به نام خدا

شناسایی الگو

گزارش پروژه چهارم

غزل زمانی نژاد

401722244

بخش اول

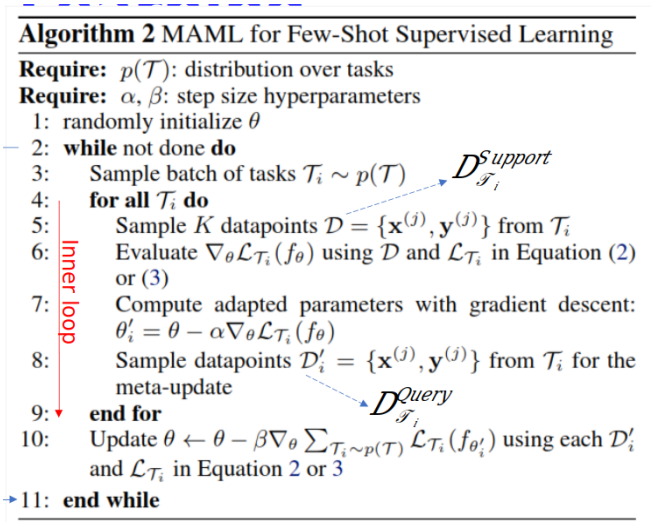
ابتدا فایل زیپ را روی درایو آپلود می کنیم تا به عکس های دیتاست دسترسی داشته باشیم. سپس فایل های متنی را می خوانیم. در هر فایل، کلاس های هر task set قرار دارد، آنها را در متغیرهای کلاس ها ذخیره می کنیم. با استفاده از transform.Compose بر روی عکس ها normalization اعمال میکنیم. سپس چند تابع برای لود کردن تصاویر از فولدرها پیاده سازی می کنیم.

با استفاده از تابع task_batch_creator یک batch از تسک ها ایجاد می کنیم. آرگومان های ورودی این تابع عبارتند از: نام کلاس های آموزش، تعداد کلاس های آموزشی، تعداد شات های آموزش (در هر کلاس)، تعداد شات های آزمون (در هر کلاس) و اندازه batch. داخل تابع یک زیرمجموعه از کلاس ها تشکیل می دهیم. سپس به اندازه N+Q تا از عکس ها لود می کنیم (به جهت اینکه داده های آموزش و آزمون نباید اشتراک داشته باشند). سپس Nتای اول را به عنوان مجموعه support و Qتای بعدی را به عنوان query در نظر می گیریم. داده ها را بر می زنیم و در پایان حلقه آن را به مجموعه تسک ها اضافه می کنیم.

```python
def task_batch_creator(C_train, n_way, N, Q, batch_size):
    """
    Creates a batch of tasks
    input:
        train classes,
        number of classes,
        number of training shots,
        number of test shots,
        batch size

    output:
        list of tasks (len(list) = batch_size), each task is a tuple of(support, query)
    """

    T = []
    for batch_idx in range(batch_size):
        Ci = random.sample(C_train, n_way)

        support, query, x_train, y_train, x_test, y_test = [], [], [], [], [], []

        for j in range(len(Ci)):
            images = load_image_name_from_folder(Ci[j])
            # support and query shouldn't have intersection
            # so sample both of them at once, then split
            tmp = random.sample(images, N + Q)
            s = load_images_of_list_from_folder(folder = Ci[j], proper = tmp)

            x_train += s[:N]
            x_test += s[N:]

            y_train += [j for i in range(N)]
            y_test += [j for i in range(Q)]

        x_train = torch.cat(x_train, 0)
        x_test = torch.cat(x_test, 0)

        y_train = torch.LongTensor(np.array(y_train))
        y_test = torch.LongTensor(np.array(y_test))

        # shuffle both train and test set
        rearranged_train = random.sample(range(x_train.shape[0]), x_train.shape[0])
        x_train = x_train[rearranged_train]
        y_train = y_train[rearranged_train]

        rearranged_test = random.sample(range(x_test.shape[0]), x_test.shape[0])
        x_test = x_test[rearranged_test]
        y_test = y_test[rearranged_test]

        # generate support and query
        support = (x_train, y_train)
        query = (x_test, y_test)

        T.append((support, query))

    return T
```

تابع calculate_accuracy_f1 با دریافت برچسب های واقعی و پیش بینی مدل به عنوان ورودی، میزان دقت مدل و همچنین امتیاز f1 را محاسبه می کند.

در بخش بعد الگوریتم MAML را پیاده سازی می کنیم:



از تابع create_model برای ایجاد مدل cnn همراه با یک لایه خطی در انتها برای پیش بینی خروجی استفاده می کنیم. سپس با صدا زدن این تابع یک مدل ایجاد می کنیم.

```python
def create_model(number_of_classes):
    """
    Create a CNN model with a classification layer in the end.
    Last layer output neurons count is number_of_classes.
    """
    model = nn.Sequential(
                            nn.Conv2d(3, 32, (3,3), (1,1)),
                            nn.BatchNorm2d(32),
                            nn.ReLU(),
                            nn.MaxPool2d((2,2)),

                            nn.Conv2d(32, 32, (3,3), (1,1)),
                            nn.BatchNorm2d(32),
                            nn.ReLU(),
                            nn.MaxPool2d((2,2)),

                            nn.Conv2d(32, 32, (3,3), (1,1)),
                            nn.BatchNorm2d(32),
                            nn.ReLU(),
                            nn.MaxPool2d((2,2)),

                            nn.Conv2d(32, 32, (3,3), (1,1)),
                            nn.BatchNorm2d(32),
                            nn.ReLU(),
                            nn.MaxPool2d((2,2)),

                            nn.Flatten(),
                            nn.Linear(288, number_of_classes)
                            )
    return model
```

در سلول بعدی هایپرپارامترها را مطابق چیزی که خواسته شده تعریف می کنیم.

از تابع train_inner_model برای آموزش مدل های درونی استفاده می کنیم. ابتدا مدل داخلی را بر روی support آموزش می دهیم و فرایند back propagation را اعمال می کنیم. سپس بر روی مجموعه query آزمایش می کنیم و دو بار عملیات back propagation را انجام می دهیم (مطابق الگوریتم MAML). گرادیان های مدل داخلی را در یک لیست ذخیره می کنیم. خطا، گرادیان ها، میزان دقت و امتیاز f1 را به عنوان خروجی برمی گردانیم.

```python
def train_inner_model(model, task, inner_epochs, device, inner_lr):
    criterion = nn.CrossEntropyLoss()
    inner_optimizer = optim.SGD(model.parameters(), lr=inner_lr)

    (x_train, y_train), (x_test, y_test) = task
    x_train, y_train = x_train.to(device), y_train.to(device)
    x_test, y_test = x_test.to(device), y_test.to(device)

    model.train()
    for i in range(inner_epochs):
        inner_optimizer.zero_grad()
        train_prediction = model(x_train)
        inner_loss = criterion(train_prediction, y_train)
        inner_loss.backward()
        inner_optimizer.step()

    inner_optimizer.zero_grad()
    test_prediction = model(x_test)
    test_loss = criterion(test_prediction, y_test)

    test_loss.backward(retain_graph=True)
    test_loss.backward()

    grads = []
    for param in list(model.parameters()):
        grads.append(param.grad)

    loss = test_loss.item()
    acc, f1_score = calculate_accuracy_f1(y_test, test_prediction)

    return loss, grads, acc, f1_score
```

از تابع train_meta_model برای آموزش مدل متا (مدل اصلی) استفاده می کنیم. گرادیان مدل اصلی را میانگینی از گرادیان های مدل های داخلی قرار می دهیم. سپس پارامترهای مدل متا را آپدیت می کنیم. میزان دقت، خطا و امتیاز f1 مدل متا میانگینی از همین معیارها برای مدل های داخلی آن است. این مقادیر را به عنوان خروجی بازمی گردانیم.

```python
def train_meta_model(meta_model, outer_lr, meta_batch_size, device, inner_models_grads, inner_models_losses, inner_models_accuracies, inner_model
    with torch.no_grad():
        meta_model.to(device)
        meta_model_grads = [torch.zeros_like(param) for param in inner_models_grads[0]]

        # update meta model gradients based on inner model grads
        for base_model_grads in inner_models_grads:
            for grad_index in range(len(base_model_grads)):
                base_model_grad = base_model_grads[grad_index]
                meta_model_grads[grad_index] += (1/meta_batch_size) * base_model_grad

        # update meta model parameters
        for parameter, grad in zip(meta_model.parameters(), meta_model_grads):
            parameter.copy_(parameter - outer_lr * grad)

    meta_accuracy = sum(inner_models_accuracies) / meta_batch_size
    meta_f1 = sum(inner_models_f1_scores) / meta_batch_size
    meta_loss = sum(inner_models_losses) / meta_batch_size

    return meta_model, meta_loss, meta_accuracy, meta_f1

```

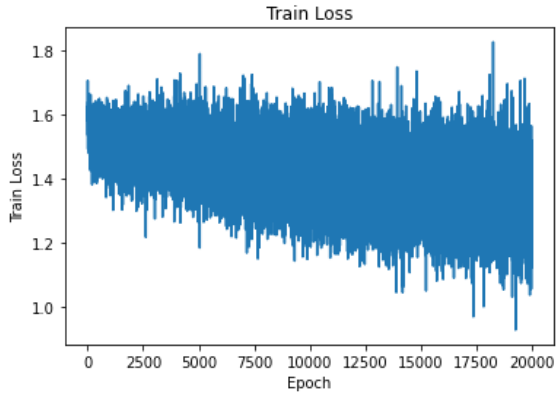
در نهایت باید از توابعی که تعریف کردیم استفاده کنیم و مدل را آموزش دهیم. در حلقه بیرونی به تعداد outer_epochs (در اینجا ۲۰هزار ایپاک) iterate می کنیم و این مراحل را تکرار می کنیم:

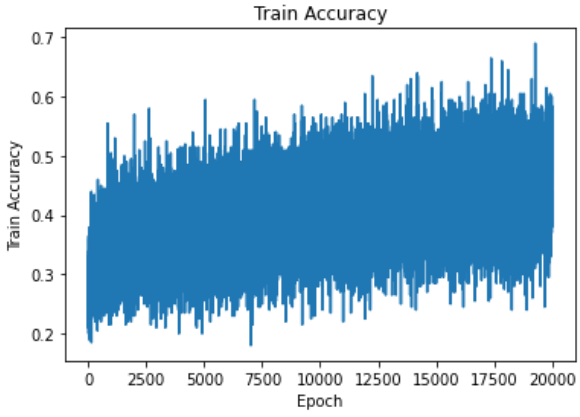
تسک ها را تشکیل می دهیم. روی تسک ها می چرخیم و مدل داخلی را به تعداد inner_train_epochs آموزش می دهیم. میزان خطا و دقت و گرادیان ها را در یک لیست ذخیره می کنیم. پس از آموزش بر روی تمامی تسک ها مدل متا را آموزش می دهیم. اطلاعات مربوط به مدل را ذخیره می کنیم تا در ادامه آنها را پلات کنیم.

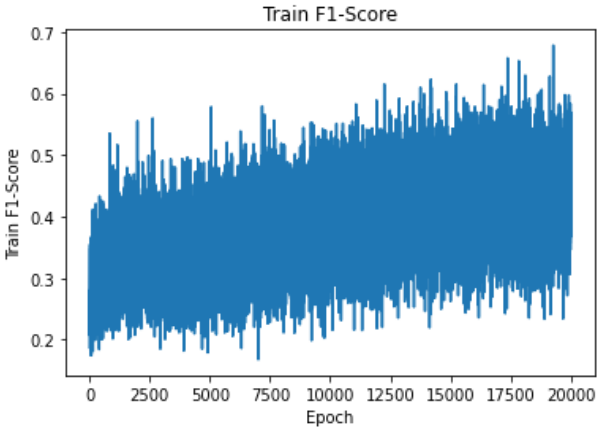
اطلاعات مربوط به آموزش مدل را پس از تعدادی epoch در کنسول چاپ می کنیم. همچنین پس از هر ۲۰۰ ایپاک باید مدل متا را بر روی test task set آزمایش کنیم. برای این کار همین مراحل را تکرار می کنیم (ساختن تعدادی batch از مجموعه تسک آزمون و آموزش مدل های داخلی). در نهایت نتیجه را چاپ می کنیم.

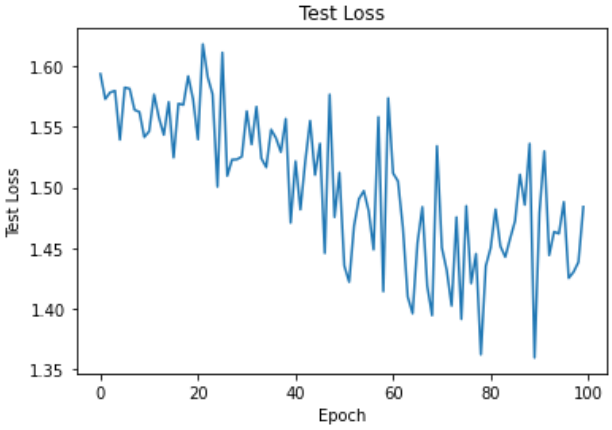
5 way, 1 shot:

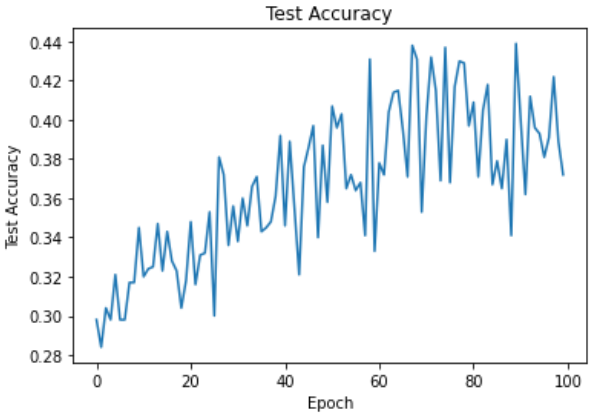
```
Testing ****** Epoch 18600: loss = 1.4440956383943557, acc = 0.4120000000000001, f1 = 0.40272643563595983
Epoch 18650: loss = 1.3892336189746857, accuracy = 0.4200000000000004, f1 = 0.40635138119086167
Epoch 18700: loss = 1.065050795674324, accuracy = 0.605, f1 = 0.6018016083765175
Epoch 18750: loss = 1.262521043419838, accuracy = 0.4600000000000001, f1 = 0.44572922499487666
Epoch 18800: loss = 1.180370956659317, accuracy = 0.53, f1 = 0.5172397226694575
Testing ****** Epoch 18800: loss = 1.4634500294923782, acc = 0.396, f1 = 0.3840584506254179
Epoch 18850: loss = 1.3378766179084778, accuracy = 0.485, f1 = 0.4679389949155801
Epoch 18900: loss = 1.2551009356975555, accuracy = 0.495, f1 = 0.4826897818383887
Epoch 18950: loss = 1.4332189559936523, accuracy = 0.37500000000000006, f1 = 0.3638106048304257
Epoch 19000: loss = 1.3943907618522644, accuracy = 0.39499999999999996, f1 = 0.37784789041367983
Testing ****** Epoch 19000: loss = 1.4619535505771637, acc = 0.39300000000000007, f1 = 0.3853737679681298
Epoch 19050: loss = 1.3617627024650574, accuracy = 0.46499999999999997, f1 = 0.45371078368244877
Epoch 19100: loss = 1.4129424393177032, accuracy = 0.385, f1 = 0.3764023340145295
Epoch 19150: loss = 1.2100639343261719, accuracy = 0.49, f1 = 0.4782340423930477
Epoch 19200: loss = 1.2110781222581863, accuracy = 0.52, f1 = 0.5041757265483204
Testing ****** Epoch 19200: loss = 1.4880663752555847, acc = 0.3810000000000001, f1 = 0.3726345317692135
Epoch 19250: loss = 1.3132221400737762, accuracy = 0.44499999999999995, f1 = 0.42923940992124393
Epoch 19300: loss = 1.5687996447086334, accuracy = 0.32, f1 = 0.3182248899228977
Epoch 19350: loss = 1.3158117681741714, accuracy = 0.46, f1 = 0.4573691642595601
Epoch 19400: loss = 1.3615043759346008, accuracy = 0.5, f1 = 0.49291079831462115
Testing ****** Epoch 19400: loss = 1.4252218425273895, acc = 0.3910000000000001, f1 = 0.37891387863155473
Epoch 19450: loss = 1.3625324368476868, accuracy = 0.465, f1 = 0.44818436921209837
Epoch 19500: loss = 1.7063796520233154, accuracy = 0.28, f1 = 0.2832943442268385
Epoch 19550: loss = 1.571099579334259, accuracy = 0.36, f1 = 0.35493412003086927
Epoch 19600: loss = 1.3493180125951767, accuracy = 0.45500000000000007, f1 = 0.4448628250273828
Testing ****** Epoch 19600: loss = 1.4302800178527832, acc = 0.42200000000000004, f1 = 0.41140552983134093
Epoch 19650: loss = 1.3891113996505737, accuracy = 0.39499999999999996, f1 = 0.3869677530445803
Epoch 19700: loss = 1.298498272895813, accuracy = 0.48500000000000004, f1 = 0.464540415108291
Epoch 19750: loss = 1.245304822921753, accuracy = 0.5050000000000001, f1 = 0.48130890103076807
Epoch 19800: loss = 1.1881439983844757, accuracy = 0.53, f1 = 0.517655307331576
Testing ****** Epoch 19800: loss = 1.4384540855884551, acc = 0.38899999999999996, f1 = 0.38241107522500295
Epoch 19850: loss = 1.3299782276153564, accuracy = 0.425, f1 = 0.4178194462094045
Epoch 19900: loss = 1.2461908459663391, accuracy = 0.52, f1 = 0.5089507250575362
Epoch 19950: loss = 1.449425369501114, accuracy = 0.37, f1 = 0.3538263065623747
Epoch 20000: loss = 1.392442524433136, accuracy = 0.42500000000000004, f1 = 0.41755694836863233
Testing ****** Epoch 20000: loss = 1.4840243101119994, acc = 0.372, f1 = 0.3679618725942023
```





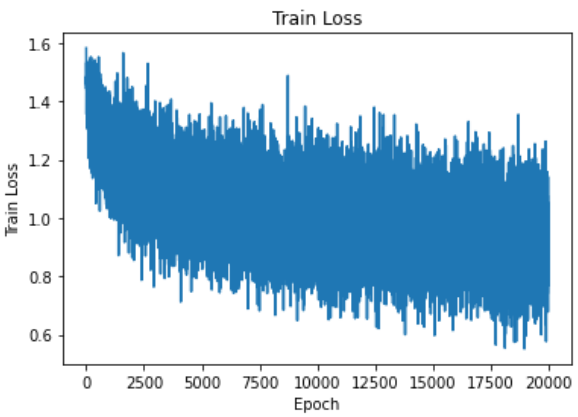

Train F1-Score
Train F1-Score
Epoch
0.7
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000

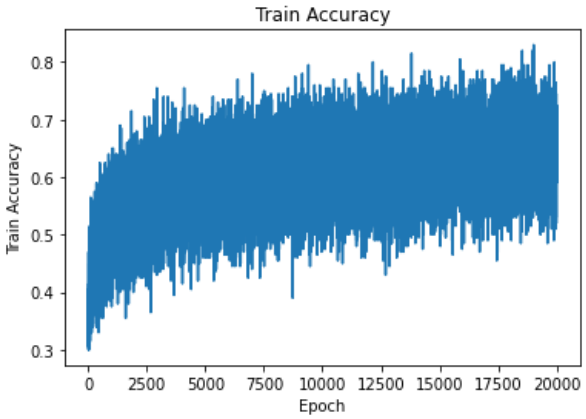

Test Loss
Test Loss
Epoch
1.60
1.55
1.50
1.45
1.40
1.35
0
20
40
60
80
100

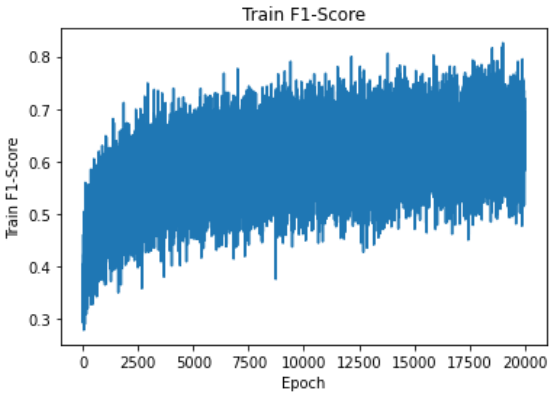

Test Accuracy
Test Accuracy
Epoch
0.44
0.42
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
20
40
60
80
100

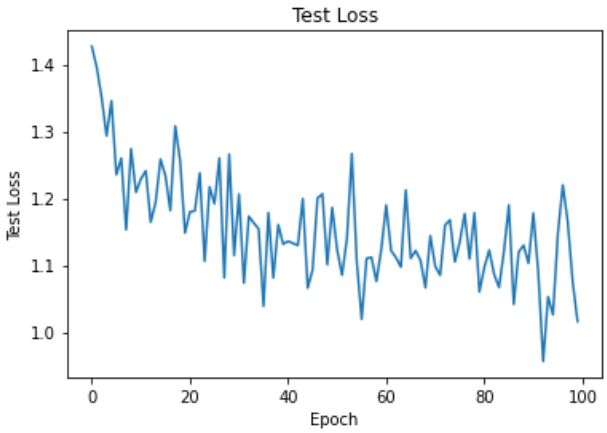
5 way, 5 shot:

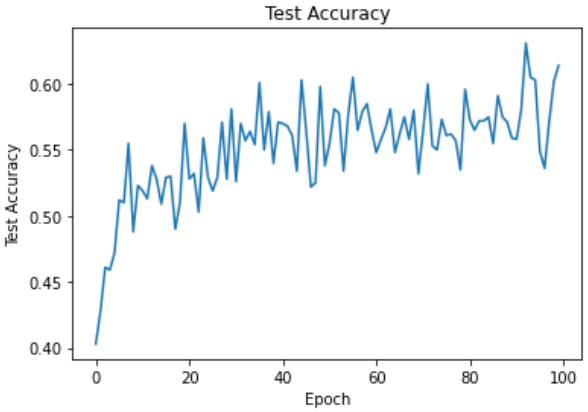
```
Testing ****** Epoch 18600: loss = 0.9561964780092239, acc = 0.6309999999999999, f1 = 0.6243422341601506
Epoch 18650: loss = 0.8257748484611511, accuracy = 0.6799999999999999, f1 = 0.6778993131126662
Epoch 18700: loss = 0.8763612061738968, accuracy = 0.665, f1 = 0.6629277739804056
Epoch 18750: loss = 0.8935050368309021, accuracy = 0.6799999999999999, f1 = 0.6752193093583254
Epoch 18800: loss = 0.9497268944978714, accuracy = 0.625, f1 = 0.6150906192863387
Testing ****** Epoch 18800: loss = 1.0527183383703231, acc = 0.605, f1 = 0.5999054025437927
Epoch 18850: loss = 0.747457891702652, accuracy = 0.73, f1 = 0.7268362928071166
Epoch 18900: loss = 0.8231830894947052, accuracy = 0.71, f1 = 0.7052669882875832
Epoch 18950: loss = 0.7822913378477097, accuracy = 0.74, f1 = 0.7319278044759266
Epoch 19000: loss = 1.032529503107071, accuracy = 0.605, f1 = 0.5964001268769071
Testing ****** Epoch 19000: loss = 1.0260397106409074, acc = 0.6029999999999999, f1 = 0.598368583552222
Epoch 19050: loss = 0.9802742451429367, accuracy = 0.62, f1 = 0.6131130279585657
Epoch 19100: loss = 0.8213019520044327, accuracy = 0.6900000000000001, f1 = 0.6867653857188115
Epoch 19150: loss = 0.948848769068718, accuracy = 0.625, f1 = 0.621036316717936
Epoch 19200: loss = 1.0782100558280945, accuracy = 0.5900000000000001, f1 = 0.573249480997159
Testing ****** Epoch 19200: loss = 1.1476296603679657, acc = 0.5489999999999999, f1 = 0.5444620409458476
Epoch 19250: loss = 1.0044443309307098, accuracy = 0.6150000000000001, f1 = 0.6024597580520316
Epoch 19300: loss = 0.8133462816476822, accuracy = 0.69, f1 = 0.6947649509148365
Epoch 19350: loss = 0.9055788218975067, accuracy = 0.67, f1 = 0.671841659873158
Epoch 19400: loss = 1.1133325599401474, accuracy = 0.56, f1 = 0.5572900701435834
Testing ****** Epoch 19400: loss = 1.219716441631317, acc = 0.5359999999999998, f1 = 0.5314279955973792
Epoch 19450: loss = 0.8834536224603653, accuracy = 0.645, f1 = 0.6380357609030715
Epoch 19500: loss = 0.7043053656816483, accuracy = 0.75, f1 = 0.745123626317596
Epoch 19550: loss = 0.9462763071060181, accuracy = 0.6799999999999999, f1 = 0.6789011375994819
Epoch 19600: loss = 0.7491015791893005, accuracy = 0.7249999999999999, f1 = 0.7272915921517348
Testing ****** Epoch 19600: loss = 1.1651746422052383, acc = 0.572, f1 = 0.564367386662036
Epoch 19650: loss = 1.067915067076683, accuracy = 0.5700000000000001, f1 = 0.5638485755580923
Epoch 19700: loss = 0.9822868406772614, accuracy = 0.6100000000000001, f1 = 0.5953537278443726
Epoch 19750: loss = 0.6982305645942688, accuracy = 0.755, f1 = 0.7528877248146328
Epoch 19800: loss = 0.9237285405397415, accuracy = 0.6, f1 = 0.5936504112427857
Testing ****** Epoch 19800: loss = 1.076124483346939, acc = 0.602, f1 = 0.5972892635176253
Epoch 19850: loss = 1.1280161440372467, accuracy = 0.5750000000000001, f1 = 0.5759295978359447
Epoch 19900: loss = 0.8472823351621628, accuracy = 0.645, f1 = 0.6429995744223769
Epoch 19950: loss = 0.8301142305135727, accuracy = 0.69, f1 = 0.691303710479407
Epoch 20000: loss = 1.04872228205204, accuracy = 0.595, f1 = 0.594293784915672
Testing ****** Epoch 20000: loss = 1.0160291016101837, acc = 0.614, f1 = 0.6078882254208003
```











هماهنطور که انتظار می رفت، دقت ۵شات در آزمون (۶۱درصد) بسیار بیشتر از دقت اشات (۳۷ درصد) است. زیرا در ۵شات مدل از هر کلاس ۵ نمونه را یاد گرفته است پس بهتر می تواند موارد یاد گرفته را تعمیم دهد.

بخش دوم

الگوریتم FOMAML یک نسخه first order از همان الگوریتم MAML است. تفاوت این دو الگوریتم در محاسبه گرادیان های داخلی است. به همین علت کافیست تنها تابع train_inner_model را اندکی تغییر دهیم.

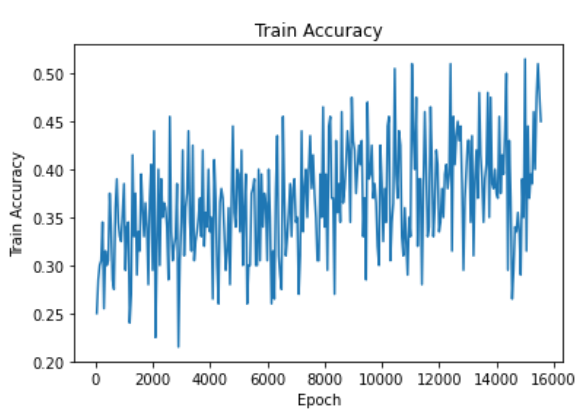
```python
def train_inner_model(model, task, inner_epochs, device, inner_lr):
    criterion = nn.CrossEntropyLoss()
    inner_optimizer = optim.SGD(model.parameters(), lr=inner_lr)

    (x_train, y_train), (x_test, y_test) = task
    x_train, y_train = x_train.to(device), y_train.to(device)
    x_test, y_test = x_test.to(device), y_test.to(device)

    model.train()
    for i in range(inner_epochs):
        inner_optimizer.zero_grad()
        train_prediction = model(x_train)
        inner_loss = criterion(train_prediction, y_train)
        inner_loss.backward()
        inner_optimizer.step()

    inner_optimizer.zero_grad()
    test_prediction = model(x_test)
    test_loss = criterion(test_prediction, y_test)

    # only this part is different from MAML
    test_loss.backward()

    grads = []
    for param in list(model.parameters()):
        grads.append(param.grad)

    loss = test_loss.item()
    acc, f1_score = calculate_accuracy_f1(y_test, test_prediction)

    return loss, grads, acc, f1_score
```

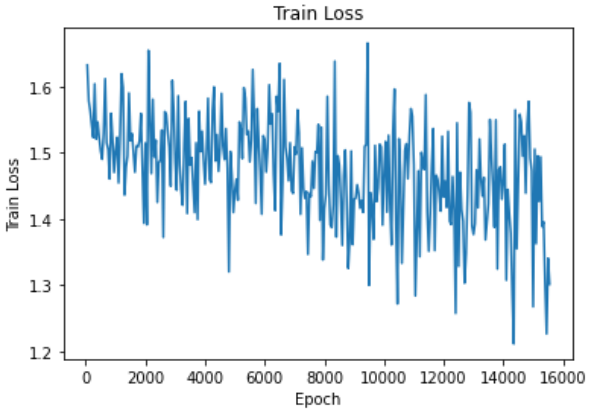
مدل ها را مطابق موارد خواسته شده اجرا کرده و نتیجه را چاپ می کنیم.

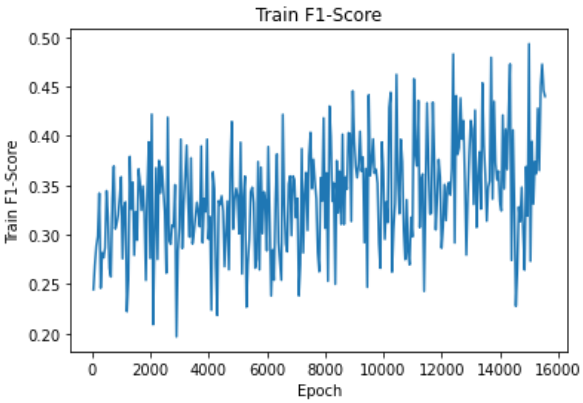
(در این بخش ها با توجه به طولانی بودن مدت زمان اجرا و محدودیت کولب در اختصاص دادن ۲۰ ،GPUهزار ایپاک به پایان نرسید.)

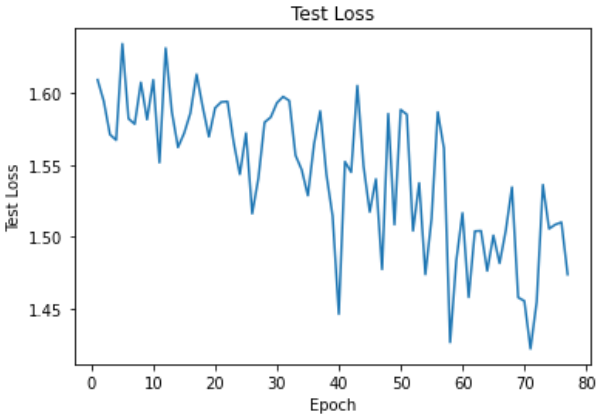
5 way, 1 shot:

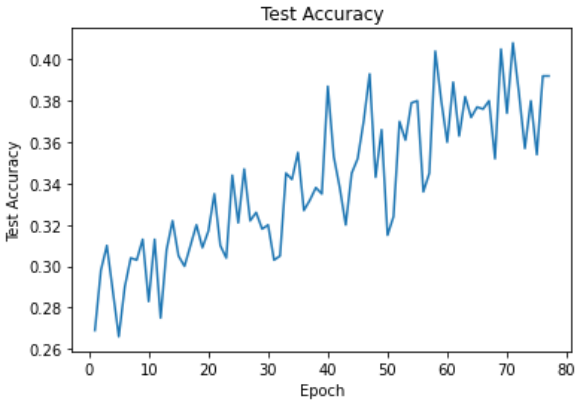
```
Testing ****** Epoch 14200: loss = 1.4224429309368134, acc = 0.40800000000000003, f1 = 0.39643002504617125
Epoch 14250: loss = 1.3784436583518982, accuracy = 0.395, f1 = 0.366017569402076
Epoch 14300: loss = 1.2882778644561768, accuracy = 0.44, f1 = 0.42404451495294815
Epoch 14350: loss = 1.2114613503217697, accuracy = 0.5, f1 = 0.4727809376289993
Epoch 14400: loss = 1.5649556517601013, accuracy = 0.295, f1 = 0.27416707913985594
Testing ****** Epoch 14400: loss = 1.4550746083259583, acc = 0.384, f1 = 0.3615086326126316
Epoch 14450: loss = 1.3550857603549957, accuracy = 0.43000000000000005, f1 = 0.4060070424744338
Epoch 14500: loss = 1.4573509693145752, accuracy = 0.37, f1 = 0.32563917079247495
Epoch 14550: loss = 1.5584344565868378, accuracy = 0.265, f1 = 0.2276424724700587
Epoch 14600: loss = 1.5467072427272797, accuracy = 0.29500000000000004, f1 = 0.26562610833949507
Testing ****** Epoch 14600: loss = 1.5362450003623962, acc = 0.35700000000000004, f1 = 0.34373413235959055
Epoch 14650: loss = 1.4951517283916473, accuracy = 0.34, f1 = 0.3281598684539861
Epoch 14700: loss = 1.5260731279850006, accuracy = 0.33499999999999996, f1 = 0.3134143835891995
Epoch 14750: loss = 1.4392695426940918, accuracy = 0.355, f1 = 0.34805053616111636
Epoch 14800: loss = 1.543229192495346, accuracy = 0.335, f1 = 0.30185111258719344
Testing ****** Epoch 14800: loss = 1.5057204961776733, acc = 0.37999999999999995, f1 = 0.3622649140217512
Epoch 14850: loss = 1.5777954757213593, accuracy = 0.29, f1 = 0.2643428106520902
Epoch 14900: loss = 1.4918261766433716, accuracy = 0.39, f1 = 0.36893653212432276
Epoch 14950: loss = 1.473460465669632, accuracy = 0.35000000000000003, f1 = 0.31901197988383345
Epoch 15000: loss = 1.2675453126430511, accuracy = 0.515, f1 = 0.4932345406322389
Testing ****** Epoch 15000: loss = 1.5086307644844055, acc = 0.3540000000000001, f1 = 0.3365475554854004
Epoch 15050: loss = 1.505707859992981, accuracy = 0.315, f1 = 0.273243166419637
Epoch 15100: loss = 1.3629459738731384, accuracy = 0.445, f1 = 0.3944986370698445
Epoch 15150: loss = 1.4957293272018433, accuracy = 0.37, f1 = 0.3311366876895954
Epoch 15200: loss = 1.4268141090869904, accuracy = 0.395, f1 = 0.37403820816864297
Testing ****** Epoch 15200: loss = 1.5102312386035919, acc = 0.392, f1 = 0.37639562570436846
Epoch 15250: loss = 1.4940739870071411, accuracy = 0.385, f1 = 0.36176685843881967
Epoch 15300: loss = 1.38871631026268, accuracy = 0.46, f1 = 0.4279025846948852
Epoch 15350: loss = 1.3958708941936493, accuracy = 0.39999999999999997, f1 = 0.3649940284946006
Epoch 15400: loss = 1.303047925233841, accuracy = 0.475, f1 = 0.450266687926871
Testing ****** Epoch 15400: loss = 1.4740155190229416, acc = 0.392, f1 = 0.3806590844535469
Epoch 15450: loss = 1.226451799273491, accuracy = 0.51, f1 = 0.47254286468480844
Epoch 15500: loss = 1.3408271372318268, accuracy = 0.48, f1 = 0.44607371011653735
Epoch 15550: loss = 1.3016452193260193, accuracy = 0.44999999999999996, f1 = 0.43998123190056837
```





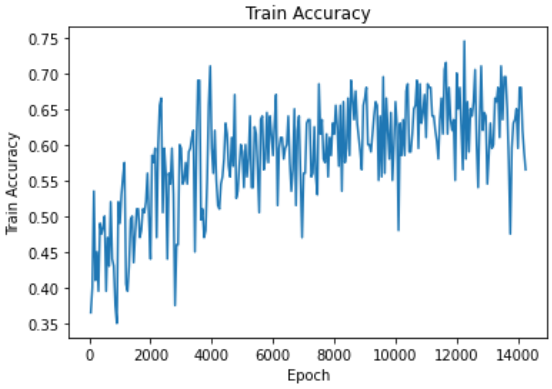

Train F1-Score
Train F1-Score
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0
2000
4000
6000
8000
10000
12000
14000
16000
Epoch

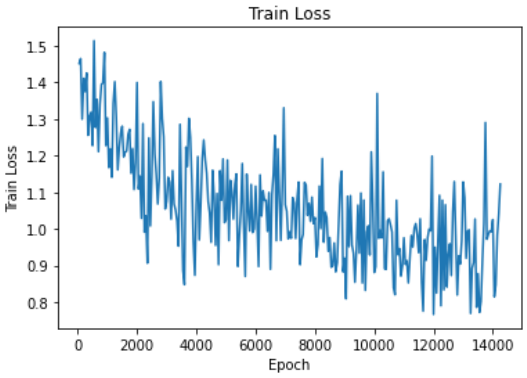

Test Loss
Test Loss
1.60
1.55
1.50
1.45
0
10
20
30
40
50
60
70
80
Epoch

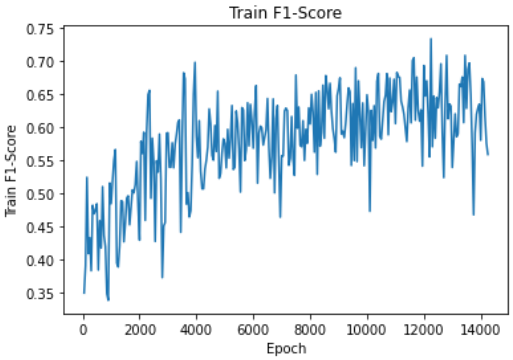

Test Accuracy
Test Accuracy
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0.26
0
10
20
30
40
50
60
70
80
Epoch

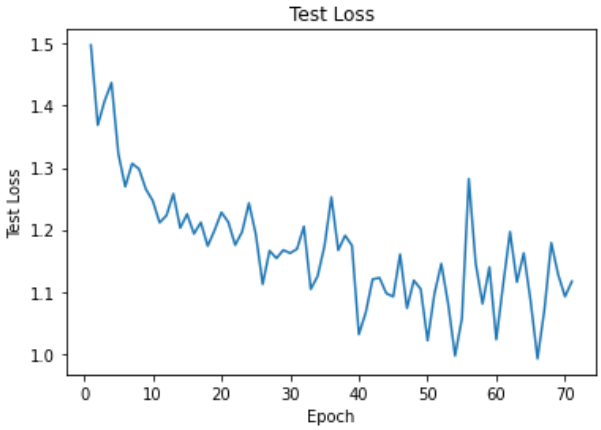
5 way, 5 shot:

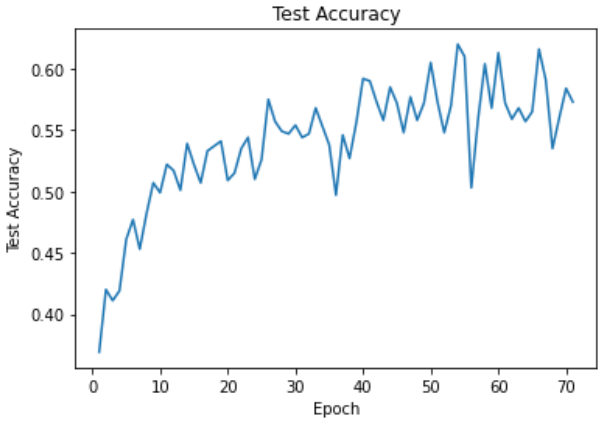
```
Testing ****** Epoch 12800: loss = 1.1624925523996352, acc = 0.557, f1 = 0.5484322427725578
Epoch 12850: loss = 0.9257949590682983, accuracy = 0.62, f1 = 0.6130646849822378
Epoch 12900: loss = 0.903805136680603, accuracy = 0.645, f1 = 0.6354063766134703
Epoch 12950: loss = 0.9669619798660278, accuracy = 0.64, f1 = 0.6322126171042579
Epoch 13000: loss = 1.128002643585205, accuracy = 0.545, f1 = 0.5395776001267992
Testing ****** Epoch 13000: loss = 1.0848445773124695, acc = 0.565, f1 = 0.5560197865243296
Epoch 13050: loss = 1.0839335918426514, accuracy = 0.5850000000000001, f1 = 0.5813872866600354
Epoch 13100: loss = 0.9197811186313629, accuracy = 0.63, f1 = 0.620582620816165
Epoch 13150: loss = 0.9940182864665985, accuracy = 0.595, f1 = 0.5859897868168833
Epoch 13200: loss = 0.9970530569553375, accuracy = 0.6, f1 = 0.5905492510921573
Testing ****** Epoch 13200: loss = 0.9922660142183304, acc = 0.616, f1 = 0.6080576797142822
Epoch 13250: loss = 0.7688104435801506, accuracy = 0.665, f1 = 0.6653225867699551
Epoch 13300: loss = 0.8917913436889648, accuracy = 0.6599999999999999, f1 = 0.661800903774588
Epoch 13350: loss = 0.9077968299388885, accuracy = 0.68, f1 = 0.6761998517478394
Epoch 13400: loss = 1.026725634932518, accuracy = 0.61, f1 = 0.6073036896958353
Testing ****** Epoch 13400: loss = 1.0709846496582032, acc = 0.591, f1 = 0.58596538772022
Epoch 13450: loss = 0.7855306714773178, accuracy = 0.71, f1 = 0.7088896182562894
Epoch 13500: loss = 0.8773947209119797, accuracy = 0.635, f1 = 0.6287959346071138
Epoch 13550: loss = 0.7717195302248001, accuracy = 0.6950000000000001, f1 = 0.688156325043019
Epoch 13600: loss = 0.8015776425600052, accuracy = 0.6950000000000001, f1 = 0.6973807183229718
Testing ****** Epoch 13600: loss = 1.1789760559797287, acc = 0.5350000000000001, f1 = 0.5295524749543392
Epoch 13650: loss = 0.9161649346351624, accuracy = 0.6499999999999999, f1 = 0.6496122646889909
Epoch 13700: loss = 1.0157598853111267, accuracy = 0.5750000000000001, f1 = 0.5712428280652806
Epoch 13750: loss = 1.2906015813350677, accuracy = 0.475, f1 = 0.467974513173968
Epoch 13800: loss = 0.9714231640100479, accuracy = 0.5950000000000001, f1 = 0.5917247071029208
Testing ****** Epoch 13800: loss = 1.1279601395130157, acc = 0.5600000000000003, f1 = 0.5561668169882197
Epoch 13850: loss = 0.9851136654615402, accuracy = 0.63, f1 = 0.6198483996261288
Epoch 13900: loss = 0.9945880472660065, accuracy = 0.635, f1 = 0.6290665893983971
Epoch 13950: loss = 0.9921182096004486, accuracy = 0.65, f1 = 0.6351470390783893
Epoch 14000: loss = 1.0251188278198242, accuracy = 0.595, f1 = 0.5807251617865092
Testing ****** Epoch 14000: loss = 1.0924732327461242, acc = 0.584, f1 = 0.5792398263056697
Epoch 14050: loss = 0.8142672777175903, accuracy = 0.68, f1 = 0.6739198709786944
Epoch 14100: loss = 0.8468272238969803, accuracy = 0.6799999999999999, f1 = 0.6689531756812009
Epoch 14150: loss = 0.982247531414032, accuracy = 0.62, f1 = 0.6161614364905521
Epoch 14200: loss = 1.0412963330745697, accuracy = 0.59, f1 = 0.5764229062056831
Testing ****** Epoch 14200: loss = 1.1168120801448822, acc = 0.573, f1 = 0.5670225068196935
Epoch 14250: loss = 1.1221460402011871, accuracy = 0.5650000000000001, f1 = 0.5592993964800399
```











این الگوریتم پیچیدگی محاسباتی کمتری دارد. عملکرد آن تقریبا مشابه الگوریتم MAML است. و این نشان دهنده آن است که بیشتر اصلاحات در MAML تحت تاثیر گرادیان اول است و مشتق دوم تاثیر چندانی ندارد. (مطابق مقاله)

همچنین تعداد شات بیشتر توانسته عملکرد مدل را بهبود بخشد و با وجود اینکه مدل دوم در تعداد اپیاک های کمتری آموزش دیده، به دقت آزمون بیشتری دست یافته است (دقت ۵۶ درصد برای مدل ۵شات و دقت ۴۴ درصد برای مدل ۱شات).

بخش سوم

قسمت اول

الگوریتم MAML

اکنون باید در حلقه درونی support با تصاویر متفاوت داشته باشیم. برای اینکه تصاویر متفاوتی داشته باشیم، متد task_batch_creator را با تعداد شات های بیشتر صدا می زنیم (در اینجا ۳ برابر حالت قبل زیرا تعداد تکرار حلقه داخلی ۳ است و ما می خواهیم تصاویر متفاوتی انتخاب کنیم) سپس متد train_inner_model را اندکی تغییر می دهیم. اکنون باید ورودی task در این تابع را به بخش های مختلف تقسیم کنیم تا support را تشکیل دهیم.

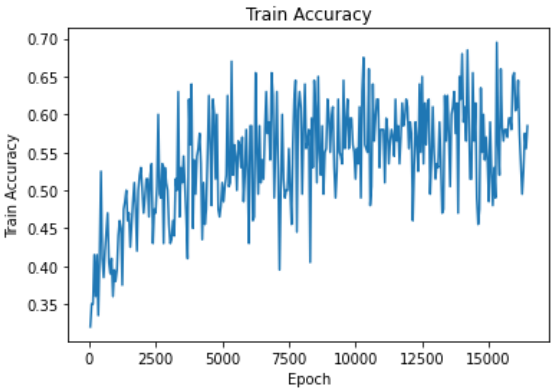
```python
def train_inner_model(model, task, inner_epochs, device, inner_lr):
    criterion = nn.CrossEntropyLoss()
    inner_optimizer = optim.SGD(model.parameters(), lr=inner_lr)

    # this part is different from Q1 and Q2
    (whole_x_train, whole_y_train), (x_test, y_test) = task[0], task[1]
    desired = whole_x_train.shape[0]//inner_epochs
    x_test, y_test = x_test.to(device), y_test.to(device)

    model.train()
    for i in range(inner_epochs):
        x_train = whole_x_train[i * desired : (i + 1) * desired]
        y_train = whole_y_train[i * desired : (i + 1) * desired]
        x_train, y_train = x_train.to(device), y_train.to(device)

        inner_optimizer.zero_grad()
        train_prediction = model(x_train)
        inner_loss = criterion(train_prediction, y_train)
        inner_loss.backward()
        inner_optimizer.step()

    model.train()
    inner_optimizer.zero_grad()
    test_prediction = model(x_test)
    test_loss = criterion(test_prediction, y_test)

    test_loss.backward(retain_graph=True)
    test_loss.backward()

    grads = []
    for param in list(model.parameters()):
        grads.append(param.grad)

    loss = test_loss.item()

    acc, f1_score = calculate_accuracy_f1(y_test, test_prediction)

    return loss, grads, acc, f1_score

```

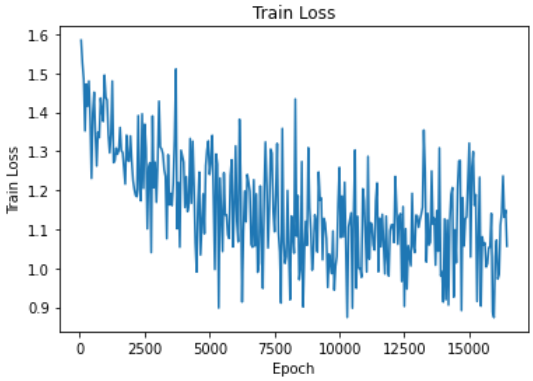
```python
import copy
metrics_hist = {}
metrics_hist['train'] = []
metrics_hist['test'] = []

for epoch in range(1, outer_epochs + 1):
    # create batches of tasks
    tasks = task_batch_creator(C_train=C_train, n_way=nway, N=support_samples*inner_epochs_train, Q=query_samples, meta_batch_size=meta_train_batch)

    inner_model_hist = {}
    inner_model_hist['grads'] = []
    inner_model_hist['losses'] = []
    inner_model_hist['accuracy'] = []
    inner_model_hist['f1'] = []

    for task_idx in range(len(tasks)):
        inner_model = copy.deepcopy(meta_model)
        inner_model.to(device)

        # train the inner model on task
        # returns (loss, grads, accuracy, f1_score)
        tmp = train_inner_model(inner_model, tasks[task_idx], inner_epochs_train, device, inner_lr)

        # save inner model history for training meta model
        inner_model_hist['losses'].append(tmp[0])
        inner_model_hist['grads'].append(tmp[1])
        inner_model_hist['accuracy'].append(tmp[2])
        inner_model_hist['f1'].append(tmp[3])
```

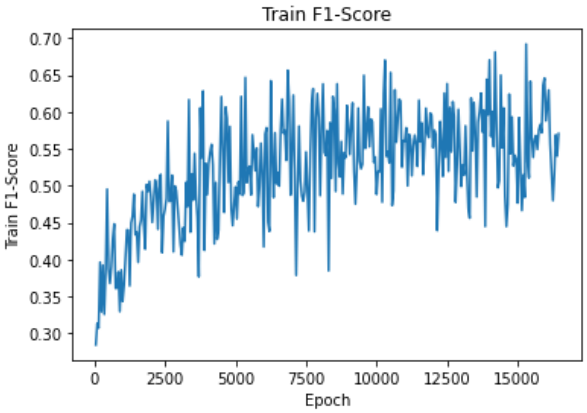
نتایج به صورت زیر است:

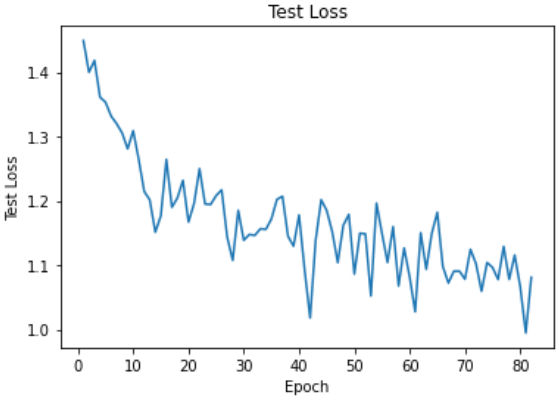
5 way, 1 shot:

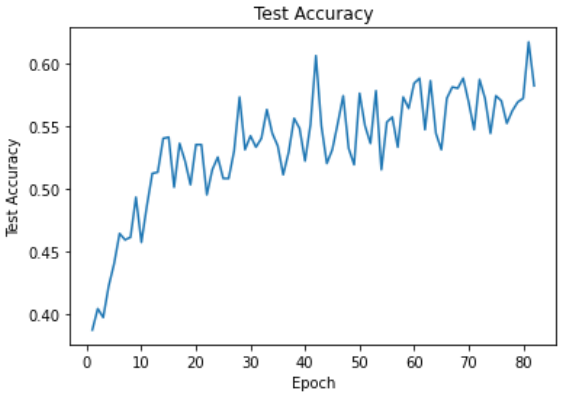
```
Testing ****** Epoch 15000: loss = 1.0962670862674713, acc = 0.5740000000000001, f1 = 0.5590318717328646
Epoch 15050: loss = 1.0285440236330032, accuracy = 0.5900000000000001, f1 = 0.5920626548934531
Epoch 15100: loss = 1.2571942210197449, accuracy = 0.525, f1 = 0.5236607821479609
Epoch 15150: loss = 1.2988402545452118, accuracy = 0.48, f1 = 0.4663921384987884
Epoch 15200: loss = 1.1611955612897873, accuracy = 0.53, f1 = 0.51472994540719
Testing ****** Epoch 15200: loss = 1.0777193397283553, acc = 0.57, f1 = 0.5597441581011494
Epoch 15250: loss = 1.1885131001472473, accuracy = 0.49, f1 = 0.4841507288720911
Epoch 15300: loss = 0.9147228598594666, accuracy = 0.6950000000000001, f1 = 0.6918671076085937
Epoch 15350: loss = 1.0772250145673752, accuracy = 0.56, f1 = 0.532630840287925
Epoch 15400: loss = 1.233194500207901, accuracy = 0.52, f1 = 0.5105340913218488
Testing ****** Epoch 15400: loss = 1.1289449840784074, acc = 0.552, f1 = 0.5373088969050185
Epoch 15450: loss = 0.9027569442987442, accuracy = 0.66, f1 = 0.6414392012431116
Epoch 15500: loss = 1.0805270075798035, accuracy = 0.5850000000000001, f1 = 0.5649671206485044
Epoch 15550: loss = 1.0607622265815735, accuracy = 0.5650000000000001, f1 = 0.5382702648341025
Epoch 15600: loss = 1.0646757632493973, accuracy = 0.5800000000000001, f1 = 0.5629543779736523
Testing ****** Epoch 15600: loss = 1.077762523293495, acc = 0.562, f1 = 0.5512493150373753
Epoch 15650: loss = 1.0030998587608337, accuracy = 0.5800000000000001, f1 = 0.567864617173407
Epoch 15700: loss = 1.011729046702385, accuracy = 0.57, f1 = 0.5492004481574073
Epoch 15750: loss = 1.0528330206871033, accuracy = 0.5950000000000001, f1 = 0.5753145530939648
Epoch 15800: loss = 1.0538006722927094, accuracy = 0.595, f1 = 0.582767641027703
Testing ****** Epoch 15800: loss = 1.1155976235866547, acc = 0.5689999999999998, f1 = 0.5574688339818101
Epoch 15850: loss = 1.1403117328882217, accuracy = 0.5800000000000001, f1 = 0.5717077832519322
Epoch 15900: loss = 0.8872643858194351, accuracy = 0.65, f1 = 0.6361526078055895
Epoch 15950: loss = 0.8733322471380234, accuracy = 0.655, f1 = 0.6458076896107587
Epoch 16000: loss = 1.0291490703821182, accuracy = 0.605, f1 = 0.5879012573963779
Testing ****** Epoch 16000: loss = 1.066386303305626, acc = 0.572, f1 = 0.5573525814232274
Epoch 16050: loss = 1.0719011574983597, accuracy = 0.61, f1 = 0.6078495522402845
Epoch 16100: loss = 0.9719695746898651, accuracy = 0.645, f1 = 0.6296740767832099
Epoch 16150: loss = 0.9838162958621979, accuracy = 0.575, f1 = 0.5568128271572881
Epoch 16200: loss = 1.1097894608974457, accuracy = 0.535, f1 = 0.5201000195120534
Testing ****** Epoch 16200: loss = 0.9945916563272477, acc = 0.6169999999999999, f1 = 0.6042854936014228
Epoch 16250: loss = 1.142640084028244, accuracy = 0.495, f1 = 0.4798621628890844
Epoch 16300: loss = 1.2364739775657654, accuracy = 0.525, f1 = 0.5154874709054276
Epoch 16350: loss = 1.1306634098291397, accuracy = 0.5750000000000001, f1 = 0.5689406970402057
Epoch 16400: loss = 1.1485795825719833, accuracy = 0.555, f1 = 0.5401560202148101
Testing ****** Epoch 16400: loss = 1.0808895885944367, acc = 0.5820000000000001, f1 = 0.5701030811752175
Epoch 16450: loss = 1.0563956052064896, accuracy = 0.5850000000000001, f1 = 0.5704773924419906
```


Train Accuracy
Train Accuracy
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2500
5000
7500
10000
12500
15000


Train Loss
Train Loss
Epoch
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0
2500
5000
7500
10000
12500
15000

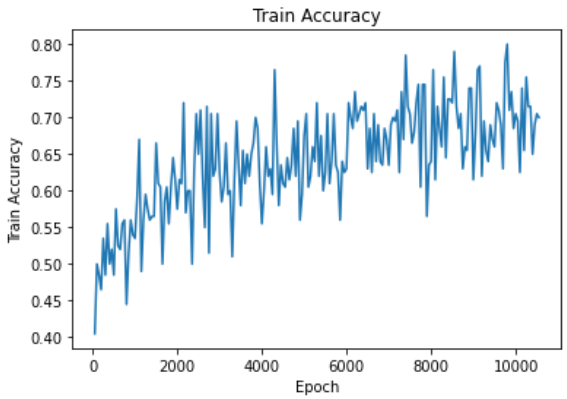

Train F1-Score
Train F1-Score
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
2500
5000
7500
10000
12500
15000

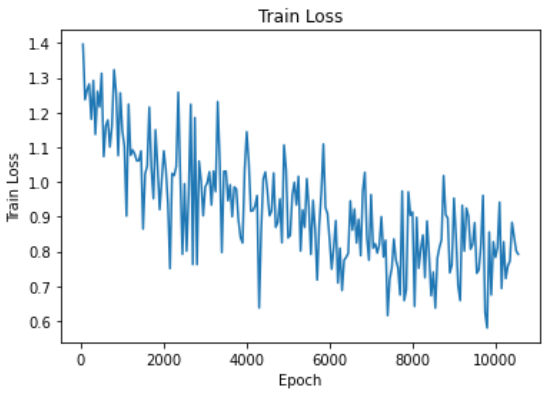

Test Loss
Test Loss
Epoch
1.4
1.3
1.2
1.1
1.0
0
10
20
30
40
50
60
70
80

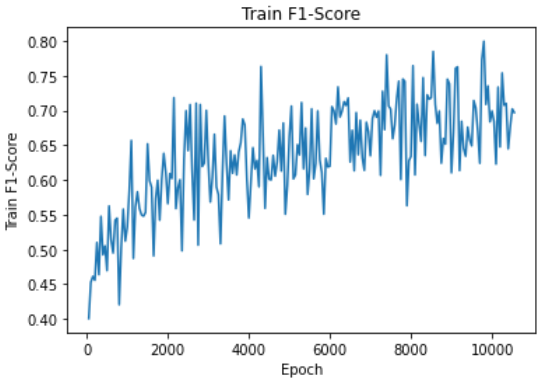

Test Accuracy
Test Accuracy
Epoch
0.60
0.55
0.50
0.45
0.40
0
10
20
30
40
50
60
70
80

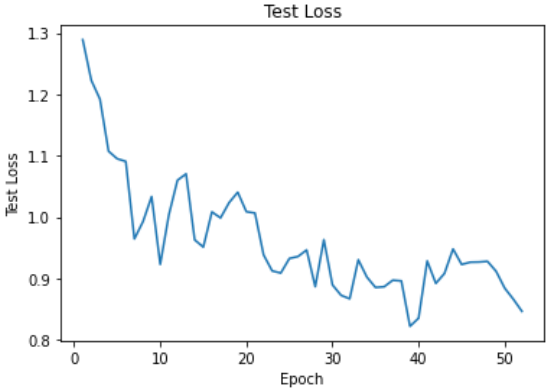
5 way, 5 shot:

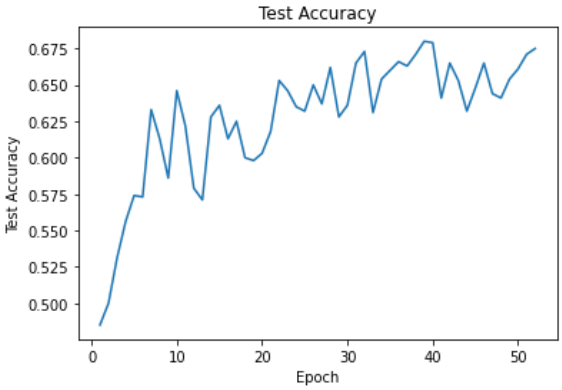
```
Testing ****** Epoch 9000: loss = 0.9230459541082382, acc = 0.648, f1 = 0.6422627302160953
Epoch 9050: loss = 0.840704545378685, accuracy = 0.6799999999999999, f1 = 0.6686613640917046
Epoch 9100: loss = 0.7012812420725822, accuracy = 0.765, f1 = 0.7604527696245962
Epoch 9150: loss = 0.6583801507949829, accuracy = 0.77, f1 = 0.762579340538016
Epoch 9200: loss = 0.9324402213096619, accuracy = 0.62, f1 = 0.613513209707987
Testing ****** Epoch 9200: loss = 0.9267538547515869, acc = 0.6649999999999999, f1 = 0.6595486681879266
Epoch 9250: loss = 0.8010314032435417, accuracy = 0.695, f1 = 0.6842420090939407
Epoch 9300: loss = 0.9243112504482269, accuracy = 0.655, f1 = 0.6451445849472166
Epoch 9350: loss = 0.9011040776968002, accuracy = 0.64, f1 = 0.6341442233227803
Epoch 9400: loss = 0.8062746077775955, accuracy = 0.6900000000000001, f1 = 0.6759123853517446
Testing ****** Epoch 9400: loss = 0.9270664751529694, acc = 0.6439999999999999, f1 = 0.6384030186046137
Epoch 9450: loss = 0.8199055194854736, accuracy = 0.67, f1 = 0.6575325367786669
Epoch 9500: loss = 0.8824223875999451, accuracy = 0.66, f1 = 0.6490086122168165
Epoch 9550: loss = 0.7370332479476929, accuracy = 0.7200000000000001, f1 = 0.7140329434665482
Epoch 9600: loss = 0.7465867400169373, accuracy = 0.7100000000000001, f1 = 0.7035300330866692
Testing ****** Epoch 9600: loss = 0.9281156793236732, acc = 0.6409999999999999, f1 = 0.6362476022459466
Epoch 9650: loss = 0.8158748671412468, accuracy = 0.6900000000000001, f1 = 0.6697050080496372
Epoch 9700: loss = 0.9609595686197281, accuracy = 0.63, f1 = 0.6239494853271943
Epoch 9750: loss = 0.6237370073795319, accuracy = 0.7750000000000001, f1 = 0.7730202589322255
Epoch 9800: loss = 0.5794319286942482, accuracy = 0.7999999999999999, f1 = 0.7994073438265119
Testing ****** Epoch 9800: loss = 0.911924883723259, acc = 0.6539999999999999, f1 = 0.651181023216176
Epoch 9850: loss = 0.8554618507623672, accuracy = 0.71, f1 = 0.7086021370270597
Epoch 9900: loss = 0.6746560707688332, accuracy = 0.7350000000000001, f1 = 0.7351141552245335
Epoch 9950: loss = 0.8273877501487732, accuracy = 0.685, f1 = 0.6835161209619414
Epoch 10000: loss = 0.7827108502388, accuracy = 0.7050000000000001, f1 = 0.6990817871424279
Testing ****** Epoch 10000: loss = 0.8845451503992081, acc = 0.661, f1 = 0.6532803872299161
Epoch 10050: loss = 0.8113124817609787, accuracy = 0.6950000000000001, f1 = 0.6837784463558456
Epoch 10100: loss = 0.9413662701845169, accuracy = 0.625, f1 = 0.6227559643509301
Epoch 10150: loss = 0.6934897750616074, accuracy = 0.74, f1 = 0.7334137162249963
Epoch 10200: loss = 0.827339842915535, accuracy = 0.655, f1 = 0.6473911877596088
Testing ****** Epoch 10200: loss = 0.8665660917758942, acc = 0.671, f1 = 0.6639605578019294
Epoch 10250: loss = 0.7220268473029137, accuracy = 0.755, f1 = 0.753745584219403
Epoch 10300: loss = 0.7583216428756714, accuracy = 0.7150000000000001, f1 = 0.7074234271060896
Epoch 10350: loss = 0.7720264941453934, accuracy = 0.7150000000000001, f1 = 0.7098262337078125
Epoch 10400: loss = 0.8833562731742859, accuracy = 0.65, f1 = 0.6447351533915002
Testing ****** Epoch 10400: loss = 0.8468244969844818, acc = 0.6749999999999999, f1 = 0.6699881576313935
Epoch 10450: loss = 0.8429587036371231, accuracy = 0.69, f1 = 0.6763244865729385
Epoch 10500: loss = 0.8017771095037746, accuracy = 0.7050000000000001, f1 = 0.702111726945823
Epoch 10550: loss = 0.7917846590280533, accuracy = 0.7, f1 = 0.6968518322660873
```











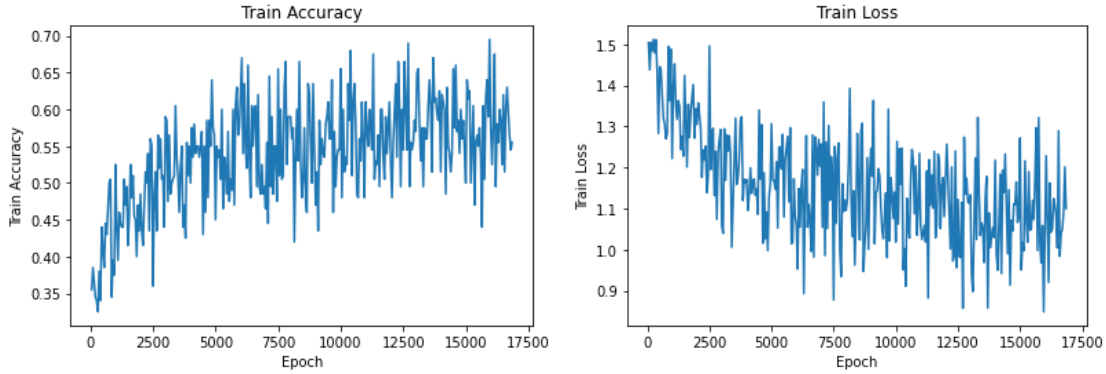
همان گونه که انتظار می رفت عملکرد مدل بهبود یافته است. چون مدل در شرایط متفاوتی آموزش دیده و تصاویر گوناگونی در هر iteration دیده است. دقت مدل روی داده آزمون در اشات از ۳۷٪ به ۵۸٪ و در ۵شات از ۶۱٪ به ۷۰٪ رسیده است (البته به طور دقیق نمی توانیم مقایسه کنیم زیرا در این بخش مدل ها بدلیل کمبود منبع GPU در ۲۰هزار اپیاک آموزش ندیده اند).

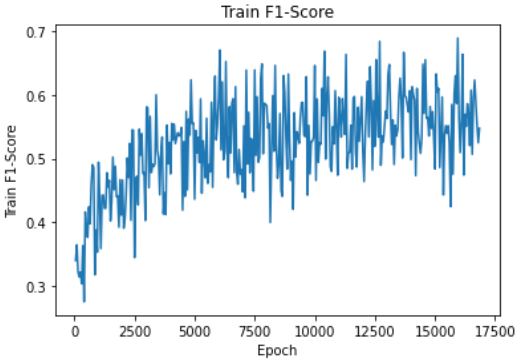
الگوریتم FOMAML

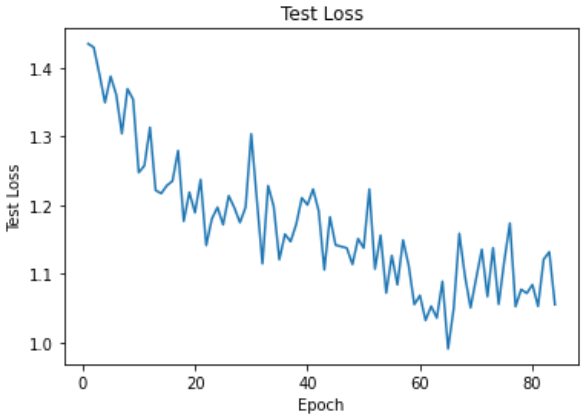
در اینجا نیز مشابه بخش قبل تابع task_batch_creator را با شات های بیشتری صدا می زنیم. نتایج به شرح زیر است:

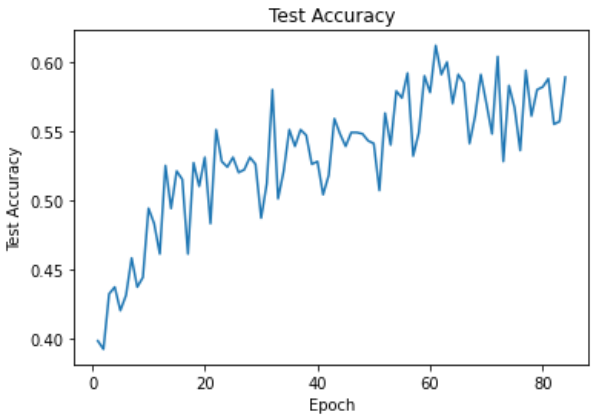
5 way, 1 shot:

```
Epoch 15350: loss = 1.1896190792322159, accuracy = 0.4700000000000003, f1 = 0.44285328920044226
Epoch 15400: loss = 1.0469628721475601, accuracy = 0.555, f1 = 0.5367021817906108
Testing ****** Epoch 15400: loss = 1.0527143955230713, acc = 0.594, f1 = 0.5859743756059478
Epoch 15450: loss = 1.050166666507721, accuracy = 0.565, f1 = 0.5516645735921547
Epoch 15500: loss = 1.119417816400528, accuracy = 0.55, f1 = 0.5395428170227942
Epoch 15550: loss = 1.073079138994217, accuracy = 0.575, f1 = 0.5515526591907666
Epoch 15600: loss = 1.168222188949585, accuracy = 0.535, f1 = 0.5237912745709892
Testing ****** Epoch 15600: loss = 1.077498835325241, acc = 0.561, f1 = 0.5486663087303325
Epoch 15650: loss = 1.296441525220871, accuracy = 0.43999999999999995, f1 = 0.4245210602577004
Epoch 15700: loss = 0.9976627975702286, accuracy = 0.6050000000000001, f1 = 0.5849668148665321
Epoch 15750: loss = 1.3212588727474213, accuracy = 0.505, f1 = 0.47548514001142217
Epoch 15800: loss = 1.0163294970989227, accuracy = 0.615, f1 = 0.6010528661225255
Testing ****** Epoch 15800: loss = 1.0720026433467864, acc = 0.5799999999999998, f1 = 0.5686032825963165
Epoch 15850: loss = 0.967360332608223, accuracy = 0.64, f1 = 0.6301581551179412
Epoch 15900: loss = 1.0584941357374191, accuracy = 0.5900000000000001, f1 = 0.5866856039968167
Epoch 15950: loss = 0.8483888655900955, accuracy = 0.695, f1 = 0.6894143667313677
Epoch 16000: loss = 1.0472166389226913, accuracy = 0.5750000000000001, f1 = 0.5632798590223882
Testing ****** Epoch 16000: loss = 1.0844124495983123, acc = 0.5820000000000001, f1 = 0.5726221366778776
Epoch 16050: loss = 1.2280979454517365, accuracy = 0.525, f1 = 0.5093110647522412
Epoch 16100: loss = 1.0746853798627853, accuracy = 0.58, f1 = 0.5575968128616869
Epoch 16150: loss = 0.9196398854255676, accuracy = 0.675, f1 = 0.6638482373394542
Epoch 16200: loss = 1.1619446575641632, accuracy = 0.495, f1 = 0.47395906459643167
Testing ****** Epoch 16200: loss = 1.0531425267457961, acc = 0.5879999999999999, f1 = 0.5765862813716697
Epoch 16250: loss = 1.0423806458711624, accuracy = 0.58, f1 = 0.5695285169056874
Epoch 16300: loss = 1.0512426942586899, accuracy = 0.5549999999999999, f1 = 0.5505339752117582
Epoch 16350: loss = 1.1238996386528015, accuracy = 0.6, f1 = 0.5862324489012132
Epoch 16400: loss = 1.1079187840223312, accuracy = 0.5850000000000001, f1 = 0.5707939111635436
Testing ****** Epoch 16400: loss = 1.121271911263466, acc = 0.555, f1 = 0.5369821494052252
Epoch 16450: loss = 1.0832119286060333, accuracy = 0.525, f1 = 0.5201767227104419
Epoch 16500: loss = 1.0039141178131104, accuracy = 0.62, f1 = 0.6073538542491296
Epoch 16550: loss = 1.289114847779274, accuracy = 0.515, f1 = 0.5067457670767336
Epoch 16600: loss = 0.9834393858909607, accuracy = 0.6, f1 = 0.5947916548386329
Testing ****** Epoch 16600: loss = 1.1320373356342315, acc = 0.557, f1 = 0.5459684460840607
Epoch 16650: loss = 1.03989939391613, accuracy = 0.63, f1 = 0.6233269875455902
Epoch 16700: loss = 1.0498437136411667, accuracy = 0.5950000000000001, f1 = 0.5862841540124822
Epoch 16750: loss = 1.0917036831378937, accuracy = 0.5650000000000001, f1 = 0.5532534220501981
Epoch 16800: loss = 1.201134443283081, accuracy = 0.545, f1 = 0.5247720695272182
Testing ****** Epoch 16800: loss = 1.055469247698784, acc = 0.5890000000000001, f1 = 0.5767650891442654
Epoch 16850: loss = 1.0996410995721817, accuracy = 0.555, f1 = 0.5474915461365186
```



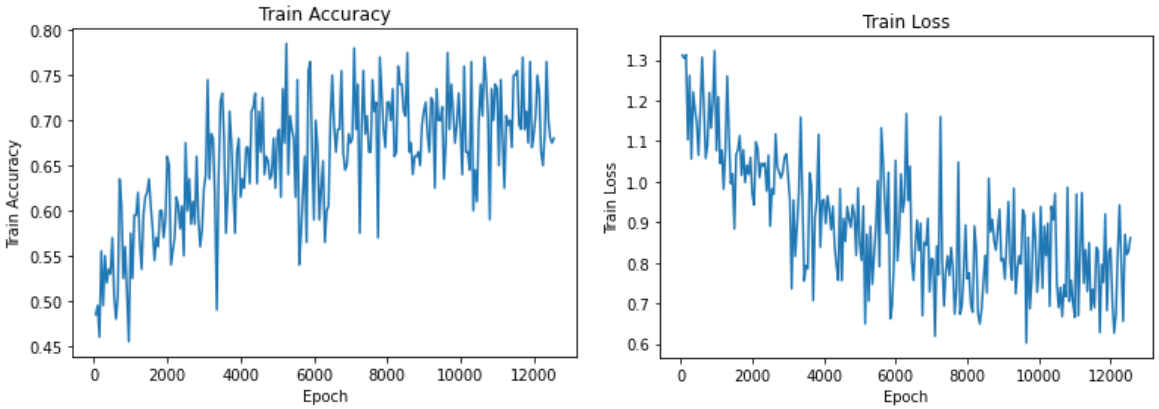

Train F1-Score
Train F1-Score
0.7
0.6
0.5
0.4
0.3
0
2500
5000
7500
10000
12500
15000
17500
Epoch

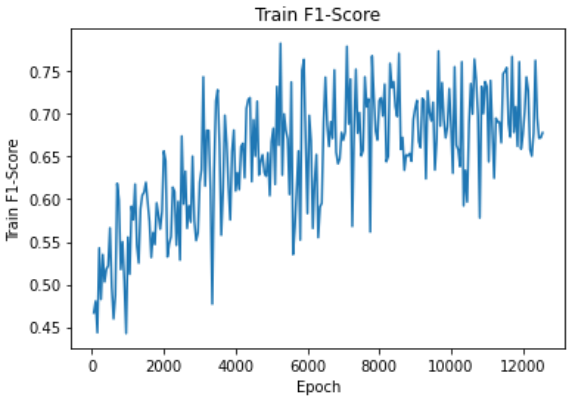

Test Loss
Test Loss
1.4
1.3
1.2
1.1
1.0
0
20
40
60
80
Epoch

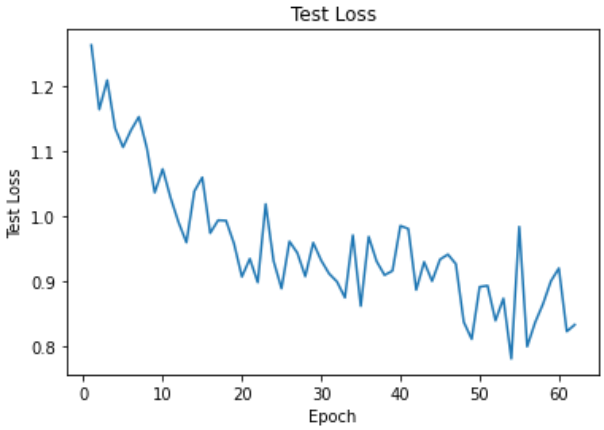

Test Accuracy
Test Accuracy
0.60
0.55
0.50
0.45
0.40
0
20
40
60
80
Epoch

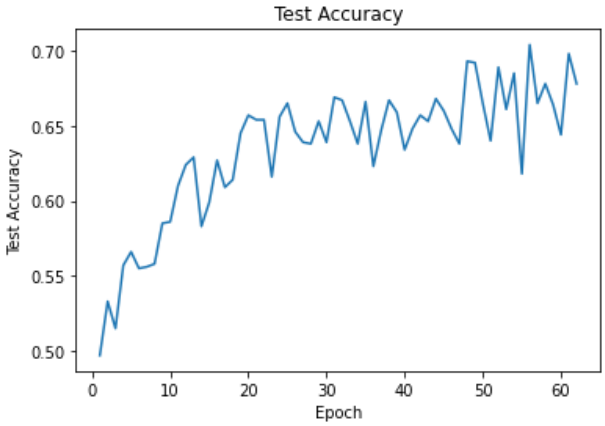
5 way, 5 shot:

```
Epoch 11000: loss = 0.0001925315850954, accuracy = 0.755, f1 = 0.7518445295057501
Testing ****** Epoch 11000: loss = 0.983582866191864, acc = 0.6179999999999998, f1 = 0.6121889575516241
Epoch 11050: loss = 0.9690072387456894, accuracy = 0.65, f1 = 0.6439384887857091
Epoch 11100: loss = 0.669749066233635, accuracy = 0.745, f1 = 0.7391337883187392
Epoch 11150: loss = 0.8276688978075981, accuracy = 0.68, f1 = 0.6806185006420571
Epoch 11200: loss = 0.9731268733739853, accuracy = 0.625, f1 = 0.624719368705807
Testing ****** Epoch 11200: loss = 0.7987016558647155, acc = 0.704, f1 = 0.7006402270012784
Epoch 11250: loss = 0.7498427033424377, accuracy = 0.7050000000000001, f1 = 0.6950036356682607
Epoch 11300: loss = 0.8317805826663971, accuracy = 0.6950000000000001, f1 = 0.6903873611506873
Epoch 11350: loss = 0.7295460850000381, accuracy = 0.7, f1 = 0.6903344165294629
Epoch 11400: loss = 0.8497441560029984, accuracy = 0.67, f1 = 0.666271661531959
Testing ****** Epoch 11400: loss = 0.835789579153061, acc = 0.6649999999999999, f1 = 0.6621216980810128
Epoch 11450: loss = 0.6836225837469101, accuracy = 0.75, f1 = 0.747044106478317
Epoch 11500: loss = 0.7349882423877716, accuracy = 0.75, f1 = 0.7499576320091867
Epoch 11550: loss = 0.6899412497878075, accuracy = 0.755, f1 = 0.7543037502012802
Epoch 11600: loss = 0.8379832059144974, accuracy = 0.6950000000000001, f1 = 0.6885171261487051
Testing ****** Epoch 11600: loss = 0.8644593387842179, acc = 0.6779999999999999, f1 = 0.676696412059994
Epoch 11650: loss = 0.8300089836120605, accuracy = 0.69, f1 = 0.6729614028192235
Epoch 11700: loss = 0.6284040287137032, accuracy = 0.7699999999999999, f1 = 0.767316951848349
Epoch 11750: loss = 0.7962169796228409, accuracy = 0.6900000000000001, f1 = 0.6784047015935561
Epoch 11800: loss = 0.7521575093269348, accuracy = 0.71, f1 = 0.708562940570445
Testing ****** Epoch 11800: loss = 0.8997363030910492, acc = 0.6639999999999999, f1 = 0.6584388911316021
Epoch 11850: loss = 0.9204892814159393, accuracy = 0.675, f1 = 0.6621605622694262
Epoch 11900: loss = 0.6824160367250443, accuracy = 0.765, f1 = 0.7610822811892942
Epoch 11950: loss = 0.8276644200086594, accuracy = 0.6699999999999999, f1 = 0.6589328434723171
Epoch 12000: loss = 0.8358007818460464, accuracy = 0.685, f1 = 0.6753282959879743
Testing ****** Epoch 12000: loss = 0.9194756239652634, acc = 0.6439999999999999, f1 = 0.6399201937015437
Epoch 12050: loss = 0.7019744962453842, accuracy = 0.7050000000000001, f1 = 0.7020544648065176
Epoch 12100: loss = 0.6274756193161011, accuracy = 0.75, f1 = 0.7433310879306303
Epoch 12150: loss = 0.6736211106181145, accuracy = 0.73, f1 = 0.7271732914787161
Epoch 12200: loss = 0.8249639719724655, accuracy = 0.665, f1 = 0.6582161285566859
Testing ****** Epoch 12200: loss = 0.8219260275363922, acc = 0.6980000000000001, f1 = 0.6929723698560555
Epoch 12250: loss = 0.9423083513975143, accuracy = 0.65, f1 = 0.650730850512315
Epoch 12300: loss = 0.8019759058952332, accuracy = 0.6849999999999999, f1 = 0.6778754197747332
Epoch 12350: loss = 0.6559267267584801, accuracy = 0.7649999999999999, f1 = 0.762709647967085
Epoch 12400: loss = 0.8701556026935577, accuracy = 0.7000000000000001, f1 = 0.695735380116959
Testing ****** Epoch 12400: loss = 0.8323712617158889, acc = 0.678, f1 = 0.6747012193061124
Epoch 12450: loss = 0.8206850439310074, accuracy = 0.6799999999999999, f1 = 0.6715184622701448
Epoch 12500: loss = 0.8281259387731552, accuracy = 0.675, f1 = 0.672166359089229
Epoch 12550: loss = 0.8613869249820709, accuracy = 0.6799999999999999, f1 = 0.6779879511613257
```









مطابق انتظار، دقت مدل روی داده آزمون در اشات از ۴۴٪ به ۵۵٪ و در ۵شات از ۵۶٪ به ۶۷٪ رسیده است (البته به طور دقیق نمی توانیم مقایسه کنیم زیرا در این بخش مدل ها بدلیل کمبود منبع GPU در ۲۰هزار ایپاک آموزش ندیده اند).

قسمت دوم

برای این کار باید از روش های learning rate scheduling استفاده کنیم. در ابتدای آموزش نرخ یادگیری زیاد باشد تا مدل فضای حالت بیشتری را ببیند، در مینیمم های محلی گیر نکند و سرعت آموزش بیشتر باشد. با پیش رفتن فرایند آموزش میزان نرخ یادگیری را کاهش می دهیم. برای این کار یک تابع lr_scheduler تعریف می کنیم که پس از هر بار آموزش نرخ یادگیری را کاهش می دهد.

```
decay_rate = 0.001

def lr_scheduler(lr, epoch, decay):
    # LearningRate = LearningRate * 1/(1 + decay * epoch)
    return lr * 1/(1 + decay * epoch)
```

```
for epoch in range(1, outer_epochs + 1):
    # create batches of tasks
    tasks = task_batch_creator(C_train=C_train, n_way=nway, N=support_samples, Q=query_samples, meta_batch_size=meta_train_batch)

    inner_model_hist = {}
    inner_model_hist['grads'] = []
    inner_model_hist['losses'] = []
    inner_model_hist['accuracy'] = []
    inner_model_hist['f1'] = []

    for task_idx in range(len(tasks)):
        inner_model = copy.deepcopy(meta_model)
        inner_model.to(device)

        # train the inner model on task
        # returns (loss, grads, accuracy, f1_score)
        tmp = train_inner_model(inner_model, tasks[task_idx], inner_epochs_train, device, inner_lr)

        # save inner model history for training meta model
        inner_model_hist['losses'].append(tmp[0])
        inner_model_hist['grads'].append(tmp[1])
        inner_model_hist['accuracy'].append(tmp[2])
        inner_model_hist['f1'].append(tmp[3])

    # train the model
    tmp = train_meta_model(meta_model, outer_lr, meta_train_batch, device, inner_model_hist['grads'], inner_model_hist['losses'], inner_model_hist['accuracy'], inner
    meta_model, meta_train_loss, meta_train_accuracy, meta_train_f1 = tmp

    # learning rate scheduling
    outer_lr = lr_scheduler(outer_lr, outer_epochs, decay_rate)

```