code review
[Train]

Train protocol

<YOLOv5-v5.0> (poc1\Yolo_EfficientNet_\3.DetectModel\yolov5-V5.0)

- **covid19.yaml** (train/val image data path, number of classes, class names setting)

- **train.py**(Detection-YOLOv5 training → save model weight file("best.pt"/"last.pt"))

<EfficientNetB4> (poc1\Yolo_EfficientNet_\Train\4.ClassifyModel\classification_EfficientNet)

- **make_csv.py** (create dataset csv → save csv("train_covid_croped", "test_covid_croped"))

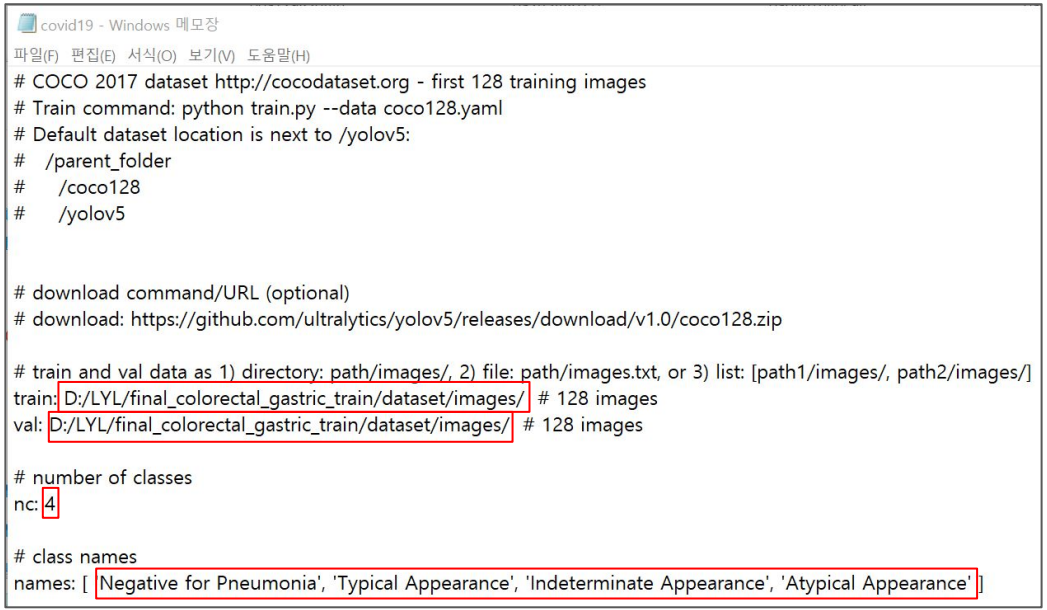
- **training.py** (Classification-EfficientNetB4 training → save model weights folder("output_b4_x"))

(+) **train_merge.py** (poc1\Yolo_EfficientNet_\Train\yolo_eff_train)

Train - YOLOv5; covid19.yaml

- train/val image data path, number of classes, class names setting

- dataset과 class 정보를 담은 yaml 파일 → train코드 중 parser 부분에 '--data'의 입력으로 사용



Train - YOLOv5; train.py

- Detection-YOLOv5 training → save model weight file("best.pt"/"last.pt")

- parameter 작성 부분

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5m6.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/covid19.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=20)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear LR')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--upload_dataset', action='store_true', help='Upload dataset as W&B artifact table')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='Set bounding-box image logging interval for W&B')
    parser.add_argument('--save_period', type=int, default=-1, help='Log model after every "save_period" epoch')
    parser.add_argument('--artifact_alias', type=str, default="latest", help='version of dataset artifact to be used')
    opt = parser.parse_args()
```

- weight pt 파일 저장 부분

```python
# Update best mAP
fi = fitness(np.array(results).reshape(1, -1))  # weighted combination of [P, R, mAP@.5, mAP@.5-.95]
if fi > best_fitness:
    best_fitness = fi
wandb_logger.end_epoch(best_result=best_fitness == fi)

# Save model
if (not opt.nosave) or (final_epoch and not opt.evolve):  # if save
    ckpt = {'epoch': epoch,
            'best_fitness': best_fitness,
            'training_results': results_file.read_text(),
            'model': deepcopy(model.module if is_parallel(model) else model).half(),
            'ema': deepcopy(ema.ema).half(),
            'updates': ema.updates,
            'optimizer': optimizer.state_dict(),
            'wandb_id': wandb_logger.wandb_run.id if wandb_logger.wandb else None}

    # Save last, best and delete
    torch.save(ckpt, last)
    if best_fitness == fi:
        torch.save(ckpt, best)
    if wandb_logger.wandb:
        if ((epoch + 1) % opt.save_period == 0 and not final_epoch) and opt.save_period != -1:
            wandb_logger.log_model(
                last.parent, opt, epoch, fi, best_model=best_fitness == fi)
    del ckpt
```

Train - EfficientNetB4; make_csv.py

- create dataset csv → save csv("train_covid_croped", "test_covid_croped")

- 각 이미지 경로와 class 정보를 csv 생성

```
for root, dirs, files in os.walk(test_img_dir):
    for file in files:
        if file.endswith(".png"):
            img_path = os.path.join(root, file)
            # print("img_path", img_path)
            if "(" in img_path:
                label_name = img_path.split('\\')[-1][:-7] + '.txt'
            else:
                label_name = img_path.split('\\')[-1][:-4] + '.txt'
            label_path = label_dir+label_name
            # print("label_name", label_name)
            # print("label_path", label_path)

            lines = open(label_path).readlines()
            cls = lines[0][0]
            # print("lines", lines)
            # print("cls", cls)
            # print("\n")

            path.append(img_path)
            label.append(cls)
print("path", path)
print("label", label)

make_csv = pd.DataFrame({"path":path,
                         "label":label})

make_csv.to_csv("test_covid_croped.csv", mode='w')
print(make_csv)
```

	A	B	C
1		path	label
1337	1335	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c99a7577bab8.png	0
1338	1336	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c9b62a594b8c.png	0
1339	1337	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c9cbab24db6c.png	0
1340	1338	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c9e358fea9ad.png	0
1341	1339	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c9e5cb5ff695.png	0
1342	1340	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩0₩c9eb5a6fcee5.png	0
1343	1341	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩000a312787f2(1).png	1
1344	1342	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩000a312787f2.png	1
1345	1343	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩0012ff7358bc(1).png	1
1346	1344	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩0012ff7358bc.png	1
1347	1345	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩001bd15d1891(1).png	1
1348	1346	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩001bd15d1891.png	1
1349	1347	D:₩LYL₩final_colorectal_gastric_train₩dataset₩classification₩train₩1₩002e9b2128d0(1).png	1

Train - EfficientNetB4; training.py

- Classification-EfficientNetB4 training → save model weights folder("output_b4_x")

- parameter 작성 부분

```
# Training settings
parser = argparse.ArgumentParser(description='PyTorch Classification')
parser.add_argument('--train_path', type=str, default="train_covid_croped.csv", metavar='P',
                    help='train label path')
parser.add_argument('--test_path', type=str, default="test_covid_croped.csv", metavar='P',
                    help='test label path')
parser.add_argument('--dataset_dir', type=str, default="D:\\LYL\\final_colorectal_gastric_train\\dataset\\classification", metavar='P',
                    help='dataset_dir')
parser.add_argument('--output_dir', type=str, default="output_b4_3", metavar='P',
                    help='output dir')
parser.add_argument('--model_name', type=str, default="efficientnet_b4", metavar='S',
                    help='model name in timm package (default: efficientnet_b4)')
```

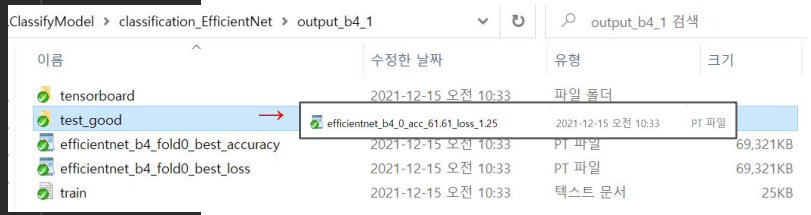
```
parser.add_argument('--train_batch_size', type=int, default=32, metavar='N',
                    help='input batch size for training (default: 32)')
parser.add_argument('--test_batch_size', type=int, default=32, metavar='N',
                    help='input batch size for validation (default: 32)')
parser.add_argument('--target_col', type=str, default="label", metavar='S',
                    help='target column name (default: label)')
parser.add_argument('--epochs', type=int, default=150, metavar='N',
                    help='number of epochs to train (default: 50)')
parser.add_argument('--train_img_size', type=int, default=320, metavar='N',   # 192, 256, 320, 380
                    help='train image size (default: 320)')
parser.add_argument('--test_img_size', type=int, default=380, metavar='N',   # 380
                    help='test image size (default: 380)')
```

```
parser.add_argument('--num_workers', type=int, default=1, metavar='N',
                    help='how many training processes to use (default: 0)')
parser.add_argument('--val_per_epochs', type=int, default=10, metavar='N',
                    help='validation per epoch (default: 10)')
parser.add_argument('--num_classes', type=int, default=4, metavar='N',
                    help='number of classes')
parser.add_argument('--n_fold', type=int, default=2, metavar='N',
                    help='number of folds (default: 4)')
```

Train - EfficientNetB4; training.py

- train 모델 실행 및 weight 파일 저장

```python
def run(args, model, device, train_dataset, valid_dataset, LOGGER, writer, test_dataset=None, fold=0):

    criterion = torch.nn.CrossEntropyLoss()
    train_fn(args=args, model=model, device=device, train_dataset=train_dataset, valid_dataset=valid_dataset,
             criterion=criterion, fold=fold, writer=writer, LOGGER=LOGGER)

    # load the best validation loss model
    if fold != 0:
        model.load_state_dict(
            torch.load(os.path.join(args.output_dir, f'{args.model_name}_fold{fold}_best_loss.pt')))

    if test_dataset is not None:
        test_loss, test_accuracy = test_fn(args, model, device, test_dataset, criterion, LOGGER)
        if not os.path.exists(os.path.join(args.output_dir, "test_good")):
            os.makedirs(os.path.join(args.output_dir, "test_good"))

        torch.save(model.state_dict(),
                   os.path.join(args.output_dir,"test_good", f'{args.model_name}_{fold}_acc_{test_accuracy:.2f}_loss_{test_loss:.2f}.pt'))

        writer.add_scalars('Loss', {'test': test_loss}, (fold+1)*args.epochs)
        writer.add_scalars('Accuracy', {'test': test_accuracy}, (fold+1)*args.epochs)
```




```python
run(args, model, device, train_dataset, valid_dataset, LOGGER, writer, test_dataset, fold)
```

Train - YOLOv5+EfficientNetB4; train_merge.py

- poc1\Yolo_EfficientNet_\Train\yolo_eff_train - ‘yolov5’ + ‘crop_resize’ + ‘efficientnetB4’

- 두 모델과 성능향상을 위한 이미지처리가 직렬로 실행되도록 하기 위한 code

```
def yolo_v5():
    logger = logging.getLogger(__name__)

    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolo/yolov5m6.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/covid19.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparameters path'
    parser.add_argument('--epochs', type=int, default=150)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[320, 320], help='[train, test] imag
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent trai
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
```

```
def efficientnet_v3():
    parser = argparse.ArgumentParser(description='PyTorch Classification')
    parser.add_argument('--train_path', type=str, default="eff/train_covid_croped.csv", metavar='P',
                        help='train label path')
    parser.add_argument('--test_path', type=str, default="eff/test_covid_croped.csv", metavar='P',
                        help='test label path')
    parser.add_argument('--dataset_dir', type=str,
                        default="D:\\LYL\\final_colorectal_gastric_train\\dataset\\classification", metavar='P',
                        help='dataset_dir')
    parser.add_argument('--output_dir', type=str, default="output_b4_3", metavar='P',
                        help='output dir')
    parser.add_argument('--model_name', type=str, default="efficientnet_b4", metavar='S',
                        help='model name in timm package (default: efficientnet_b4)')
    parser.add_argument('--pretrained', action='store_true', default=True,
                        help='load pretrained model')
    parser.add_argument('--pretrained_path', type=str, default="", metavar='S',
                        help='pretrained model path (default: "")')
    parser.add_argument('--train_batch_size', type=int, default=16, metavar='N',
                        help='input batch size for training (default: 32)')
```

```
def crop_resize():
    raw_path = "D:/LYL/final_colorectal_gastric_train/dataset/"
    dst_folder = raw_path + "classification"  # (수정)class별 폴더를 생성할 상위폴더
    image_path2 = raw_path + "images/"  # (수정)image파일 존재하는 위치
    label_path2 = raw_path + "labels/"  # (수정)label txt파일이 존재하는 위치

    num_of_class = 4  # (수정)class개수 --> 이 숫자에 따라 폴더 class별 폴더 생성

    crop_main()
```

```
if __name__ == '__main__':
    yolo_v5()
    crop_resize()
    efficientnet_v3()
```