목차

목차

주제 소개

저는 손으로 쓴 글자의 이미지를 입력 받으면 해당 손글씨체를 성별과 나이를 판별하여 결과를 돌려주는 인공지능 응용프로그램을 개발해보는 것을 목표로 삼았습니다. 또한 찍어 놓은 사진을 통해 답을 받을 수 있으며 streamlit 라이브러리로 구현한 WebApp에서 글씨를 입력 받아 opencv를 통해 편집하여 실시간으로 답을 받을 수 있게 구현하였습니다.

프로그램 사용 방법

```
# 모델 학습 및 테스트 필요 라이브러리(기본적으로 설치하라 하셨던 것들)
# tensorflow: 2.8.0 (pip install tensorflow==2.8.0)
# numpy: 1.21.5 (conda install numpy==1.21.5)
# sklearn: 1.0.2 (pip install sklearn==1.0.2)

# 손글씨 사진을 실시간으로 받아오기 위한 WebApp을 실행하기 위한 라이브러리들
# cv2: 4.1.2 (pip install opencv-python==4.1.2)
# pandas: 1.3.5 (pip install pandas==1.3.5)
# streamlit: 1.10.0 (pip install streamlit==1.10.0)
# streamlit_drawable_canvas: 0.9.1 (pip install streamlit_drawable_canvas==0.9.1)
# matplotlib: 3.2.2 (conda install matplotlib=3.2.2)
```

```
#각종 변수를 따로 지정하여 실행할 수 있으나 기본적으로 지정되어 있기에
#따로 값을 지정하지 않아도 실행 할 수 있게 하였습니다

# 실행문 - python "train3(vgg11).py"
# -bs = 배치사이즈
# -er = 목표 에러율 해당 에러율이 될때까지 반복하게 되지만 현재 코드에서는 쓰지 않는상태입니다.
# -ep = 해당 값만큼 반복하고 로그를 출력
# -cl = Conv2d Layer 개수 한 계층당 2개의 conv 레이어 포함하며 기본으로 한개가 존재 (ex clayer = 2 -> 1 + 2 * 2 - 6개) 또한 pooling 계층 포함
# -img = 학습하게 될 이미지 사이즈
# -u = 학습 간격간에 유닛 크기 배율
# -p = 해당 모델 이름을 받게되며 해당 모델이름을 통해 폴더를 생성 후 해당 폴더에 모델과
# -a = 기본값은 train으로 학습을 1회 진행하게 되며
#              test를 입력하게 되면 학습을 하지 않고 데이터셋에 일부분을 받아와 테스트를 진행후 models/'path'에 테스트 기록을 저장하게 됩니다.
```

학습을 실행할 경우 단순히 파이썬 파일을 실행시키게 되면 해당 코드는 가중치 파일이 있는지 확인 후 없으면 새로 학습하게 되며 파일들이 있으면 가중치 파일을 불러와 해당 가중치를 기반으로 학습을 진행하게 됩니다

테스트를 실행할 경우 실행문 뒤에 -a test를 붙여서 실행하게 되면 해당 코드는 학습을 진행하지 않고 폴더에서 가중치를 불러오게 됩니다. 다만 가중치 파일이 존재하지 않으면 오류를 일으키게 됩니다.

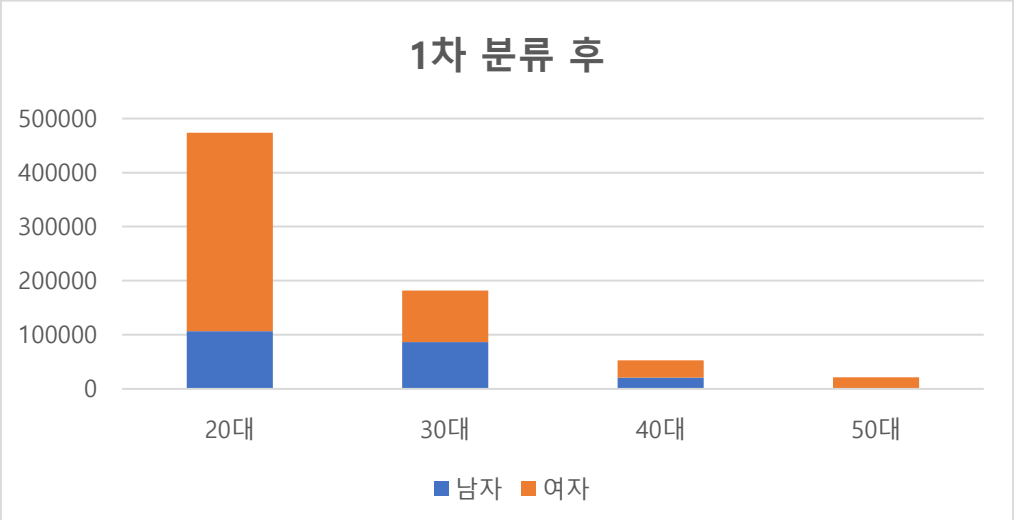
나머지 argument들은 파라미터들을 조정해가며 학습하기위해 추가한 것들로 설명은 해당 사진과 코드에 적어 놓았습니다.

또한 WebApp.py 파일이 있는데 해당 파일은 python webapp.py 를 통해 실행할 수 있으며 웹페이지는 http://localhost:8051 로 접속할 수 있습니다.

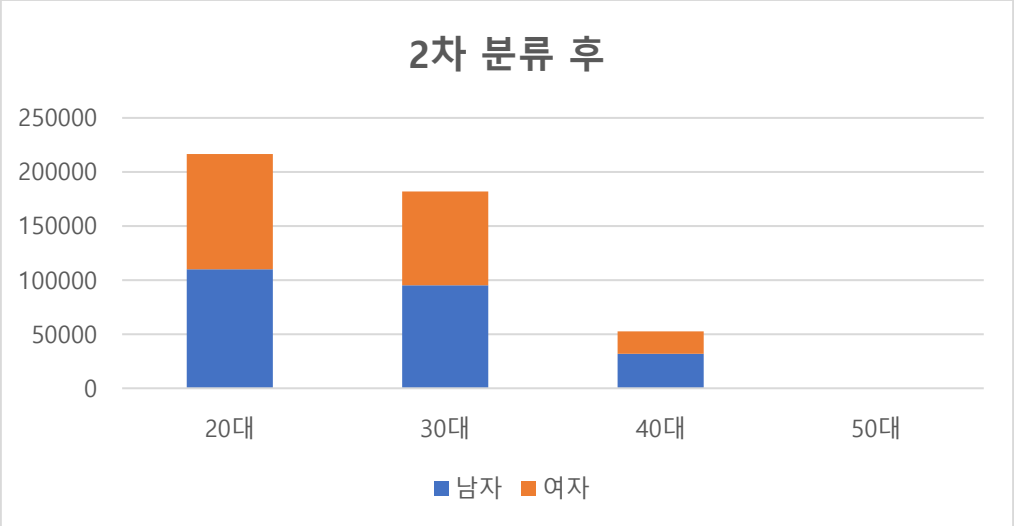
프로그램 구현 내용

데이터 세트

AIHUB에서 제공하는 한국어 글자체 이미지 데이터셋을 사용하였으며 해당 데이터셋에서 손글씨 그 중에서도 남녀 20~40대에 해당되는 이미지들을 분류하여 사용하였습니다



해당 데이터셋 처음 상태는 20대 여자가 과도하게 많으며 50대는 여자 데이터만 있는 상태였습니다. 해당 상태에서는 남자 50대는 분류를 못하고 20대 여자로 과적합이 우려되는 상황이기에 다시 데이터셋 분류가 필요한 상태였습니다.



2차 분류에선 50대 데이터를 제거하였고 20대 이미지들 중 11만개만 사용하도록 하였습니다 여전히 40대 데이터들이 부족하나 제거하게 되면 분리항목이 너무 적어지기에 그대로 진행하게 되었습니다.

Imgs.zip = http://naver.me/5x0xcryJ

Imgs.z01 = http://naver.me/x787JV6L

해당 분류된 데이터셋 이미지들은 해당 링크에서 다운받을 수 있으며 용량이 커 분할압축을 하였습니다.

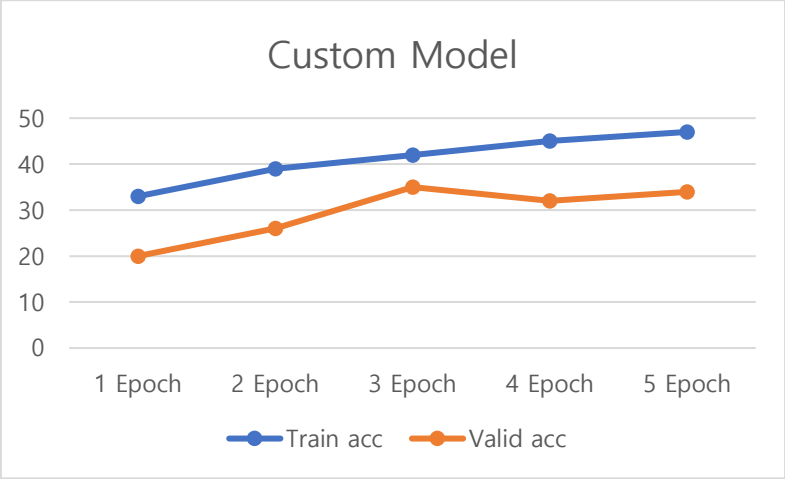
테스트용도의 이미지는 메일로 코드와 같이 첨부하였습니다.

경로는 오른쪽과 같이 분리하였으며

keras.utils.image_dataset_from_directory()를 통해 해당 데이터셋을 불러오게 하여 총 6개의 클래스로 분류되게 학습을 진행하였습니다.

이론적 배경

처음에는 이때까지 하였던 과제를 활용해 해당 신경망을 구성하려 했으나 해당 모델에서는 학습률과 검증률이 50%를 밑돌고 있으며 과적합이 바로 나타났기에 14주차 수업에서 배웠던 RESNET과 VGG 모델을 조사하게 되었습니다.



먼저 **VGGNet**은 (Karen Simonyan, Andrew Zisserman) Very Deep Convolutional Networks for Large-Scale Image Recognition - https://arxiv.org/abs/1409.1556

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

해당 표를 참고하여 직접 구현하게 되었습니다. 이중 **A모델(VGG11)**과 **D모델(VGG16)을** 구현하였습니다.

ResNet은 (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun) Deep Residual Learning for Image Recognition - https://arxiv.org/abs/1512.03385

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

해당 표를 참고하여 직접 구현하였으며 이중 18-layer(ResNet18)모델과 50-layer(ResNet50)을 구현하였습니다.

실제 구현 모델 구조

해당 신경망 계층들을 그대로 적용하기에는 해당 신경망들은 큰 이미지를 대상으로 하여 pooling 계층을 통해 이미지 크기를 적극적으로 줄여 나가기에 약간씩 수정할 필요가 있었습니다.

입력계층에서는 모두 (32, 32, 1)크기에 흑백 이미지를 입력하게 됩니다.

왼쪽 이미지는 VGGNet, 오른쪽 이미지는 ResNet을 구현하고 과적합을 해결하기 위해 마지막에 DropOut 계층을 추가한 모습입니다.

또한 따로 모델에 대한 계층과 각각에 파라미터 개수를 해당 폴더에 summary 텍스트 파일로 저장하였습니다.

ConvNet	
A	D
11 weight layers	16 weight layers
input (32, 32) GrayScale	
Rescaling(1./255)	
conv2d-64	conv2d-64 conv2d-64
MaxPooling(2,2)	
conv2d-64	conv2d-64 conv2d-64
MaxPooling(2,2)	
conv2d-64 BN conv2d-64 BN	conv2d-64 conv2d-64 conv2d-64
MaxPooling(2,2)	
conv2d-64 BN conv2d-64 BN	conv2d-64 BN conv2d-64 BN conv2d-64 BN
None	
None	conv2d-64 BN conv2d-64 BN conv2d-64 BN
None	MaxPooling(2,2)
Flatten	
FC(2048)	FC(4096)
DropOut(.5)	
FC(2048)	FC(4096)
DropOut(.5)	
softmax(6)	

ResNet	
ResNet18	ResNet50
18 weight layers	50 weight layers
input (32, 32) GrayScale	
Rescaling(1./255)	
Conv2d(pool=(7,7), stride(2,2))	
MaxPooling(pool=(3,3), stride=(2,2))	
Conv Block Conv Iden	Conv BN Block Conv BN Iden * 2
Conv Block Conv Iden	Conv BN Block Conv BN Iden * 3
Conv Block Conv Iden	Conv BN Block Conv BN Iden * 5
Conv Block Conv Iden	Conv BN Block Conv BN Iden * 2
GlobalAveragePooling2D	
FC(4096)	
DropOut(.5)	
FC(4096)	
DropOut(.5)	
softmax(6)	

VGG11: 끝에 마지막 conv계층과 Polling계층을 없앴으며 끝에 DropOut 계층을 통해 과적합을 어느정도 해소하게 된다.

VGG16: 중간 Pooling 계층을 없앴으며 11계층 모델과 비슷하게 DropOut계층을 통해 과적합을 해소한다

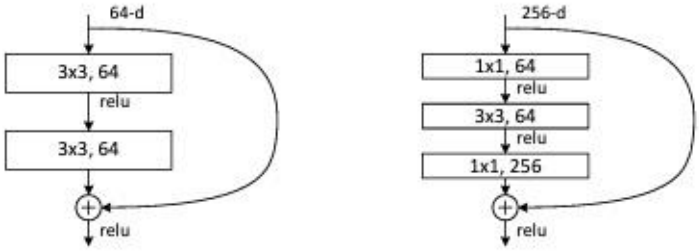


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

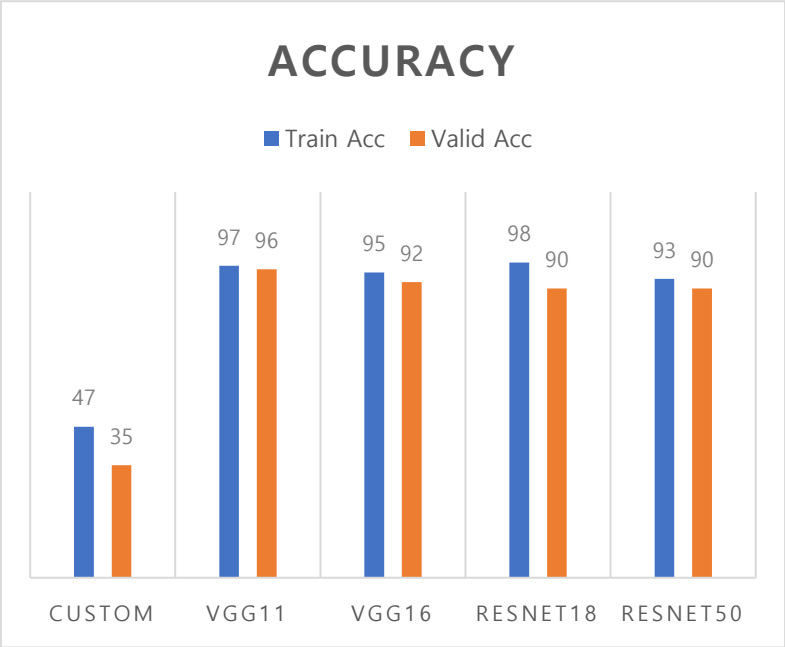
```python
def ConvBlock(self, x, units, stride=1):
    shortcut = x
    shortcut = tf.keras.layers.Conv2D((pow(2, units)*self.unit), (3, 3), strides = (stride, stride), padding = 'same', use_bias = False)(shortcut)
    shortcut = tf.keras.layers.BatchNormalization()(shortcut)
    #
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), strides = (stride, stride), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    #
    x = tf.keras.layers.Add()([x, shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x

def ConvIden(self, x, units):
    shortcut = x
    #
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    #
    x = tf.keras.layers.Add()([x, shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x
def ConvBlock(self, x, units, stride=1):
    shortcut = x
    shortcut = tf.keras.layers.Conv2D((pow(2, units)*self.unit)*4, (1, 1), strides = (stride, stride), padding = 'valid', use_bias = False)(shortcut)
    shortcut = tf.keras.layers.BatchNormalization()(shortcut)
    #
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (1, 1), strides = (stride, stride), padding = 'valid', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D((pow(2, units)*self.unit)*4, (1, 1),  padding = 'valid', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    #
    x = tf.keras.layers.Add()([x, shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x

def ConvBNIden(self, x, units):
    shortcut = x
    #
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (1, 1), padding = 'valid', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Dropout(.1)(x)
    x = tf.keras.layers.Conv2D(pow(2, units)*self.unit, (3, 3), padding = 'same', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Dropout(.1)(x)
    x = tf.keras.layers.Conv2D((pow(2, units)*self.unit)*4, (1, 1),  padding = 'valid', use_bias = False)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    #
    x = tf.keras.layers.Add()([x, shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x
```

ResNet18: 5개의 conv계층중 1,4,5번째 계층에서 이미지 크기를 반으로 줄이게 하며 DropOut을 통해 과적합을 해소하려 하였다.

ResNet50: ResNet18과 같이 1,4,5번째 계층에서 이미지 크기를 반으로 줄이게 되며 DropOut을 통해 과적합을 해소하려 하였다.

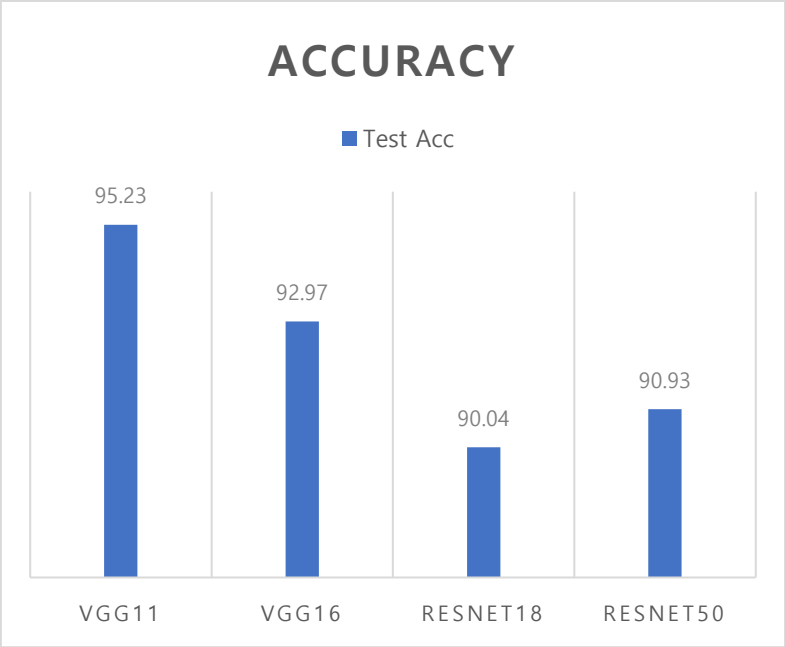
구현 모델 성능평가

첫번째로 단순히 학습률과 검증률을 비교하였으며 학습을 끝낸 기준은 학습데이터셋에 정확도보단 검증 데이터셋에 정확도가 더 이상 유의미하게 오르지 않거나 과적합이 발생하는 경우 학습을 중단하게 되었습니다.



첫번째 성능 평가에서는 VGG11모델이 가장 높은 검증률을 보여주었습니다.

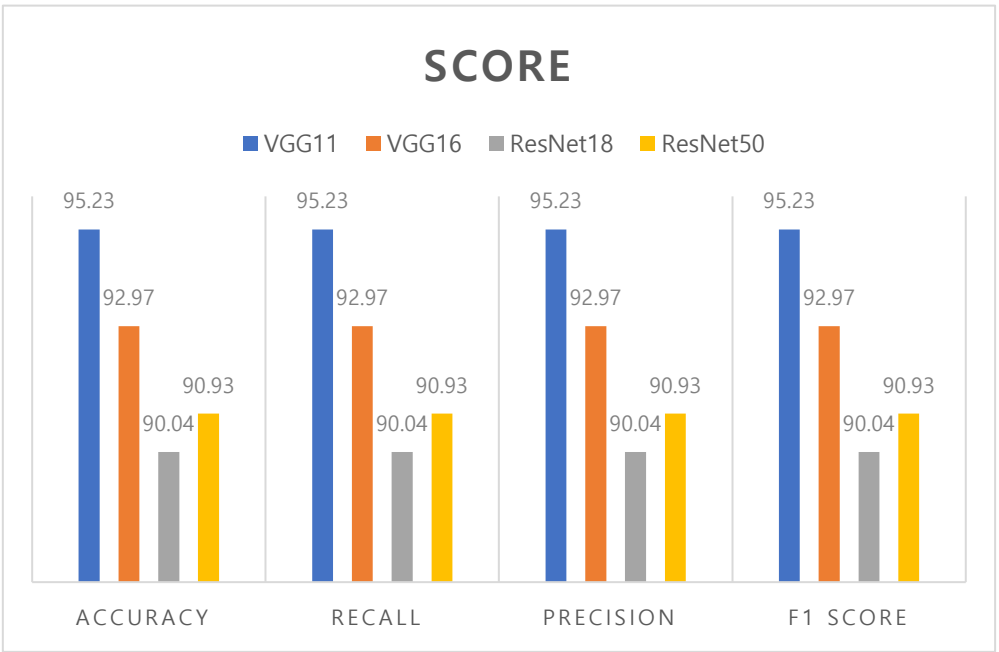
두번째로 정확도를 비교하게 되었으며 테스트 데이터셋은 전체 데이터셋에 1/100 총 4510개의 이미지를 예측하게 하였습니다.



두번째 성능평가에서도 VGG11이 가장 높은 정확도를 보여주게 되었습니다. 해당 테스트 로그는 폴더마다 따로 텍스트파일로 저장하였습니다.

세번째 성능평가는 사이킷런 패키지에서 지원하는 각종 성능평가 함수를 통해 얻은 점수들입니다.

종류는 Accuracy (정확도), Recall (재현율), Precision (정밀도), F1 Score입니다.



해당 성능 평가에서도 VGG11이 가장 높은 점수를 보여주었습니다. 다만 모든 성능 평가 방법에 따른 결과가 똑같은데 원래는 이진분류에서 사용하는 방법이라 똑같게 나온 것 같습니다.

결론

따로 구현하였던 모델을 합쳐 총 5가지 모델을 텐서플로우 라이브러리로 구현하고 학습 후 테스트해보았으며 가장 성능이 높다 판단된 것은 VGG11 이였습니다. 예상으로는 학습은 느려도 ResNet50이 가장 높은 성능을 보일 것이라 예상하였지만 한 번 학습 반복에 시간이 가장 오래 걸리는 것은 맞으나 성능은 3번째였습니다.

이유는 작은 이미지를 처리하는 데에 있어 계층 최적화를 부족하게 한 것과 과적합을 해결하지 못한 것 그리고 데이터셋에 대한 전처리가 부족한 것 이 3가지로 보고 있습니다.

첫번째로 vgg11은 여러 conv계층들을 없애고 이미지 사이즈를 줄이는 부분을 줄여나가며 최적화를 진행하였으나 다른 계층은 이미지 사이즈를 줄이는 부분만 수정하였기에 최적화가 부족하지 않았나 생각이 듭니다.

두번째로 VGG16과 ResNet이 학습을 진행할수록 과적합이 나타나게 되는데 마지막 DropOut 계층을 추가한 것만으로는 부족하였던 것으로 생각하게 됩니다.

세번째로 데이터셋 이미지가 배경은 흰색에 글자가 검은색 이였는데 이를 출력해보면 흰색부분이 255 검은색이 0으로써 MNIST 데이터셋처럼 배경을 검은색 글자를 흰색으로 하는 전처리를 좀더 해야 됐다고 생각됩니다.

또 개인적인 아쉬움으로 코드를 더 깔끔히 쓰고 싶었으나 구현이 우선되어 중간중간 쓸모없는 변수가 있게 되어 아쉬움이 남게 되었습니다.

다만 그래도 VGG11에 각 클래스별 정확도가 높아 구현 결과로서는 목표인 90%에 도달하였다고 생각됩니다.