d/1GQpI5HSHF_cg8mvBsBxc89ohSa5fDqEv/view?usp=sharing

https://drive.google.com/file/d/1K73sdUVBVx29VX8cBABH0AInev920sT0/view?usp=sharing

https://drive.google.com/file/d/1PBTq5xRgadmgc_MocZIHGhCFY80QBAg2/view?usp=sharing

https://drive.google.com/file/d/1VzWoWbjjMeptGSLgqqBxSdwiQ28CFmp9/view?usp=sharing