**Artificial Neural Network**

**Project Report**

**4조**

**조장-20180615 조무곤**

**조원-20180565 박성민**

**조원-20180605 임현만**

**조원-20180611 전태호**

**목차**

1. **프로젝트 개요 및 목표**

**가. 개요 및 목표**

**나. 수행 계획 및 역할 분담**

1. **프로젝트 내용**

**가. MNIST extended dataset 개요 및 분석**

**나. Dataset 분류**

**다. Baseline 학습 및 결과분석**

**-1 LeNet5**

**-2 ResNet-50**

**라. 학습에 사용할 모델 선택**

**-1 Pre Trained Model**

**-2 Handmade Model**

**마. 모델 최적화 및 분석**

**-1 Pre Trained Model**

**-2 Handmade Model**

1. **결과**

**1. 프로젝트 개요 및 목표**

**가. Overview**

MNIST extended dataset를 직접 설계한 CNN 또는 pretrained CNN모델을 이용하여 학습시키고 성능을 파악하며, 이를 최적화하여 가장 좋은 성능을 나타내는 학습모델 찾아내기.

**Target of Project**

MNIST extended dataset을 이용하여 직접 설계한 CNN 또는 pretrained CNN모델을 이용하여 학습시키고 accuracy와 inference time의 적절한 조합을 찾는다. 직접 준비한 data instance에 대해 inference 결과를 얻어 학습 데이터의 테스트 결과와 비교하고 차이를 분석한다

**나. 수행계획서 및 역할 분담 (수정한 계획 및 수행일지)**

**수행일지**

| 역할 | 조장 | 팀원 | 팀원 | 팀원 |
| --- | --- | --- | --- | --- |
| 이름 | 조무곤 | 박성민 | 임현만 | 전태호 |
| 모임일자 | 발표내용 | | | |
| 5.3 | Class dataset, Merge dataset | Digits dataset | Balanced dataset | Letters dataset, mnist dataset |
| 5.10 | training/validation/test dataset  비율 조정 및 코드 작성 | training/validation/test dataset  비율 조정 및 코드 작성 | training/validation/test dataset  비율 조정 및 코드 작성 | training/validation/test dataset  비율 조정 및 코드 작성 |
| 5.11 | Baseline 학습 및 결과분석2 (LeNet5코드 작성 및 수정) | Baseline 학습 및 결과분석 (ResNet-50 코드작성 및 수정) | Baseline 학습 및 결과분석2 (LeNet5코드 작성 및 수정) | Baseline 학습 및 결과분석 (ResNet-50 코드작성 및 수정) |
| 5.13 | LeNet5코드 작성 및 수정 + 중간 발표자료 준비 및 ppt 제작 | ResNet-50 코드작성 및 수정)　+중간 발표자료 준비 및 ppt 제작 | LeNet5코드 작성 및 수정 + 중간 발표자료 준비 및 ppt 제작 | ResNet-50 코드작성 및 수정)　+중간 발표자료 준비 및 ppt 제작 |
| 5.15 | baseline model 정하기 위한LeNet5의 optimizer 수정 | baseline model 정하기 위한ResNet-50의 optimizer수정 | baseline model 정하기 위한LeNet5의 optimizer  수정 | baseline model 정하기 위한ResNet-50의 optimizer수정 |
| 5.22 | 학습에 선택할 모델 선택-Pre Trained Model 코드 짜기 | 학습에 선택할 모델 선택-Pre Trained Model 코드 짜기 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기 |
| 5.29 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기2 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기2 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기2 | 학습에 선택할 모델 선택- Hadnemade Model 설계 및 코드 짜기2 |
| 6.3 | 선택한 모델의 최적화 진행 learning rate 관련작업 | 선택한 모델의 최적화 진행 learning rate 관련작업 | 선택한 모델의 최적화 진행 optimizer 관련작업 | 선택한 모델의 최적화 진행  optimizer 관련작업 |
| 6.7 | 선택한 모델의 최적화 진행 hyper parameter 변경 2 | 선택한 모델의 최적화 진행 hyper parameter 변경 2 | 선택한 모델의 최적화 진행 hyper parameter 변경 2 | 선택한 모델의 최적화 진행 hyper parameter 변경 2 |
| 6.8 | 최종 모델의 성능 평가, 보고서 작성 | 최종 모델의 성능 평가, 보고서 작성 | 최종 모델의 성능 평가, 보고서 작성 | 최종 모델의 성능 평가, 보고서 작성 |
| 6.10 | 보고서 마무리 및 발표 준비 | 보고서 마무리 및 발표 준비 | 보고서 마무리 및 발표 준비 | 보고서 마무리 및 발표 준비 |

## 

## **수행계획서**

## **5월 3일 : discover and visualize the data to gain insight**

목표) 새로운 data instance를 추가하기 위한 data format 분석 (Get the data/Discover and visualize the data) - 여러 종류의 dataset 중 하나를 선택: 근거 제시.

박성민 - EMNIST By Class dataset, EMNIST By Merge dataset format 분석 및 조사

전태호 - EMNIST Balanced dataset, EMNIST Digits dataset 분석 및 조사

조무곤 - EMNIST Letters dataset, EMNIST mnist dataset 분석 및 조사

박성민,전태호,조무곤 임현만공동 - 각자 찾아온 dataset 중 가장 적절하다고 판된되는 것을 찾기 위한 토론 진행 (LeNet 또는 ResNet-50을 사용하여 간단한 실행 포함)

**5월 10일 : Dataset 분류**

## 목표) training/validation/test dataset으로 분배 (Prepare the data)

박성민,전태호 -training/validation/test dataset으로 분배

조무곤, 임현만- Baseline 학습 및 결과분석 (LeNet5/ResNet-50)

박성민,전태호,조무곤 임현만공동 - 중간발표전 발표 준비 작업

**5월 17일: Select and train a model**

목표) 각자 학습에 사용될 적절한 모델을 골라 최적의 모델 찾기

박성민,전태호,조무곤 임현만공동 - 학습에 사용할 모델 선택: 각자 model 다수개를 정해 실행 해본뒤 그 중 가 장 성능이 좋은 최종 모델 선택. 근거와 학습 계획 제시.

이후 선택된 최종 model에 대한 근거와 학습 계획의 타당성 및 적절성 검토, 필요시 각 모델의 성능에 대한 재분석과 모델 변경.

**5월 26일 : 모델 최적화 및 분석 목표) 모델 최적화 및 분석**

박성민 - Model과 training hyperparameter의 최적화를 통해 성능 개선.

전태호 - 최적화 과정 제시 및 결과 분석

조무곤 - 학습시간, 예측시간(inference time), 정확도 측면에서 분석

임현만 - 위 작업들을 통한 Model 전체적 성능 분석 및 피드백

**6월 5일: 모델 최적화 마무리**

목표) 모델 최적화 마무리

박성민 - 이전 작업을 통한 분석 결과를 바탕으로 최종 모델 최적화 및 마무리

전태호,조무곤- EMNIST dataset에 포함된 test dataset을 기반으로 한 성능 평가: 주어진 dataset에 대한 학습 최적화 결과 평가 - Epoch에 따른 learning curve 제시.

박성민,전태호,조무곤 임현만 공동 - 각 조의 손글씨 데이터에 대한 prediction 결과 분석 -> 학습 dataset 선택을 포함한 전반적인 CNN 개발 결과에 대한 평가

**2. 프로젝트 내용**

**가. MNIST extended dataset 개요 및 분석**

mnist extended dataset 중에서 byclass를 사용하기로 결정

이유는 0~ｚ 까지 모든 숫자와 문자 총 62개를 분류해둔 데이터셋이 byclass가 유일하게 때문 중간발표 때는 그래도 이런 byclass보다 balanced dataset이 class별로 데이터수 개수가 일정하기 때문에 balanced dataset으로 진행하는 것이 더 유리할 것으로 생각하였으나, 결국 최종 목표가 모든 대소문자와 숫자를 제대로 구별하는 것이 목적이기에 byclass dataset으로 최종 결정하였다.

dataset을 tensorflow로 받기로 하였는데 이유는 csv의 파일 dataset의 크기가 너무 커서 다양한 곳에서 ipynb 프로그램을 돌리는데 번거롭게 때문에 tensorflow로 데이터를 받아오는 것으로 정하였다. 이 과정에서 기존 csv로 받았을때는 np 형태로 바꿔주는 형태로 하였는데, tensorflow의 경우에는 numpy로 들어오기에 이에 따라 코드 수정을 다시 진행하였다.

**문제점**

현재 데이터셋을 다운받아서 한번에 돌릴려고 하면 데이터셋이 너무 크다면서 모델이 돌아가지않고 에러가 뜨는 현상이 발생하였다.

해결 방법으로 의논을 해봤는데, 우선 하나는 데이터셋의 크기가 너무 커서 생긴 문제이니 데이터셋 일부를 나눠 한쪽은 A라는 모델에서 돌리고 다른 일부는 B라는 모델에 돌린 후 이 모델들을 transfer learning 으로 나중에 합치자는 의견이 나왔다.근데 이러한 방법은 학습율이 떨어지는 문제가 있다.resnet-50 모델로 돌려보았는데 데이터셋을 심지어 20만개까지 줄여도 여전히 gpu 부족으로 에러가 발생하였다.

그래서 hands on machine learning 3 책의 13장을 참고한 결과

dataset = tf.data.Dataset.list files(filepaths).repeat(repeat) 으로 데이터를 dataset으로 가져오는 코드를 통해 이 gpu 부족 문제를 해결하였다. 이 코드를 통해 데이터가 메모리에 한 번에 로드되지 않고 필요할 때만 로드됨 따라서 대규모 데이터셋을 다룰 때 유용한데 이를 통홰 돌려보지도 못하는 문제를 해결하였다.

**나. Dataset 분류**

앞서 언급한대로 학습에 사용하는 데이터의 형태는 tf.data.Dataset으로 진행한다. 위 형식으로 만들기 위해서 다음과 같은 과정을 거친다

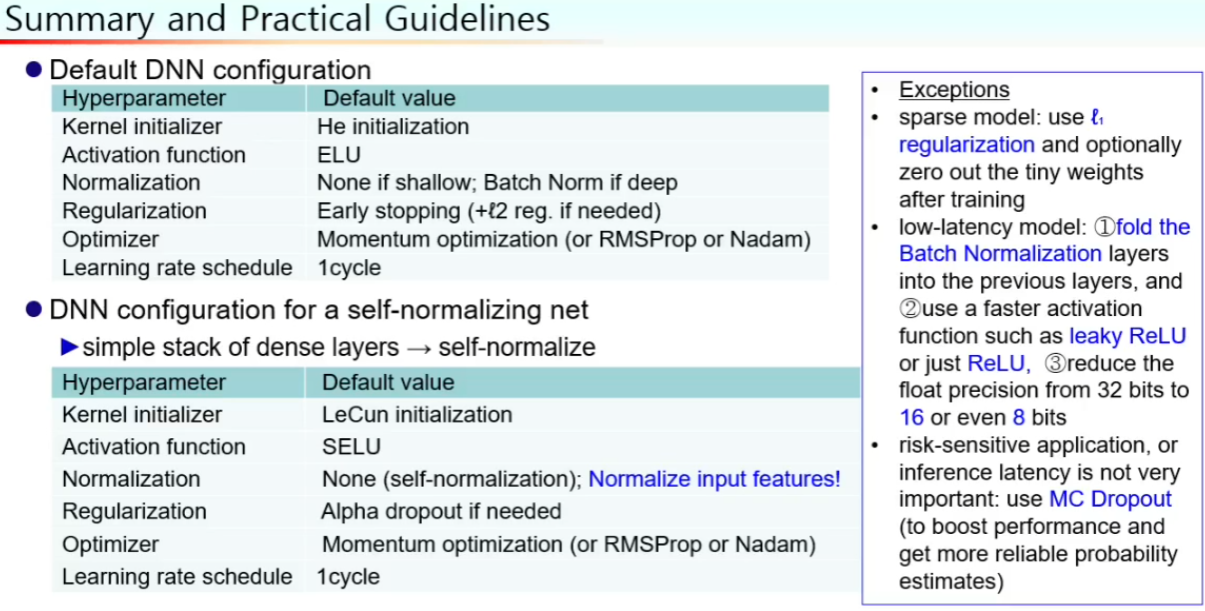
먼저 byclass의 csv로부터 데이터를 읽어 들인다. 이후 데이터를 분류하는데 train\_test\_split 으로 분류하는 대신에 StratifiedShuffleSplit를 이용한다. 이는 byclass의 경우 class마다 들어있는 class의 개수가 서로 상이하여 분류할때 어느 한쪽으로의 쏠림의 현상을 방지하고 초기의 비율을 유지하여 학습을 진행하기 위함이다.

이러한 데이터는이후 npz의 형태로 변한한다.이후 handson machine learning 3 책의 13장의 save\_to\_multiple\_csv\_files의 함수를 이용하여 csv를 10등분으로 나눠 분산처리 가능하게 한다. 마찬가지로 13장의 csv\_reader\_dataset의 함수를 이용하여 tf.data.Dataset의 형태로 변환된다. 이 과정에서 preprocess를 이용하여 모델에 맞게 크기를 조절한다. 또한 각 모델에 내장되어 있는 preprocess\_input 함수를 이용해 scaling을 진행하고 직접만드는 모델의 경우 255로 나눠주어 0~1사이의 값을 가지도록 해준다.

총 data의 수 814255개의 data를 training, validation, test set을 각각 8:1:1 비율로 나눠준다.

**다. Baseline 학습 및 결과분석**

이후 basemodel인 LeNet5와 ResNet-50의 parameter를 여럿 바꾸어 테스트를 진행해보았다. 또한 여러 optimizer를 사용하여 nesteroV,adamW, SGD등을 비교했다. 이 테스트를 위해서 parameter를 바꾸는 기준은 Hands on Machine Learning 3판 의 Practical Guidelines을 따라 바꾸었다.



\*위 그림은 Hands on Machine Learning 2판의 guideline이다.

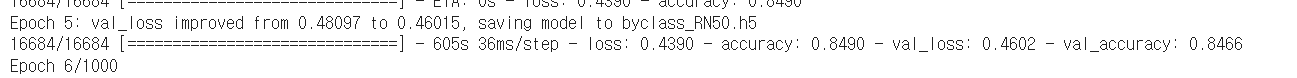
빠른 학습과 과적합을 예방하기 위하여 earlystop을 epoch 5로 할당하였다.

**1)Resnet-50**

**resnet50에서 optimizer를 SGD에서 default parameter**

이렇게 돌려본 결과 epoch 10정도에서 0.67 까지 밖에 안올라가는 것을 확인하여 아무리 SGD여도 너무 오래걸리고 낮기 때문에 SGD로 돌리는 것을 중단하였다.

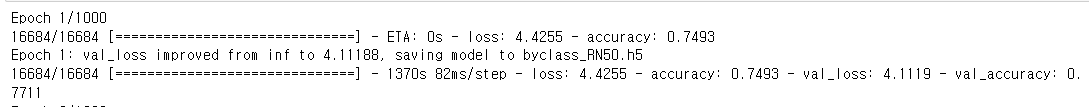
**resnet50에서 optimizer를 Adam 사용**





resnet 50에 optimizer adam을 적용하였다. val\_loss는 0.46 val\_accuracy는 84%로 나왔다.

**Resnet-50 에서 optimizer를 adamax 사용**





**Resnet-50 에서 optimizer를 adamW를 사용**

Epoch 13: val\_loss improved from 0.46503 to 0.46279, saving model to byclass\_RN50.h5

16684/16684 [==============================] - 601s 36ms/step - loss: 0.4383 - accuracy: 0.8475 - val\_loss: 0.4628 - val\_accuracy: 0.8402

**optimizer를 RMSprop 사용**

resnet50 deafult에선 에포크가 5일 때 가장 좋은 결과가 나왔고 train\_loss 0.4390 train\_accuracy 0.8490 val\_loss 0.3832 val\_accuracy 0.8466

| optimizer | loss | accuracy | val\_loss | val\_accuracy |
| --- | --- | --- | --- | --- |
| SGD | 너무 긴 시간이 걸려 중단 | | | |
| adamax | 4.4255 | 0.7493 | 4.119 | 0.7711 |
| adamW | 0.4383 | 0.8475 | 0.4628 | 0.8402 |
| RMSprop | 0.4390 | 0.8490 | 0.3832 | 0.8466 |

표1. optimizer별 model의 성능(Resnet-50)

**2) LeNet-5**

**optimzer: adamW**

epoch 18에서 제일 좋은 결과가 train\_loss 0.3495 train\_accuracy 0.8678 val\_loss 0.3832, val\_accuracy가 0.8860으로 step당 6ms가 나왔다.

**optimizer: nesteroV**

epoch 32에서 제일 좋은 결과가 89s 5ms/step - loss: 0.3377 - accuracy: 0.8720 - val\_loss: 0.3805 - val\_accuracy: 0.8618

**optimizer: adam**

epoch 5에서 제일 좋은결과 train\_loss 0.3669 train\_accuracy 0.8863 val\_loss 0.3932 val\_accuray 0.8554 step당 6ms가 나왔다.

| optimizer | loss | accuracy | val\_loss | val\_accuracy |
| --- | --- | --- | --- | --- |
| adamW | 0.3495 | 0.8678 | 0.3832 | 0.8860 |
| nesteroV | 0.3377 | 0.8720 | 0.3805 | 0.8618 |
| adam | 0.3669 | 0.8863 | 0.3932 | 0.8554 |

표2. optimizer별 model의 성능(LeNet-5)

**activation function: selu**

Epoch 13/1000 16683/16684 [============================>.] - ETA: 0s - loss: 0.6486 - accuracy: 0.7904 Epoch 13: val\_loss improved from 0.47165 to 0.45630, saving model to byclass\_lenet\_selu.h5 16684/16684 [==============================] - 216s 13ms/step - loss: 0.6486 - accuracy: 0.7904 - val\_loss: 0.4563 - val\_accuracy: 0.8430

**activation function: elu**

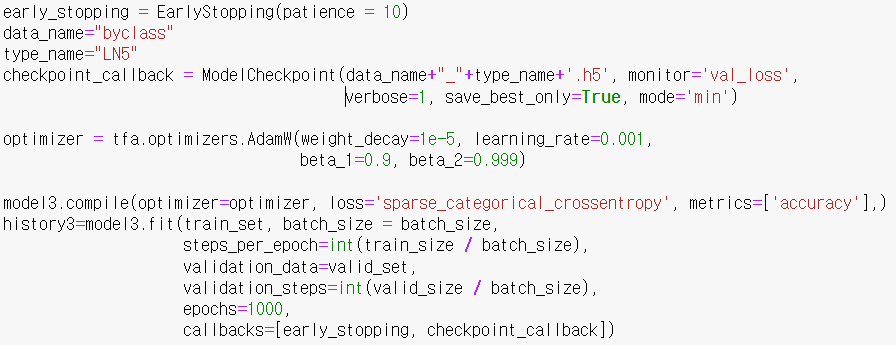
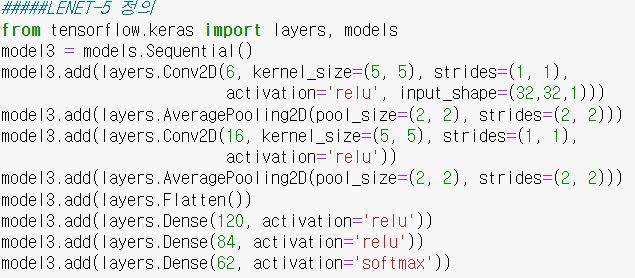
Epoch 6/1000 16680/16684 [============================>.] - ETA: 0s - loss: 0.5392 - accuracy: 0.8215 Epoch 6: val\_loss improved from 0.44117 to 0.42640, saving model to byclass\_lenet\_elu.h5 16684/16684 [==============================] - 210s 13ms/step - loss: 0.5392 - accuracy: 0.8216 - val\_loss: 0.4264 - val\_accuracy: 0.8508

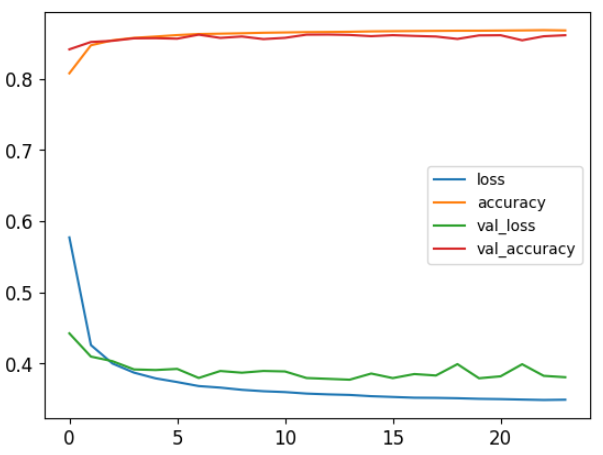
진행하는 과정에서 nesteroV는 정확도의 의미있는 변화가 없어서 중단하였다.

| activation | loss | accuracy | val\_loss | val\_accuracy |
| --- | --- | --- | --- | --- |
| selu | 0.6486 | 0.7904 | 0.4563 | 0.8430 |
| elu | 0.5392 | 0.8216 | 0.4264 | 0.8508 |

표3. activation 별 model의 성능(LeNet-5)

여러 시도 중 가장 좋은 결과가 나온 LeNet-5 에서 optimizer를 adamW를 적용한 모델을 최종 Baseline 모델로 사용하기로 결정하였다.



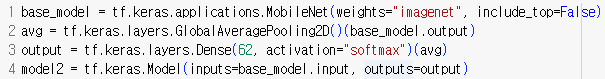
****

**\*Base Line 모델인 lenet-5의 history plot**

**라. 학습에 사용할 모델 선택**

**A) Pre Trained Model**

**<MobileNet>**



**모든 레이어 잠금하였을 때의 결과-**

Epoch 4/1000

16683/16684 [============================>.] - ETA: 0s - loss: 0.4433 - accuracy: 0.8443

Epoch 4: val\_loss improved from 0.47292 to 0.46950, saving model to byclass\_mobile\_adam.h5

16684/16684 [==============================] - 292s 18ms/step - loss: 0.4433 - accuracy: 0.8443 - val\_loss: 0.4695 - val\_accuracy: 0.8399

**모든 레이어 잠금을 풀었을 때의 결과-**

Epoch 1/1000 16681/16684 [============================>.] - ETA: 0s - loss: 0.4465 - accuracy: 0.8441 Epoch 1: val\_loss improved from inf to 0.46571, saving model to byclass\_mobile\_adam\_all\_no\_freeze.h5 16684/16684 [==============================] - 216s 13ms/step - loss: 0.4466 - accuracy: 0.8440 - val\_loss: 0.4657 - val\_accuracy: 0.8382

**나눠서 레이어 잠금을 풀었을 때의 결과-**

for layer in base\_model.layers[83:]:

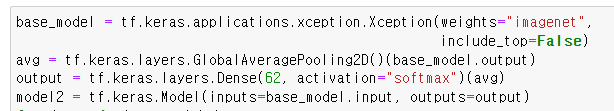
layer.trainable = True

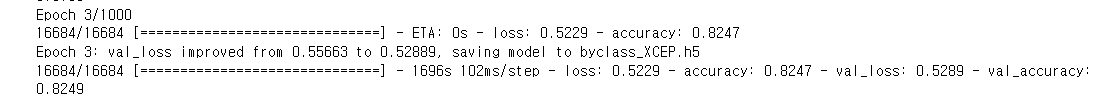
for layer in base\_model.layers[77:]:

layer.trainable = True

Epoch 1/1000 16678/16684 [============================>.] - ETA: 0s - loss: 0.4463 - accuracy: 0.8441 Epoch 1: val\_loss improved from inf to 0.44001, saving model to byclass\_mobile\_adam\_no\_freeze.h5 16684/16684 [==============================] - 207s 12ms/step - loss: 0.4464 - accuracy: 0.8441 - val\_loss: 0.4400 - val\_accuracy: 0.8440

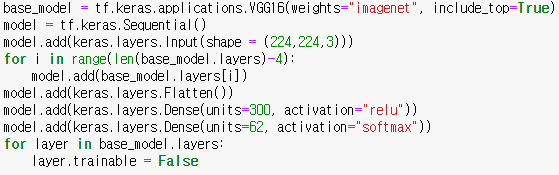
**<Xception >**





**<VGG-16>**

모델은 pre trained model을 비교하기 위해 VGG-16 모델을 사용하였다.



**모든 레이어 잠금하였을 때의 결과-**

Epoch 1: val\_loss improved from inf to 0.60571, saving model to byclass\_vgg16.h5

16684/16684 [==============================] - 545s 33ms/step - loss: 0.6665 - accuracy: 0.8022 - val\_loss: 0.6057 - val\_accuracy: 0.8232

**모든 레이어 잠금을 풀었을 때의 결과-**

Epoch 1/1000

16684/16684 [==============================] - ETA: 0s - loss: 0.6246 - accuracy: 0.8167

Epoch 1: val\_loss improved from inf to 0.69927, saving model to byclass\_vgg16\_no\_freeze.h5

16684/16684 [==============================] - 1021s 61ms/step - loss: 0.6246 - accuracy: 0.8167 - val\_loss: 0.6993 - val\_accuracy: 0.8127

오히려 모든 모델을 풀었을 때의 결과가 더 안좋아진 것을 확인 할 수 있다.

| pre-trained-model | loss | accuracy | val\_loss | val\_accuracy |
| --- | --- | --- | --- | --- |
| mobilenet(전) | 0.4433 | 0.8443 | 0.4695 | 0.8399 |
| mobilenet(후) | 0.4466 | 0.8440 | 0.4657 | 0.8382 |
| xception(전) | 0.5229 | 0.8247 | 0.5289 | 0.8249 |
| vgg16(전) | 0.6665 | 0.8022 | 0.6057 | 0.8232 |
| vgg16(후) | 0.6246 | 0.8167 | 0.6993 | 0.8127 |

표4. pre trained model 별 model의 성능(풀기 전/후)

표4의 결과 위 3가지의 모델의 결과에서는 mobilenet을 이용하였을때 가장 좋은 성능을 보이는 것을 확인 할 수 있었다. pre trained model중 가장 좋은 성능을 보이는 모델로 mobilnet을 골랐다.

**B) handmade model**

**모델을 만들 때 두가지를 고민하였는데,**

1. 모델이 받아들이는 입력을 입력이미지의 크기에 맞춘 28x 28로 진행하거나.

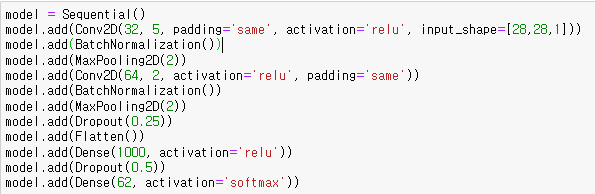
2. 모델이 받아들이는 입력을 224x224로 정하고 입력이미지를 그 크기에 맞추어서 스케일링 해줄지를 고민하였는데, 고민 후 둘다 돌려보기로 결정하였다.

돌려본 결과 한 epoch에 걸리는 시간이 hour단위로 걸리고 accuracy도 오르는 속도가 느리기 때문에 이 선택은 포기하고 1번으로 가기로 결정하였다.

만든 모든 모델에 Early Stopping을 10을 주어서 결과에 변함이 없으면 학습을 중단하기로 설정하였다. 그리고 어떤 모델에는

**모델 1-1**

에포크는 9에포크일 때 제일 좋은 결과가 나왔고, 123s 7ms/step - loss: 0.3845 - accuracy: 0.8610 - val\_loss: 0.3749 - val\_accuracy: 0.8652 가 나왔다.

**모델 1-2**

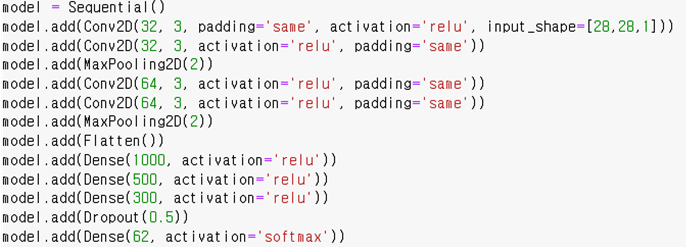
Epoch 6/1000

16674/16684 [============================>.] - ETA: 0s - loss: 0.4145 - accuracy: 0.8527

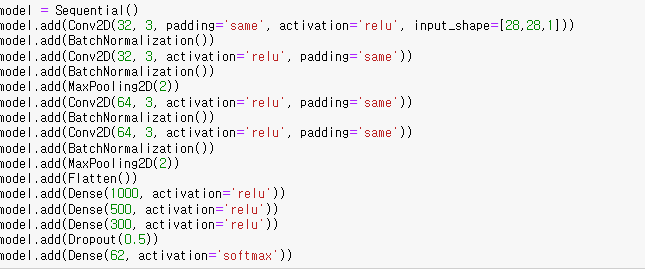
Epoch 6: val\_loss improved from 0.37264 to 0.36589, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 114s 7ms/step - loss: 0.4146 - accuracy: 0.8527 - val\_loss: 0.3659 - val\_accuracy: 0.8678

**모델 2-1**



에포크는 3에포크일 때 제일 좋은 결과가 나왔고, 143s 9ms/step - loss: 0.4832 - accuracy: 0.8339 - val\_loss: 0.4072 - val\_accuracy: 0.8543 이 나왔다.

**모델 2-2**

Epoch 2/1000

16684/16684 [==============================] - ETA: 0s - loss: 0.4142 - accuracy: 0.8554

Epoch 2: val\_loss improved from 0.40585 to 0.38100, saving model to byclass\_make\_adam.h5

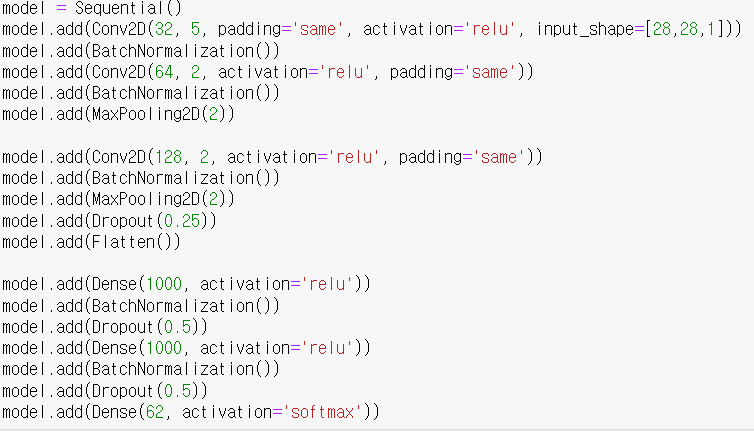
16684/16684 [==============================] - 187s 11ms/step - loss: 0.4142 - accuracy: 0.8554 - val\_loss: 0.3810 - val\_accuracy: 0.8628

**모델 3-1**



에포크는 4에포크일 때 제일 좋은 결과가 나왔고, 80s 5ms/step - loss: 0.3800 - accuracy: 0.8636 - val\_loss: 0.3865 - val\_accuracy: 0.8624 이 나왔다.

**모델 3-2**



Epoch 14/1000

16681/16684 [============================>.] - ETA: 0s - loss: 0.3307 - accuracy: 0.8751

Epoch 14: val\_loss improved from 0.33412 to 0.33223, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 344s 21ms/step - loss: 0.3307 - accuracy: 0.8751 - val\_loss: 0.3322 - val\_accuracy: 0.8764

**모델 4**



에포크는 3에포크일 때 제일 좋은 결과가 나왔고, 463s 28ms/step - loss: 0.4246 - accuracy: 0.8516 - val\_loss: 0.3689 - val\_accuracy: 0.8673 이 나왔다.

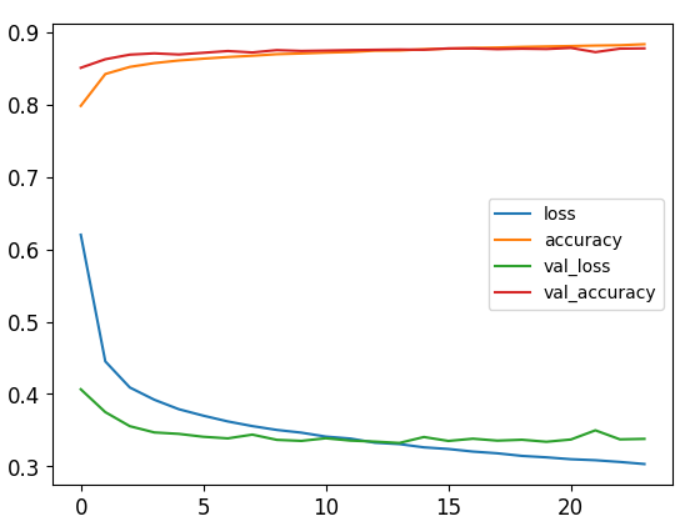
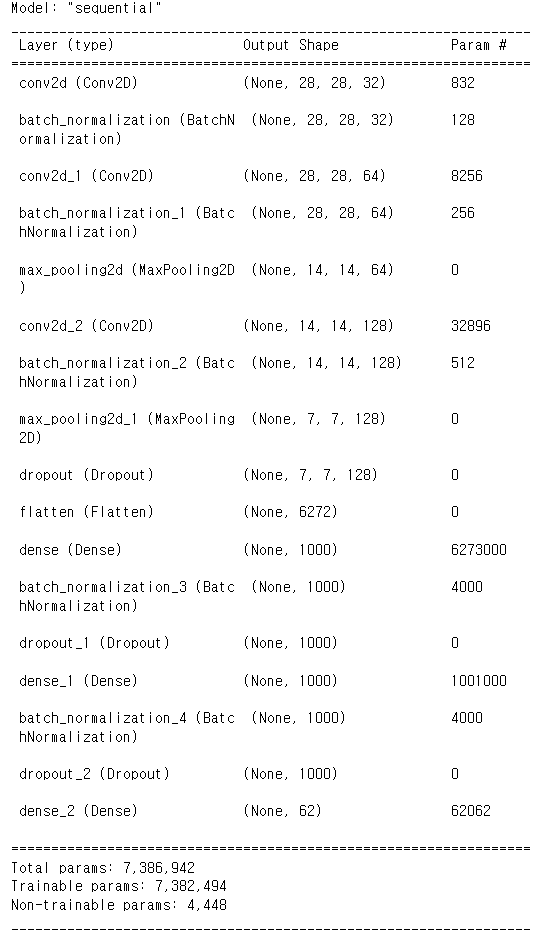
| handmade | loss | accuracy | val\_loss | val\_accuracy |
| --- | --- | --- | --- | --- |
| 모델1-1 | 0.3845 | 0.8610 | 0.3749 | 0.8652 |
| 모델1-2 | 0.4146 | 0.8527 | 0.3659 | 0.8678 |
| 모델2-1 | 0.4832 | 0.8339 | 0.4072 | 0.8543 |
| 모델2-2 | 0.4142 | 0.8554 | 0.3810 | 0.8628 |
| 모델3-1 | 0.3800 | 0.8636 | 0.3865 | 0.8624 |
| 모델3-2 | 0.3307 | 0.8751 | 0.3322 | 0.8764 |
| 모델4 | 0.4246 | 0.8516 | 0.3689 | 0.8673 |

표4. handmade model 별 model의 성능

각 모델 뒤에 ( - ) 가 붙은 모델들은 기존 모델에서 추가로 부분 수정을 진행한 모델들이다.

직접 설계한 모델 중 모델 3-2 가 val\_loss가 0.3322로 가장 좋은 성능을 보이고 있기에 최종 모델로 모델 3-2를 사용하기로 결정하였다.

**모델 3-2** 가 가장 좋은 학습 결과가 나와 3-2를 모델을 선택하여 이 모델을 최적화를 진행하기로 하였다.



**마. 모델 최적화 및 분석**

**1. parameter 하나만 변경**

**1) Learning rate 관련 최적화**

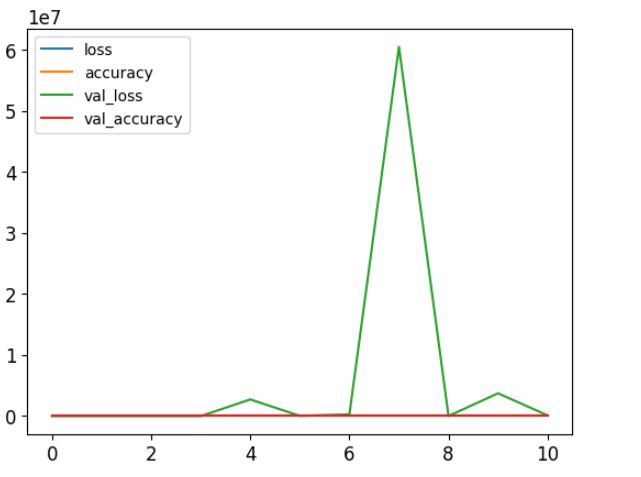
**-OneCycle Scheduler 사용-**

Epoch 1/1000

16683/16684 [============================>.] - ETA: 0s - loss: 0.7652 - accuracy: 0.7705

Epoch 1: val\_loss improved from inf to 0.52115, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 274s 16ms/step - loss: 0.7652 - accuracy: 0.7705 - val\_loss: 0.5211 - val\_accuracy: 0.8333

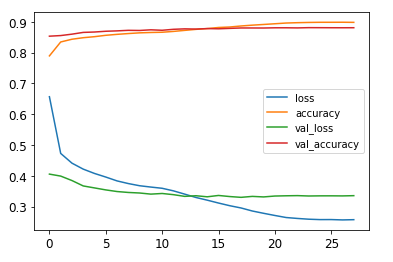
****

**lr설정에 문제가 있어서 이런 이상한 그래프가 등장하였다.**

**-OneCycle Scheduler 개선후 재사용-**

Epoch 18: val\_loss improved from 0.33161 to 0.32986, saving model to byclass\_make\_one\_cy\_re.h5

16684/16684 [==============================] - 204s 12ms/step - loss: 0.2953 - accuracy: 0.8866 - val\_loss: 0.3299 - val\_accuracy: 0.8801

****

****

이 코드에서 max rate가 너무 큰것으로 확인하여 0.001로 바꾸어 학습을 하였더니 정상적인 결과를 출력하였다. 하지만 그래프에서 보이는 것처럼 loss와 val\_loss의 차이가 큰 overfitting이 발생한 것을 확인 할 수 있다. 이러한 위험점 때문에 이 scheduler는 사용하지 않도록 하였다.

**- Exponential\_decay scheduler 사용-**

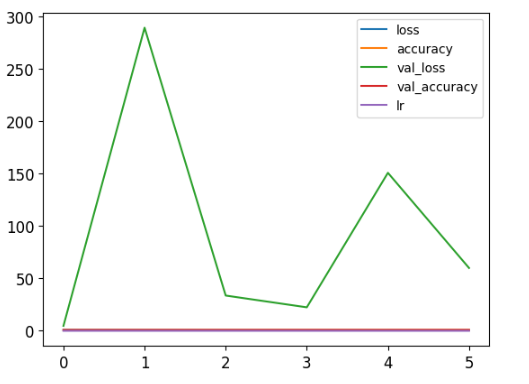
**초기값: learning rate : 0.1 s=20 지정**

Epoch 1/1000

16683/16684 [============================>.] - ETA: 0s - loss: 0.9148 - accuracy: 0.7419

Epoch 1: val\_loss improved from inf to 4.72746, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 243s 14ms/step - loss: 0.9148 - accuracy: 0.7419 - val\_loss: 4.7275 - val\_accuracy: 0.8166 - lr: 0.0100

****

**-> 수정1**

**learning rate : 0.001 , s=20으로 지정**

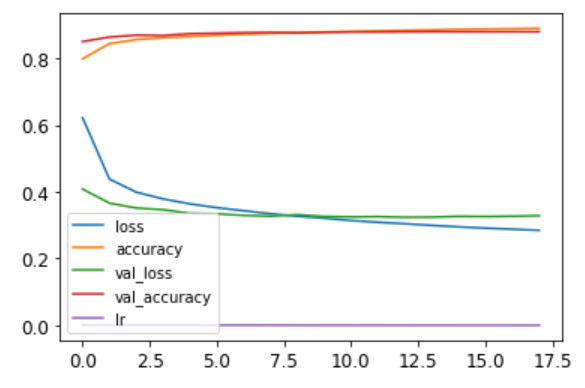
****

Epoch 13/1000

16682/16684 [============================>.] - ETA: 0s - loss: 0.3043 - accuracy: 0.8836

Epoch 13: val\_loss improved from 0.32469 to 0.32356, saving model to byclass\_make\_exp\_re.h5

16684/16684 [==============================] - 329s 20ms/step - loss: 0.3043 - accuracy: 0.8836 - val\_loss: 0.3236 - val\_accuracy: 0.8792 - lr: 2.5119e-04

****

**->수정2**

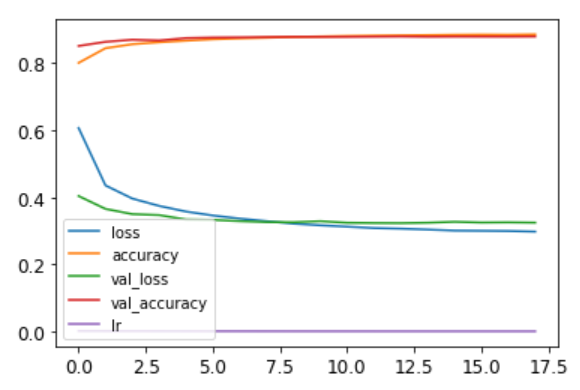
**learning rate: 0.001 , s값을 10으로 지정**

Epoch 13/1000

16683/16684 [============================>.] - ETA: 0s - loss: 0.3063 - accuracy: 0.8832

Epoch 13: val\_loss improved from 0.32311 to 0.32278, saving model to byclass\_make\_exp\_re\_ver2.h5

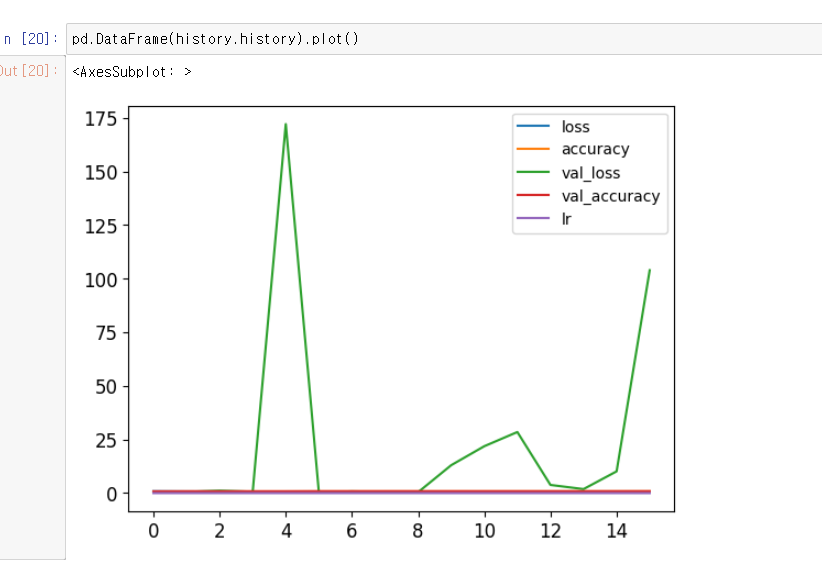
16684/16684 [==============================] - 326s 20ms/step - loss: 0.3063 - accuracy: 0.8832 - val\_loss: 0.3228 - val\_accuracy: 0.8799 - lr: 6.3096e-05

****

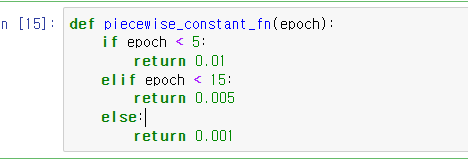
**- Piecewise scheduler 사용-**

Epoch 6: val\_loss improved from 0.57068 to 0.41097, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 246s 15ms/step - loss: 0.5176 - accuracy: 0.8252 - val\_loss: 0.4110 - val\_accuracy: 0.8579 - lr: 0.0050

****

초기에 5에포크까지 learning\_rate 0.01 15에포크까지 learinig\_rate 0.005 그 이후 에포크에 대해서는 0.001의 learning\_rate를 적용하였다.

****

에포크 설정과 각 에포크당 learning\_rate를 수동적으로 설정해야하기에 optimize한 결과를 얻어내기 힘들다고 판단하였다**.**

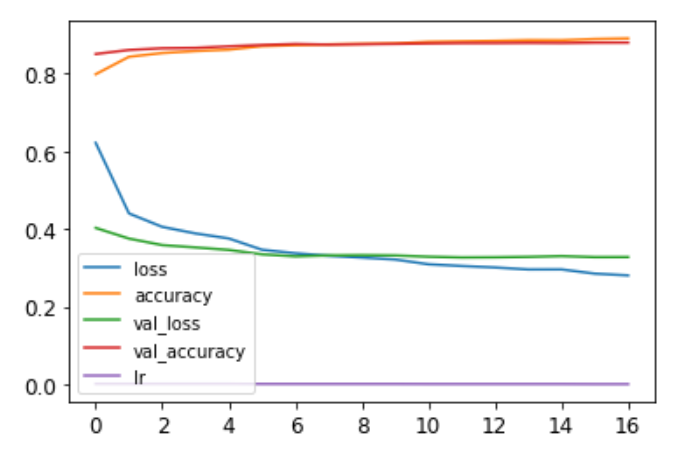
**-수정 1**

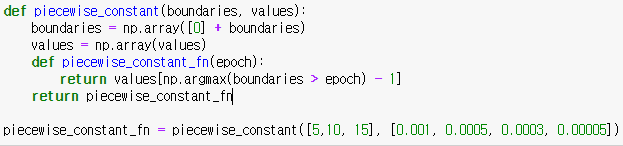
Epoch 12/1000

16682/16684 [============================>.] - ETA: 0s - loss: 0.3045 - accuracy: 0.8833

Epoch 12: val\_loss improved from 0.32844 to 0.32645, saving model to byclass\_make\_adam\_piecewise.h5

16684/16684 [==============================] - 163s 10ms/step - loss: 0.3045 - accuracy: 0.8833 - val\_loss: 0.3265 - val\_accuracy: 0.8787 - lr: 3.0000e-04

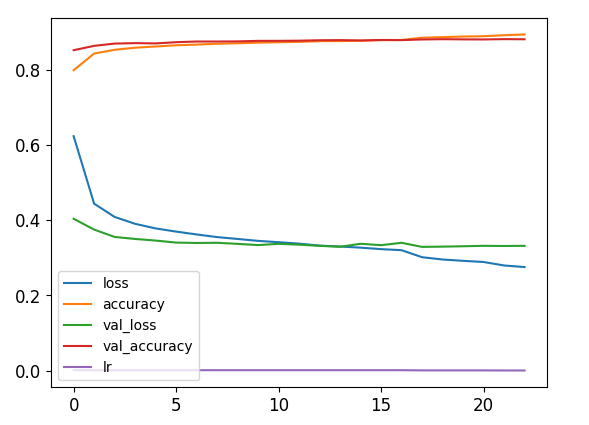
****

****

**- Performance LR scheduler 사용-**

Epoch 18: val\_loss improved from 0.32900 to 0.32865, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 127s 8ms/step - loss: 0.3012 - accuracy: 0.8845 - val\_loss: 0.3287 - val\_accuracy: 0.8798 - lr: 5.0000e-04

****

lr = keras.callbacks.ReduceLROnPlateau(factor=0.5,patience=3)

가장 좋은 val\_ac와 val\_loss 값을 가짐.

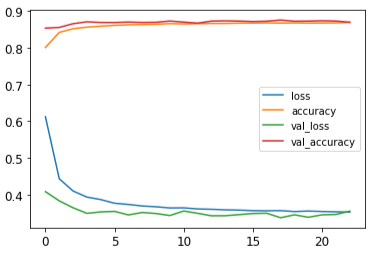
현재 학습 상태에 따라 LR를 변경시켜주기에 결과 도출에 용이

**2) optimizer 변경**

**adamW 사용**

Epoch 18: val\_loss improved from 0.34261 to 0.33731, saving model to byclass\_make\_adamW.h5

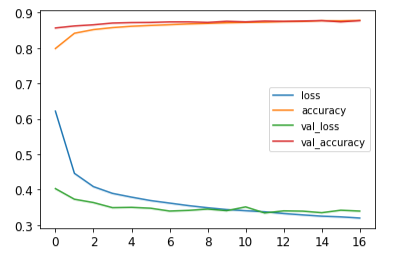
16684/16684 [==============================] - 216s 13ms/step - loss: 0.3567 - accuracy: 0.8675 - val\_loss: 0.3373 - val\_accuracy: 0.8758

****

optimizer 중 adam 에서 개선된 optimizer인 adamW를 사용하였지만 기존 모델의 adam에 비해 성능이 개선되지는 않았다.

**3) activation function 변경**

activation function- swish 사용

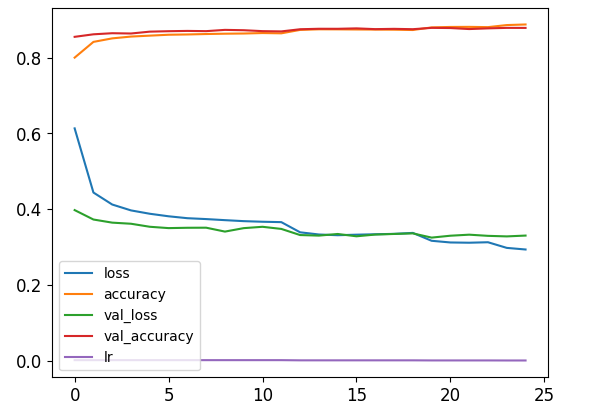
****

이런 그래프가 그려졌는데, 기존 모델에서 성능이 개선되지 않아 activation function 변경은 포기하기로 하였다.

따라서 기존의 activation function인 relu 함수를 이용하기로 하였다.

**2.parameter 두 개 변경**

**LR: Performance scheduler , optimizer: adamW 사용**

****

Epoch 20/1000

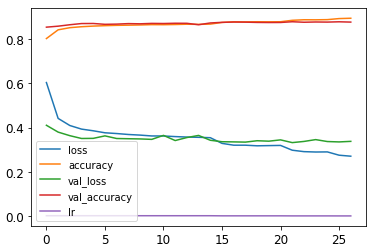
16679/16684 [============================>.] - ETA: 0s - loss: 0.3164 - accuracy: 0.8803

Epoch 20: val\_loss improved from 0.32828 to 0.32482, saving model to byclass\_make\_adam.h5

16684/16684 [==============================] - 139s 8ms/step - loss: 0.3164 - accuracy: 0.8803 - val\_loss: 0.3248 - val\_accuracy: 0.8791 - lr: 2.5000e-04

**기존 model인 optimizer: adam과 LR scheduler: Performance scheduler 를 사용한 것보다 성능이 낮음.**

**LR: performance scheduler, optimizer:adamW activation function: swish 사용**

****

Epoch 22: val\_loss improved from 0.33424 to 0.33197, saving model to byclass\_make\_adamw&swish.h5

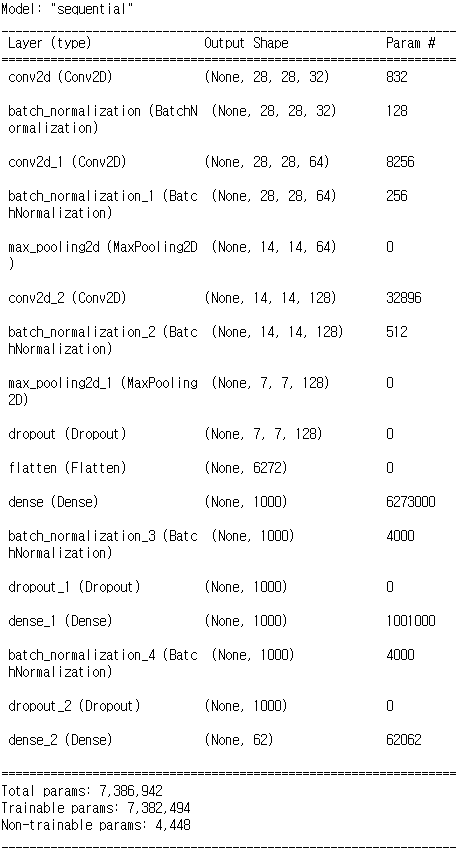
16684/16684 [==============================] - 218s 13ms/step - loss: 0.2976 - accuracy: 0.8857 - val\_loss: 0.3320 - val\_accuracy: 0.8792 - lr: 2.5000e-04

val\_accuray가 0.8950까지 올라간 경우도 있었지만 val\_loss를 기준으로 정하기로 하였기에 가장 좋은 성능을 보여주지는 못하고 있다.

**기존 model인 optimizer: adam과 LR scheduler: Performance scheduler 를 사용한 것보다 성능이 낮음.**

**3. 결과**

**최종 모델**

****

위 사진이 최종적으로 완성한 모델의 summary이다.

최종적으로 우리가 만든 모델 중 가장 최적화가 잘 된 모델은 3-2의 변형본이다.

이 모델의 경우 optimizer와 learning rate scheduler는 각각 adam, Performance LR scheduler를 사용하였다.

baseline 중 가장 성능이 높았던 LeNet-5 의 adamW를 사용한 model의 predict를 진행하였다.

손글씨 데이터 2480개를 기준으로 하여 최종 model의 predict를 진행하였다.

우리가 만든 최종 모델을 손으로 직접 작성한 EMIST DataSet을 읽어들여 모델 평가를 진행하였다.

그 결과 아래와 같은 결과가 나왔다.

**80/80 [==============================] - 1s 7ms/step - loss: 0.3502 - accuracy: 0.8770**

**[0.3502311408519745, 0.876953125]**

**-핸드메이드 데이타에 대한 최종 모델의 loss와 accuracy-**

나온 결과와 비교하기 위해 우리의 Base Line 모델의 Lenet5-adamW 에 대해서도 evaluate를 진행해보았다.

아래와 같은 결과가 나왔습니다.

**80/80 [==============================] - 1s 5ms/step - loss: 0.3723 - accuracy: 0.8460**

**[0.372334678152693, 0.846015789]**

Base Line 모델보다 1.06%정도 나은 결과가 나왔습니다.