در این فاز از ما خواسته شده است که تصاویر دیتاست Oxford flowers را با شبکه عمیق طبقه بندی کنیم.

پیاده سازی ساختار شبکه

کلاس CNN شبکه ماست که ساختار آن را طبق خواسته‌ی پروژه پیاده سازی میکنیم:

```
class CNN(nn.Module):
    def __init__(self, num_classes=80):
        super(CNN, self).__init__()
```

لایه convolution اول را با in_channels=3 و out_channels=64، با اندازه کرنل ۳*۳ ، stride=1 و padding=1 میسازیم. همچنین لایه batch norm و ReLU را به صورت زیر برای آن تعریف میکنیم:

```
        # conv1
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu1 = nn.ReLU()
```

این سه لایه، لایه کانولوشن اول را تشکیل میدهند.

Padding=1 باعث میشود ابعاد لایه‌ی جدید توسط kernel با سایز ۳ کاهش پیدا نکند.

لایه های کانولوشن دوم و سوم و چهارم و پنجم باید چهار بار تکرار شوند. بنابراین به عنوان نمونه لایه کانولوشن سوم را به صورت زیر تعریف میکنیم. هر کدام از کانولوشن ها batch norm و ReLU مخصوص به خود را دارند. in_channels برای هر لایه باید برابر با out_channels لایه قبلی آن باشد.

```
        # conv3
        self.conv3_1 = nn.Conv2d(64, 96, kernel_size=3, stride=1, padding=1)
        self.bn3_1 = nn.BatchNorm2d(96)
        self.relu3_1 = nn.ReLU()

        self.conv3_2 = nn.Conv2d(96, 96, kernel_size=3, stride=1, padding=1)
        self.bn3_2 = nn.BatchNorm2d(96)
        self.relu3_2 = nn.ReLU()

        self.conv3_3 = nn.Conv2d(96, 96, kernel_size=3, stride=1, padding=1)
        self.bn3_3 = nn.BatchNorm2d(96)
        self.relu3_3 = nn.ReLU()

        self.conv3_4 = nn.Conv2d(96, 96, kernel_size=3, stride=1, padding=1)
        self.bn3_4 = nn.BatchNorm2d(96)
        self.relu3_4 = nn.ReLU()
```

لایه های pooling را نیز به صورت زیر تعریف میکنیم:

```
        # pool1
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
```

لایه آخر fully connected است، با استفاده از nn.linear() آن را تعریف میکنیم. از آنجایی که ورودی این لایه باید flat باشد پس یک Flatten() برای آن تعریف میکنیم که با استفاده از آن داده های لایه pooling قبل آن را در یک بُعد مسطح کند و سپس به لایه فولی کانکتد بدهد.

```
        # foolly connected
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(256*4*4, num_classes)
```

خروجی این لایه به اندازه تعداد کلاس های داده هاست.

مشابه این نمونه ها بقیه لایه ها را آنگونه که در سورس کد آمده است تعریف میکنیم.

پس از تعریف کردن لایه های مورد نیاز، در تابع forward ساختار شبکه را با استفاده از این لایه ها میسازیم. برای نمونه اینجا لایه های کانولوشن اول و دوم را به صورت زیر تعریف کرده ایم. به این صورت که مطابق خواسته صورت پروژه ترتیب لایه ها را مشخص میکنیم و داده های ورودی(inputs) را وارد شبکه میکنیم. داده ها را از یک لایه عبور میدهیم و سپس نتیجه آن را به لایه بعد وارد میکنیم.

```
    def forward(self, inputs, debug=False):
        # conv 1
        conv1 = self.conv1(inputs)
        bn1 = self.bn1(conv1)
        relu1 = self.relu1(bn1)

        # conv 2
        conv2_1 = self.conv2_1(relu1)
        bn2_1 = self.bn2_1(conv2_1)
        relu2_1 = self.relu2_1(bn2_1)

        conv2_2 = self.conv2_2(relu2_1)
        bn2_2 = self.bn2_2(conv2_2)
        relu2_2 = self.relu2_2(bn2_2)

        conv2_3 = self.conv2_3(relu2_2)
        bn2_3 = self.bn2_3(conv2_3)
        relu2_3 = self.relu2_3(bn2_3)

        conv2_4 = self.conv2_4(relu2_3)
        bn2_4 = self.bn2_4(conv2_4)
        relu2_4 = self.relu2_4(bn2_4)
```

تابع freeze_except_FC_layer در کلاس CNN برای بخش اول فاز دوم پروژه استفاده میشود. به این صورت که ابتدا همه‌ی لایه ها را فریز میکند و سپس لایه فولی کانکتد را unfreeze میکند.

```
def freeze_except_FC_layer(self):
    for param in self.parameters():
        param.requires_grad = False
    for param in self.fc.parameters():
        param.requires_grad = True
```

تابع freeze_FC_except_last_20_neurons برای بخش دوم فاز دوم استفاده میشود. به این صورت که همه‌ی پارامتر های لایه فولی کانکتد را به جز پارامتر های ۲۰ نورون آخر آن فریز میکند. در کد، قبل از این تابع، تابع freeze_except_FC_layer صدا زده میشود تا ابتدا همه لایه ها بجز فولی کانکتد را فریز کند و سپس با صدا زدن این تابع تنها پارامترهای ۲۰ نورون آخر آن unfreezed باقی بمانند.

```
def freeze_FC_except_last_20_neurons(self):
    # Freeze all parameters in the fully connected layer
    for param in self.fc.parameters():
        param.requires_grad = False

    # Unfreeze the parameters of the last 20 neurons
    self.fc.weight.requires_grad = True
    self.fc.bias.requires_grad = True
    self.fc.weight.data[:80, :].requires_grad = False
    self.fc.bias.data[:80].requires_grad = False
```

تابع unfreeze_all_layers همه لایه ها را از حالت فریز خارج میکند.

```
def unfreeze_all_layers(self):
    for param in self.parameters():
        param.requires_grad = True
```

با استفاده از تابع train_one_epoch مدل را با دیتاست مورد نظر خود آموزش میدهیم.

Train model

```python
def train_one_epoch(model: nn.Module, optim: torch.optim.Optimizer,
                    dataloader: DataLoader, loss_fn):

    num_samples = len(dataloader.dataset)
    num_batches = len(dataloader)
    running_corrects = 0
    running_loss = 0.0

    model.train()

    for batch_indx, (inputs, targets) in enumerate(dataloader): # Get a batch of Data
        inputs = inputs.to(device)
        targets = targets.to(device)

        outputs = model(inputs) # Forward Pass
        loss = loss_fn(outputs, targets) # Compute Loss

        loss.backward() # Compute Gradients
        optim.step() # Update parameters
        optim.zero_grad() # zero the parameter's gradients

        _, preds = torch.max(outputs, dim=1)
        running_corrects += torch.sum(preds == targets).cpu()
        running_loss += loss.item()

    epoch_acc = (running_corrects / num_samples) * 100
    print("train: ", running_corrects, num_samples)
    epoch_loss = (running_loss / num_batches)

    return epoch_acc, epoch_loss
```

به عنوان خروجی epoch_acc, epoch_loss را ریترن میکنیم تا در توابع evaluate که جلوتر بررسی میکنیم بتوانیم اکیوریسی و لاس را برای هر ایپاک بدست آوریم.

با استفاده از تابع test_model اکیوریسی و لاس شبکه را برای دیتاست مورد نظر بررسی میکنیم.

Test model

```python
def test_model(model: nn.Module,
               dataloader: DataLoader, loss_fn):

    num_samples = len(dataloader.dataset)
    num_batches = len(dataloader)
    running_corrects = 0
    running_loss = 0.0

    true_labels = torch.empty(0).to(device)
    pred_labels = torch.empty(0).to(device)

    # we call `model.eval()` to set dropout and batch normalization layers to evaluation mode before running inference.
    model.eval()

    with torch.no_grad():
        for batch_indx, (inputs, targets) in enumerate(dataloader): # Get a batch of Data
            inputs = inputs.to(device)
            targets = targets.to(device)

            outputs = model(inputs) # Forward Pass
            loss = loss_fn(outputs, targets) # Compute Loss

            _, preds = torch.max(outputs, 1) #
            running_corrects += torch.sum(preds == targets).cpu()
            running_loss += loss.item()

            true_labels = torch.cat([true_labels, targets], dim=0)
            pred_labels = torch.cat([pred_labels, preds], dim=0)

    test_acc = (running_corrects / num_samples) * 100
    print("test: ", running_corrects, num_samples)
    test_loss = (running_loss / num_batches)

    return test_acc, test_loss, true_labels, pred_labels
```

در این تابع علاوه بر epoch_acc, epoch_loss، مقادیر true_labels, pred_labels را نیز ریترن میکنیم تا در توابع evaluate بتوانیم از آنها برای بدست آوردن ماتریس گمراهی در هر ایپاک استفاده کنیم.

با استفاده از تابع evaluate_A شبکه را روی دیتاهای درون A_train_dl آموزش میدهیم و روی دیتاهای درون A_test_dl تست میکنیم. از بهینه ساز Adam در این فاز استفاده میکنیم. همچنین مقادیر پارامترهای num_epochs, learning_rate را با توجه به تست هایی که در ادامه به نتایج آنها میپردازیم مشخص میکنیم. در نهایت شبکه آموزش دیده را به صورت یک فایل pth. ذخیره میکنیم.

از تابع evaluate_B_1 برای بخش اول فاز دو پروژه استفاده میکنیم. به این صورت که با استفاده از این قسمت کد، شبکه آموزش دیده فاز ۱ را در model جدید که لایه آخر آن ۱۰۰ نورون دارد کپی میکنیم:

```
# loading phase1 model
model1 = CNN(80)
model2 = CNN(100)
model1 = model1.to(device)
model2 = model2.to(device)
model_path = './CNN_model_ph1.pth'
model1.load_state_dict(torch.load(model_path))

# Copy parameters from model1 to model2 (excluding the fully connected layer)
for layer1, layer2 in zip(model1.children(), model2.children()):
    if isinstance(layer1, nn.Linear):
        continue  # Skip copying parameters for fully connected layer

    # Copy parameters from layer1 to layer2
    for param1, param2 in zip(layer1.parameters(), layer2.parameters()):
        param2.data.copy_(param1.data)

model2.fc.weight.data[:80, :] = model1.fc.weight.data
model2.fc.bias.data[:80] = model1.fc.bias.data
```

در این فاز از بهینه ساز SGD استفاده میکنیم.

با استفاده از سلول check network copy بررسی میکنیم که پارامترهای شبکه فاز ۱ به درستی به شبکه جدید منتقل شوند.

از تابع evaluate_B_2 برای بخش دوم فاز دو استفاده میکنیم. روند کار آن مشابه evaluate_B_1 است با این تفاوت که اینبار همه لایه های شبکه بجز فولی کانکتد فریز خواهند شد:

```
# freeze all the layers exept fully connected layer
model2.freeze_except_FC_layer()
```

از تابع evaluate_B_3 برای بخش سوم فاز سه استفاده میکنیم. روند کار آن مشابه evaluate_B_1 است با این تفاوت که اینبار همه پارامترهای همه لایه های شبکه بجز پارامتر های ۲۰ نورون آخر فولی کانکتد فریز خواهند شد:

```
# freeze all the layers exept last 20 neurons of fully connected layer
model2.freeze_except_FC_layer()
model2.freeze_FC_except_last_20_neurons()
```

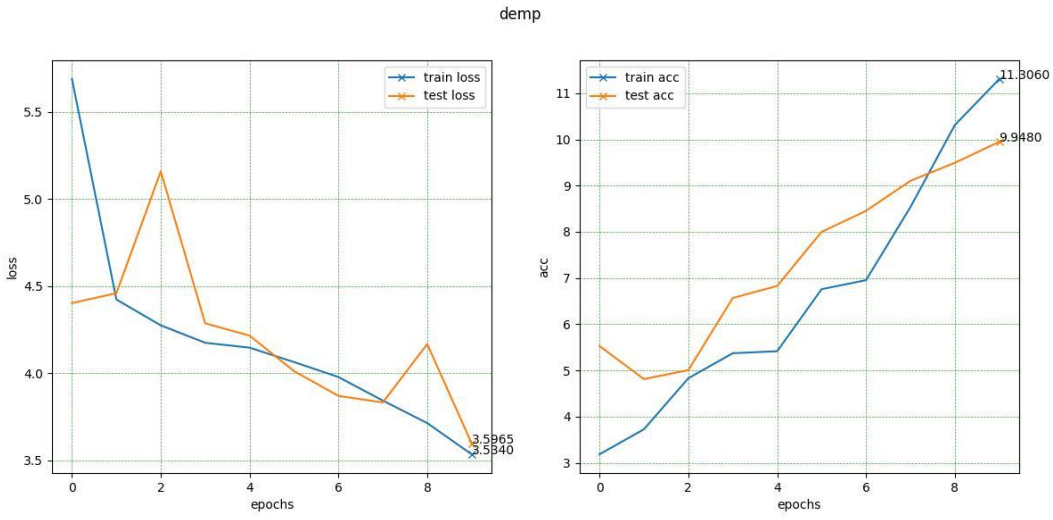
بررسی منحنی یادگیری و دقت برای دیتاست A – فاز اول

برای بدست آوردن مقادیر بهینه برای پارامتر های یادگیری شبکه حالات مختلف را تست کرده ایم. بر اساس نتایج این آزمایش ها بهترین مقادیر را ارزیابی میکنیم.

در ابتدا با مقدار بچ سایز ۱۲۸ (مقدار دیفالت) و تغییر مقادیر لرنینگ ریت و تعداد ایپاک ها نتایج آموزش و تست شبکه را بررسی کردیم:

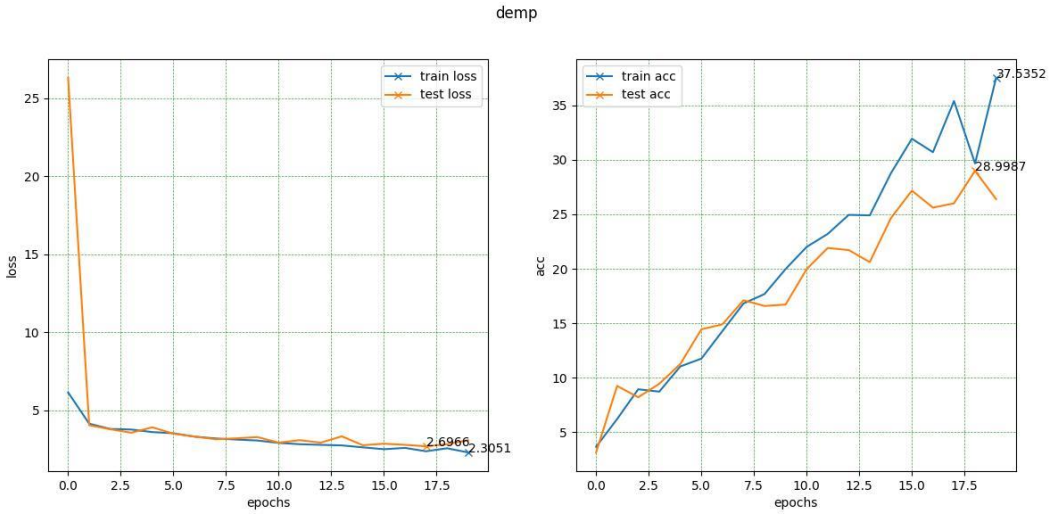
ابتدا با مقادیر کوچک num epochs:

Batch size=128, learning rate=0.005, num epochs=10

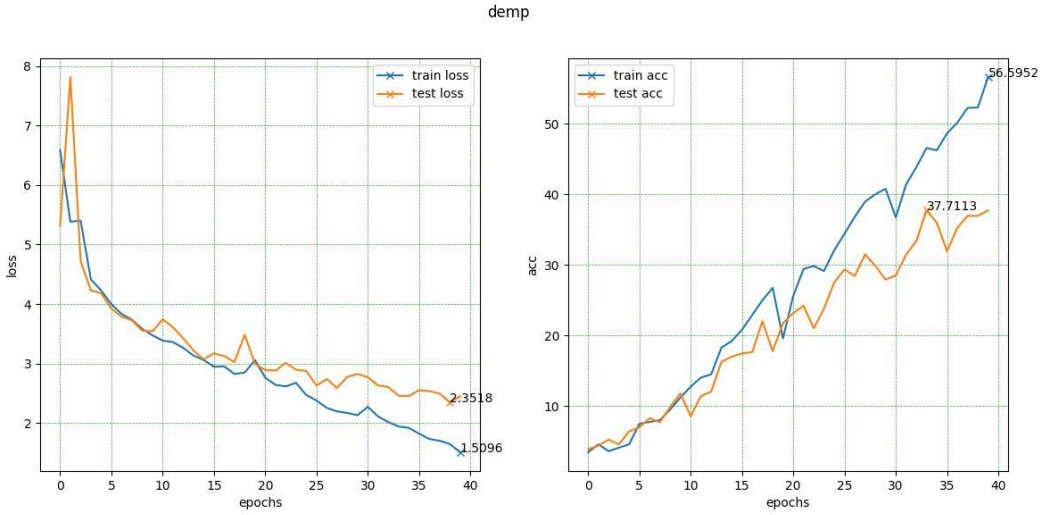


دقت train بسیار کم است، پس تعداد ایپاک ها را افزایش دادیم تا شبکه برای بارهای بیشتری داده ها را مشاهده کند:

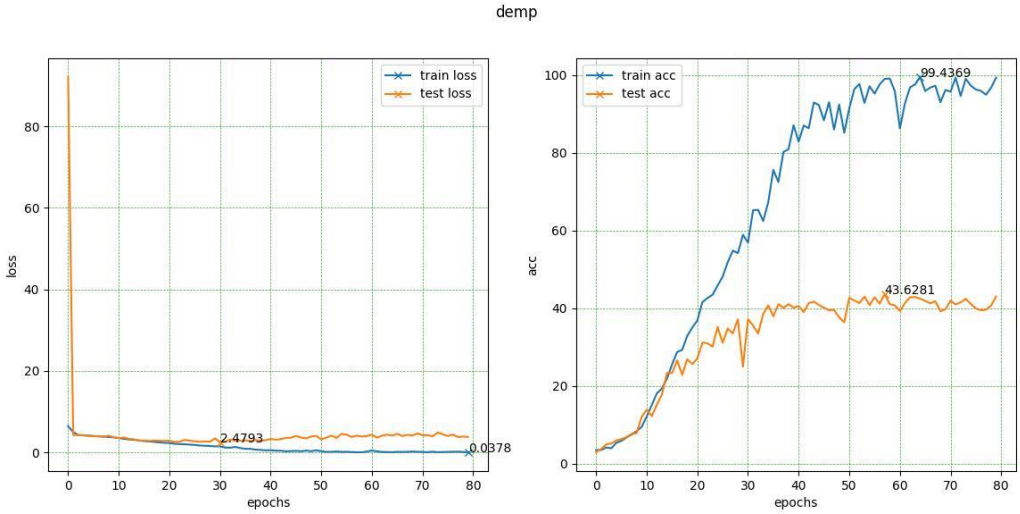
Batch size=128, learning rate=0.005, num epochs=20



Batch size=128, learning rate=0.005, num epochs=40



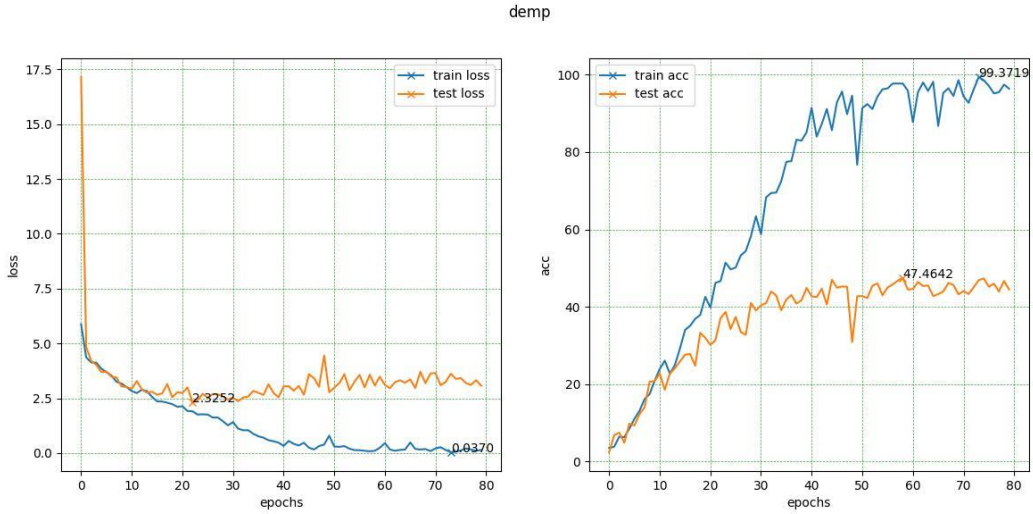
Batch size=128, learning rate=0.005, num epochs=80



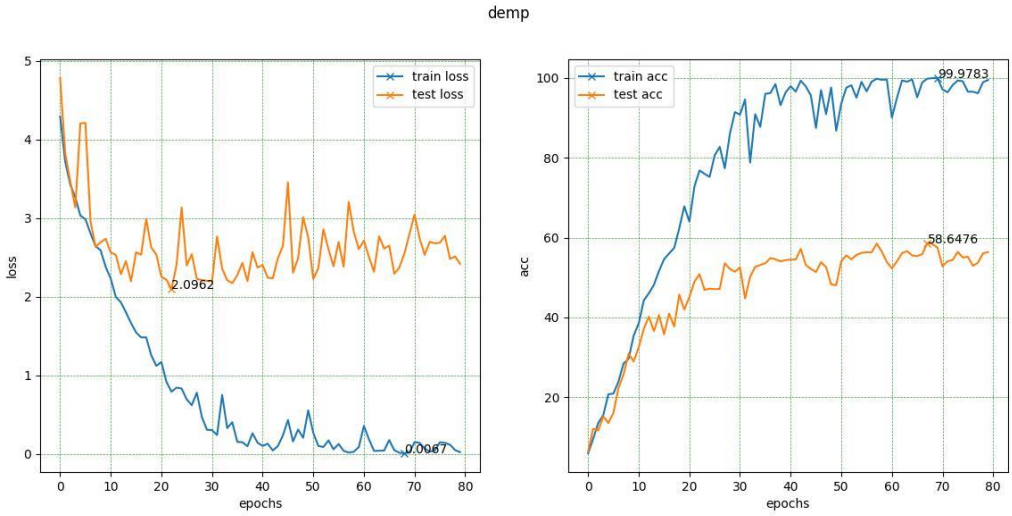
حالا دقت train به نزدیک ۱۰۰ رسید.

مقادیر learning rate را تغییر دادیم تا شاید باعث افزایش دقت تست شود:

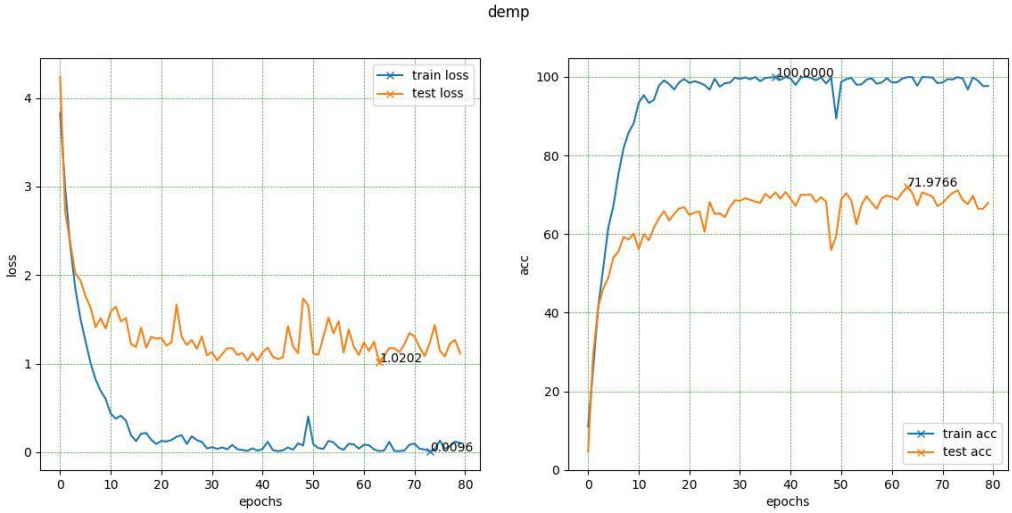
Batch size=128, learning rate=0.003, num epochs=80



Batch size=128, learning rate=0.001, num epochs=80



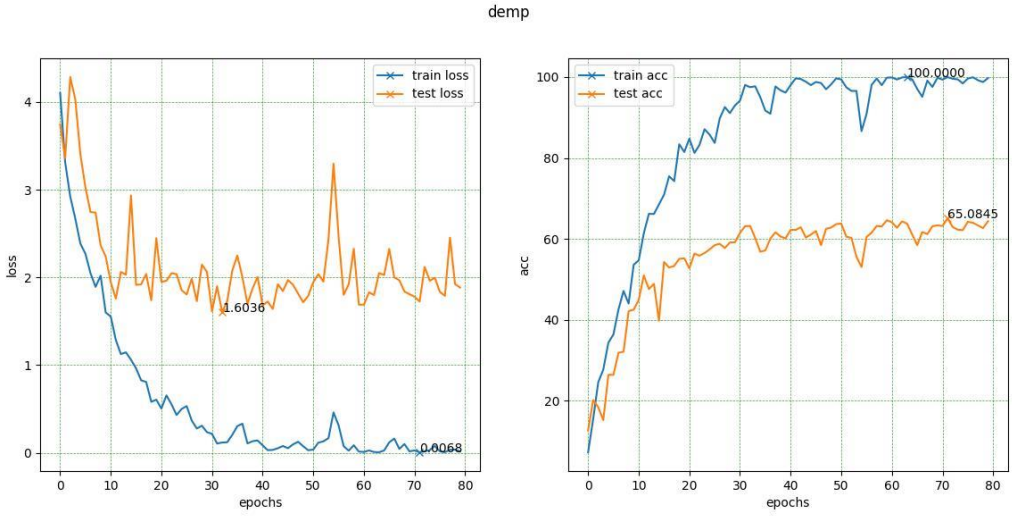
Batch size=128, learning rate=0.0001, num epochs=80



همانطور که مشاهده میشود به ازای این مقدار لرنینگ ریت در کنار مقادیر سایر پارامتر ها بهترین اکیوریسی را تا اینجای بررسی داشته. **نتیجه میشود که کاهش مقدار لرنینگ ریت به دقت شبکه کمک میکند.**

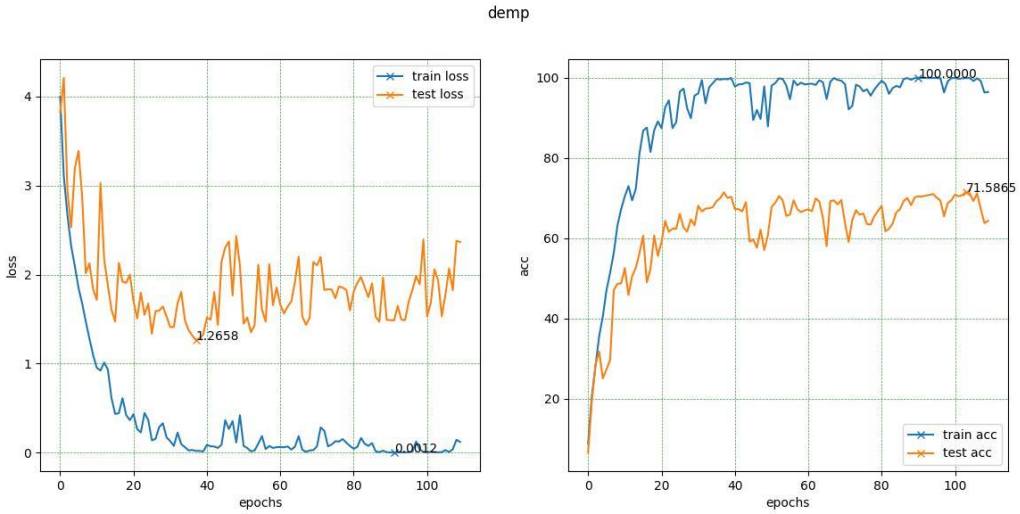
دو تغییر مقدار دیگر لرنینگ ریت را نیز بررسی میکنیم:

Batch size=128, learning rate=0.0005, num epochs=80



تعداد ایپاک هارا به ۱۱۰ افزایش میدهیم تا تاثیر آن را ببینیم:

Batch size=128, learning rate=0.0005, num epochs=110

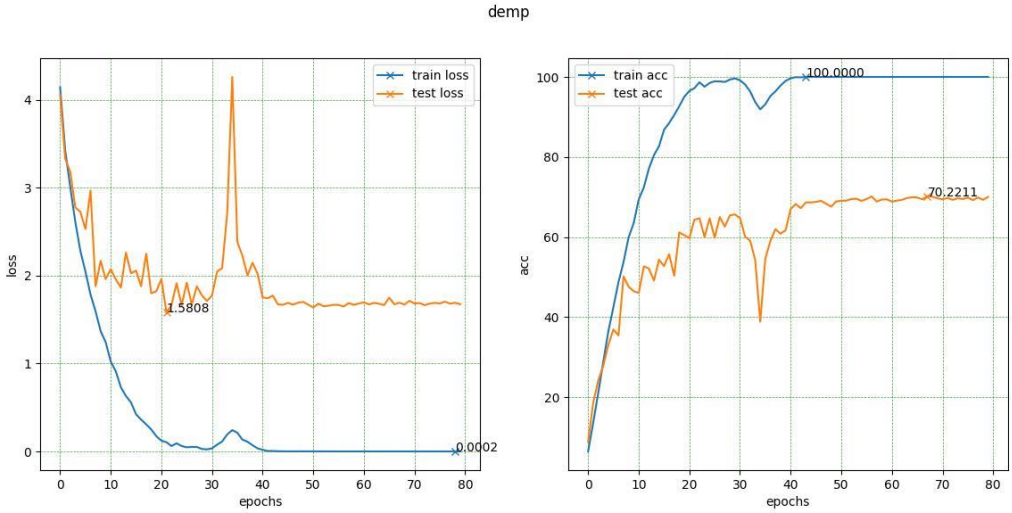


میبینیم که دقت تست نسبت به حالت قبل افزایش داشته، اما باید این را در نظر بگیریم که ما این افزایش دقت را با افزایش ۳۰ ایپاک بدست آورده ایم که باعث افزایش زمان پردازش میشود. پس این افزایش دقت مطلوب ما نیست. این در حالیست که با مقادیر Batch size=128, learning rate=0.0001, num epochs=80 یعنی با ۳۰ ایپاک کمتر توانسته بودیم به همین دقت ۷۱٪ برسیم.

تا اینجای کار توانستیم دقت ۷۱ درصدی برای داده های تست بدست بیاوریم، ولی این نتیجه با سربار زمانی بسیار بالای ۸۰ ایپاک بدست آمده است. پس این نتیجه بدست آمده مطلوب ما نیست و باید سعی کنیم با تغییر سایر پارامترها دقت مناسبی را با تعداد ایپاک کمتر بدست بیاوریم.

برای این هدف، یکبار مقدار بچ سایز را کاهش میدهیم تا نتیجه دقت شبکه را بررسی کنیم:

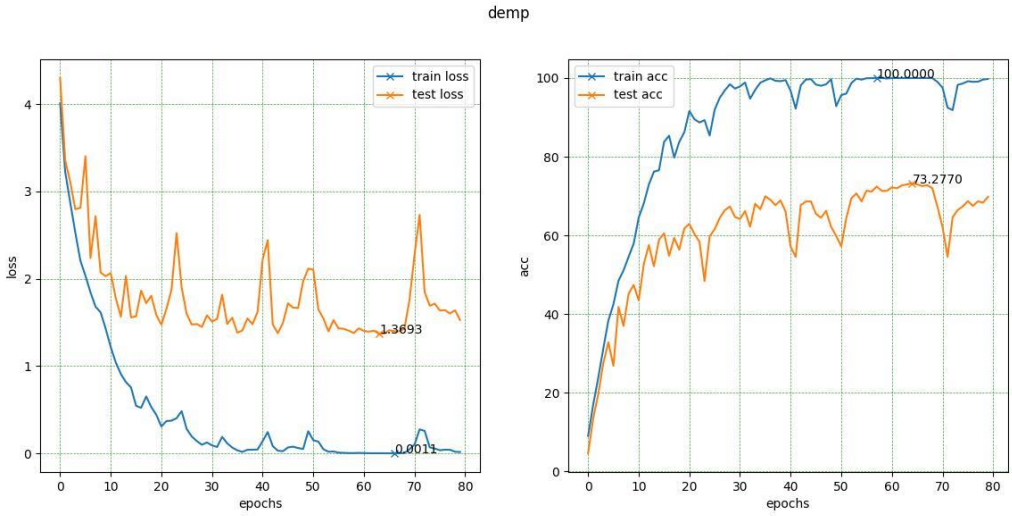
Batch size=75, learning rate=0.0005, num epochs=80



مشاهده میشود که مقدار اکیوریسی حدود ۷۰٪ بدست میاید، اما باز هم در تعداد ایپاک بالا این نتیجه میشود، پس این حالت یعنی کمتر کردن مقدار بچ سایز نیز مطلوب ما نیست.

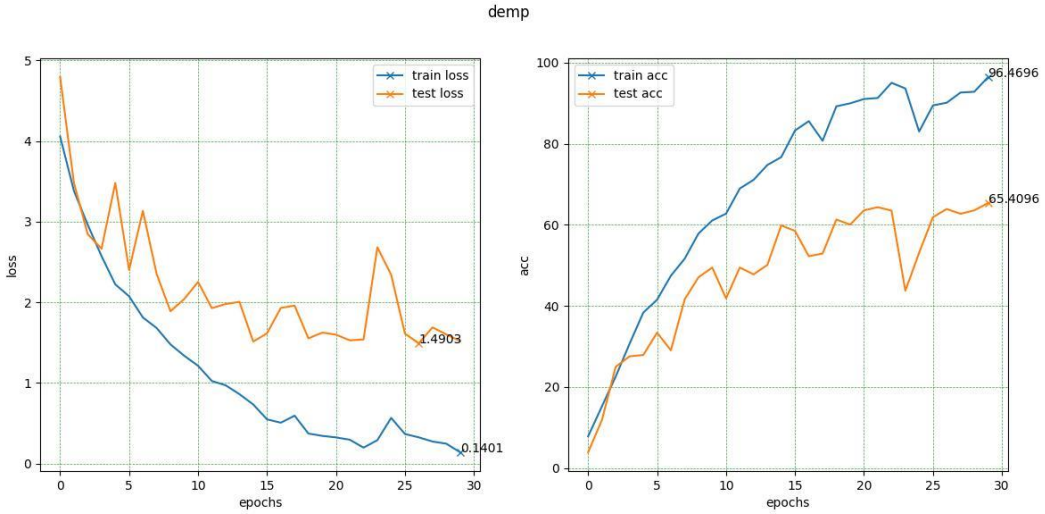
این بار دقت شبکه را با افزایش مقدار بچ سایز به ۲۰۰ بررسی میکنیم:

Batch size=200, learning rate=0.0005, num epochs=80



مشاهده میشود که اینبار دقت تست شبکه در تعداد ایپاک های کمتر حدود ۷۰٪ است. پس دوباره با همین مقادیر لرنینگ ریت و بچ سایز، تعداد ایپاک هارا به ۳۰ عدد کم میکنیم و نتیجه را بررسی میکنیم:

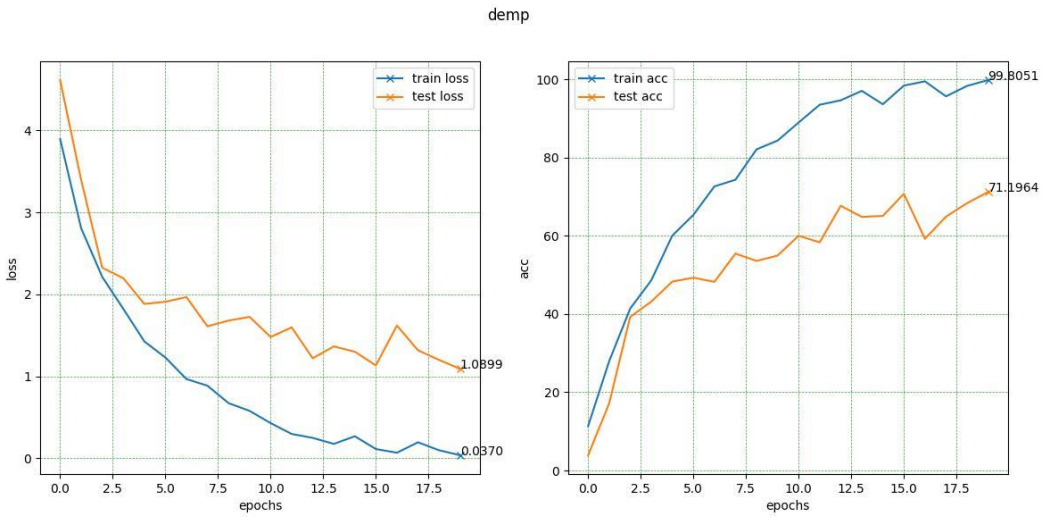
Batch size=200, learning rate=0.0005, num epochs=30



مشاهده میشود که با اینکه تعداد ایپاک هارا خیلی کاهش دادیم، اما به دلیل افزایش بچ سایز دقت تست شبکه آنچنان کم نمیشود. **پس نتیجه میشود افزایش مقدار بچ سایز به دقت شبکه کمک کرده است.**

یکبار هم با مقدار ثابت ۲۰۰ بچ سایز، لرنینگ ریت را کاهش میدهیم تا تاثیر آن را بررسی کنیم:

Batch size=200, learning rate=0.0003, num epochs=20

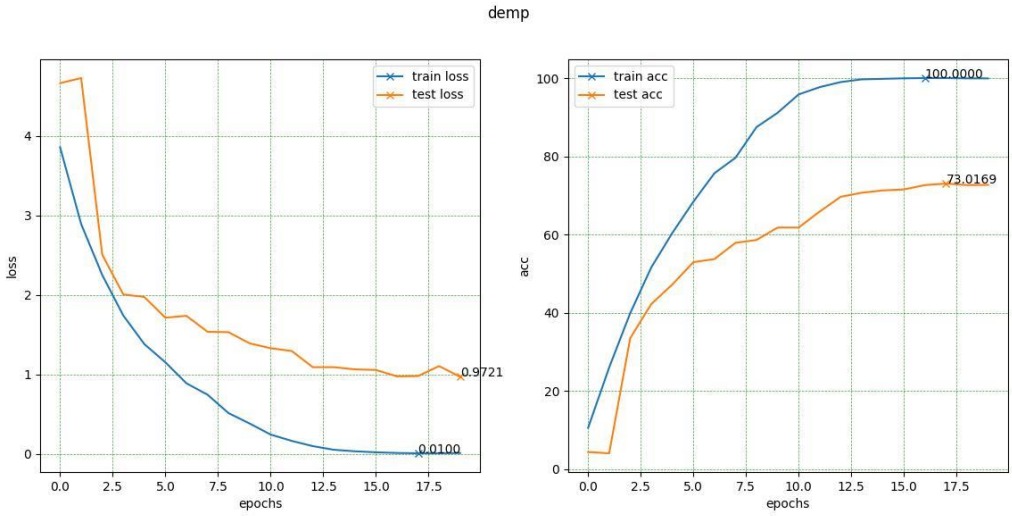


این بررسی دوباره تایید میکند که کاهش مقدار لرنینگ ریت باعث بهبود عملکرد شبکه میشود.

با توجه به دو نتیجه ای که تا الان حاصل شده است، یعنی اینکه افزایش بچ سایز و کاهش لرنینگ ریت به بهبود عملکرد شبکه کمک کرده است، دوباره چند حالت جدید را برای رسیدن به مقادیر بهینه برای پارامتر ها بررسی میکنیم.

نسبت به حالت قبلی، تنها مقدار بچ سایز را به ۲۵۰ افزایش میدهیم تا تاثیر آن را بررسی کنیم:

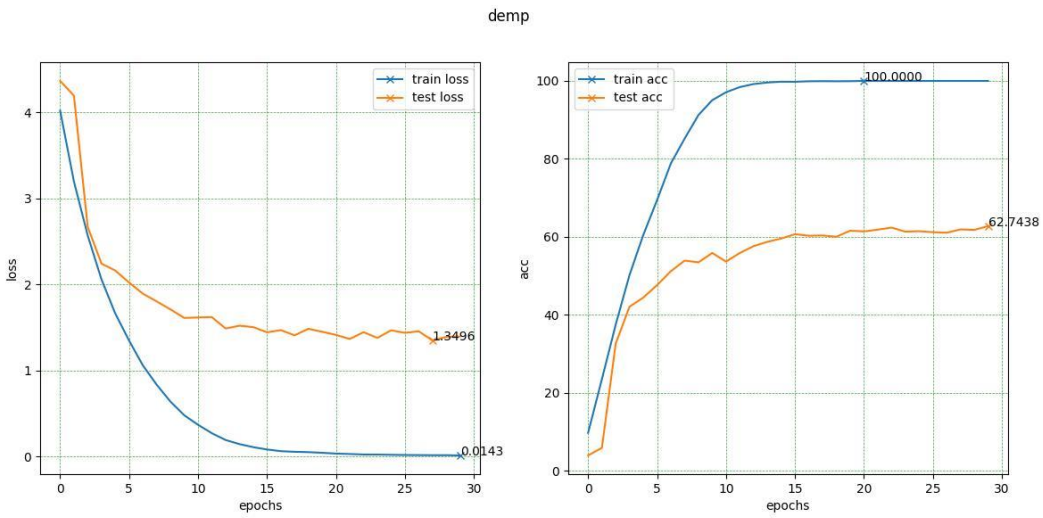
Batch size=250, learning rate=0.0003, num epochs=20



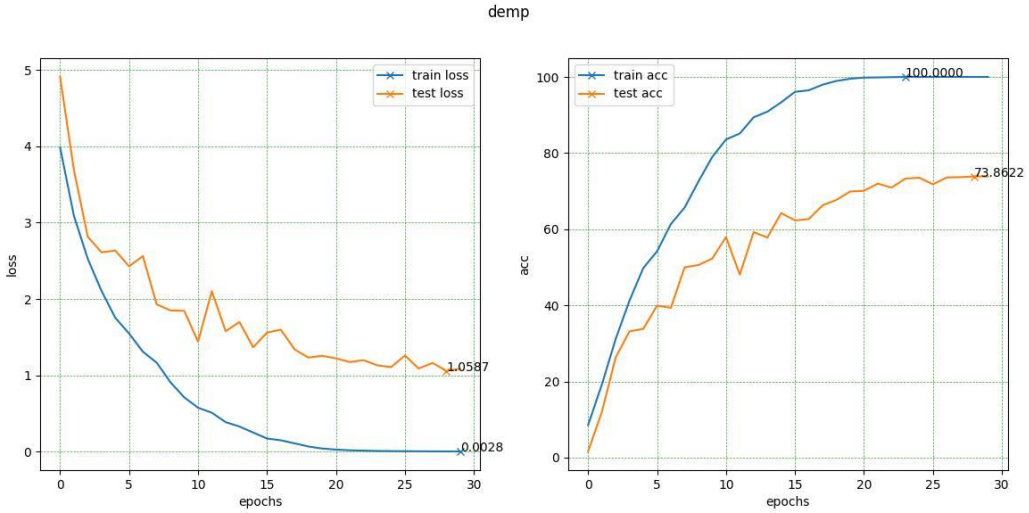
دوباره این افزایش بچ سایز باعث بهبود عملکرد شبکه شده است. **پس تا الان نتیجه میشود که مقدار ۲۵۰ برای بچ سایز بهترین مقدار است.**

برای اینکه بهترین مقدار لرنینگ ریت را پیدا کنیم، به ازای برابر بودن مقادیر سایر پارامتر ها، یکبار آنرا به ۰.۰۰۰۱ کاهش و یکبار به ۰.۰۰۰۵ افزایش میدهیم تا تاثیر آن را بررسی کنیم:

Batch size=250, learning rate=0.0001, num epochs=30



Batch size=250, learning rate=0.0005, num epochs=30

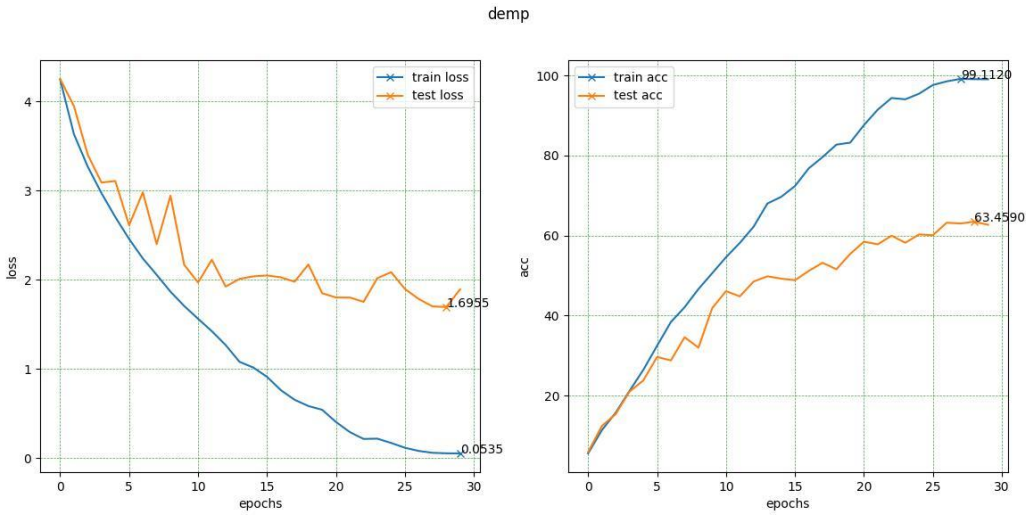


این بررسی نشان میدهد به ازای مقادیر برابر سایر پارامتر ها، کاهش لرنینگ ریت از ۰.۰۰۰۳ به ۰.۰۰۰۱ باعث بدتر شدن عملکرد شبکه (کاهش دقت تست از ۷۳٪ به ۶۲.۷٪) و افزایش آن از ۰.۰۰۰۳ به ۰.۰۰۰۵ باعث بهتر شدن عملکرد شبکه (افزایش دقت تست از ۷۳٪ به ۷۳.۸٪) شده است.

پس نتیجه میشود بهترین مقدار برای لرنینگ ریت ۰.۰۰۰۵ است.

برای اطمینان از اینکه این مقدار بهینه است، یکبار هم مقدار آن را برابر ۰.۰۰۰۷ قرار میدهیم و بررسی میکنیم:

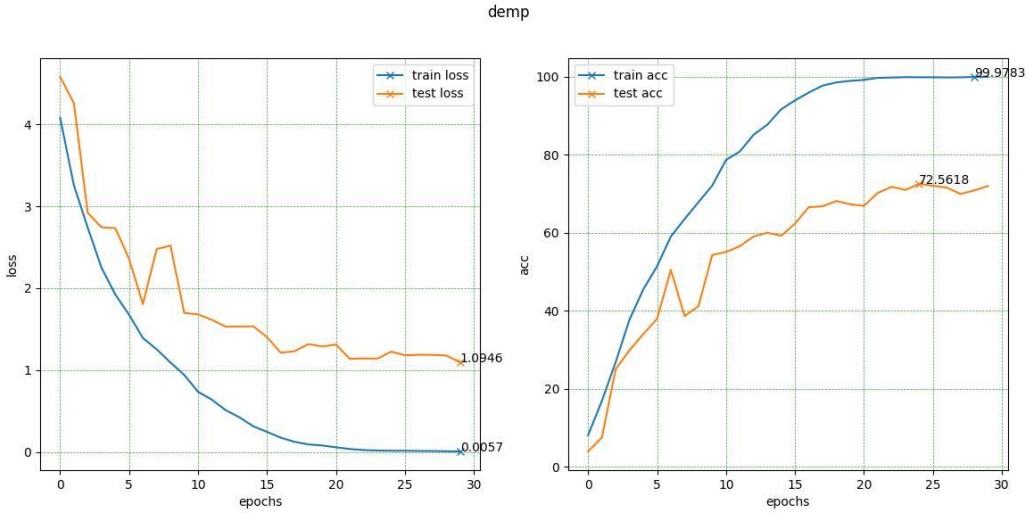
Batch size=250, learning rate=0.0007, num epochs=30



پس اطمینان حاصل شد که مقدار گفته شده برای لرنینگ ریت بهترین مقدار است.

یکبار هم مقدار بچ سایز را به ازای این مقدار لرنینگ ریت افزایش میدهیم تا تاثیر آن را بررسی کنیم:

Batch size=300, learning rate=0.0005, num epochs=30



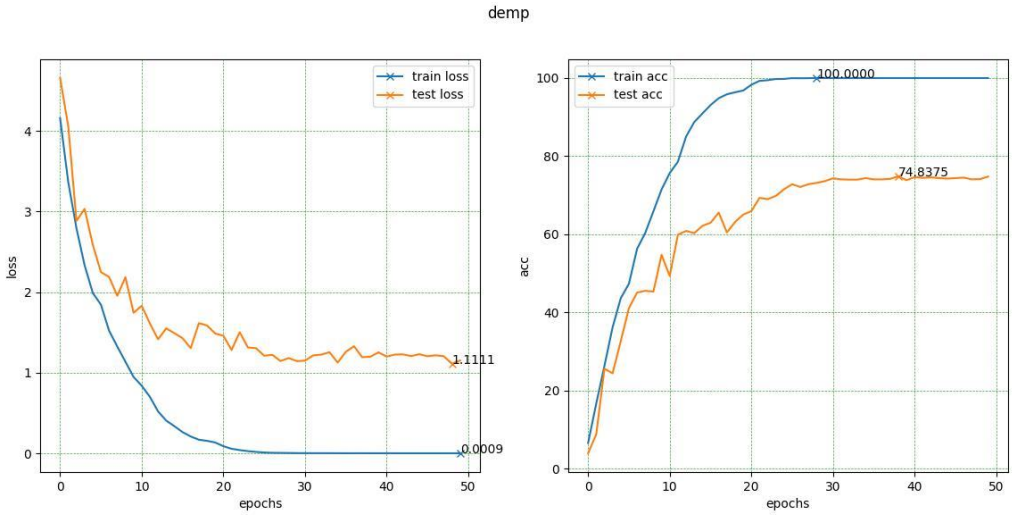
مشاهده میشود که دقت تست در این حالت نسبت به حالتی که مقادیر پارامتر ها برابر Batch size=250, learning rate=0.0005, num epochs=30 است حدود ۱٪ کاهش میابد.

پس اطمینان حاصل میشود که بهترین مقدار برای بچ سایز ۲۵۰ است.

در نهایت بهترین مقادیر را برای learning rate=0.0005 و batch size=250 بدست آوردیم.

حال برای اینکه ببینیم با این مقادیر آیا افزایش تعداد ایپاک تاثیری در بهبود عملکرد شبکه دارد یا نه، حالت زیر را بررسی میکنیم:

Batch size=250, learning rate=0.0005, num epochs=50



مشاهده میشود که این افزایش ۲۰ عددی تعداد ایپاک ها باعث افزایش ۱ درصدی دقت تست شبکه به ازای مقادیر برابر لرنینگ ریت و بچ سایز میشود، اما این ۱ درصد افزایش دقت در برابر ۲۰ عدد ایپاک و زمان پردازشی که اضافه شده است برای ما مطلوب نیست.

پس از تمام بررسی ها نتیجه میشود بهترین مقادیر برای پارامتر های شبکه اینگونه است:

Batch size=250, learning rate=0.0005, num epochs=30

بررسی منحنی یادگیری و دقت برای دیتاست B – فاز دوم

برای بدست آوردن مقادیر بهینه برای پارامتر های یادگیری شبکه حالات مختلف را تست کرده ایم. بر اساس نتایج این آزمایش ها بهترین مقادیر را ارزیابی میکنیم.

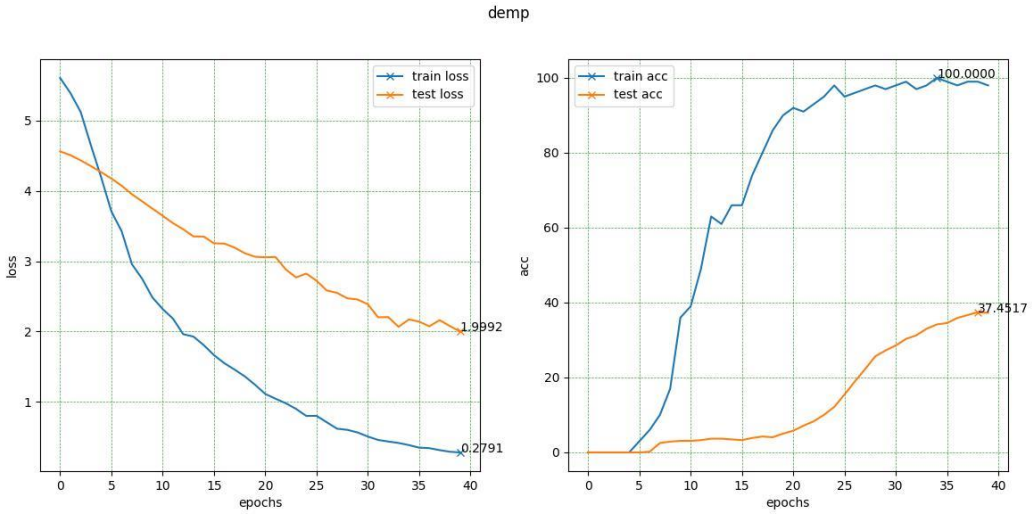
بخش اول

در روش اول هیچ محدودیتی بر وزن های شبکه نداریم و بعد از کپی کردن پارامتر های شبکه فاز اول در شبکه جدید، آن را به طور عادی با داده های آموزشی دیتاست B آموزش می دهیم.

بررسی پارامترها را از مقادیر لرنینگ ریت و بچ سایز بدست آمده در فاز قبل شروع میکنیم. همچنین ابتدا از بهینه ساز SGD استفاده میکنیم.

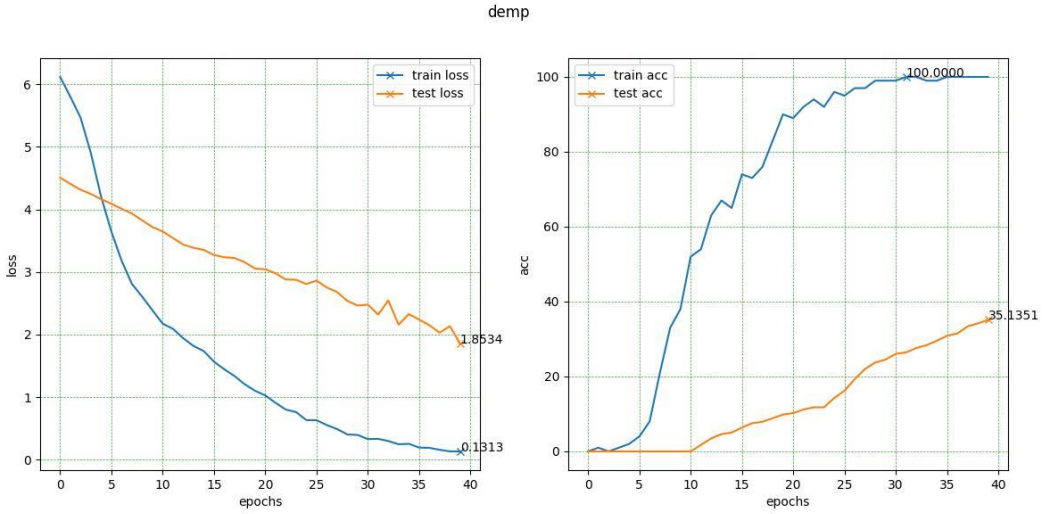
در ابتدا این بررسی را برای B_test_dl انجام میدهیم.

Batch size=250, learning rate=0.0005, num epochs=40, momentum=0.9, opt=SGD

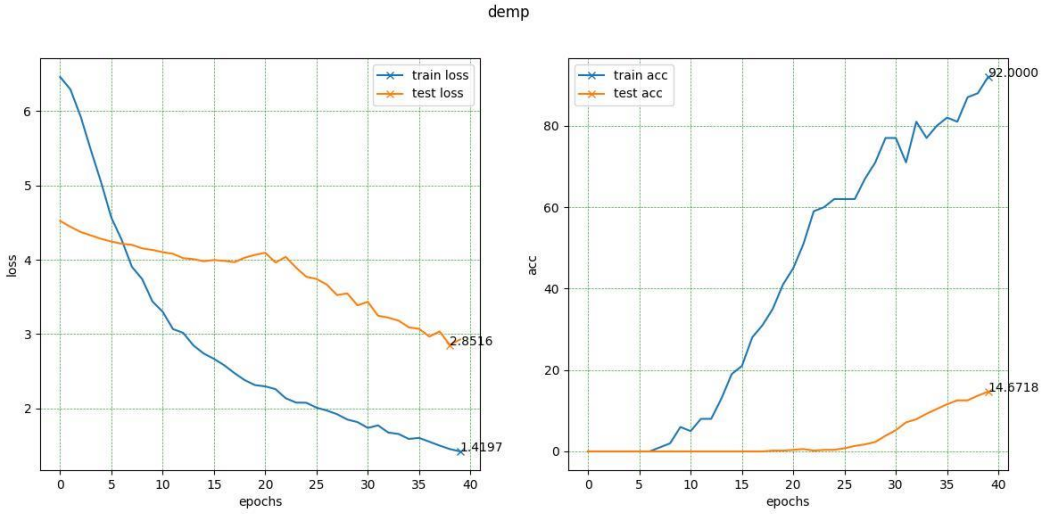


مقدار momentum را تغییر میدهیم و اثر آن را بررسی میکنیم:

Batch size=250, learning rate=0.0005, num epochs=40, momentum=0.95, opt=SGD



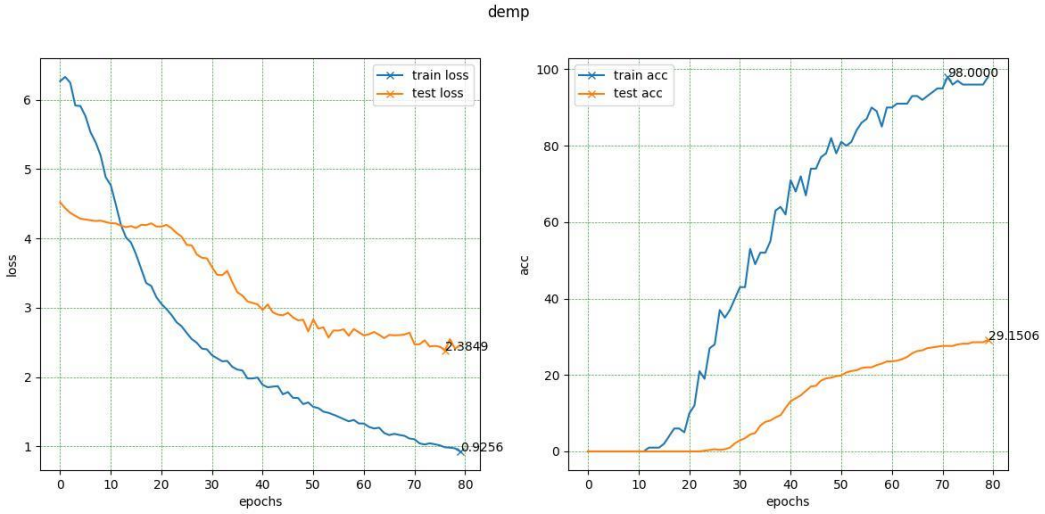
Batch size=250, learning rate=0.0005, num epochs=40, momentum=0.6, opt=SGD



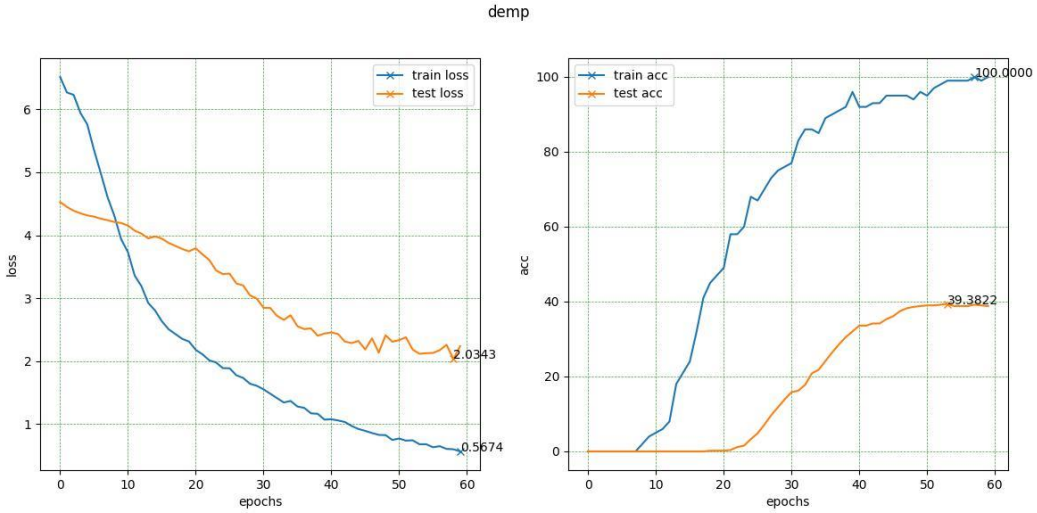
مشاهده میشود که مقدار ۰.۹ برای momentum مناسب تر است و منتج به دقت تست بیشتری میشود.

لرنینگ ریت را تغییر میدهیم تا تاثیر آن را ببینیم:

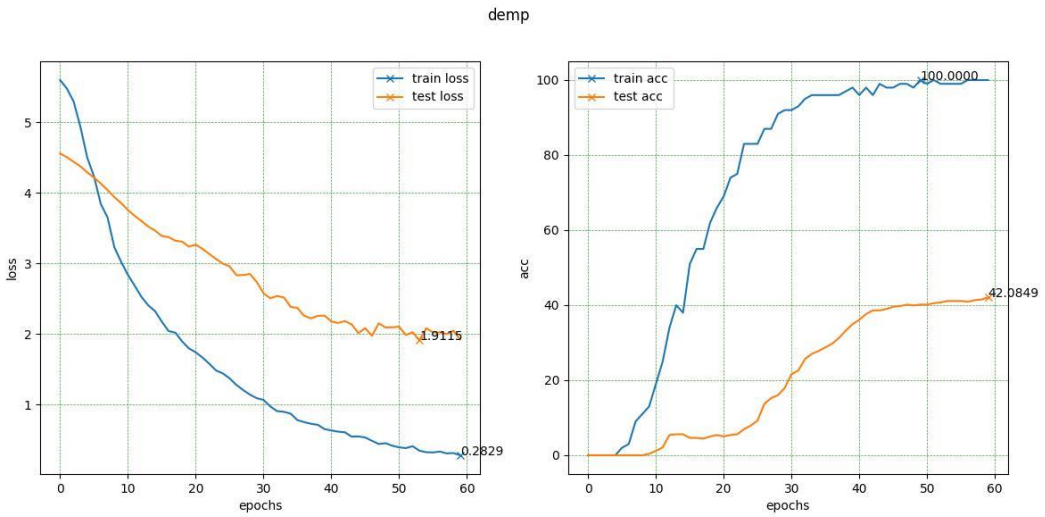
Batch size=250, learning rate=0.0001, num epochs=80, momentum=0.9, opt=SGD



Batch size=250, learning rate=0.0002, num epochs=60, momentum=0.9, opt=SGD



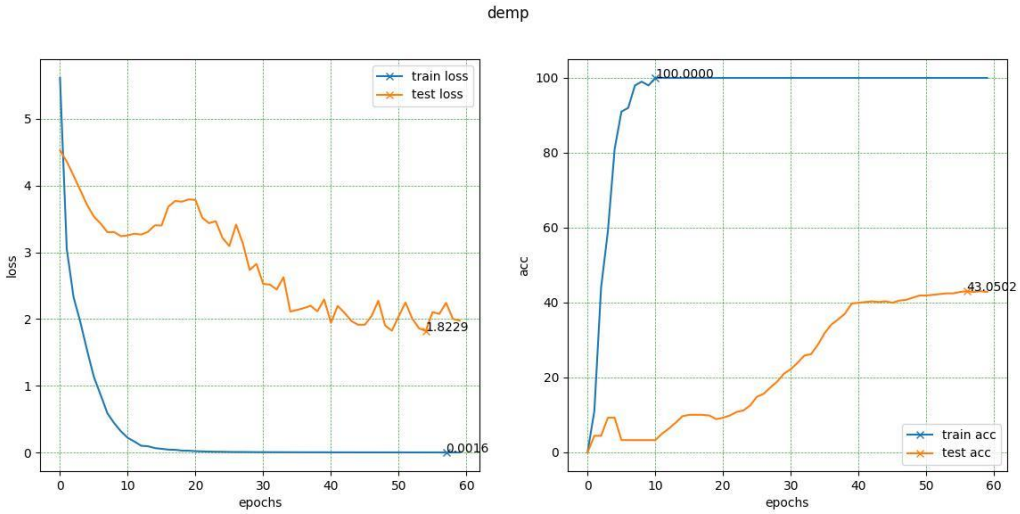
Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD



مشاهده میشود که لرنینگ ریت ۰.۰۰۰۳ بهترین دقت تست را بدست میدهد.

یکبار هم با بهینه ساز Adam این نتیجه را مقایسه میکنیم:

Batch size=250, learning rate=0.0003, num epochs=60, opt=Adam



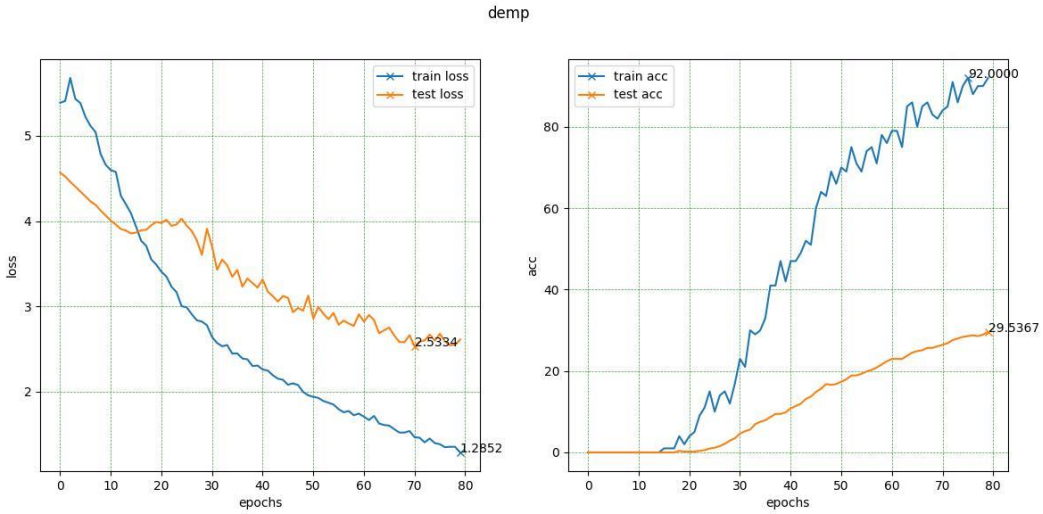
در این دو حالت آخر، زمانی که از Adam استفاده کردیم به اندازه ۱٪ دقت بیشتر شده است، اما اگر به منحنی trian این بیهنه ساز نگاه کنیم و آن را با قبلی مقایسه کنیم، متوجه میشویم که دومی خیلی زودتر به صد رسیده و از آن به بعد تعداد ایپاک زیادی را در ۱۰۰٪ مانده است، پس احتمال اورفیت شدن در این حالت بیشتر است.

در نتیجه حالت قبلی با این پارامتر ها مطلوب ماست:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD

برای اطمینان از بهینه بودن پارامتر ها، مقدار لرنینگ ریت را کاهش میدهیم و اثر آن را بررسی میکنیم:

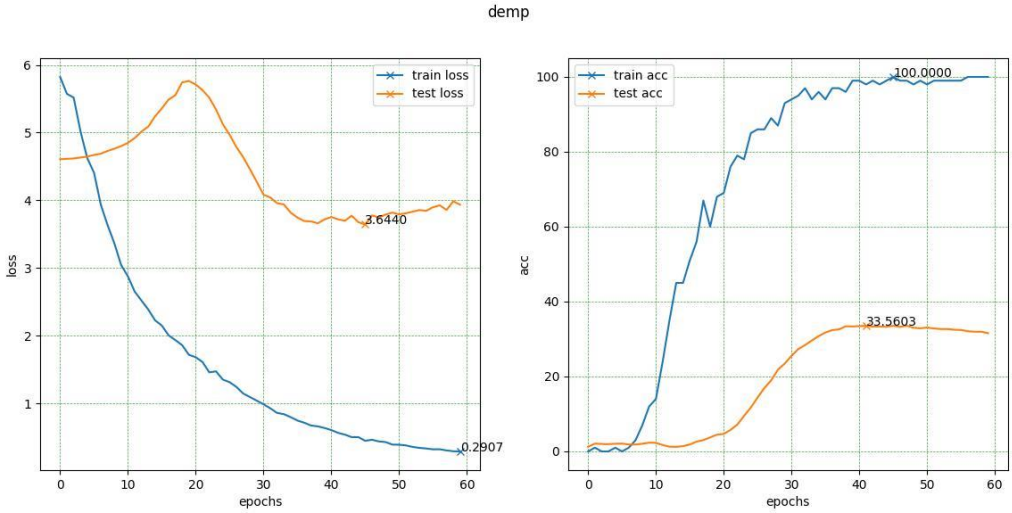
Batch size=250, learning rate=0.00007, num epochs=60, momentum=0.9, opt=SGD



در نتیجه همان مقادیر مشخص شده بهترین مقادیر هستند.

اکنون با استفاده از این مقادیر، دقت داده های test_all را بررسی میکنیم:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD

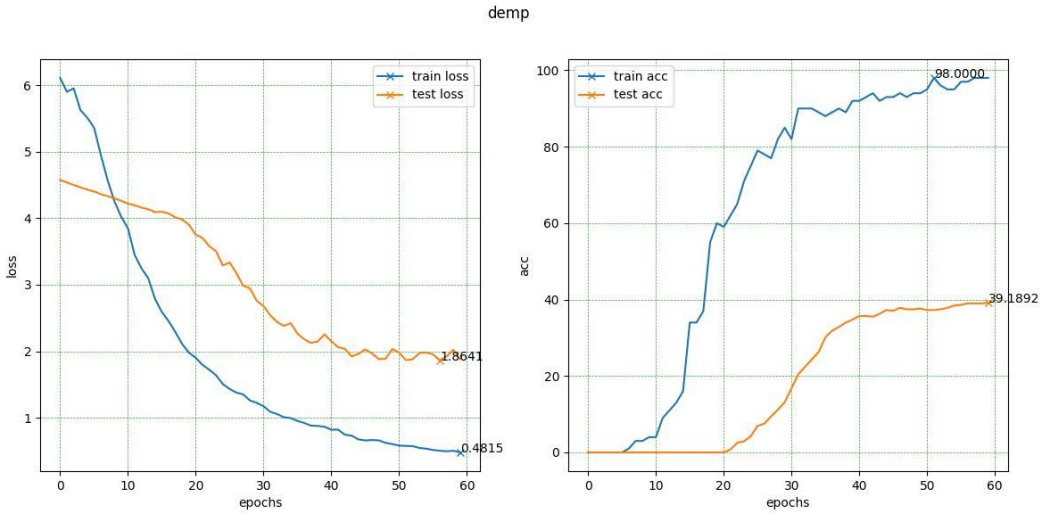


بخش دوم

در روش دوم حین آموزش شبکه پارامتر های لایه پیچشی شبکه جدید را آپدیت نمی کنیم. در واقع تنها پارامتر های لایه آخر در آموزش آپدیت می شوند.

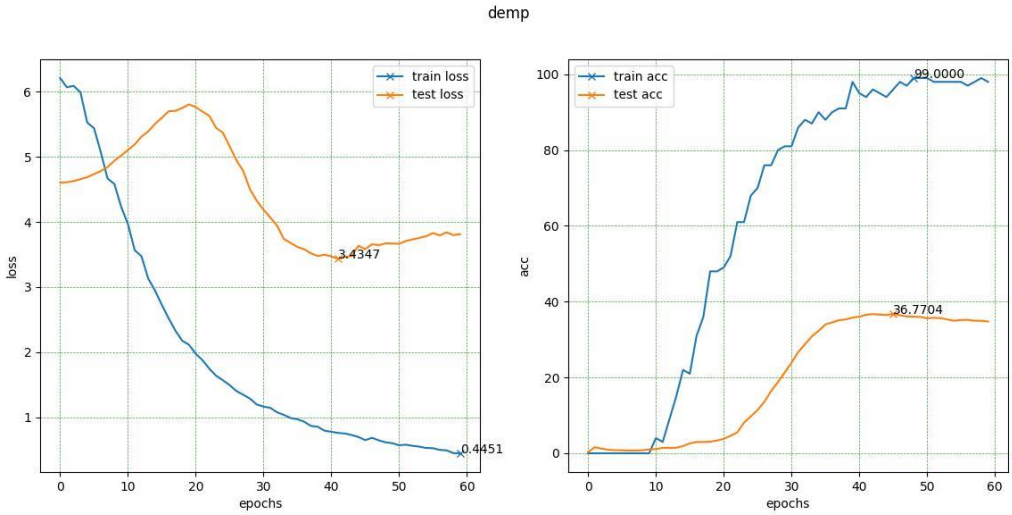
دقت تست B_test_dl در این حالت را بدست میاوریم:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD



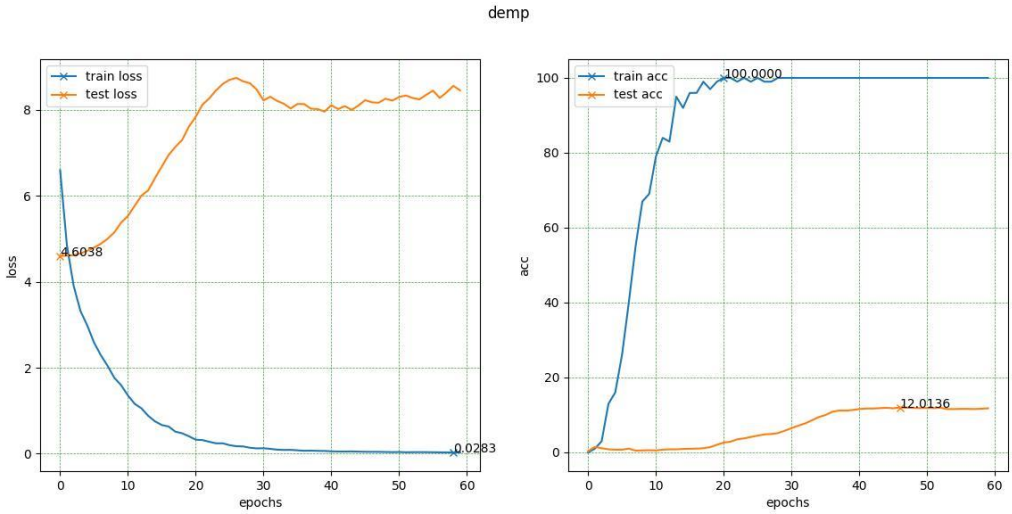
اکنون دقت داده های test_all را بررسی میکنیم:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD



از بهینه ساز Adam نیز استفاده میکنیم و نتیجه را مقایسه میکنیم:

Batch size=250, learning rate=0.0003, num epochs=60, opt=Adam



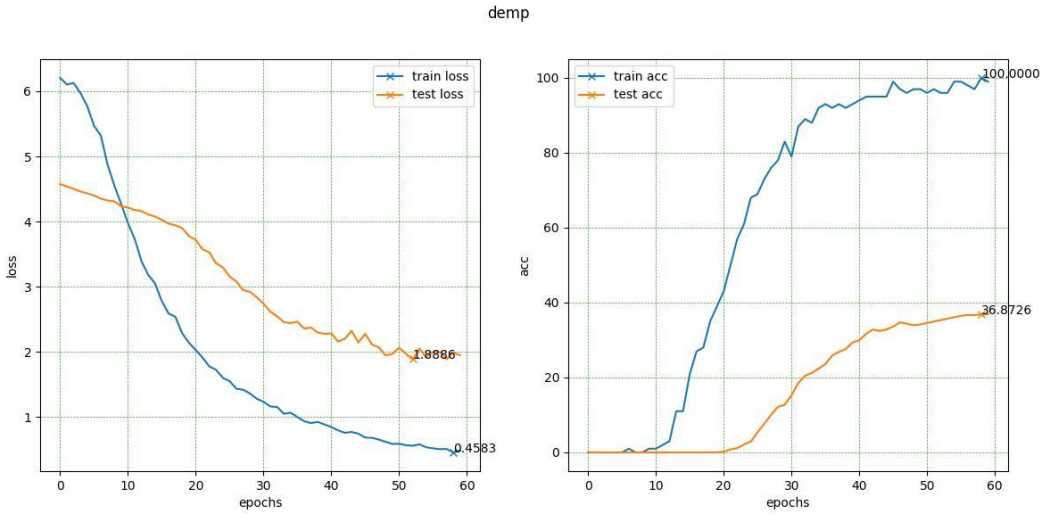
در نتیجه بهینه ساز SGD در این بخش هم عملکرد بهتری داشته است.

بخش سوم

در روش آخر علاوه بر پارامتر های لایه پیجشی بخشی از وزن های لایه آخر که از شبکه فاز ۱کپی کردیم را هم فریز می کنیم.

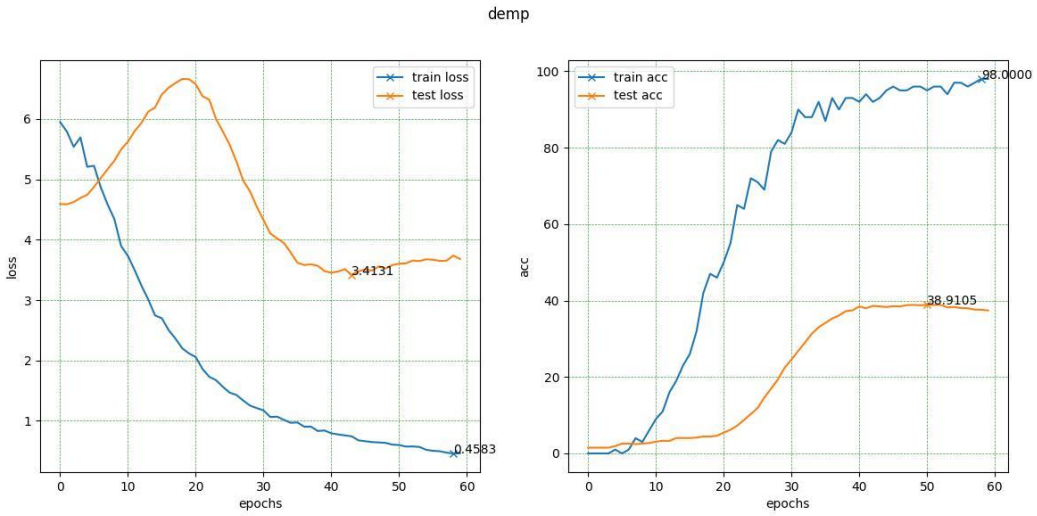
دقت تست B_test_dl در این حالت را بدست میاوریم:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD

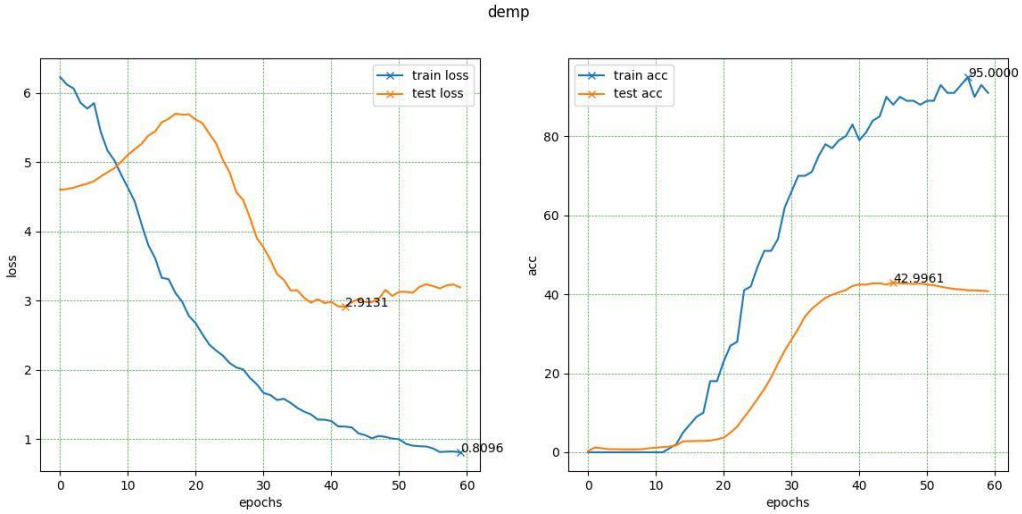


اکنون دقت داده های test_all را بررسی میکنیم:

Batch size=250, learning rate=0.0003, num epochs=60, momentum=0.9, opt=SGD



Batch size=250, learning rate=0.0002, num epochs=60, momentum=0.9, opt=SGD



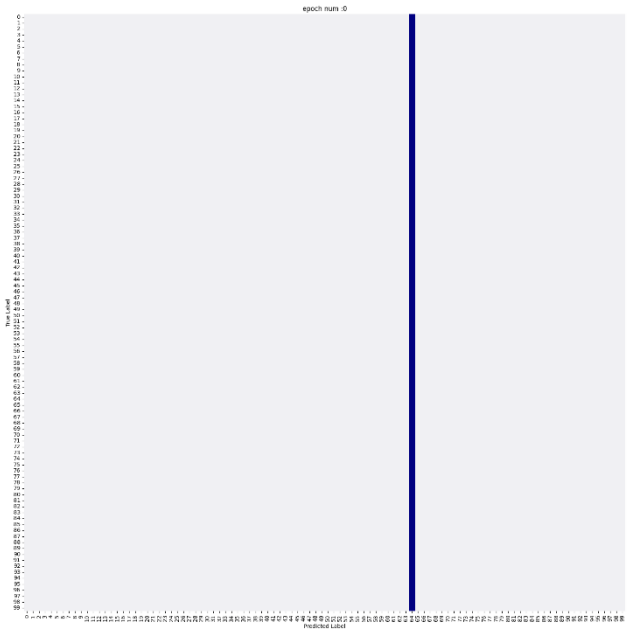
در نتیجه در این بخش لرنینگ ریت ۰.۰۰۰۲ عملکرد بهتری دارد. مقادیر سایر پارامتر ها هم که مشخص شده است.

بررسی ماتریس گمراهی

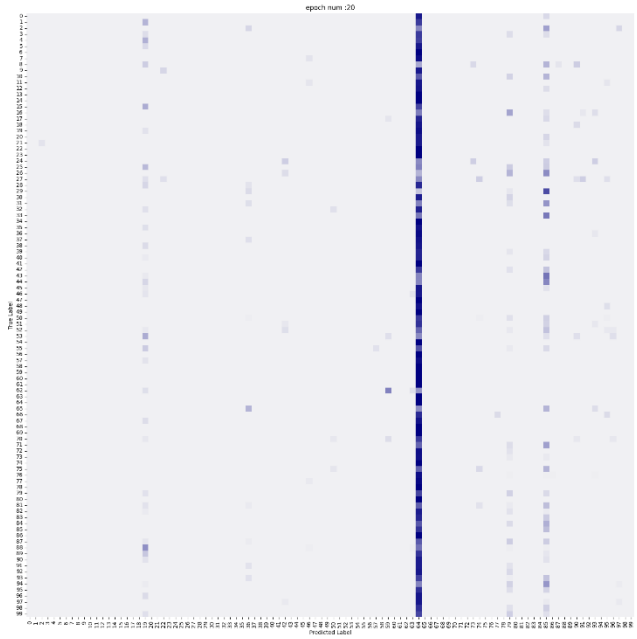
بخش اول فاز ۲

ماتریس های گمراهی در این بخش، همانطور که در زیر نشان داده شده اند تغییر میکنند. (همراه با این داکیومنت سه عدد گیف ارسال شده است که هرکدام روند تغییر ماتریس گمراهی را برای هر یک از سه بخش فاز ۲ نشان میدهند.)

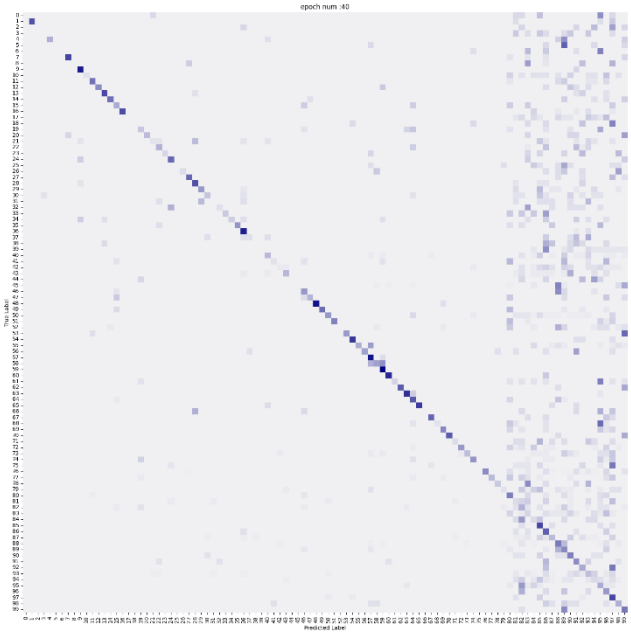
Epoch 0



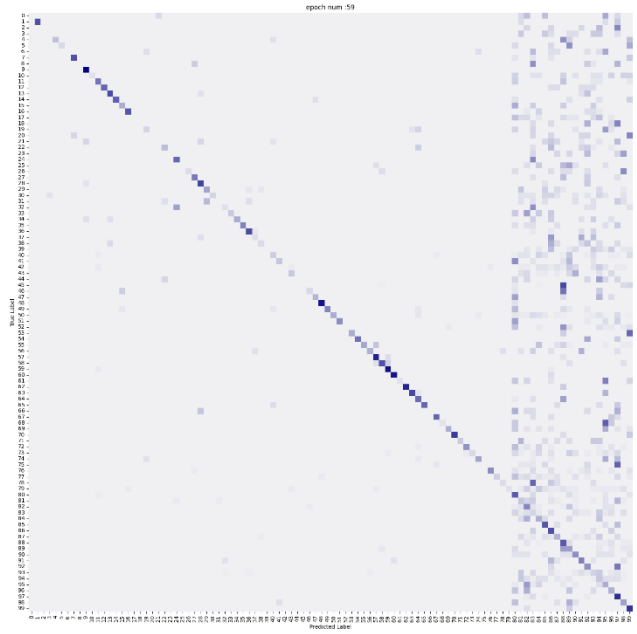
Epoch 20



Epoch 40

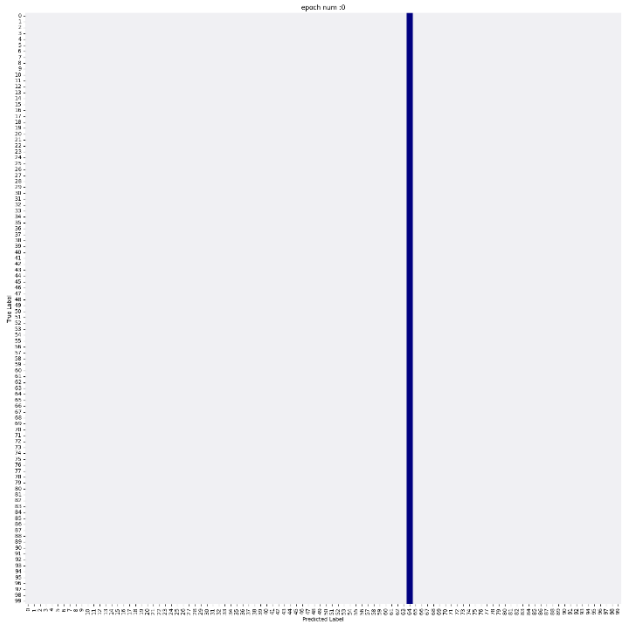


Epoch 59

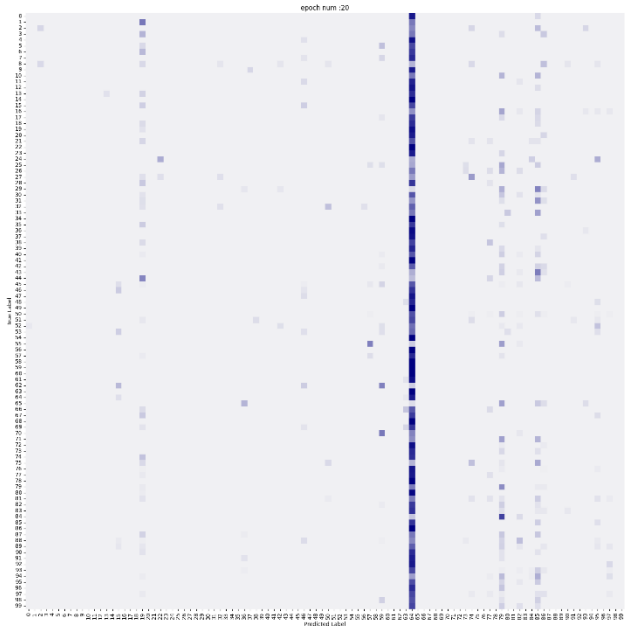


بخش دوم فاز ۲

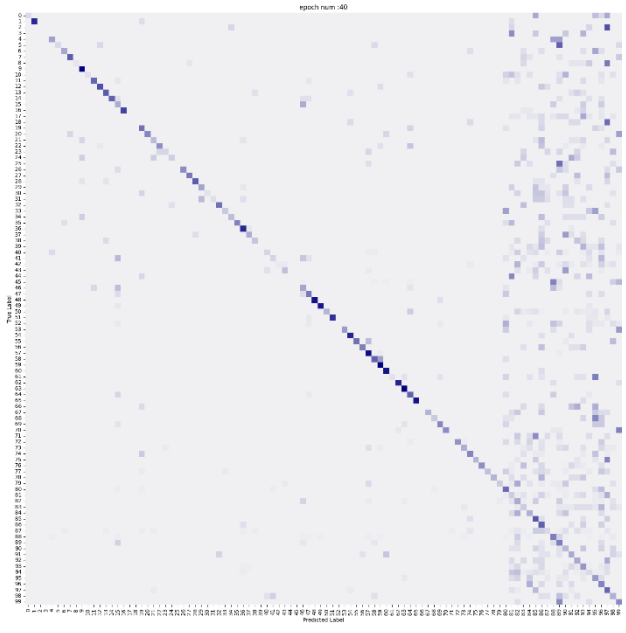
Epoch 0



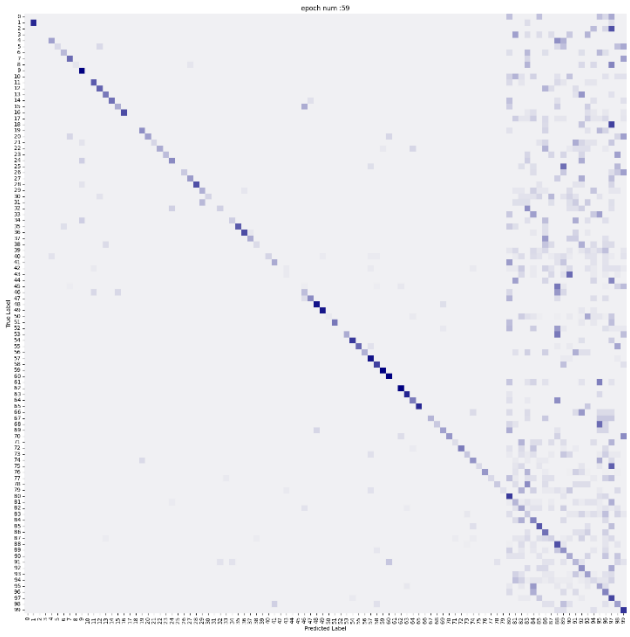
Epoch 20



Epoch 40

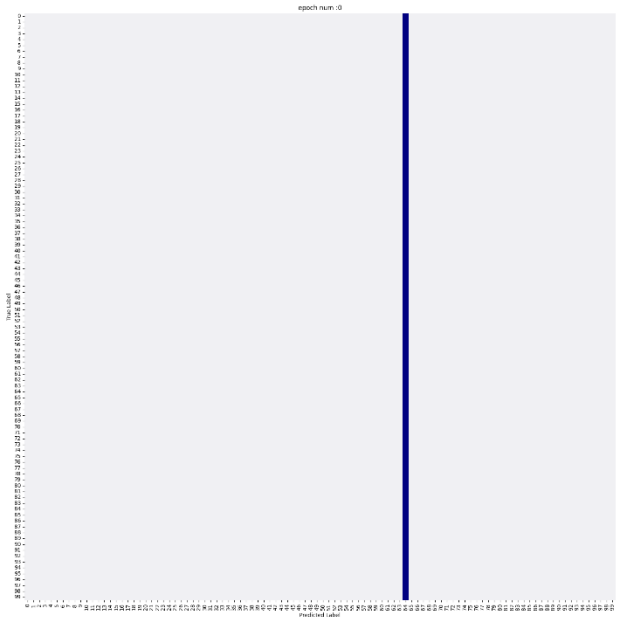


Epoch 59

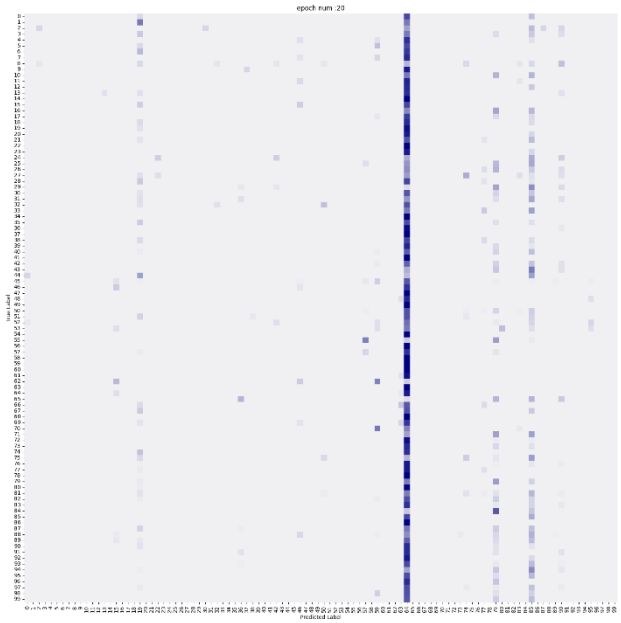


بخش سوم فاز ۲

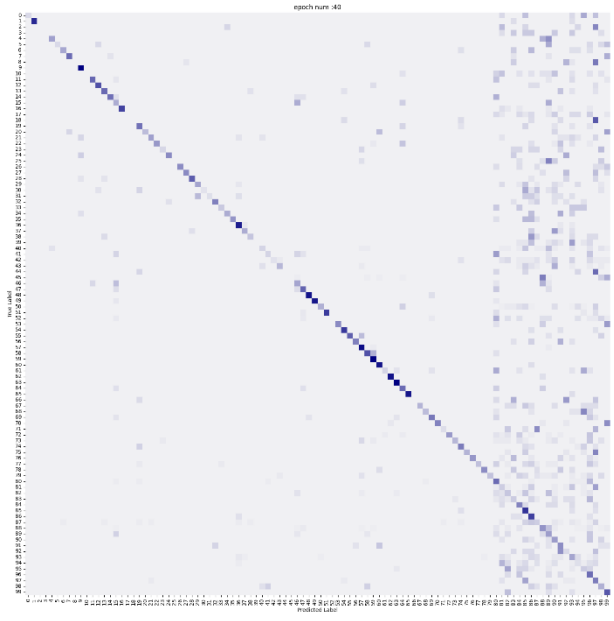
Epoch 0



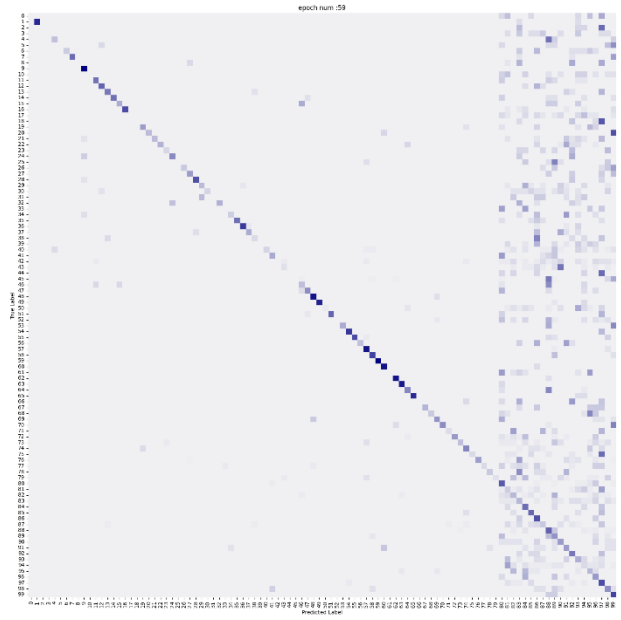
Epoch 20



Epoch 40



Epoch 59



میزان دقت کلاس هایی که از قبل آموزش دیده اند (۸۰ کلاس اول) در ابتدا خیلی کم است اما به تدریج در هر سه روش این دقت بیشتر میشود. اما همانطور که میتوان دید لیبل بسیاری از داده های ۸۰ کلاس اول به اشتباه با لیبل ۲۰ کلاس آخر پیشبینی شده است. دلیل آن این است که ما کلاس های از قبل آموزش دیده را دوباره با ۲۰ کلاس آخر آموزش میبینیم و در این حالت شبکه سعی میکند وزن هارا طوری تغییر دهد که بهتر بتواند لیبل داده های کلاس های دیتاست B را پیشبینی کند و از آنجا که در این حالت دیگر داده آموزشی از ۸۰ کلاس اول نداریم، آموزشی که شبکه روی آنها دیده است کمی تخریب میشود و شبکه آنها را بیشتر با داده های ۲۰ کلاس آخر لیبل میزند.

میزان این افت دقت داده های ۸۰ کلاس اول در بخش اول فاز ۲ که همه شبکه بدون فریز شدن توسط دیتاست B آموزش میبیند بیشتر از حالت دوم است که بخشی از شبکه فریز میشود. در بخش دوم نیز این افت بیشتر از حالت سوم است که پارامتر های بیشتری فریز میشوند.
در نتیجه و با توجه به ماتریس های گمراهی میزان افت دقت داده های ۸۰ کلاس اول در بخش اول فاز ۲ بیشتر از همه و میزان افزایش دقت داده های ۲۰ کلاس آخر در بخش اول بیشتر همه است. همینطور میزان افزایش دقت داده های ۸۰ کلاس اول در بخش سوم فاز ۲ بیشتر از همه و میزان افت دقت داده های ۲۰ کلاس آخر در بخش سوم بیشتر همه است.