활동 순서

1. 모델 수정

목표는 run을 통해 나온 result의 정확도 향상이므로 이를 위해 저는 우선 모델의 크기를 키워봤습니다.

BaseModel에 resnet50을 이용해보았고, 결과는 모델 자체의 용량이 55M를 넘겨 run을 돌려보지 못하고 실패하였습니다.

이후에 resnet18이 44M정도 나오는 것을 확인하고 제한 용량 내에서 효율적일 것이라 판단하여 resnet18을 이용하고자 하였습니다.

이 때 저는 test.py의 역할을 잘못 이해하여 validation용 데이터를 따로 할당해주어 train : valid : test = 6 : 2 : 2로 설정하였습니다.

```
# you can change input size(don't forget to change linear layer!)
custom_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

마지막으로 이미지를 normalize해보기도 하였습니다. 이 값은 사람들이 수백만번의 실험을 하며 찾아낸 가장 이상적인 normalize 값으로 알고 있었고, 어쩌면 도움이 될 것 같아 이를 이용하려 하였으나 validation 정확도가 20가량 감소하는 결과를 보여 제거하였고 그 이유를 찾아보니 사진이 동일한 방식으로 편향되어 있으면 정규화해도 괜찮으나 사진의 형태가 다양한 경우 새로운 normalize가 필요하다는 의견을 찾을 수 있었습니다.

(출처: https://stackoverflow.com/questions/58151507/why-pytorch-officially-use-mean-0-485-0-456-0-406-and-std-0-229-0-224-0-2)

2. Optuna를 통한 파라미터 탐색

다음은 하이퍼 파라미터를 수정해보는 것이었습니다.

우선은 초기에 하이퍼 파라미터가 좋은 성능을 내는지 파악하기 위해 예를 들어 learning rate를 0.1로 설정하는 등의 결과를 확인했으나 validation의 결과가 중구난방이었으며 기본 코드의 결과보다 validation 정확도가 낮은 것으로 보여 epoch이 5 이상으로 진행될 때 오히려 validation이

감소한다는 점을 확인한 것을 제외하면 얻은 정보가 없어 진행에 차질이 있을 것으로 판단하였고, optuna라는 라이브러리를 이용하게 되었습니다.

```
"""
TODO: You can change the hyperparameters as you wish.
      (e.g. change epochs etc.)
"""

# hyperparameters
args.epochs = 7

# custom model
model = resnet18(weights=ResNet18_Weights)

# you have to change num_classes to 10
num_features = model.fc.in_features # edited
model.fc = nn.Linear(num_features, num_classes) # edited
model.to(device)
print(model)

# Training The Model
study = optuna.create_study(direction="maximize")
study.optimize(lambda trial: objective(model, trial, args), n_trials=100, callbacks=[save_intermediate_results])
```

Optuna는 하이퍼파라미터를 베이지안 추론을 통해 찾아나가는 방식으로, epoch을 제외한 나머지 하이퍼 파라미터 전부를 optuna를 이용해 탐색하고자 하였습니다.

결과 즉, validation을 maximize하는 direction으로 create_study 하였고, optimize의 파라미터는 순서대로 다음과 같습니다.

Objective

```
def initializer(model, learning_rate, weight_decay, gamma):
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
    scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=gamma)
    return model, optimizer, scheduler


def objective(model, trial: optuna.Trial, args):
    # 모델마다 최적의 하이퍼파라미터값은 다르다.
    learning_rate = trial.suggest_float('learning_rate', low=5e-5, high=0.1, log=True)
    weight_decay = trial.suggest_float('weight_decay', low=4e-5, high=0.1, log=True)
    train_batch_size = trial.suggest_categorical("per_device_train_batch_size", [32, 64, 128, 256, 512])
    gamma = trial.suggest_float('gamma', 0, 1)

    # Make Data loader and Model
    args.batch_size = train_batch_size
    train_loader, valid_loader, _ = make_data_loader(args)

    model, optimizer, scheduler = initializer(model, learning_rate, weight_decay, gamma)
    epoch_valid_acc = train(model, optimizer, scheduler, train_loader, valid_loader)

    gc.collect()   # cuda memory 부족 방지
    return max(epoch_valid_acc)   # 검증데이터중 가장 큰 값 return
```

Objective의 내부에는 optuna.Trial.suggest_*함수들이 선언되어 있으며 이를 통해 각 변수들의 저장될 이름, 최저 범위, 최고 범위, log 여부, 범위 등을 설정할 수 있습니다.

예를 들어 learning_rate는 실수의 범위로 5e-5~0.1까지의 범위중 log 스케일로 범위 내의 값을 추천하도록 해줍니다. 또한 Train_batch_size는 주어진 범위 내에서 하나가 추천됩니다. 마지막으로 gamma는 0과 1 내에서 uniform하게 실수가 추천됩니다. 정해진 하이퍼 파라미터들은 train을 진행하며 에폭당 valid_acc를 저장한 배열을 반환하고, 이중 가장 큰 값을 반환합니다.

Initializer 함수는 추천받은 인자들을 바탕으로 optimizer과 scheduler을 정의해 새롭게 모델을 학습할 수 있게 해줍니다.

N_trials

Optuna가 실행되는 횟수를 지정합니다. 해당 코드에서 n_trials가 100인 것은 하이퍼 파라미터 탐색을 100회 진행하도록 함을 의미합니다. 따라서 1번의 탐색동안 앞에서 정의한 7 epoch까지 확인하므로 전체 epoch은 700임을 계산할 수 있습니다.

Callback

Callback 함수는 매 탐색이 끝날 때마다 지정된 함수를 실행합니다. 현재 코드에서 쓰인 코드는 callbacks=[save_intermediate_results]이며, 이는 매 탐색 후 save_intermediate_results 함수를 실행하라는 의미입니다.

```
def save_intermediate_results(study, trial: optuna.Trial):
    best = study.best_params
    df_best = pd.DataFrame([best])
    df_trials = study.trials_dataframe()
    df_best.to_csv('current_best.csv', index=False)
    df_trials.to_csv('optuna_trials.csv', index=False)
```

save_intermediate_results 함수는 현재까지 optuna를 통해 찾은 결과중 최고의 파라미터를 찾아 best로 저장하고, 모든 로그를 df_trials에 저장한 후 이를 csv 타입으로 반환하여줍니다.

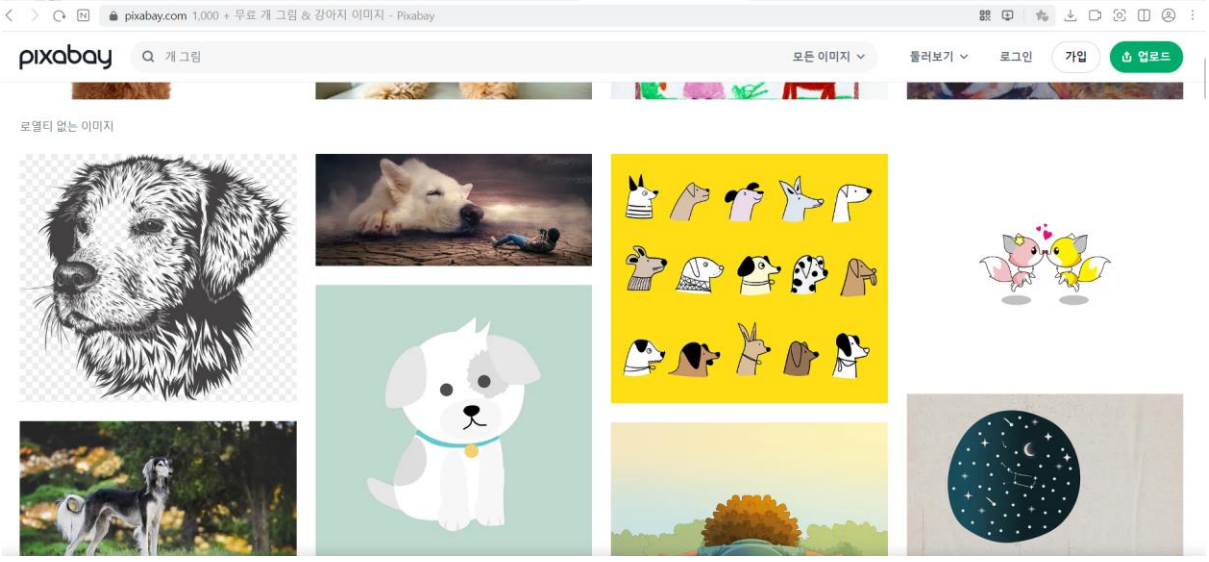
이를 통해 100번동안 실행하다 중간에 중단되어 메모리가 날라가더라도 하이퍼파라미터와 결과를 확인할 수 있도록 하였습니다.

number	value	datetime_start	datetime_complete	duration	gamma	lr	batch_size	weight_decay	state
0	0.7470869149952245	2023-11-29 18:48:26.744123	2023-11-29 18:59:18.196743	0 days 00:10:51.452620	0.8247036977594391	0.00010255908153643282	256	0.0004986383586760189	COMPLETE
1	0.8022922636103151	2023-11-29 18:59:18.206744	2023-11-29 19:10:39.690827	0 days 00:11:21.484083	0.11482974958346504	0.0014015948308571788	128	0.0003488927746184961 4	COMPLETE
2	0.21795606494746897	2023-11-29 19:10:39.696829	2023-11-29 20:09:01.437409	0 days 00:58:21.740580	0.54410139565941 91	0.012687495009278155	512	0.0001260287335686427	COMPLETE
3	0.21623686723973257	2023-11-29 20:09:01.437409	2023-11-29 20:18:16.172610	0 days 00:09:14.735201	0.39410339422888168	0.0003274093252122271	32	7.4063924986781 76e-05	COMPLETE
4	0.210697230181470 88	2023-11-29 20:18:16.174114	2023-11-29 21:00:08.755800	0 days 00:41:52.581686	0.16062528996927 87	0.000104955314286764 9	128	6.446306620685052e-05	COMPLETE
5	0.1933142311365807	2023-11-29 21:00:08.755800	2023-11-29 21:09:16.740700	0 days 00:09:07.984900	0.060767334964559216	0.007329518551283494	32	0.0008382151811810741	COMPLETE
6	0.19522445081184336	2023-11-29 21:09:16.745700	2023-11-29 22:07:32.111896	0 days 00:58:15.366196	0.5818184031819159	0.0004297564743016767 6	512	0.0001688807861869517 4	COMPLETE
7	0.7921680993314231	2023-11-29 22:07:32.111896	2023-11-29 23:05:43.500390	0 days 00:58:11.388494	0.9885442323449 97	0.000724730321761 5042	512	0.0549104339648 2422	COMPLETE
8	0.32951289398280803	2023-11-29 23:05:43.516764	2023-11-29 23:18:35.050124	0 days 00:12:51.533360	0.9798781489136421	0.033473097342679 87	256	0.0001042862909022908	COMPLETE
9	0.18242597898758356	2023-11-29 23:18:35.055126	2023-11-29 23:32:00.511461	0 days 00:13:25.456335	0.19148469363311083	0.0101683568911167 43	256	8.561828656910504e-05	COMPLETE
10	0.2082139446036294	2023-11-29 23:32:00.516463	2023-11-30 00:05:47.391811	0 days 00:33:46.875348	0.02034185879271222	0.0021447289813602 79	128	0.0035507025979992566	COMPLETE
11	0.20687679083094557	2023-11-30 00:05:47.398814	2023-11-30 00:14:46.425705	0 days 00:08:59.026891	0.7486688493347041	0.0012690081144521778	64	0.0728515153530966 4	COMPLETE
12	0.2099331423113658	2023-11-30 00:14:46.431706	2023-11-30 00:56:31.699380	0 days 00:41:45.267674	0.3617056692173 55	0.000914767482783853 7	128	0.0864500870688639 2	COMPLETE
13	0.2674307545367717 3	2023-11-30 00:56:31.705385	2023-11-30 01:56:15.772134	0 days 00:59:44.066749	0.7215780522511609	0.0037254881711141249	512	0.0068194328285677 28	COMPLETE
14	0.33791786055396 37	2023-11-30 01:56:15.778136	2023-11-30 02:05:16.396110	0 days 00:09:00.617974	0.9917966196878673	0.000457390889561638	64	0.0178700545906134 26	COMPLETE
15	0.26762176504299797	2023-11-30 02:05:16.402112	2023-11-30 03:05:40.604016	0 days 01:00:24.201904	0.3653772248172205 7	0.0029400904853927037	512	0.0010766819252536 35	COMPLETE
16	0.2968481375358166	2023-11-30 03:05:40.609018	2023-11-30 03:35:17.214362	0 days 00:29:36.605344	0.6317518469098931	5.383796705623375e-05	128	0.0003103994816833402	COMPLETE
17	0.18319006685768863	2023-11-30 03:35:17.219365	2023-11-30 04:18:38.809541	0 days 00:43:21.590176	0.4781782395509069	0.0637202167325570 4	128	0.0023429065716610 95	COMPLETE
18	0.18968481375358165	2023-11-30 04:18:38.819545	2023-11-30 05:16:52.876665	0 days 00:58:14.057120	0.8745137344400445	0.0011791887622042176	512	0.0281045357293851 95	COMPLETE
19	0.18872970391595034	2023-11-30 05:16:52.882667	2023-11-30 05:26:11.978043	0 days 00:09:19.095376	0.2759060302696359	0.004862729594656104	32	0.0063728591387605 57	COMPLETE
20	0.1894937917860554	2023-11-30 05:26:11.978043	2023-11-30 05:35:03.622694	0 days 00:08:51.644651	0.6869808803057952	0.0019456900370174643	64	0.0011008273625882433	COMPLETE
21	0.18185291308500479	2023-11-30 05:35:03.628696	2023-11-30 05:47:49.351287	0 days 00:12:45.722591	0.8416401966284056	0.0001941198428847 1247	256	0.0003406259488106037	COMPLETE
22	0.21700095510983763	2023-11-30 05:47:49.351287	2023-11-30 06:00:35.846283	0 days 00:12:46.494996	0.873653163478486	0.0006277760526694912	256	0.0004500326221689824 6	COMPLETE
23	0.2259789875835721	2023-11-30 06:00:35.852286	2023-11-30 06:13:23.494641	0 days 00:12:47.642355	0.7989429618847382	0.0001764282745545848 4	256	0.0006484692407447186	COMPLETE
24	0.22865329512893984	2023-11-30 06:13:23.510995	2023-11-30 06:26:11.733905	0 days 00:12:48.222910	0.9038883006301621	0.0002595921366447546	256	0.0018204367197576495	COMPLETE
25	0.23476599808978033	2023-11-30 06:26:11.739907	2023-11-30 07:24:23.692780	0 days 00:58:11.952873	0.7741721884808631	0.0007029872330296665	512	0.0002280254139478422 6	COMPLETE
26	0.24584527220630373	2023-11-30 07:24:23.692780	2023-11-30 07:53:21.449326	0 days 00:28:57.756546	0.6600937115529724	0.0001066293203710004 4	128	0.0004138136535184724 3	COMPLETE
27	0.2565425023877746	2023-11-30 07:53:21.458038	2023-11-30 08:06:10.697309	0 days 00:12:49.239271	0.9406429431249287	0.0014311028825487351	256	0.0002331727374677887 6	COMPLETE
28	0.5965616045845272	2023-11-30 08:06:10.705312	2023-11-30 08:38:40.075586	0 days 00:32:29.370274	0.9991457232475455	0.0005931195609917113	128	4.448119290386562e-05	COMPLETE
29	0.6515979751671443	2023-11-30 08:38:40.084590	2023-11-30 09:36:47.597206	0 days 00:58:07.512616	0.8187597598457288	0.0009187100289305365	512	0.0001670943069449 0735	COMPLETE
30	0.6737344794651385	2023-11-30 09:36:47.605210	2023-11-30 10:36:05.815977	0 days 00:59:18.210767	0.9112369728509261	0.0003304407985417811 4	512	0.0005758903277264353	COMPLETE
31	0.6901623686723973	2023-11-30 10:36:05.821980	2023-11-30 11:39:03.041449	0 days 01:02:57.219469	0.9465563302000111	0.0003633064758272 76	512	0.0007501265752867223	COMPLETE
32	0.6718242597898758	2023-11-30 11:39:03.048452	2023-11-30 12:38:13.231336	0 days 00:59:10.182884	0.9279016484676594	0.0004087036131369874	512	0.0008246285536 8896	COMPLETE
33	0.6819484240687679	2023-11-30 12:38:13.231336	2023-11-30 13:37:13.197522	0 days 00:58:59.966186	0.8121177074274963	0.0002271400417357 8865	512	0.0012387035763707593	COMPLETE
34	0.6760267430754536	2023-11-30 13:37:13.203524	2023-11-30 13:46:46.799848	0 days 00:09:33.596324	0.4831487613666 5463	0.0001117644668 68676629	32	0.0005747004930564388	COMPLETE
35	0.6819484240687679	2023-11-30 13:46:46.806850	2023-11-30 14:45:26.842762	0 days 00:58:40.035912	0.5465316367677535	5.128175977949463 5e-05	512	0.0001408607911430 1916	COMPLETE
36	0.6647564469914404	2023-11-30 14:45:26.849766	2023-11-30 14:54:53.280260	0 days 00:09:26.430494	0.9561168576012652	0.0002815661946075393	32	0.0003375428743887926 3	COMPLETE
37	0.6830945558739255	2023-11-30 14:54:53.288262	2023-11-30 15:53:41.312136	0 days 00:58:48.023874	0.8560193059918507	0.0001446066842916416	512	0.0016579694023132541	COMPLETE
38	0.6494746895893028	2023-11-30 15:53:41.318138	2023-11-30 16:23:32.678293	0 days 00:29:51.360155	0.1115493721532537 6	0.0004262883627463715	128	0.0001132889319500 6666	COMPLETE
39	0.6913085004775549	2023-11-30 16:23:32.684296	2023-11-30 16:35:45.713098	0 days 00:12:13.028802	0.7399513772021058	8.269094292172537e-05	64	0.0008070278315235579	COMPLETE
40	0.6777459407831901	2023-11-30 16:35:45.720100	2023-11-30 16:45:31.283040	0 days 00:09:45.562940	0.7451379583074729	0.0001014165801938706 3	64	0.0002239160294065585 7	COMPLETE
41	0.6897803247373447	2023-11-30 16:45:31.291042	2023-11-30 16:54:47.177639	0 days 00:09:15.886597	0.9071698170840103	7.073181024355274e-05	64	0.0007238442824357215	COMPLETE
42	0.6769818523 13085	2023-11-30 16:54:47.184640	2023-11-30 17:03:59.675452	0 days 00:09:12.490812	0.9518367144325464	0.0001684752265438634	64	0.0005009404405433035	COMPLETE
43	0.6855778414517669	2023-11-30 17:03:59.682454	2023-11-30 17:13:05.932525	0 days 00:09:06.250071	0.7877175584018061	7.509496516751406e-05	64	0.0008216203347201147	COMPLETE
44	0.6748806112702961	2023-11-30 17:13:05.939527	2023-11-30 17:26:10.363319	0 days 00:13:04.423792	0.7088063198231754	0.0002848330186001 2547	256	0.0031952949729521 85	COMPLETE
45	0.679847182425979	2023-11-30 17:26:10.371323	2023-11-30 17:58:49.806447	0 days 00:32:39.435124	0.6168378666105434	0.0001343417129056 9637	128	0.0013383479783462833	COMPLETE
46	0.6697230181470869	2023-11-30 17:58:49.812449	2023-11-30 18:07:54.800168	0 days 00:09:04.987719	0.0128274801113682 78	0.0002271068632249337	64	0.0010245705359085472	COMPLETE
47	0.7159503342884431	2023-11-30 18:07:54.807169	2023-11-30 19:09:43.849473	0 days 01:01:49.042304	0.9835969935130001	0.0005276398013307 74	512	8.795245792037 87e-05	COMPLETE

위의 학습 결과로 얻은 데이터입니다. 코드의 구현 난이도와 시간 문제로 인해 100회를 진행하지 못하였으나 callback을 구현하였기에 데이터를 확보할 수 있었습니다.

3. 데이터 추가

앞서 normalize를 하며 결과가 떨어지는 이유를 찾던 도중 사진의 종류를 확인하게 되었고, 이때 train에 사용되는 데이터와 run에 사용되는 데이터를 확인해본 결과 run에는 train에 사용되는 데이터보다 더 다양한 종류의 사진들이 있는 것들을 확인하고 Pixabay 사이트를 이용해 각 카테고리에 어울리는 사진들을 추가해주어 학습을 진행하여 더 좋은 결과를 얻도록 하였습니다.



4. 모델 생성 코드 구현

Train 함수

```
5. def train(model, optimizer, scheduler, train_loader,
   valid_loader):
6.     """
7.     TODO: Change the training code as you need. (e.g.
   different optimizer, different loss function, etc.)
8.             You can add validation code. -> This will increase
   the accuracy.
9.     """
10.          gc.collect()   # cuda memory 부족 방지
11.
12.          valid_acc_list = []  # 추가: 각 Epoch 의 검증 정확도를
   저장할 리스트
13.
14.          criterion = torch.nn.CrossEntropyLoss()
15.
16.          optimizer = optimizer
17.          scheduler = scheduler
18.
19.          for epoch in range(args.epochs):
20.              train_losses = []
21.              train_acc = 0.0
22.              total=0
23.              print(f"[Epoch {epoch+1} / {args.epochs}]")
24.
25.              model.train()
26.              print('training')
27.              pbar = tqdm(train_loader)
28.              for i, (x, y) in enumerate(pbar):
29.                  image = x.to(args.device)
30.                  label = y.to(args.device)
31.                  optimizer.zero_grad()
32.
33.                  output = model(image)
34.
35.                  label = label.squeeze()
36.                  loss = criterion(output, label)
37.                  loss.backward()
38.                  optimizer.step()
39.                  scheduler.step()
```

```
40.
41.                 train_losses.append(loss.item())
42.                 total += label.size(0)
43.
44.                 train_acc += acc(output, label)
45.
46.             epoch_train_loss = np.mean(train_losses)
47.             epoch_train_acc = train_acc/total
48.
49.             # add valid course.
50.             model.eval()
51.             valid_acc = 0.0
52.             valid_total = 0
53.             with torch.no_grad():
54.                 print('valid calculating')
55.                 pbar = tqdm(valid_loader)
56.                 for i, (x, y) in enumerate(pbar):
57.                     image = x.to(args.device)
58.                     label = y.to(args.device)
59.                     output = model(image)
60.                     valid_total += label.size(0)
61.                     valid_acc += acc(output, label)
62.             epoch_valid_acc = valid_acc/valid_total
63.
64.             valid_acc_list.append(epoch_valid_acc)   # 추가:
    리스트에 검증 정확도 추가
65.
66.             print(f'Epoch {epoch+1}')
67.             print(f'train_loss : {epoch_train_loss}')
68.             print('train_accuracy :
    {:.3f}'.format(epoch_train_acc*100))
69.             print('valid_accuracy :
    {:.3f}'.format(epoch_valid_acc*100))  # Print validation
    accuracy
70.
71.             # 검증 세트의 정확도를 반환합니다.
72.
73.             print("==============================")
74.             print("Save path:", args.save_path)
75.             print('Using Device:', device)
```

```
            print('Number of usable GPUs:',
  torch.cuda.device_count())

            # Print epoch
            print("Epoch:", epoch)
            print("==============================")

        return valid_acc_list
```

main

```
    parser = argparse.ArgumentParser(description='2023 DL Term
Project')
    parser.add_argument('--save-path', default='checkpoints/',
help="Model's state_dict")
    parser.add_argument('--data', default='data/', type=str,
help='data folder')
    args = parser.parse_args()

    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
    args.device = device
    num_classes = 10 # edited

    """
    TODO: You can change the hyperparameters as you wish.
            (e.g. change epochs etc.)
    """

    # hyperparameters
    args.epochs = 7

    # custom model
    model = resnet18(weights=ResNet18_Weights)

    # you have to change num_classes to 10
    num_features = model.fc.in_features # edited
    model.fc = nn.Linear(num_features, num_classes) # edited
    model.to(device)
    print(model)
```

```
    # Training The Model
    study = optuna.create_study(direction="maximize")
    study.optimize(lambda trial: objective(model, trial, args),
n_trials=100, callbacks=[save_intermediate_results])
```

save

```
def save(model, optimizer, scheduler, train_loader):

    criterion = torch.nn.CrossEntropyLoss()

    for epoch in range(args.epochs):

        train_losses = []
        train_acc = 0.0
        total=0
        print(f"[Epoch {epoch+1} / {args.epochs}]")

        model.train()
        pbar = tqdm(train_loader)
        for i, (x, y) in enumerate(pbar):
            image = x.to(args.device)
            label = y.to(args.device)
            optimizer.zero_grad()

            output = model(image)

            label = label.squeeze()
            loss = criterion(output, label)
            loss.backward()
            optimizer.step()
            scheduler.step()

            train_losses.append(loss.item())
            total += label.size(0)

            train_acc += acc(output, label)

        torch.save(model.state_dict(),
f'{args.save_path}/model_epoch{epoch+1}.pth')
```

epoch마다 torch.save를 진행하여 run.py에 각 epoch 모두 실행할 수 있도록 구현하였습니다.

Train2

```
    parser = argparse.ArgumentParser(description='2023 DL Term
Project')
    parser.add_argument('--save-path', default='checkpoints/',
help="Model's state_dict")
    parser.add_argument('--data', default='data/', type=str,
help='data folder')
    args = parser.parse_args()

    device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
    args.device = device
    num_classes = 10 # edited

    """
    TODO: You can change the hyperparameters as you wish.
            (e.g. change epochs etc.)
    """

    # hyperparameters
    args.epochs = 7

    # custom model
    model = resnet18(weights=ResNet18_Weights)

    # you have to change num_classes to 10
    num_features = model.fc.in_features # edited
    model.fc = nn.Linear(num_features, num_classes) # edited
    model.to(device)
    print(model)


    # 최적의 하이퍼파라미터 출력
    best_learning_rate = 0.0007247303217615042
    best_weight_decay = 0.05491043396482422
    best_train_batch_size = 512
    best_gamma = 0.988544232344997
    args.batch_size = best_train_batch_size

    train_loader, valid_loader = make_data_loader(args)
```

```
    optimizer = torch.optim.Adam(model.parameters(),
lr=best_learning_rate, weight_decay=best_weight_decay)
    scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer,
gamma=best_gamma)
    save(model, optimizer, scheduler, train_loader, valid_loader)
```

불러온 모델에 하이퍼 파라미터들을 입력해가며 최적의 결과를 찾아고자 하였습니다.

4. 한계점

- 우선 아쉬운 점으로 데이터 학습 전에 사진을 넣고 입력했더라면 좀 더 적절한 하이퍼 파라미터를 얻을 수 있을 것 같았으나 사진을 추가하는 생각을 늦게 하여 추가되는 데이터를 하이퍼 파라미터 탐색에 반영하지 못하였다는 한계가 존재합니다.
- 최종적으로 callback 함수를 이용해 계속 결과를 반환하였으나 이전 코드에는 그런 부분이 없었고, 컴퓨터가 종료되는 일이 생기면서 기록이 날라가였습니다. 이로인해 최소 횟수라고 생각했던 100회의 절반도 달성하지 못하였다는 한계가 존재합니다.
- 직접 하이퍼파라미터를 찾아가며 epoch이 5정도면 충분하다고 여겼으나 optuna를 이용해 찾을 때는 epoch이 7일 때까지 valid값이 계속 상승하던 경험이 있었습니다. 가장 결과가 좋아 제출하게된 모델 또한 지속적으로 loss는 감소하고 valid acc와 train acc 모두 상승하는 모습을 보였으나 종료하게 되었고, 추후에 하고자 하였으나 시간상의 문제로 추가 학습을 진행하지 못하였다는 한계가 존재합니다.