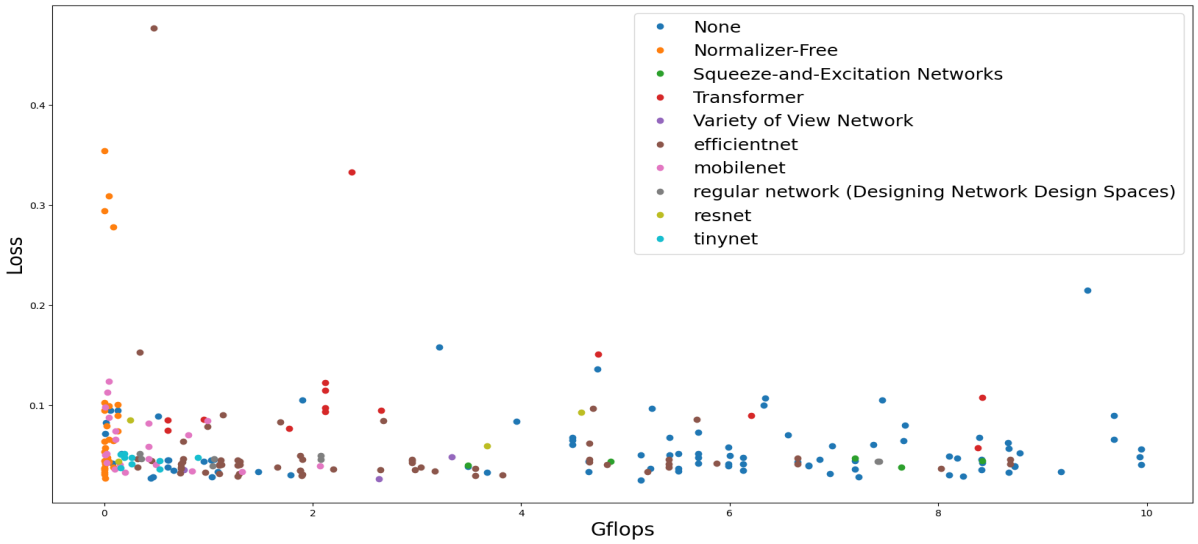
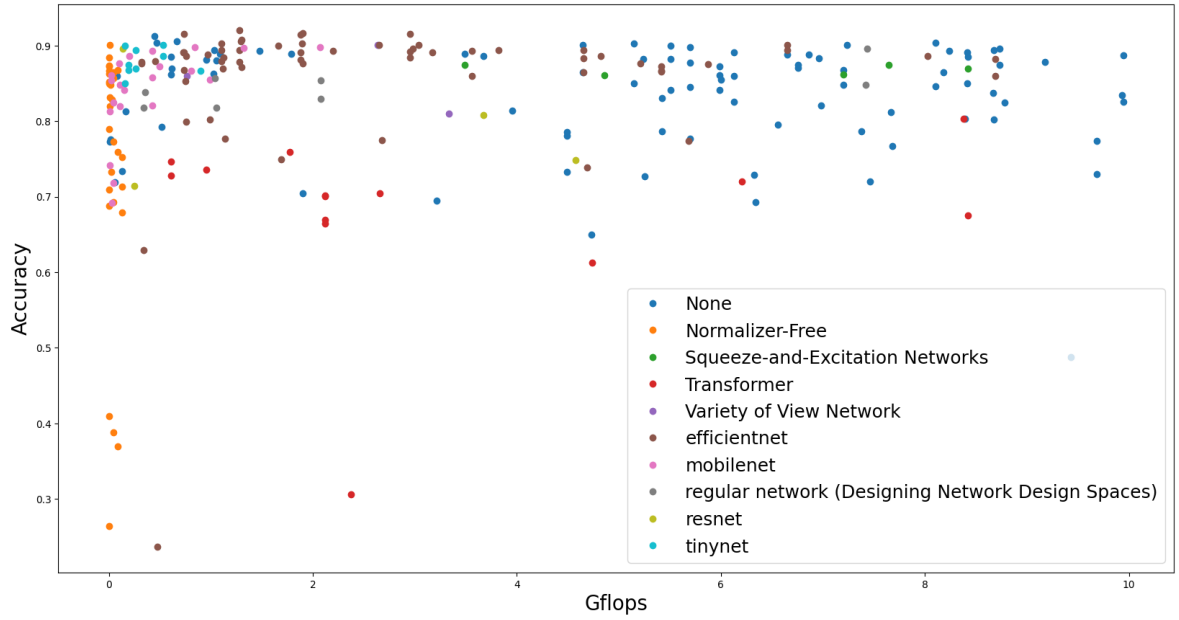
20185124 나영훈 기말 프로젝트 보고서.

Step 1: timm 모델 분석

기존의 Baseline으로 제공된 efficientNet보다 더 효율적이며, Model의 gflops가 더 낮은 Model Architecture가 있을 것이라 생각하였습니다. Papers with code에서 제공하는 SOTA model을 적용해 보았으나 성능이 좋지 않았고, timm 라이브러리에서 제공하는 모델 약 960여개 중 Pretrain model이 없는 경우와, gflops가 너무 큰 경우, 각종 오류를 야기하는 경우를 제외 한 300개의 모델을 돌려본 후 초기 모델을 설정하였습니다. 실험 결과를 요약하기 위해 Gflops를 x축으로 고정시킨 후 loss 및 accuracy를 시각화 하였습니다.

정확도 측면에서는 갈색 점의 efficientnet 계열의 모델이 성능이 좋았으며, Loss 측면에서는 주황색의 Normalizer-Free 방식을 사용한 모델이 좋았습니다. 따라서 Loss Normalizer-Free 방식을 사용한 모델 중 가장 낮은 Loss를 기록한 nf_regnet_b1 모델을 사용하여 학습을 진행하였습니다.





Step 2: Input Image shape

우선 이미지 크기 부분입니다. 학습의 사용되는 이미지의 평균 크기는 가로 약 850, 세로 약 570 정도의 이미지의 크기를 가지고 있으며, 이미지의 최대/최소 크기는 가로 기준 (1605, 787)/(280, 392), 세로 기준 (1137, 787)/(320, 196)으로 차이가 심한 것을 알 수 있습니다.

따라서 이미지의 크기를 특정 크기로 Resize 후, RandomCrop을 하는 방법을 사용하였습니다.

이미지 크기를 Resize 후 학습 과정에서 아래의 결과와 같이 Resize 후 학습하였을 때의 loss, accuracy, gflops는 가장 높은 정확도를 기록한 256x256으로 하였습니다.

	112x112	224x224	256x256	284x284	384x384	512x512	1024x1024
gflops	0.0088147	0.0091473	0.0092865	0.0094328	0.0100289	0.0110683	0.0181953
loss	0.338894	0.292937	0.305487	0.312202	0.337522	0.359611	0.448291
accuracy	0.882844	0.889613	0.90935	0.908338	0.886956	0.883034	0.834767

추가적으로 자체적인 einops 모듈의 Reduce 함수를 이용한 Gray Scale을 진행하였습니다. Albumentation에서의 Gray Scale은 3개의 채널이 모두 같은 값을 가지게 만들어주어 Gray scale이 되더라도 Gflops의 변동이 없습니다. 따라서 이 부분은 직접 구현하여 적용하였습니다.

Step 3: Data Augmentation

Data augmentation은 삼성 메디슨에서 제공한 기본 Augmentation인 HorizontalFlip(), VerticalFlip(), Rotate(), Crop() 중에서 가장 성능이 좋은 조합인 Rotate(), RandomCrop()을 사용하였습니다. 추가적으로 Albumentation 모듈 중 가장 성능이 좋았던 FancyPCA()까지 총 3개를 사용하였습니다.

CutMix와 MixUp을 사용하기 위해 최대한 간단하게 DataLoader에 Transform을 진행하였습니다.

Normalize의 경우에는 성능이 비슷하였지만, 학습을 더 길게 가져가며, 최적 모델 저장 기준을 Loss로 잡았기 때문에 Train data의 평균과 표준편차를 통해 Normalize를 진행하였습니다.

Normalize	**Loss**	**Accuracy**	**종료 Epoch (Early stop 5)**
ImageNet 평균, 표준편차	0.279744	0.906868	14
Train data 평균, 표준편차	0.278171	0.905428	25

Albumentation 모듈에 실험에 대한 결과는 엑셀에 첨부하였습니다.

Step 4: Model, Optimizer, loss function 선택.

Model은 앞서 설명한 실험 결과를 바탕으로 Normalizer Free 방식을 사용한 nf_regnet_b1을 사용하였습니다. Loss Function은 Classification 문제이기 때문에 가장 보편적으로 사용되는 CrossEntropy를 사용하였습니다. Stratified Kfold를 사용하여 Scheduler, Optimizer, learning rate, epochs를 설정하는 과정을 거쳤으며, Optimizer는 Adam보다 수렴속도가 더 빠르고 성능이 좋은 AdamW를 사용하였습니다. Scheduler는 ReduceLROnPlateau의 성능이 가장 뛰어났습니다.

추후에 Knowledge distillation을 Student Model에 적용하기 위해 Teacher Model의 출력값과 Student 모델의 출력 값에 KLDivLoss()를 적용시킨 후, Student 모델의 CrossEntropyLoss()와 일정 비율로 섞은 Knowledge distillation Loss를 사용하였습니다.

학습 시 일정한 확률로 CutMix와 MixUp을 진행하였습니다. 따라서 CutMix와 Mixup이 적용된 비율에 맞게 soft label로 변해야 하며, Loss function 또한 이를 반영하여 적용하였습니다.

"nf_regnet_b1" 모델의 학습을 하기위해 기존의 모델의 학습 방법과 동일하게 Optimizer를 AGC로 구현하여 학습하고 싶었으나 구현하지는 못하였습니다.

Teacher model (Loss function, Optimizer, Scheduler)

CrossEntropyLoss(), AdamW(learning rate = 0.001), ReduceLROnPlateau()

Student model & Student model Low Rank Feature map fine-tuning

Loss function: KLDivLoss() & CrossEntropyLoss(), CutmixCriterion(),

Optimizer, Scheduler: AdamW(learning rate = 0.001), ReduceLROnPlateau()

Step 5: 구성한 모델에 대해 Training 진행.

전체적인 학습의 틀은 Teacher Model과, Student Model을 이용한 Knowledge distillation을 진행하였습니다. Student Model로는 "nf_regnet_b1" 모델을, Teacher Model은 위의 timm 패키지 중 실험을 한 300개 모델과 Pytorch_efficientnet 모델 중 gflops가 높으면서, Model의 정확도가 가장 높은 모델을 산정하여 진행하여 최종 모델으로 Pytorch_EfficientNet의 b4를 Teacher Model로 선정하였습니다. (teacher-student model이 동일한 mean-teacher보다 Knowledge distillation의 성능이 더 좋았습니다.)

Teacher Model 학습 시 When Does label smoothing help? 논문의 내용을 바탕으로 label smoothing을 진행하지 않았습니다. 또한 앞서 설명한 Data augmentation 외의 CutMix, MixUp등의 Augmentation을 진행하지 않았습니다.

Student Model 학습 시에는 모델의 성능 향상을 위해 매 배치마다 균일분포를 가지는 0부터 1사이의 난수를 발생시켜 1/4 비율로 CutMix, 1/4확률로 MixUp이 발생되게 하여 Augmentation을 진행하였습니다. Validation accuracy가 가장 높은 비율은 CutMix와 Mixup을 0.5씩 사용할 때, 하지만 Kaggle의 Public score가 너무 많이 떨어지게 되어 각각 1/4씩 적용하였습니다.

CutMix 및 Mixup 비율에 따른 Teacher 모델의 Accuracy.

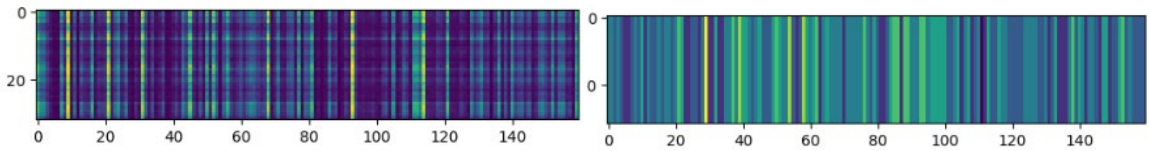
	Cutmix-0, MixUp-0	Cutmix-0.25, MixUp-0.25	Cutmix-0.5, MixUp-0.5
Pytorch-efficientnet_b4	92.85	90.50	91.86
bat_regnet_ts	92.38	91.53	90.79
Tf_efficientNetv2_l	91.48	90.99	90.50

CutMix 및 Mixup 비율에 따른 student 모델의 Accuracy. (Teacher model의 Cutmix, mixup 0)

	Cutmix-0.25, MixUp-0.25	Cutmix-0.5, MixUp-0.5
Pytorch-efficientnet_b4	94.72	94,86
bat_regnet_ts	93.13	91.84
Tf_efficientNetv2_l	94.64	

추가적으로 학습 후 Filter Pruning을 진행하여 모델 경량화를 하는 것이 목표였으나, Pretrain Model을 불러오는 과정에서 Filter연산을 Skip하는 과정이 불가능하여 gflops의 변동이 없습니다. 따라서 filter pruning 대신, Feature map의 Rank를 계산하여 Low Rank를 가지는 filter의 Weight를 초기화 시킨 후 재 학습하는 과정을 거쳤습니다. (Lin, HRank_CVPR_2020)을 참고하여 모델의 후반부의 Low rank filter를 초기화 하기로 하였으며, nf_regnet_b1 모델의 3번째 block의 마지막 conv layer의 filter의 weight를 초기화 후 재 학습을 하였습니다.

아래 사진은 nf_regnet_b1의 3번째 block과 4번째 block의 마지막 layer의 filter별 feature map의 rank를 시각화 한 그림입니다. x축은 layer의 몇 번째 필터인지를 나타내며, y축은 배치 별 이미지를 나타냅니다. 밝기가 낮을수록 Rank가 낮으며, 밝기가 높을수록 Rank가 높은 것을 의미합니다. 아래 그림은 배치 내부에서 어떤 label의 그림이 들어오던, Feature map에서의 Rank는 매우 유사하다는 것을 보여줍니다. 따라서 Filter Pruning을 통해 중요도가 낮은 필터(Low Rank filter)를 없애 gflops를 줄이거나 혹은 Weight initialization 후 재 학습을 통해 중요도가 낮은 필터를 중요도가 높은 필터(High Rank filter)로 학습을 통한 변환이 가능하다는 것을 의미합니다.



Weight 초기화 후 재 학습 Accuracy 변화

	Initialization ratio	기존	재 학습 후
Pytorch-efficientnet_b4 – nf_regnet_b1	1/8	94,64	94.88
Tf_efficientNetv2_l – nf_regnet_b1	1/8	91.64	94.00
Tf_efficientNetv2_l – nf_regnet_b1	1/4	91.64	94.40
Tf_efficientNetv2_l – nf_regnet_b1	1/2	91.64	94.65
Tf_efficientNetv2_l – nf_regnet_b1	5/8	91.64	93.73

따라서 최종 모델으로 teacher model은 pytorch efficientNet 패키지의 b4 모델, student 모델은 nf_regnet_b1 모델을 사용하였으며, weight 초기화는 Rank기준으로 정렬한 후 1/8을 초기화 후 재 학습한 모델을 사용하여 Test data 예측을 하였습니다.

Step 6: 시도하였지만 실패한 방법.

Step 5에서 설명한 최종 모델선택 방식으로 teacher model - pytorch efficientNet 패키지의 b4를 사용하였습니다. Model의 앙상블을 위해 다양한 teacher Model로 Knowledge distillation을 한 후 Confusion Matrix를 시각화 하여 기존의 pytorch efficientNet b4 모델과 성격이 다른 모델을 찾아

Test Time Augmentation 및 soft voting을 진행하였습니다.

```
[ 70.,   0.,   0.,   0.,   0.,   0.,   0.,   6.]
[  0., 165.,   4.,   8.,   0.,   0.,   0.,   6.]
[  0.,   4.,  71.,   0.,   0.,   0.,   0.,   2.]
[  0.,   6.,   0.,  51.,   0.,   0.,   0.,   2.]
[  0.,   0.,   0.,   0.,  98.,   0.,   0.,   6.]
[  0.,   0.,   0.,   0.,   0., 210.,   0.,   8.]
[  0.,   0.,   0.,   0.,   0.,   0., 196.,   2.]
[  1.,   0.,   0.,   0.,   5.,   2.,   0., 287.]
```

```
[ 71.,   0.,   0.,   0.,   0.,   0.,   0.,   7.]
[  0., 174.,  23.,   0.,   0.,   0.,   0.,   3.]
[  0.,   0.,  52.,   0.,   0.,   0.,   0.,   1.]
[  0.,   0.,   0.,  58.,   0.,   0.,   0.,   1.]
[  0.,   0.,   0.,   0., 100.,   0.,   0.,   1.]
[  0.,   0.,   0.,   0.,   0., 211.,   0.,   3.]
[  0.,   0.,   0.,   0.,   0.,   0., 196.,   2.]
[  0.,   1.,   0.,   1.,   3.,   1.,   0., 301.]
```

왼쪽이 Pytorch Efficientnet b4, 오른쪽은 timm tf_efficientnetv2_l 모델의 confusion matrix입니다. Confusion Matrix는 Matplotlib으로 시각화 한 그림은 너무 작아서 보이지가 않아 numpy 배열을 첨부하였습니다. x축이 모델의 예측, y축은 정답입니다. other 클래스 예측에 약점을 보이는 efficientnet b4 모델의 단점을 soft voting을 통해 보완할 수 있기를 기대하였으나, Public score의 향상을 보지 못하여 사용하지 않았습니다. (Private에서 TTA와 결합하여 사용시 89.851까지 올랐습니다.) Test Time Augmentation 으로는 원본 이미지, FancyPCA(p=1)을 사용하였습니다.

Automatic mixed precision: nf_regnet_b1에서 Gradient clipping을 사용하였다는 점을 보고 Gradient clipping방식을 알아보던 중 학습 중 Gradient를 32비트에서 16bit로 Clipping하며, 동시에 정확도도 유지하기 위해 32bit 연산을 섞어서 사용하는 amp를 사용해 보았습니다. 하지만 TITAN X GPU에서는 큰 효과를 보지 못했으며 3090 GPU에서만 동작을 잘 하였기 때문에 사용을 하지 않았습니다.

- 아래 코드 및 결과 스크린샷 보고서에 추가 필수

```
# !pip install thop
from thop import profile
import torch
# Creates once at the beginning of training
# scaler = torch.cuda.amp.GradScaler()

inputs = torch.randn(1, 1, 256, 256).to(device) # 자신의 model input size
macs, params = profile(model.module, inputs=(inputs, ))
flops = macs*2
gflops = round(flops/1000000000,7)

print("내 모델의 FLOPs : ",gflops, "GFLOP")
```

```
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register count_avgpool() for <class 'torch.nn.modules.pooling.AvgPool2d'>.
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register count_adap_avgpool() for <class 'torch.nn.modules.pooling.AdaptiveAvgPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
내 모델의 FLOPs :  0.0092865 GFLOP
```

참조1 GPU 정보: Computer1: [GeForce RTX 3090] x4, Computer2: [GeForce GTX TITAN X] x4

KLDivLoss() 참조: https://re-code-cord.tistory.com/entry/Knowledge-Distillation-1

Cutmix 참조: https://github.com/sonkt98/Dacon_artist/blob/main/skf_tta_cutmix.ipynb

Mixup 참조: https://dacon.io/codeshare/2452