

최종 발표

7조

박태현, 송준규, 양은주, 정하연

목차

1. 프로젝트 개요 및 목표

2. 수행일정 및 역할 분담

2.1 최종 일정표

2.2 최종 역할분담 및 기여도

3. 실험 환경

3.1 분산학습 설정

3.2 파이썬 모듈

4. 프로젝트 수행 과정

4.1 데이터셋 선정 및 전처리

4.2 Baseline 모델 학습 및 결과 분석

4.2.1 LeNet5

4.2.2 ResNet50

4.3 후보 모델 선정

4.4 최종 기반 모델 선정

4.5 DenseNet121 기반 하이퍼파라미터 선정

4.5.1 학습률, 옵티마이저

4.5.2 활성화 함수

4.5.3 Dropout

4.5.4 Augmentation

4.6 DenseNet 변형 모델 설계

4.6.1 레이어 수정

4.6.2 전이 학습

5. 최종 정량적 성능

5.1 최종 모델 구조

5.2 최종 모델 학습 정보

5.3 최종 모델 테스트 결과

5.4 Baseline 모델과 성능 비교

6. Challenges and Solutions

7. Lessons Learned

8. 결론

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. 수행일정 및 역할 분담

2.2 최종 역할분담 및 기여도

이름	역할	기여도
박태현	<div><div><ul style="list-style-type: none">• 다양한 CNN 모델 분석• 최종 모델 선정</div><div><ul style="list-style-type: none">• 평가 지표 조사• 최종 발표자료 제작자 1</div></div>	15
송준규	<div><div><ul style="list-style-type: none">• 데이터 분석 및 전처리• 최종 모델 튜닝</div><div><ul style="list-style-type: none">• 중간 발표자 1• 중간 발표자료 제작자 1</div></div>	27
양은주	<div><div><ul style="list-style-type: none">• LeNet5 및 ResNet50 학습• 튜닝방법 조사 및 Baseline 학습</div><div><ul style="list-style-type: none">• 최종 발표자 1• 최종 발표자료 제작자 2</div></div>	23
정하연	<div><div><ul style="list-style-type: none">• 팀장 / 실험 및 분석 총괄• 중간, 최종 발표자 2</div><div><ul style="list-style-type: none">• 중간, 최종 발표자료 제작자 2, 3• 최종 보고서 작성자 2</div></div>	35

3. 실험 환경

3.1 분산학습 설정

- NVIDIA A5000 24GB GPU 4개를 사용하여 분산학습을 진행함

3. 실험 환경

3.2 파이썬 모듈

- 상황별로 사용할 수 있는 함수를 .py 파일로 모듈화
- 커널 종료 상황에서의 효율을 높임

파일 이름	함수 이름
datasets_utils.py	process_image
	prepare_datasets
	plot_class_examples
	class_counts_plot
distribution_utils.py	num_of_gpu
	distribute_dataset
	set_strategy
train_and_test_utils.py	get_model_path
	set_before_train
	plot_learning_curves
	test_loss_and_accuracy

4. 프로젝트 수행 과정

4.1 데이터셋 선정 및 전처리

(1) 데이터셋 선정

- EMNIST 논문을 통해 정확한 정보를 파악하고, 종류별로 plot 후 관찰
- EMNIST Balanced
 - 학습 대상으로 선정 : 클래스별 데이터 개수의 불균형에서 오는 성능 저하를 예방하기 위함
- EMNIST bymerge, byclass
 - 전이학습 대상으로 선정 : 더 많은 클래스를 가짐

(2) 전처리

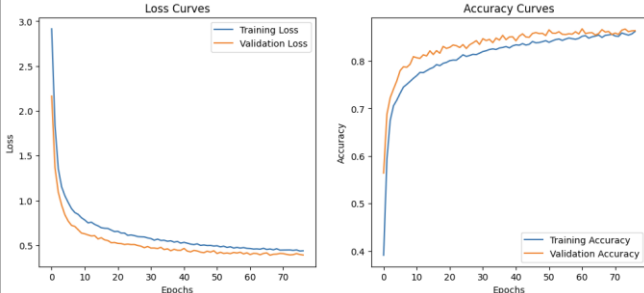
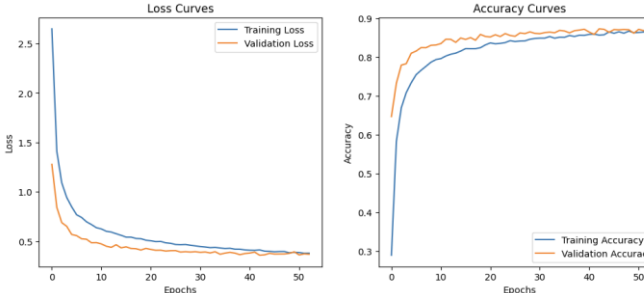
- EMNIST Balanced의 학습/검증/테스트 데이터셋 분리
- 데이터셋 크기 조정 함수 정의

4. 프로젝트 수행 과정

4.2 Baseline 모델 학습 및 결과 분석

4.2.1 LeNet-5

- early stopping, 학습률 스케줄러 사용

	LeNet-5-old	LeNet-5-new
Activation Function	tanh, RBF	ReLU, Softmax
Learning Curve		
Training Time	3.5 sec/epoch	3 sec/epoch
	266.94 sec (4m 27.0sec)	171.13 sec (2m 51.1sec)
Test Accuracy	86.10 %	86.90%

4. 프로젝트 수행 과정

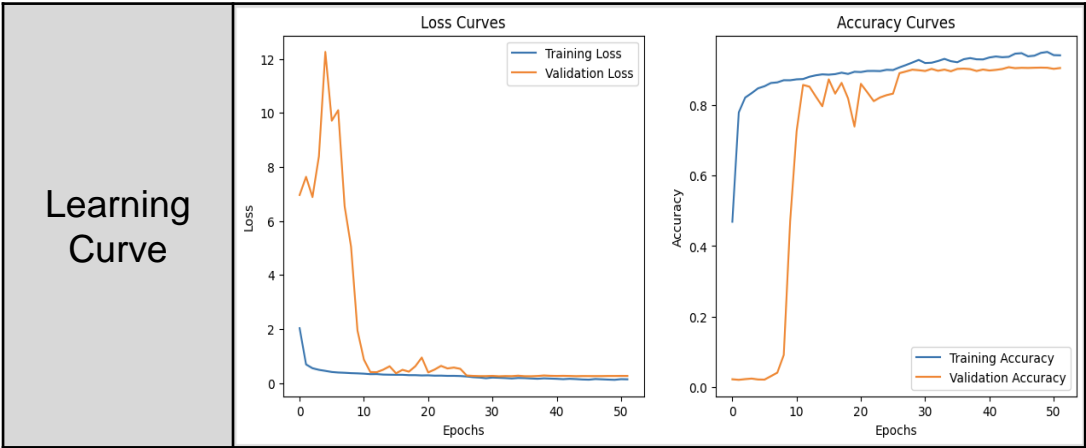
4.2 Baseline 모델 학습 및 결과 분석

4.2.2 ResNet-50

- 배치 사이즈 변경 → 학습률 스케줄러 사용 → 활성화 함수 변경 → dropout 추가 → augmentation 적용

↓
Best Test Acc

↓
성능 하락



Batch size	2048
Parameters	23,684,015
Training Time	1449.46 sec
Test Accuracy	90.40%

4. 프로젝트 수행 과정

4.3 후보 모델 선정

- 모델 구조, 파라미터 수, Top-1 및 Top-5 정확도, GPU 추론 속도 등 고려

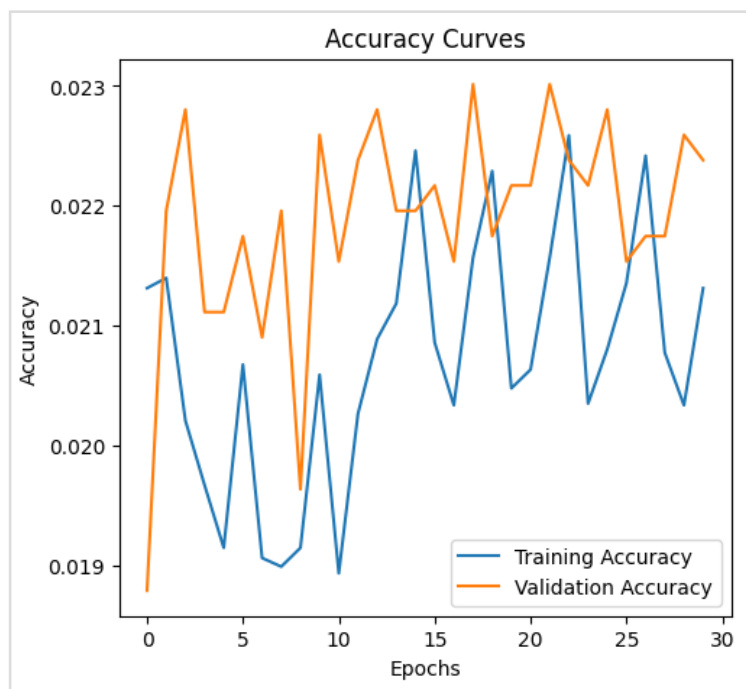
EfficientNetB1	파라미터 수가 너무 많지 않으면서도 높은 정확도를 유지	110
----------------	--------------------------------	-----

- 모델 구조, EMNIST 데이터셋의 특징 고려

VGG16	3x3 커널을 사용하여 EMNIST 이미지의 작은 영역에서 세밀한 패턴을 인식하고, 깊은 구조를 통해 다각도 피처를 추출 가능	16
MobileNet	깊이별 분리 합성곱을 사용하여 파라미터 수와 연산량을 줄이면서도 작은 이미지에서 높은 정확도를 유지할 수 있어 EMNIST 분류에 적합	28-53
ShuffleNet	채널 셔플링과 그룹 컨볼루션을 통해 연산 효율성을 극대화하면서도 작은 이미지에서도 높은 성능을 유지하기 때문에 EMNIST 분류에 적합	50 - 164

4. 프로젝트 수행 과정

4.3 후보 모델 선정



대부분의 모델에서 **그레이디언트 소실 문제 발생**

1. 지나치게 큰 **batch size**

2. LeNet보다 레이어가 조금만 늘어나도

이전레이어로부터 전달되는 그레이디언트 값이 점차 작아져

바로 그레이디언트 소실이 발생함



∴ 이미지의 크기만 고려하는 것은 옳지 않음

4. 프로젝트 수행 과정

4.3 후보 모델 선정

- 이전 레이어들의 정보가 다음 레이어에도 반영되는 모델 선정
- 모델이 크더라도, 작은 크기의 EMNIST 에서의 그레이디언트 소실 문제를 방지할 것이라 기대

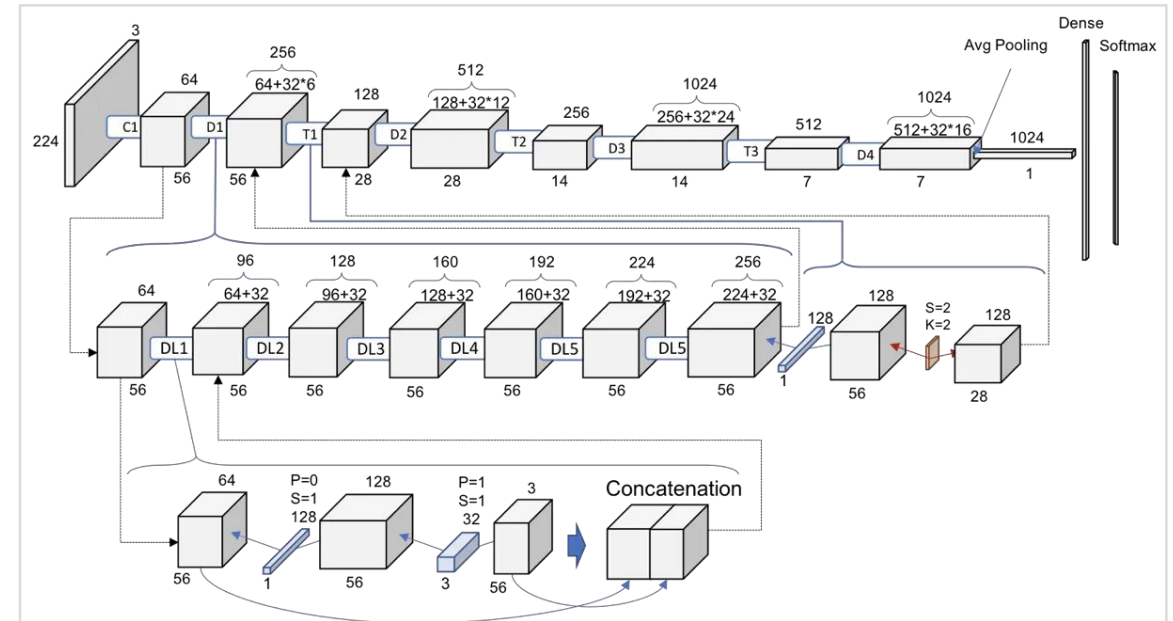
Wide ResNet (WRN)	<ul style="list-style-type: none">- ResNet의 변형으로, 네트워크 깊이를 증가시키는 대신 너비를 늘린 모델- 기존 ResNet의 성능을 유지하며 넓은 레이어를 통해 더 다양한 특징을 추출할 수 있을 것으로 예상
DenseNet	<ul style="list-style-type: none">- 각 레이어가 모든 이전 레이어의 출력을 입력으로 받음- 특징 재사용이 극대화되어 그레이디언트 소실을 방지할 수 있을 것으로 예상
Dual Path Networks (DPN)	<ul style="list-style-type: none">- ResNet과 DenseNet의 장점을 결합한 모델- 병렬 경로를 통해 두 가지 특징 추출 방식을 사용하기에 그레이디언트 소실 방지에 더 효과적일 것으로 예상

4. 프로젝트 수행 과정

4.4 최종 기반 모델 선정

DenseNet

WRN보다 더 적은 파라미터로 높은 효율성 제공.
 DPN보다 단순한 구조로 구현이 간단함.
 동일한 수준의 그레디언트 소실 방지 효과 제공.
 개발 및 유지보수 유리.



4. 프로젝트 수행 과정

4.5 DenseNet121 기반 하이퍼파라미터 선정

4.5.1 학습률, 옵티마이저

- batch size = 2048로 설정하여 학습 시간 단축
- 학습 시간이 오래 걸린 Nadam, RMSProp 제외

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

- batch size = 1024로 줄여 다시 학습

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

4. 프로젝트 수행 과정

4.5 DenseNet121 기반 하이퍼파라미터 선정

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

- 테스트 데이터셋으로 Top-1 및 Top-5 정확도, 평가 시간, 추론 시간 추가 측정 및 비교

1) 평가 시간 : 테스트 데이터를 사용하여 수행하는 전체 과정의 시간

2) 추론 시간 : 학습된 모델이 새로운 데이터를 입력받아 출력을 생성하는 데 걸리는 시간

4. 프로젝트 수행 과정

4.5 DenseNet121 기반 하이퍼파라미터 선정

4.5.3 Dropout

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.

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

4. 프로젝트 수행 과정

4.5 DenseNet121 기반 하이퍼파라미터 선정

4.5.4 Augmentation

- 왜곡을 적용한 aug_3의 경우 **평가 및 추론 시간이 단축됨**.
→ augmentation 적용이 좋은 개선방안이라고 판단
- **But 학습 시간 약 100% 증가**
→ 테스트 성능이 거의 비슷하나,
학습 시간이 약 50%만 증가한 aug_4 적용 결정
- 학습 시간이 오래 걸리는 문제 해결을 위해
 - ReduceLROnPlateau 스케줄러 사용
 - Early Stopping 사용
- **회전 각도 10**으로 지정
 - 비슷한 숫자나 문자의 혼동을 방지하기 위해

실험 번호	적용한 Augmentation 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

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



4. 프로젝트 수행 과정

4.6 DenseNet 변형 모델 설계

4.6.1 레이어 수정

- 목표
 - DenseNet121 모델보다 파라미터 수가 적어 학습 속도가 빠르면서도 분류 성능이 높은 모델 설계
 - 평가 및 추론 시간의 단축
- 변경사항
 - Dense block의 개수
 - 각 Dense block 내의 합성곱 층 개수
 - Transition Layer에서 줄이는 채널 수 등

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

4. 프로젝트 수행 과정

4.6 DenseNet 변형 모델 설계

4.6.1 레이어 수정

layer_4	Dense block 합성곱 층 개수 변경 (3-6-12-6)
---------	------------------------------------

- 테스트 정확도 : 88.18%
- 평가 및 추론 시간이 약 2배로 가장 크게 감소

• 최종 모델 구조 후보

layer_4_1	(32,32,1)로 입력 데이터 크기 변경
-----------	-------------------------

- 테스트 정확도 : 89.84%

layer_4_2	Dense block 합성곱 층 개수 변경 (2-4-8-4)
-----------	-----------------------------------

- 테스트 정확도 : 89.26
- 파라미터 수가 layer_4_1보다 약 2배 적음

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

4. 프로젝트 수행 과정

4.6 DenseNet 변형 모델 설계

4.6.2 전이 학습

transfer_1	bymerge 데이터셋을 2048 배치, 10에포크, 0.001 학습률로 학습
transfer_2	전이학습 (2048, 0.1학습률, 5epoch → 1024, 0.001학습률, 100+early+scheduler)
transfer_3	byclass 전이학습 3epoch씩 (2048, 0.1) → 기존과 동일
transfer_4	layer_4_2, 나머지 조건 transfer_2와 동일

표 9. 전이 학습 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	-				
	transferred		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	-				
	transferred		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	-				
	transferred		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	-				
	transferred		1140.10		0.9013	0.8875	0.8875	0.9932	4.653	4.203

- 마지막 레이어에서 클래스 개수만 수정하여 전이학습 진행
 - Dense block 내 합성곱 층을 과도하게 줄인 경우(layer_4_2) 테스트 정확도 향상에 한계
 - bymerge를 batch size = 2048, 5epoch 동안 학습한 결과가 가장 효율적임

5. 최종 정량적 성능

5.1 최종 모델 구조 및 파라미터 확인

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

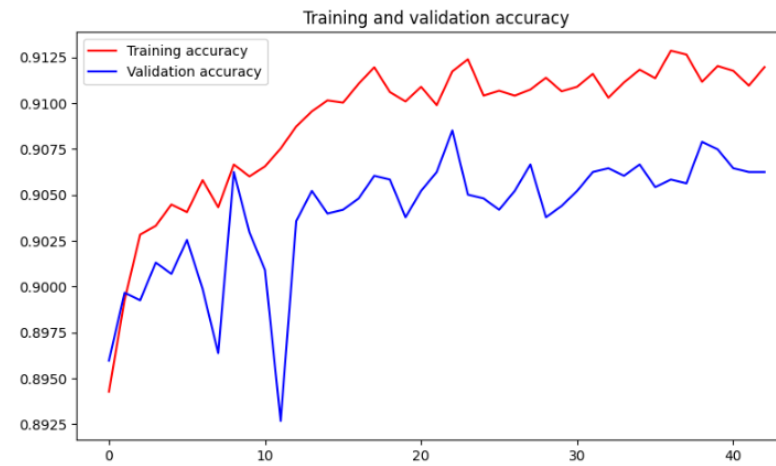
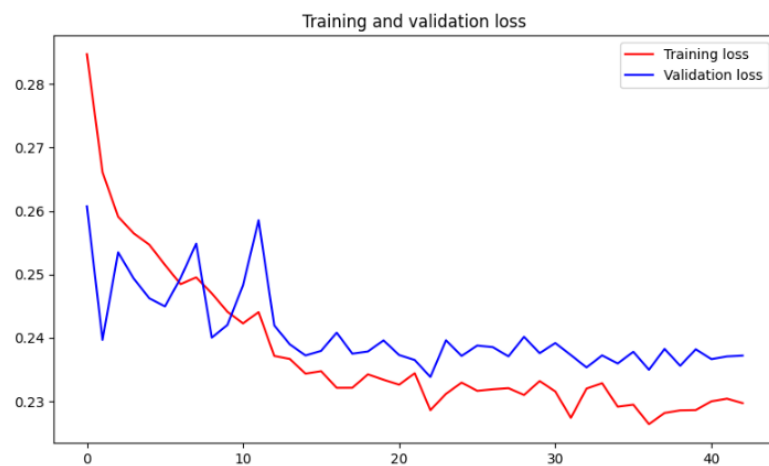
Non-trainable params: 23,912

- Total params
 = Trainable params + Non-trainable params
 → 모델의 모든 파라미터가 누락 없이 포함되었음

5. 최종 정량적 성능

5.2 최종 모델 학습 정보

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



5. 최종 정량적 성능

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

- 모델 성능 개선에 가장 큰 영향을 끼친 방법은 **전이 학습**
- 최종 모델의 기준점이 된 DenseNet121보다 평가 및 추론 시간이 2배 단축됨

5. 최종 정량적 성능

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%

- LeNet5

파라미터가 적고 학습 시간이 짧지만, 테스트 정확도가 90%를 넘지 못함

- ResNet50

비교적 높은 정확도를 보였지만 파라미터 수가 많고, 학습 시간이 오래 걸림

- DenseNet을 변형한 프로젝트 모델

ResNet 보다 약 10배 적은 파라미터

배치 사이즈가 ResNet50 학습 시보다 작다는 것을 고려하면 약 2배 단축됨
Baseline 모델들보다 높은 분류 성능을 보임

6. Challenges and Solutions

Challenges 데이터셋의 특성에 맞는 모델 선정 방법

EMNIST 데이터셋은 작은 이미지로 구성됨



작은 커널을 사용하는 모델 선택



그레이디언트 소실 문제 발생

Solutions 이전 레이어들의 정보를 반영하는 모델 선정

7. Lessons Learned

- 데이터셋과 모델의 특성을 함께 고려하는 것이 중요함
- 하이퍼파라미터의 세밀한 조정이 성능에 큰 변화를 가져올 수 있음
- 데이터셋의 특성을 깊이 있게 이해하고, 이를 바탕으로 논리적인 추론을 통해 적절한 활성화 함수와 하이퍼파라미터를 설정하는 감각을 익힘

Ex) 금융 데이터의 경우 전년동기대비, 전분기대비 지표를 활용한 데이터셋을 사용하기에 음수 데이터에 대한 가중치가 필요함

따라서 ReLU나 선형 함수보다는 LeakyReLU와 같은 활성화 함수를 사용하는 것이 적합하다고 추론 가능

- 다양한 모델이 가지는 기본적인 특성을 이해함으로써 목적에 맞는 모델과 하이퍼파라미터를 빠르게 찾을 수 있는 능력을 얻음

8. 결론

- 다양한 CNN모델을 학습시켜보며 EMNIST 분류기를 설계
- 학습률, 옵티마이저, 활성화 함수, Augmentation, 레이어 수정, 전이학습이 모델의 분류 성능에 어떤 영향을 미치는지 파악함
- LeNet5와 ResNet50을 **Baseline 모델**로 선정, 모델 개발 시 성능 개선의 기준으로 사용
- 최종적으로 경량화된 DenseNet 모델을 설계
- 학습 시간, 테스트 정확도, 평가 및 추론 시간 측면에서 가장 우수한 모델을 완성

감사합니다