Sprint - I

Date	13 November 2022
Team ID	PNT2022TMID14661
Project Name	AI POWERED NUTRITION ANALYZER FORFITNESS ENTHUSIASTS

The Configuration Class - This class contains a list of all the parameters and hyperparameters of the data and the model

Config

```
# ====================================================
CFG
# ====================================================
class CFG: #the configuration where all parameters/hyperparameters are stored
    debug=False
    apex=False
    print_freq=100
    num_workers=4
    model_name='tf_efficientnet_b0_ns'
    size=256
    scheduler='CosineAnnealingWarmRestarts'
    criterion='CrossEntropyLoss'
    epochs=2
    T_0=10
    lr=1e-4
    min_lr=1e-6
    batch_size=32
    weight_decay=1e-6
    gradient_accumulation_steps=1
    max_grad_norm=1000
    seed=42
    target_size = 5
    target_col = 'label'
    n_fold=5
    trn_fold = [0, 1, 2, 3, 4]
    train=True
    smoothing=0.05

if CFG.debug:
    CFG.epochs = 1
    train_data = train_data.sample(n=1000, random_state=CFG.seed).reset_index(drop=True)
```

Python

The Model Architecture Class – This is the class where the architecture of the model is defined. The pretrained weights are used from the timm library and a custom head is attached to the model to fit the number of classes

```python
# ====================================================
# MODEL
# ====================================================
class CustomEfficientNet(nn.Module):
    """ using the efficientnet arch and modifying the final classification head"""
    def __init__(self, model_name=CFG.model_name, pretrained=False):
        super().__init__()
        self.model = timm.create_model(CFG.model_name, pretrained=pretrained)
        n_features = self.model.classifier.in_features
        self.model.classifier = nn.Linear(n_features, CFG.target_size)

    def forward(self, x):
        x = self.model(x)
        return x
```

[ ] Python

```python
## for debug only
model = CustomEfficientNet(model_name=CFG.model_name, pretrained=False)
train_dataset = TrainDataset(train_data, transform=get_transforms(data='train'))
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True,
                          num_workers=4, pin_memory=True, drop_last=True)

for image, label in train_loader:
    output = model(image)
    print(output)
    break
```

[ ] Python

```
tensor([[ 3.6885e-02,  8.5828e-02,  9.2741e-02,  2.1368e-01, -4.4491e-01],
        [-8.7685e-02, -1.0964e-01,  6.5145e-02,  9.0393e-02,  1.0179e-02],
        [-1.1202e-01, -1.6264e-02,  5.9678e-03,  2.7062e-02,  2.2634e-04],
        [-1.2348e-01, -1.0647e-01,  5.4219e-02, -2.9790e-02, -2.3000e-02]],
       grad_fn=<AddmmBackward0>)
```

Train script – This block of code contains how the data is passed to the model for training

```python
def train_fn(train_loader, model, criterion, optimizer, epoch, scheduler, device):
    """the main train function is defined here, gradient clipping is done and final loss average is returned"""
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    scores = AverageMeter()
    # switch to train mode
    model.train()
    start = end = time.time()
    global_step = 0
    for step, (images, labels) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        images = images.to(device)
        labels = labels.to(device)
        batch_size = labels.size(0)
        y_preds = model(images)
        loss = criterion(y_preds, labels)
        # record loss
        losses.update(loss.item(), batch_size)
        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        if CFG.apex:
            with amp.scale_loss(loss, optimizer) as scaled_loss:
                scaled_loss.backward()
        else:
            loss.backward()
        grad_norm = torch.nn.utils.clip_grad_norm_(model.parameters(), CFG.max_grad_norm)
        if (step + 1) % CFG.gradient_accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()
            global_step += 1
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
```

Valid script – This block of code freezes the model weights and tests the performance of the model on the current validation set

```python
def valid_fn(valid_loader, model, criterion, device):
    """ the validation fucntion that returns the average validation loss and the prediction probability for each class(through softmax)"""
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    scores = AverageMeter()
    # switch to evaluation mode
    model.eval()
    preds = []
    start = end = time.time()
    for step, (images, labels) in enumerate(valid_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        images = images.to(device)
        labels = labels.to(device)
        batch_size = labels.size(0)
        # compute loss
        with torch.no_grad():
            y_preds = model(images)
        loss = criterion(y_preds, labels)
        losses.update(loss.item(), batch_size)
        # record accuracy
        preds.append(y_preds.softmax(1).to('cpu').numpy())
        if CFG.gradient_accumulation_steps > 1:
            loss = loss / CFG.gradient_accumulation_steps
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if step % CFG.print_freq == 0 or step == (len(valid_loader)-1):
            print('EVAL: [{0}/{1}] '
                  'Data {data_time.val:.3f} ({data_time.avg:.3f}) '
                  'Elapsed {remain:s} '
                  'Loss: {loss.val:.4f}({loss.avg:.4f}) '
                  .format(
                   step, len(valid_loader), batch_time=batch_time,
```

Training log – Here is a log of the model training

```python
if __name__ == '__main__': #The entry point for the program to run
    main()
```

Python

```
========== fold: 0 training ==========
INFO:__main__:========== fold: 0 training ==========
Downloading: "https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-weights/tf_efficientnet_b0_ns-c0e6a31c.pth" to
/root/.cache/torch/hub/checkpoints/tf_efficientnet_b0_ns-c0e6a31c.pth
Criterion: CrossEntropyLoss()
INFO:__main__:Criterion: CrossEntropyLoss()

Epoch: [1][0/65] Data 14.182 (14.182) Elapsed 0m 21s (remain 23m 20s) Loss: 1.5587(1.5587) Grad: 2.1206
Epoch: [1][64/65] Data 13.430 (3.503) Elapsed 4m 10s (remain 0m 0s) Loss: 0.1789(0.7073) Grad: 1.0743
EVAL: [0/17] Data 13.731 (13.731) Elapsed 0m 13s (remain 3m 40s) Loss: 0.0570(0.0570)

Epoch 1 - avg_train_loss: 0.7073  avg_val_loss: 0.0500  time: 315s
INFO:__main__:Epoch 1 - avg_train_loss: 0.7073  avg_val_loss: 0.0500  time: 315s
Epoch 1 - Accuracy: 1.0
INFO:__main__:Epoch 1 - Accuracy: 1.0
Epoch 1 - Save Best Score: 1.0000 Model
INFO:__main__:Epoch 1 - Save Best Score: 1.0000 Model

EVAL: [16/17] Data 4.179 (3.711) Elapsed 1m 4s (remain 0m 0s) Loss: 0.0311(0.0500)
Epoch: [2][0/65] Data 0.647 (0.647) Elapsed 0m 0s (remain 1m 1s) Loss: 0.0823(0.0823) Grad: 0.5457
Epoch: [2][64/65] Data 0.000 (0.011) Elapsed 0m 15s (remain 0m 0s) Loss: 0.0298(0.0603) Grad: 0.2402
EVAL: [0/17] Data 0.611 (0.611) Elapsed 0m 0s (remain 0m 12s) Loss: 0.0118(0.0118)

Epoch 2 - avg_train_loss: 0.0603  avg_val_loss: 0.0109  time: 18s
INFO:__main__:Epoch 2 - avg_train_loss: 0.0603  avg_val_loss: 0.0109  time: 18s
Epoch 2 - Accuracy: 1.0
INFO:__main__:Epoch 2 - Accuracy: 1.0
```

Infer Scripts – This block of code contains loading the trained weights to run inference on the test set

```python
# ====================================================
# Helper functions
# ====================================================
def load_state(model_path):
    """loading the state of model dictionary"""
    model = CustomEfficientNet(CFG.model_name, pretrained=False)
    model.load_state_dict(torch.load(model_path)['model'], strict=True)
    state_dict = torch.load(model_path)['model']
    return state_dict

def inference(model, states, test_loader, device):
    """making predictions on the test set"""
    model.to(device)
    tk0 = tqdm(enumerate(test_loader), total=len(test_loader))
    probs = []
    for i, (images) in tk0:
        images = images.to(device)
        avg_preds = []
        for state in states:
            model.load_state_dict(state)
            model.eval()
            with torch.no_grad():
                y_preds = model(images)
                #print(f'ypreds is : {y_preds}')
            avg_preds.append(y_preds.softmax(1).to('cpu').numpy())
            #print(f'avgpreds is : {avg_preds}')
        avg_preds = np.mean(avg_preds, axis=0)
        probs.append(avg_preds)
    probs = np.concatenate(probs)
    return probs
```

Python

```python
# ====================================================
# inference
# ====================================================
model = CustomEfficientNet(CFG.model_name, pretrained=False)
MODEL_DIR = '/content/'
states = [load_state(MODEL_DIR+f'{CFG.model_name}_fold{fold}_best.pth') for fold in CFG.trn_fold]
test_dataset = TestDataset(test_data, transform=get_transforms(data='valid'))
test_loader = DataLoader(test_dataset, batch_size=CFG.batch_size, shuffle=False,
                         num_workers=CFG.num_workers, pin_memory=True)
predictions = inference(model, states, test_loader, device)

test_data['model_preds'] = predictions.argmax(1) #argmax returns the highest index(class label)
test_data.head()
```

[ ]

Python

...

</>

	fruit	path	model_preds
0	APPLES	/content/drive/MyDrive/ibm_data/TEST_SET/APPLE...	0
1	APPLES	/content/drive/MyDrive/ibm_data/TEST_SET/APPLE...	0
2	APPLES	/content/drive/MyDrive/ibm_data/TEST_SET/APPLE...	0
3	APPLES	/content/drive/MyDrive/ibm_data/TEST_SET/APPLE...	0
4	APPLES	/content/drive/MyDrive/ibm_data/TEST_SET/APPLE...	0