2024 ANN 프로젝트 최종보고서

7조

박태현 송준규 양은주 정하연

20201588 20201590 20201596 20201610

〈 목차 〉

1. 프로젝트 개요 및 목표

MNIST extended dataset을 이용하여 직접 설계한 CNN 또는 pretrained CNN 모델을 학습시키고, Accuracy와 Inference Time의 적절한 조합을 찾는다.

2. 수행일정 및 역할 분담

2.1 최종 일정표

일시	프로젝트 진행 일정
4/29	프로젝트 개요 파악, 역할 분담, 수행계획서 작성
5/6	EMNIST 분석, LeNet-5 및 ResNet-50 스터디 및 학습
5/13	LeNet-5 및 ResNet-50 하이퍼파라미터 변경, pretrained-CNN 조사
5/20	pretrained-CNN 모델 학습, 중간발표 PPT 제작, 대본 작성, 리허설
5/27	자체 모델 개발 및 파인튜닝
6/3	자체 모델 수정 및 평가 지표 추가
6/10	최종발표 PPT 제작, 대본 작성, 리허설, 보고서 작성

2.2 최종 역할분담 및 기여도

이름	역할	기여도
박태현	- 다양한 CNN 모델 분석 - 최종 모델 선정 - 평가 지표 조사 - 최종 발표자료 제작자 1	15
송준규	- 데이터 분석 및 전처리 - 최종 모델 튜닝 - 중간 발표자 1 - 중간 발표자료 제작자 1 - 최종보고서 작성자 1	27
양은주	- LeNet5 및 ResNet50 학습 - 튜닝방법 조사 및 Baseline 학습 - 최종 발표자 1 - 최종 발표자료 제작자 2	23
정하연	- 팀장 / 실험 및 분석 총괄 - 중간, 최종 발표자 2 - 중간, 최종 발표자료 제작자 2, 3 - 최종 보고서 작성자 2	35

3. 실험 환경

3.1 분산학습 설정

NVIDIA A5000 24GB GPU 4개를 사용하여 분산학습을 진행하였습니다.

3.2 파이썬 모듈

Jupyter Notebook 커널이 알 수 없는 오류로 종료될 때마다 모든 함수를 다시 선언해야 하는 번거로움을 줄이기 위해, 상황별로 사용할 수 있는 함수를 작성하여 .py 파일로 모듈화하였습니다. 이를 통해 필요할 때마다 함수를 호출하여 효율성을 높일 수 있었습니다.

4. 프로젝트 수행 과정

4.1 데이터셋 선정 및 전처리

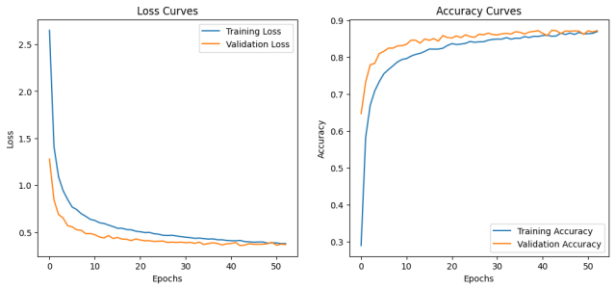
EMNIST 논문[1]을 참고하여 정확한 정보를 파악하고, Kaggle[2]로부터 6종류의 EMNIST 데이터셋을 모두 다운받았습니다. 종류 별 데이터셋을 아래 이미지와 같이 클래스 별로 모두 plot 후 관찰하였습니다. 클래스 별 데이터 개수의 불균형에서 오는 성능 저하를 예방하기 위해 Balanced 데이터셋을 학습 대상으로 선정하였으며, 더 많은 클래스를 가진 bymerge나 byclass 데이터셋을 전이학습 대상으로 선정하였습니다.

EMNIST Balanced의 테스트 데이터셋은 학습 데이터셋의 마지막 부분에 검증 데이터셋과 동일한 개수 만큼 지정되어 있기에, 학습 데이터셋으로부터 분리하는 작업을 진행하였습니다.

또한, CNN 모델마다 요구하는 최소 입력값 크기가 다르기 때문에, 원하는 크기로 데이터셋을 조정할 수 있는 함수를 정의하였습니다.

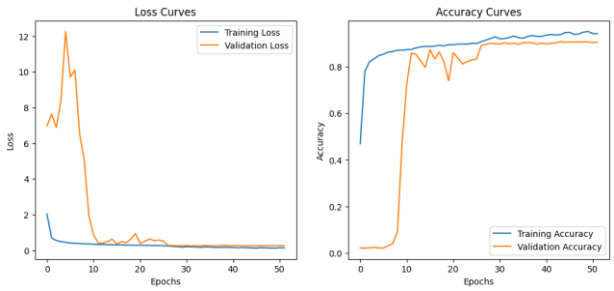
4.2 Baseline 모델 학습 및 결과 분석

4.2.1 LeNet5



LeNet-5은 early stopping과 학습률 스케줄러를 이용하여 최대 성능을 낼 수 있도록 학습했습니다. Tanh, RBF를 사용한 경우는 학습 시간이 266.94초 소요되었고, 테스트 정확도는 86.10%이었습니다. ReLU, softmax를 사용한 경우는 학습 시간이 171.13초로 더 짧았고, 테스트 정확도는 86.90%로 더 높았습니다. 위 그래프는 ReLU와 softmax를 사용한 모델의 학습 곡선이며, 매우 안정적으로 수렴함을 확인할 수 있습니다.

4.2.2 ResNet50



테스트 정확도 향상을 목표로 배치 사이즈 변경, 학습률 스케줄러 사용, 활성화 함수 변경, dropout 추가,

augmentation 적용의 과정을 순차적으로 거치며 ResNet50을 학습시켜보았습니다. Dropout을 추가한 단계에서 가장 좋은 테스트 정확도를 보였으며, augmentation은 오히려 성능 하락의 결과를 보였습니다.

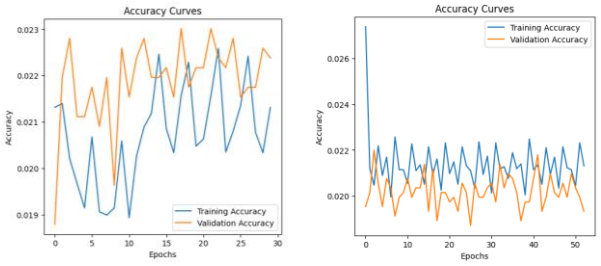
파라미터 수는 23,684,015개였으며, batch size가 2048일 때 학습 시간이 1449.46초가 소요되었고, 테스트 정확도는 90.40%를 보였습니다. 위 그래프는 해당 경우의 학습 곡선입니다.

4.3 후보 모델 선정

이미 논문을 통해 알려진 모델 구조와 파라미터 수, Top-1 및 Top-5 정확도, 그리고 GPU 추론 속도 등을 참고하여 EfficientNetB1[3]을 후보 모델 중 하나로 선정하였습니다.

추가로 모델 구조와 EMNIST 데이터셋의 특징을 고려하여 3가지 모델을 더 선정하였습니다. EMNIST는 이미지 크기가 작기에, 좁은 커널 크기(3x3)를 사용하여 작은 영역에서 세밀한 패턴을 학습할 수 있는 모델인 VGG16[4], MobileNet[5], ShuffleNet[6]을 후보로 결정하였습니다.

모델 이름	모델 특징에 기반한 선정 이유
EfficientNetB1	이미 알려진 다양한 CNN 모델 중 파라미터 수가 너무 많지 않으면서도 높은 정확도를 유지하는 모델임. (110 layers)
VGG16	3x3 커널을 사용하여 EMNIST 이미지의 작은 영역에서 세밀한 패턴을 인식하면서도, 깊은 구조를 통해 다각도 피처를 추출하여 높은 정확도를 달성할 것으로 예상됨. (16 layers)
MobileNet	깊이별 분리 합성곱을 사용하여 파라미터 수와 연산량을 줄이면서도 작은 이미지에서 높은 정확도를 유지할 수 있어 EMNIST 분류에 적합할 것으로 예상됨. (28-53 layers)
ShuffleNet	채널 셔플링과 그룹 컨볼루션을 통해 연산 효율성을 극대화하면서도 작은 이미지에서도 높은 성능을 유지하기 때문에 EMNIST 분류에 적합할 것으로 예상됨. (50-164 layers)



그러나 대부분의 모델에서 그레디언트 소실 문제가 발생하여 학습이 거의 진행되지 않았습니다. 위 그래프의 왼쪽은 VGG16, 오른쪽은 MobileNet의 학습 곡선을 나타내며, 매우 낮은 정확도 수치에서 심한 진동이 확인되었습니다. 모델의 레이어를 다양한 경우로 줄여보았지만, 문제는 여전히 해결되지 않았습니다. 해당 실험은 .ipynb 파일에 자세하게 기록되어있습니다.

원인은 크게 두 가지로 추정됩니다. 첫째, 배치 사이즈가 지나치게 커서 그레디언트 추정이 정확하지 않아 학습이 불안정해진 것으로 보입니다. 둘째, 선정된 후보 모델들은 7개의 레이어를 가진 Baseline 모델 LeNet에 비해 레이어 개수가 많지만, 레이어 간의 상호작용이 부족하여 이전 레이어의 정보가 충분히 반영되지 않았다는 점입니다.

그래서, 후보 모델을 다시 선정하였습니다. ResNet50이 매우 많은 파라미터를 가지고도 높은 성능을 보이는 점에 착안하여, 이전 레이어들의 정보가 다음 레이어에도 반영되는 모델들을 선택했습니다. 이러한 접근 방식은 모델이 깊더라도, 작은 크기의 EMNIST 이미지를 사용할 때 발생하는 기울기 소실 문제를 방지할 수 있을 것이라고 생각했습니다. 최종적으로 Wide ResNet (WRN) [7], DenseNet[8], Dual Path Networks (DPN)[9]을 후보로 선정했습니다.

모델 이름	모델 특징에 기반한 선정 이유
Wide ResNet (WRN)	ResNet의 변형으로, 네트워크 깊이를 증가시키는 대신 너비를 늘린 모델임. 기존 ResNet의 성능을 유지하며 넓은 레이어를 통해 더 다양한 특징을 추출할 수 있을 것으로 예상됨.
DenseNet	각 레이어가 모든 이전 레이어의 출력을 입력으로 받기 때문에 특징 재사용이 극대화되어 그레디언트 소실을 방지할 수 있을 것으로 예상됨.
Dual Path Networks (DPN)	ResNet과 DenseNet의 장점을 결합한 모델로, 병렬 경로를 통해 두 가지 특징 추출 방식을 사용하기에 그레디언트 소실 방지에 더 효과적일 것으로 예상됨.

4.4 최종 기반 모델 선정

DenseNet은 파라미터 수와 연산 효율성 측면에서 WRN보다 뛰어납니다. WRN은 네트워크의 너비를 증가시켜 다양한 특징을 추출할 수 있지만, 이는 파라미터 수가 증가하고 효율성이 떨어지는 단점이 있습니다. 반면, DenseNet은 더 적은 파라미터로 높은 효율성을 제공합니다.

또한, DenseNet은 DPN보다 구현이 간단합니다. DPN은 ResNet과 DenseNet의 장점을 결합한 모델로, 구현이 복잡하고, 개발 및 유지보수에 더 많은 노력이 필요합니다. 반면, DenseNet은 단순한 구조로 동일한 수준의 그레디언트 소실 방지 효과를 제공하면서도 구현이 용이하여 개발 및 유지보수에 유리합니다. 따라서, DenseNet을 자체 모델의 기반 모델로 선정하였습니다.

4.5 DenseNet121 기반 하이퍼파라미터 선정

DenseNet121을 기반으로 한 자체 모델을 개발하기 전, 안정적인 학습 환경을 설정하여 일관되게 학습하기 위해 하이퍼파라미터를 결정하는 실험을 진행했습니다.

4.5.1 학습률, 옵티마이저

표 1. batch_size=2048인 경우 학습률 및 옵티마이저 조합 별 학습 결과

lr \ optimizer		Adam	AdaMax	Nadam	AdamW	RMSprop	Nesterov	Momentum
0.01	training_time	544.56	550.91	760.89	567.72	620.66	520.53	529.66
	val_loss	0.364	0.332	0.425	0.484	0.601	0.569	0.577
	val_accuracy	0.873	0.884	0.871	0.871	0.872	0.893	0.892
0.001	training_time	537.02	569.95	774.14	561.67	614.44	536.29	519.2
	val_loss	0.616	0.879	0.952	0.808	0.869	1.120	1.111
	val_accuracy	0.892	0.889	0.888	0.886	0.887	0.887	0.885
0.0001	training_time	545.23	565.66	751.28	557.15	613.29	542.48	519.65
	val_loss	1.109	1.119	1.133	0.766	0.802	0.922	0.928
	val_accuracy	0.887	0.887	0.888	0.887	0.890	0.889	0.889

표 2. batch_size=1024인 경우 학습률 및 옵티마이저 조합 별 학습 결과

lr \ optimizer		Adam	AdaMax	AdamW	Nesterov	Momentum
0.01	training_time	902.40	915.48	976.46	869.01	866.92
	val_loss	0.35	0.3042	0.3891	0.311	0.3194
	val_accuracy	0.88	0.8906	0.8766	0.9009	0.8993
	test_accuracy	0.87	0.869	0.868	0.904	0.895
0.001	training_time	895.94	913.5	929.1	906.08	868.29
	val_loss	0.34	0.6115	0.578	0.5875	0.5953
	val_accuracy	0.90	0.8925	0.8896	0.8892	0.8888
	test_accuracy	0.89	0.893	0.886	0.89	0.891

최대한 많은 경우의 수를 고려하기 위해 batch size를 2048로 설정하여 학습 시간을 단축했고, 학습시간이 비교적 오래 걸린 Nadam과 RMSprop을 대상에서 제외했습니다. Batch size 1024로 줄여 다시 학습한 결과, 학습률이 0.01이고 옵티마이저가 Nesterov인 경우의 테스트 정확도가 90.4%로 가장 높았습니다.

4.5.2 활성화 함수

표 3. batch_size=1024, learning_rate=0.001, optimizer='Nesterov'의 경우 활성화함수 별 학습 결과

\ evaluate activation	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
ReLU	892.24	0.2893	0.8976	0.8867	0.8867	0.9922	11.858	10.724
LeakyReLU	892.01	0.2805	0.8997	0.8936	0.8936	0.9932	12.534	11.473
ELU	900.86	0.2632	0.903	0.8945	0.8945	0.9961	11.979	11.272
Swish	1168.56	0.303	0.898	0.8984	0.8984	0.9922	13.849	13.366

ReLU, LeakyReLU, ELU, Swish 중 어떤 함수가 가장 효과적인지 탐구하였습니다. 검증 손실이 0.2632로 최소값이고, 검증 정확도가 90.3%로 최댓값인 ELU가 가장 우수한 함수임을 확인했습니다.

활성화 함수를 탐색하는 과정부터 테스트 데이터셋으로 top-1 및 top-5 정확도, 평가 시간, 그리고 추론시간을 측정하여 성능을 비교하기 시작했습니다. 평가 시간이란 테스트 데이터를 사용하여 수행하는 전체 과정의 시간을 의미하고, 추론 시간은 학습된 모델이 새로운 데이터를 입력 받아 출력을 생성하는 데 걸리는 시간을 의미합니다.

4.5.3 Dropout

표 4. Dropout 위치에 따른 학습 결과

\ evaluate dropout	train_time	val_loss	val_acc
dropout_1	943.93	0.3013	0.8917
dropout_2	940.85	0.3642	0.8748
dropout_3	973.92	0.4051	0.8705
dropout_4	1037.98	0.4292	0.8553
dropout_5	973.22	0.2948	0.8942

표 5. Dropout 비율에 따른 학습 및 테스트 결과

\ evaluate dropout	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
dropout_1_1	974.06	0.2847	0.8974	0.9072	0.9072	0.9912	12.504	11.357
dropout_5_1	973.52	0.2926	0.8939	0.8906	0.8906	0.9912	15.929	11.117

실험 번호	모델 내 Dropout이 배치된 위치
dropout_1	Fully connected layer 뒤에 Dropout(0.5)
dropout_1_1	1번에서 비율을 0.5에서 0.3로 감소시킴.
dropout_2	각 Dense block 뒤에 Dropout(0.5)
dropout_3	각 Transition Block 뒤에 Dropout(0.5)
dropout_4	Dense Blocks 내부 Convolutional Layers 뒤에 Dropout(0.2)
dropout_5	처음 Convolution Layer 뒤에 Dropout(0.5)
dropout_5_1	5번에서 비율을 0.5에서 0.2로 감소시킴.

모델의 상위 층부터 하위 층까지 순차적으로 내려가면서 모델 내 dropout의 위치에 따른 성능 변화를 실험해 보았습니다. 표 4에서 최상위 층 혹은 최하위 층에 dropout을 배치했을 때 가장 높은 성능을 보임을 확인할 수 있습니다. 해당 경우에서 dropout 비율을 달리할 경우 성능이 개선되는지 추가 시험을 진행하였고, 최종적으로 최상위 층에서 0.3 비율로 dropout을 사용하는 것이 테스트 정확도가 90.7%로 가장 좋은 성능을 보임을 확인했습니다.

4.5.4 Augmentation

표 6. Augmentation 방법에 따른 학습 및 테스트 결과

\ evaluate aug	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
aug_1	1698.6555	0.3539	0.875	0.8818	0.8818	0.9902	12.207	10.174
aug_2	1920.1612	0.305	0.8863	0.8936	0.8936	0.9922	10.959	10.101
aug_3	1986.3978	0.2885	0.8924	0.8936	0.8936	0.9932	10.659	10.114
aug_4	1429.6499	0.2878	0.8935	0.8906	0.8906	0.9932	10.708	10.868
aug_5	1994.2567	5.1104	0.024	0.0273	0.0273	0.1104	10.521	9.975

실험 번호	적용한 Augmenation 방법
aug_1	회전
aug_2	회전, 수평 및 수직 이동
aug_3	회전, 수평 및 수직 이동, 왜곡
aug_4	회전, 수평 및 수직 이동, 왜곡, 확대
aug_5	회전, 수평 및 수직 이동, 왜곡, 확대, 밝기

다양한 augmentation 방법 중 어떠한 방법이 분류 성능 개선에 가장 큰 영향을 미치는 지 실험을 진행했습니다. 왜곡을 적용한 aug_3의 경우 모든 테스트 성능 지표에서 우수한 성능을 보였고, top-1 정확도는 dropout_1_1 모델보다 조금 낮은 89.36%이지만, top-5 정확도가 더 높고 평가 시간 및 추론 시간이 10초대로 크게 단축되어 augmentation을 적용하는 것이 좋은 개선 방법이라고 판단하였습니다.

그러나 학습 시간이 augmentation을 적용하지 않은 경우보다 약 100% 증가하는 단점이 있었습니다. 그래서 테스트 성능이 거의 비슷하고 학습 시간이 약 50%만 증가한 aug_4 방법을 사용하기로 결정했습니다. 하지만 여전히 학습 시간이 오래 걸리는 문제를 해결하기 위해, 다음 실험부터는 성능에 기반하여 학습률을 줄여나가는 ReduceLROnPlateau 스케줄러과 Early Stopping을 사용하기로 결정했습니다.

참고로, EMNIST의 특성 상 '회전'을 적용하면 6과 9와같이 비슷한 숫자나 문자가 혼동될 수 있다는 점을 고려하여 회전 각도는 10으로 지정하여 진행하였습니다.

4.6 DenseNet 변형 모델 설계

4.6.1 레이어 수정

기반 모델로 선택한 DenseNet은 여러 개의 Dense Block과 Transition Layer로 구성되어 있습니다. DenseNet121의 경우 4개의 Dense Block은 각각 4-8-16-8개의 합성곱 층으로 이루어져 있으며, 각 층은 이전 모든 층의 출력을 입력으로 받아 특징 재사용을 극대화하고 그레디언트 소실을 방지합니다. Transition Layer는 1x1 컨볼루션과 평균 풀링(Average Pooling)으로 차원을 줄여 모델의 크기를 조정하고 연산량을 줄입니다. Transition Layer에서 Compression Factor을 조정하면 차원을 얼마나 줄일 지 설정할 수 있습니다. 또한, DenseNet121 모델은 약 7.98M 파라미터를 가지고 있습니다.

Dense block의 개수, 각 dense block 내의 합성곱 층 개수, Transition Layer에서 줄이는 채널 수 등을 변경하여 DenseNet121 모델보다 파라미터 수가 적어 학습 속도가 빠르면서도 분류 성능이 높은 모델을 만드는 것을 목표로 설정했습니다. 더불어 평가 및 추론 시간의 단축도 목표로 하였습니다.

표 7. 레이어 수정 방법에 따른 학습 및 테스트 결과

\ evaluate layer	parameters	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
layer_1	3,346,743	2002.07	0.3182	0.8865	0.8799	0.8799	0.9893	7.268	6.853
layer_2	1,337,967	1944.93	0.4567	0.8539	0.8418	0.8418	0.9824	4.572	4.137
layer_3	5,747,858	2079.75	0.3065	0.8896	0.8809	0.8809	0.9912	12.179	9.182
layer_4	2,158,939	1989.10	0.3256	0.883	0.8818	0.8818	0.9902	5.946	5.427

표 8. 레이어 수정 방법에 따른 추가 학습 및 테스트 결과

\ evaluate layer	parameters	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
layer_4_1	2,152,667	1145.1703	0.2962	0.8918	0.8984	0.8984	0.9932	5.917	6.399
layer_4_2	1,201,151	1097.9399	0.3224	0.8894	0.8926	0.8926	0.9932	5.377	4.199
layer_4_3	1,201,151	1093.25	0.3181	0.8869	0.8877	0.8877	0.9922	5.309	4.215

실험 번호	변경한 모델 구조
layer_1	Dense block 합성곱 층 개수 줄이기 (4-8-16-8)
layer_2	Dense block 2개, Transition block 1개만 남기기
layer_3	Transition block의 Compression Factor 변경
layer_4	Dense block 합성곱 층 개수 변경 (3-6-12-6)
layer_4_1	(32,32,1)로 입력 데이터 크기 변경
layer_4_2	Dense block 합성곱 층 개수 변경 (2-4-8-4)
layer_4_3	처음 제로 패딩, 합성곱 층 줄이기

layer_4에서 테스트 정확도가 88.18%로 비교적 높게 유지되면서 평가 및 추론 시간이 가장 크게 감소하여 비슷한 구조로 추가 실험을 진행했습니다. Dense block이나 transition block의 개수를 줄이는 것은 파라미터 수 감소에는 도움이 되었지만, 성능이 유지되지 않았습니다. Dense block 내 합성곱 층의 개수를 줄이는 방법이 성능을 유지하면서 파라미터 수를 유지하는 데 가장 효과적이었습니다. EMNIST 데이터셋이 중앙에 위치한다는 점을 감안해 제로 패딩을 크게 할 필요가 없다고 생각하여 layer_4_3도 학습해보았지만, 성능 하락이 발생하여 최종적으로 Dense block 내 합성곱 층 개수가 3-6-12-6인 구조가 가장 우수한 모델임을 확인하였습니다.

layer_4_1의 테스트 정확도가 89.84%로 가장 높았지만, 2배 정도 적은 파라미터 수를 가지는 모델 또한 비슷한 성능을 보였기 때문에 layer_4_1과 layer_4_2을 최종 모델 구조 후보로 지정했습니다.

4.6.2 전이 학습

표 9. 전이학습 방법에 따른 학습 및 테스트 결과

\ evaluate transfer	model	parameters	train_time	val_loss	val_acc	test_acc	top-1 acc	top-5 acc	eval_time	inf_time
transfer_1	base	2,152,667	1287.07	-	0.8894			-		
	transfered		1760.59	0.2562	0.9038	0.9082	0.9082	0.9951	5.864	5.391
transfer_2	base	2,152,667	705.18	-	0.8894			-		
	transfered		979.62	0.2339	0.9085	0.9072	0.9072	0.9980	5.885	5.361
transfer_3	base	2,159,462	713.32	-	0.8638			-		
	transfered		1207.75	0.2441	0.9039	0.9072	0.9072	0.9922	6.139	6.850
transfer_4	base	1,201,151	118.34	0.3568	0.8777			-		
	transfered		1140.10		0.9013	0.8875	0.8875	0.9932	4.653	4.203

실험 번호	전이 학습 방법
transfer_1	layer_4_1, bymerge 데이터셋을 2048 배치, 10 에포크, 0.001 학습률로 학습
transfer_2	layer_4_1, bymerge (2048, 0.1학습률, 5epoch -> 1024, 0.001학습률, 100+early+scheduler)
transfer_3	layer_4_1, byclass 3epoch씩 (2048, 0.1) -> 기존과 동일
tranfer_4	layer_4_2, 나머지 조건 transfer_2와 동일.

전이학습으로 추가 실험을 진행한 결과, Dense block 내 합성곱 층을 과도하게 줄인 layer_4_2의 경우 테스트 정확도 향상에 한계가 있음을 확인하였습니다. 전이학습은 마지막 레이어에서 클래스 개수만 수정하여 진행되었습니다. 최종적으로 bymerge를 batch size 2048로 5 epoch 동안 학습한 결과가 가장 효율적임을 확인했습니다.

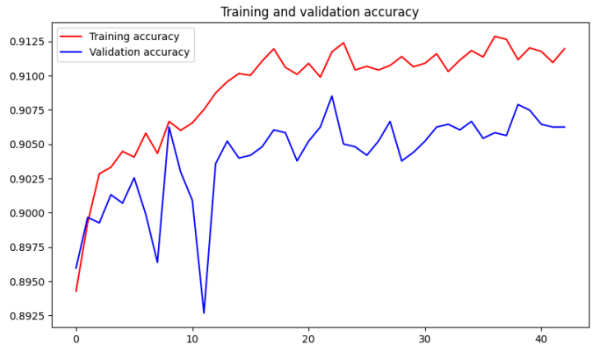
5. 최종 정량적 성능

5.1 최종 모델 구조

Layer (type)	Output Shape	Param #
input_27 (InputLayer)	(None, 32, 32, 1)	
zero_padding2d_48 (ZeroPadding2D)	(None, 38, 38, 1)	
conv1/conv (Conv2D)	(None, 16, 16, 64)	3136
conv1/bn (BatchNormalization)	(None, 16, 16, 64)	256
conv1/elu (ELU)	(None, 16, 16, 64)	
zero_padding2d_49 (ZeroPadding2D)	(None, 18, 18, 64)	
pool1 (AveragePooling2D)	(None, 8, 8, 64)	
conv2: 3 blocks		
conv2/block1	(Conv2D, ELU, BatchNormalization, Concatenate)	
conv3: 6 blocks		
conv3/block1	(Conv2D, ELU, BatchNormalization, Concatenate)	
conv4: 12 blocks		
conv4/block1	(Conv2D, ELU, BatchNormalization, Concatenate)	

conv5: 6 blocks		
conv5/block1	(Conv2D, ELU, BatchNormalization, Concatenate)	
bn (BatchNormalization)	(None, 1, 1, 452)	1808
elu (ELU)	(None, 1, 1, 452)	0
avg_pool (GlobalAveragePooling2D)	(None, 452)	0
dropout (Dropout)	(None, 452)	0
fc (Dense)	(None, 47)	21291
Total params: 2,152,667		
Trainable params: 2,128,755		
Non-trainable params: 23,912		

5.2 최종 모델 학습 정보



Batch	lr	Scheduler	Optimizer	act_func	aug
1024	0.001	성능 기반	Nesterov	ELU	aug_4

5.3 최종 모델 테스트 결과

test_acc	top-1 acc	top-5 acc	eval_time	inf_time
0.9072	0.9072	0.9980	5.885	5.361

Top-1 Accuracy는 90.72%, Top-5 Accuracy는 99.80%를 달성했습니다. 모델 성능 개선에 가장 큰 영향을 끼친 방법은 전이 학습입니다. 또한 평가 시간은 5.885초, 추론 시간은 5.361초를 기록하여, 최종 모델의 기준점이 된 DenseNet121의 평가 시간 11.858초, 추론 시간 10.724초보다 약 2배 빨라진 성능을 보였습니다.

5.4 Baseline 모델과 성능 비교

	Param #	Train time	Test Acc
LeNet5	65,104	171.12sec	86.9%
ResNet50	23,684,015	1449.46sec (batch 2048)	90.40%
Ours	2,152,667	**1684.80sec (batch 1024)**	**90.72%**

LeNet5은 파라미터가 적고 학습 시간이 짧다는 장점이 있지만, 테스트 정확도가 90%를 넘지 못하는 한계점이 있었습니다. 반면, ResNet은 90.40%의 정확도를 보였지만 파라미터 수가 많고, 학습 시간이 오래 걸린다는 단점이 있었습니다.

DenseNet을 변형한 저희 팀의 모델은 ResNet50보다 약 10배 작은 파라미터 수를 가지며, 배치 사이즈가 ResNet50 학습 시보다 작다는 점을 고려하면 약 2배 정도 단축되었습니다. 또한, 테스트 정확도가 90.72%로 Baseline 모델들보다 높은 분류 성능을 보입니다.

6. Challenges and Solutions

데이터셋의 특성에 맞는 모델을 선택하는 과정이 가장 어려웠습니다. EMNSIT 데이터셋이 작은 이미지로 구성되어 있다는 점에 기반하여 작은 커널을 사용하는 모델을 선택했지만, 그레디언트 소실 문제가 발생했습니다.

그래서 이전 레이어들의 정보를 반영하는 모델을 사용하기로 결정했습니다. 이러한 특징을 가지는 CNN 모델의 종류도 여러가지가 존재하지만, 파라미터 수와 구현 난이도를 함께 고려하여 DenseNet을 기반으로 변형 모델을 설계했습니다.

7. Lessons Learned

데이터셋과 모델의 특성을 함께 고려하는 것이 중요함을 깨닫게 되었습니다. 또한, 하이퍼파라미터의 세밀한 조정이 모델 성능에 큰 변화를 가져올 수 있음을 알게 되었습니다.

데이터셋의 특성을 고려하여 적절한 활성화 함수와 하이퍼파라미터를 설정하는 감각을 익힐 수 있었습니다. 예를 들어, 금융 데이터의 경우 전년동기대비, 전분기대비 지표를 활용한 데이터셋을 사용하기에 음수 데이터에 대한 가중치가 필요한데, ReLU 함수나 선형 함수를 사용하는 것보다는 LeakyReLU와 같은 활성화 함수가 더 적합할 것으로 추론할 수 있습니다. 즉, grid search와 같은 방식으로 모든 경우의 수를 고려하는 최적화 방식이 아닌 데이터셋의 특성을 깊이 있게 이해하고, 논리적인 추론을 통해 적절한 활성화 함수와 하이퍼파라미터를 선택하는 능력을 개발할 수 있었습니다.

8. 결론

다양한 CNN 모델을 학습시켜보며 EMNIST 분류기를 만들었습니다. 또한 학습률, 옵티마이저, 활성화 함수, Augmentation, 레이어 수정 및 전이학습이 모델의 분류 성능에 어떠한 영향을 미치는지 알아보았습니다. LeNet5와 ResNet50을 Baeseline 모델로 선정하고, 모델 개발 시 성능 개선의 기준점으로 활용했습니다. 최종적으로 경량화된 DenseNet 모델을 설계하여 학습 시간과 테스트 정확도, 평가 및 추론 시간 측면에서 가장 우수한 모델을 완성하였습니다.

Reference

[1] Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017). EMNIST: Extending MNIST to handwritten letters. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 2921-2926). IEEE. doi:10.1109/IJCNN.2017.7966217.

[2] https://www.kaggle.com/datasets/crawford/emnist, 2024년 6월 10일 접속.

[3] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the 36th International Conference on Machine Learning (pp. 6105-6114).

[4] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[6] Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6848-6856).

[7] Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks. arXiv preprint arXiv:1605.07146.

[8] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[9] Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., & Feng, J. (2017). Dual path networks. In Advances in neural information processing systems (pp. 4470-4478).