REPORT

인공지능

-Termproject-

제출일	2021. 06. 14	전 공	글로벌SW융합
과 목	인공지능	학 번	2017114696
담당교수	박혜영	이 름	서혜교

목 차

Ⅰ. Task 1 데이터 분석

1. Data info

데이터는 총 5000개로 5개의 고유라벨을 가지며, 라벨마다 각각 1000개씩의 데이터를 가지고 있었습니다.

2. Data Split

pca와 lda를 위해 데이터를 정렬하고, train과 test set을 8:2 로 나누어 주었습니다.

Ⅱ. 데이터 스케일링

데이터 스케일링에는 여러 종류가 있는데, 잘 알려진 4가지 종류를 적용한 후 비교해보겠습니다.
모든 Scaling은 test data가 포함된 전체 dataset이 아닌 오로지 train data에 대해서만 fit되어야합니다. 이후 train data와 test data 각각을 스케일링합니다. 모델이 학습을 하는 과정에서 어떠한 방식으로든 test data를 사용하게 되어서는 안됩니다. 이 점 유의하며 스케일링 하지 않은 것으로 정확도를 측정해보겠습니다.
base model은 scaling에 민감한 SVM로 지정하였습니다

1. RobustScaler

각 특성들의 중앙값을 0, IQR(제3사분위수-제1사분위수)을 1로 스케일링합니다. StandardScaler와 비슷하지만, 이상치의 영향을 최소화합니다.

2. StandardScaler (Standardization, 표준화)

각 특성의 평균을 0, 분산을 1로 스케일링합니다. 즉 데이터를 정규분포로 만듭니다. 하한값과 상한값이 존재하지 않을 수 있기에, 어떤 알고리즘에서는 문제가 있을 수 있습니다. 회귀보다 분류에 유용합니다

3. MinMaxScaler (Normalization, 정규화)

각 특성의 하한값을 a, 상한값을 b로 스케일링합니다. a=0, b=1일 경우 Normalization으로 표기할 때도 있습니다. 분류보다 회귀에 유용합니다.

4 MaxAbsScaler

각 특성을 절대값이 0과 1사이가 되도록 스케일링합니다. 즉, 모든 값은 -1과 1사이로 표현되며, 데이터가 양수일 경우 MinMaxScaler와 같습니다.

5. SVM을 이용한 Accuracy비교

Scaler종류	미적용	Robust	Standard	MinMax	MaxAbs
accuracy	0.921	0.713	0.914	0.921	0.921

dataset이 양수값을 가지고 있기에 MaxAbs와 MinMax Scaler적용결과가 같음을 확인할 수 있습니다.
Scaling 분석결과 스케일링하지 않은 것과 같거나, 오히려 나쁘기도 하여, 스케일링은 적용하지 않기로 했습니다.

Ⅲ. 특징 추출

1. PCA

Unsupervised learning(비지도 학습)의 일종으로, 독립변수들 사이에 correlation을 없애고, 숨은 latent variable을 찾아내거나, 노이즈(noise)를 줄일 때 사용합니다.

2. LDA

LDA는 Classification과 차원 축소까지 동시에 사용하는 알고리즘입니다.LDA는 지도학습으로서 label값도 학습시킬 것 입니다. LDA에서의 핵심은 classification을 할 때 클래스 내의 분산은 최소가 되도록 하되, 클래스끼리의 분산은 최대가 되도록 한다는 것입니다. 즉, LDA를 통해 하나의 축으로 transformation된 데이터들이 같은 클래스 내에는 그 값의 차가 최소가 되도록 하며, 다른 클래스끼리는 값의 차가 크게 하는 축을 찾는 것입니다.

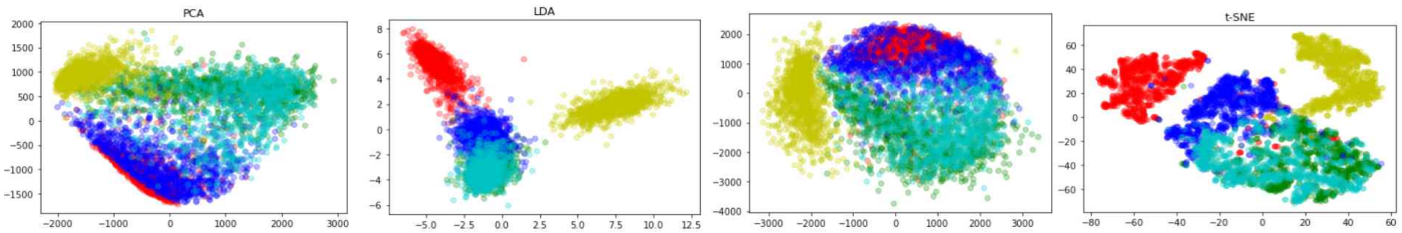
3. MDS

Unsupervised learning(비지도 학습)의 일종으로, numeric할 때만 사용가능하며, 거리(euclidean)를 바탕으로 차원축소합니다. MDS를 이용하여 데이터를 시각화 하는 방법의 가장 큰 장점은 바로 데이터들의 유사도를 확인할 수 있다는 점입니다. 비슷한 값을 가지는 데이터들은 가까이 있을 것입니다.

4. t-SNE

Unsupervised learning(비지도 학습)의 일종으로, 거리기반(확률적 거리)을 최대한 보존합니다, Gaussian이 아닌 t-분포를 쓰고, 단점은 시간이 많이 걸립니다.

아래는 전체 데이터를 시각화한 것입니다. 추후 실제 모델학습 시 사용할 PCA 적용입력값과 LDA 적용입력값은 코드 상에 있습니다.



PCA,LDA,MDS,t-SNE(n_component=2)

Ⅳ. 모델링

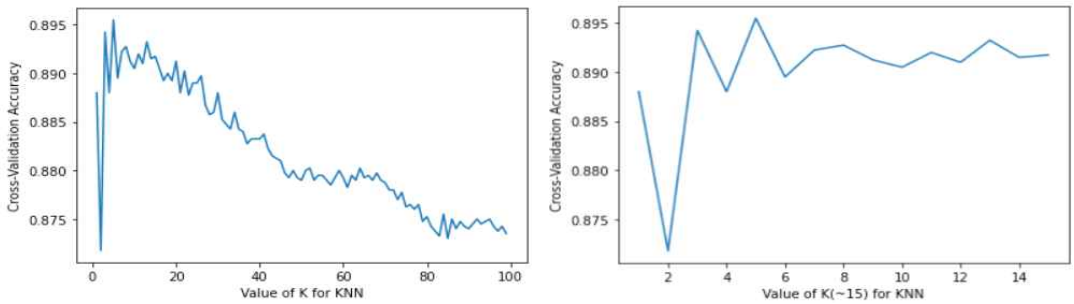
1. Bayes Classifier

Bayes 분류기에 아무 처리도 하지 않은 데이터와, PCA, LDA를 한 데이터를 넣고 학습시켰을 때, 예측값-실제값을 뺀 절대값의 총합을 오차로 하여 비교해보았을 때 LDA를 거친 입력이 가장 좋은 성능을 내었습니다.

	Bayes	PCA input Bayes	LDA input Bayes
오차	0.287	0.114	0.109

2. K-NN

최적의 K값을 찾기 위하여, 10 Fold cross-validation을 통해 1~100까지의 k값의 Accuracy를 비교해보았습니다.



결과는 위 그래프에서 확인할 수 있듯, k=5일 때 가장 좋은 score를 냈으며 이 최적의 k값으로, K-NN 실행한 결과 아래와 같습니다. Bayes와 같은 오차산출방법으로 비교한 결과, PCA를 거친 입력이 가장 좋은 성능이였습니다.

	Bayes	PCA input Bayes	LDA input Bayes
오차	0.11	0.107	0.118

3. MLP

다층 퍼셉트론(multi-layer perceptron)은 퍼셉트론으로 이루어진 층(layer) 여러 개를 순차적으로 붙여놓은 형태입니다. 다층 퍼셉트론은 Feed Forward Neural Network의 가장 기본적인 형태이며, 이는 입력층에서 출력층으로 오직 한 방향으로만 연산 방향이 정해져 있는 신경망을 말합니다.

3.1 Model Architecture

모델 생성시 activation function은 'relu'로 128개 node로 이루어진 2개의 hidden layer와 activation function='softmax'인 5개 node로 이루어진 output layer, batchsize=128, epoch=100으로 고정했습니다. 대신 callback method로 'earlystopping'을 적용하였습니다.

아래 3가지 조건을 달리하여 결과(accuracy)를 비교분석해보겠습니다.

1) Dropout (0.2), (0.5) 두가지 case + BatchNormalization(여기선 BN이라 명시) 적용
2) momentum(0.5) (0,9) 조정 (momentum 조정을 위해 optimizer=RMSprop 사용)
3) one-hot encoding 적용

original	dropout & BN X	dropout(0.2) & BN	dropout(0.5) & BN
momentum 0.0	0.8980000019073486	0.9079999923706055	0.9139999747276306
momentum 0.5	0.9020000100135803	0.9169999957084656	0.9210000038146973
momentum 0.9	0.9020000100135803	0.9039999842643738	0.9229999780654907

One-hot vector	dropout & BN X	dropout(0.2) & BN	dropout(0.5) & BN
momentum 0.0	0.9100000262260437	0.9049999713897705	0.8939999938011169
momentum 0.5	0.9160000085830688	0.9110000133514404	0.8519999980926514
momentum 0.9	0.9039999842643738	0.8759999871253967	0.8970000147819519

one-hot encoding을 하지않은 원래 데이터에, 모델 구조에는 Dropout(0.5)와 BatchNormalization을 적용하고, optimizer의 momentum parameter를 0.9로 조정하였을 때 가장 좋은 정확도(accuracy)가 나왔습니다.

4. CNN

CNN(Convolutional Neural Network)은 이미지를 분석하기 위해 패턴을 찾는데 유용한 알고리즘으로 데이터에서 이미지를 직접 학습하고 패턴을 사용해 이미지를 분류합니다. CNN의 핵심적인 개념은 이미지의 공간정보를 유지하며 학습을 하며,CNN은 필터링 기법을 인공 신경망에 적용함으로써 이미지를 더욱 효과적으로 처리할 수 있습니다.

4.1 Model Architecture

모델 생성시 특징추출을 위한 단계로, activation function은 'relu'로 32개의 3x3 kernel(filter)를 지닌 convolution layer와 MaxPooling사용한 2x2필터를 쓰는 pooling layer, 다시 64개(3x3)의 filter를 지닌 convolution layer, pooling layer, 또 다시 64개(3x3) convolutioin layer로 이루어져있으며, 이미지 분류단계로 Fully connected된 Flatten Layer와 5개 노드의 Softmax Layer로 이루어져있습니다.

optimizer='adam', batchsize=128, epoch=100으로 고정하고 callback method로 'earlystopping'을 적용하였습니다. Dropout&Batch Normalization만 조정한 그 결과(accuracy,정확도)는 아래의 표와 같습니다.

dropout & BN X	dropout(0.2) & BN	dropout(0.5) & BN
0.9280	0.9400	0.9340

5. AutoEncoder

딥러닝에서의 비지도 학습이라고 할 수 있는 AutoEncoder는 출력이 입력과 동일한 특수한 유형의 신경망 아키텍처입니다.아래 Usecase에서 시행될 내용으로 hidden layer의 뉴런 수를 input layer(입력층)보다 작게해서 데이터를 압축(차원 축소)한다거나, 입력 데이터에 노이즈(noise)를 추가한 후 원본 입력을 복원할 수 있도록 네트워크를 학습시키는 AutoEncoder가 있습니다. 그 외에도 여러 AutoEncoder가 있지만 이 두가지만 사용해볼 예정입니다.

5.1 AutoEncoder Architecture

- Encoding Architecture : 인코더 아키텍처는 노드 수 감소하고 궁극적으로 Latent View Representation으로 감소하는 일련의 계층으로 구성됩니다.
- Latent View Repersentation: latent view는 input이 감소되고 정보가 보존되는 가장 낮은 level입니다.
- Decoding Architecture: 디코딩 아키텍처는 인코딩 아키텍처의 미러 이미지이지만(node수 반전형태처럼 같아야함), 모든 계층의 노드 수가 증가하고 궁극적으로 거의 유사한 입력을 출력합니다.

5.2 AutoEncoder UseCase 1 : Image Reconstruction

1) AutoEncoder Architecture 생성

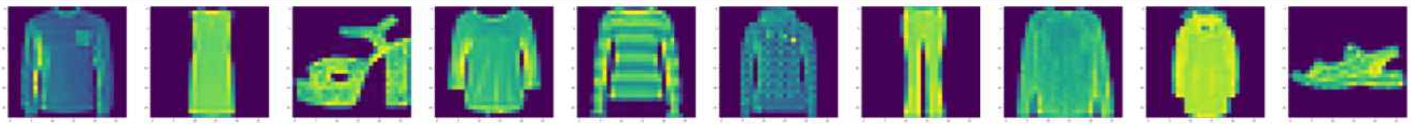
AutoEncoder Architecture를 생성하겠습니다. 인코딩 부분은 1500, 1000, 500개의 노드가 있는 세 개의 계층으로 구성됩니다. 인코딩 아키텍처는 5개의 노드로 구성된 latent view에 연결되며, 이후 500, 1000 ,1500개의 노드로 디코딩 아키텍처에 연결됩니다. output layer는 input layer와 같은 정확한 노드 수로 구성됩니다.

2) Model 학습

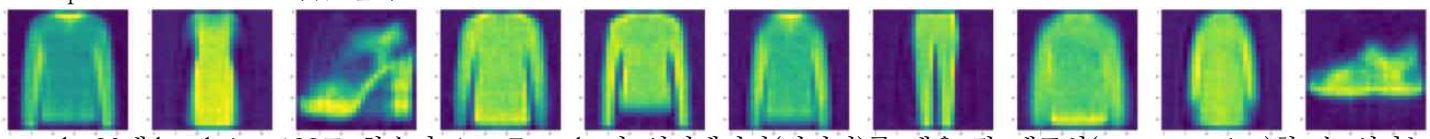
optimizer='adam' loss function은 mse로 early_stopping을 callback함수로 사용했습니다.

3) 결과분석

Input: original data



Output: AutoEncoder학습 결과



epoch=20개,batchsize=128로 학습된 AutoEncoder가 입력데이터(이미지)를 매우 잘 재구성(reconstruction)할 수 있다는 것을 확인할 수 있습니다.

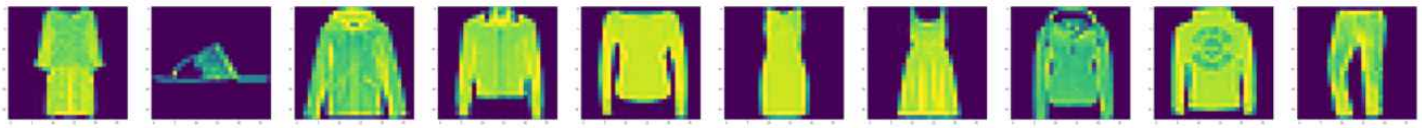
5.3 AutoEncoder UseCase 2 : Image Denoising

입력데이터(이미지)에 노이즈(noise)가 포함되어 있는 경우 AutoEncdoer를 활용하여 노이즈를 제거할 수 있습니다.

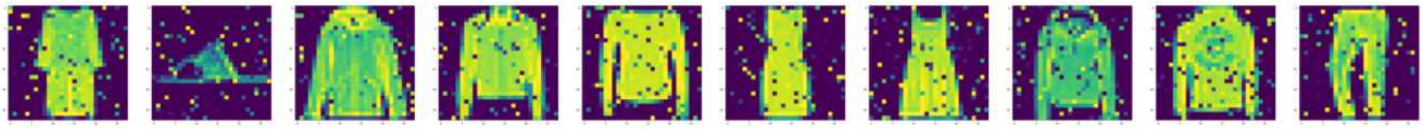
1) Noise 추가

Gaussian Noise를 주로 쓰지만 여기선 데이터에 Salt and Pepper Noise를 일부러 데이터에 적용하겠습니다.

noise 추가 전



noise 추가 후



2) AutoEncoder Architecture

-Encoding Architecture:인코딩 아키텍처는 3개의 convolution layer와 3개의 Max pooling layer가 하나씩 쌓여 구성됩니다. activation function으로는 Relu를 사용, padding은 "same"로 유지됩니다. Max pooling layer은 이미지 차원을 축소, 이 layer는 max filter를 원본이미지의 overlapping 되지않은 부분에 적용합니다.

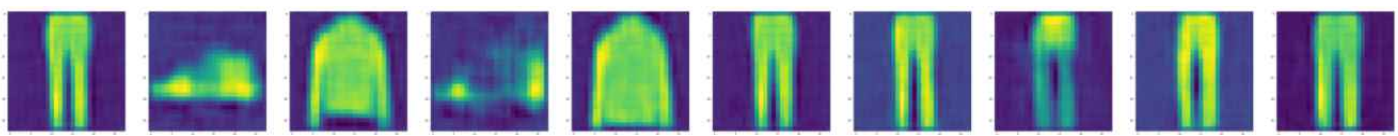
-Decoding Architecture: 디코딩 아키텍처에서도 마찬가지로, convolution layer은 인코딩 아키텍처와 동일한 차원(대신 반대로 16->32->64)을 가지고 사용됩니다. 하지만 3개의 Max pooling layer 대신 3개의 UpSampling layer를 사용합니다. activation function은 똑같이 Relu로, padding도 'same'으로 같습니다. UpSampling의 역할은 입력 벡터의 치수를 고차원(고해상도)으로 바꾸는 것입니다. Max pooling은 되돌릴 수 없지만 각 pooling 영역 내 maxima의 위치를 기록하면 대략적인 근사치를 얻을 수 있습니다.이를 통해 Umsampling layer는 저차원을 재구성(reconstruction)합니다.

3) Model 학습

optimizer='adam' loss function은 mse로 early_stopping을 callback함수로 사용했습니다.

4) 결과 분석

보시면 noise가 추가된 dataset이 깔끔하게 특징만 추출된 것을 확인 할 수 있습니다.



목 차

Ⅰ. Task 2 데이터 분석

1. Data info

CIFAR-10 dataset : tensorflow, pytorch dataset에 내장되어 따로 다운받을 필요가 없습니다
CIFAR-10 dataset은 10개 클래스의 60000 개의 32x32 컬러 이미지로 구성되며, 클래스 당 6000개의 이미지가 있습니다. 50000개의 training 이미지와 10000개의 test 이미지가 있습니다.

Ⅱ. 모델링

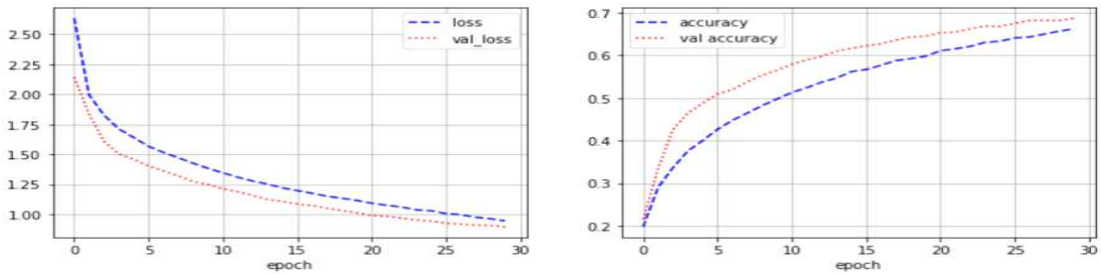
1. CNN

1.1 Model Architecture

모델 생성시 특징추출을 위한 단계로, activation function은 'relu'로 32개의 3x3 kernel(filter)를 지닌 convolution layer와 2x2 max pooling이 stride 2로 시행됩니다.다시 64개(3x3)의 filter를 지닌 convolution layer, 2x2, stride=2 pooling layer, 또 다시 128개(3x3) convolutioin layer, 2x2, stride=2 Pooling Layer로 이루어져있으며, 이미지 분류단계로 Fully connected된 Flatten Layer와 Dense Layer를 사용해 감소된 차원의 Featuremap들만 input으로 하여 Output과 Fullyconnected합니다. Cifar는 10개의 class를 가졌기에 10개 노드의 activation='softmax' Output Layer로 이루어져있습니다.

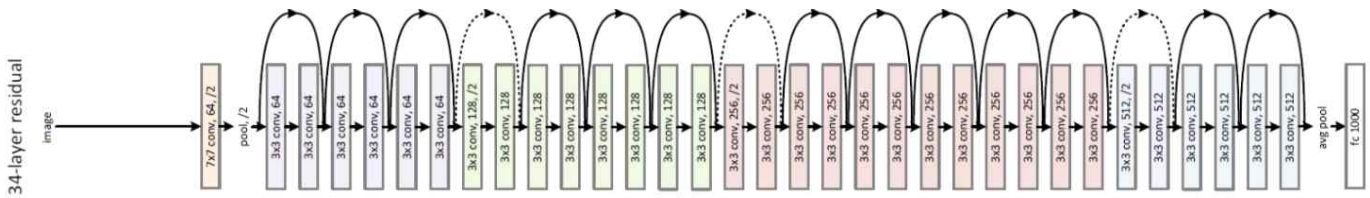
optimizer='adam', learning_rate=1e-4, loss='sparse_categorical_crossentropy', batchsize=256, epoch=100으로 파라미터 조정,Dropout(0.5)&Batch Normalization도 적용한 그 결과값입니다.

loss	accuracy
0.8951	0.6872



〈epoch 진행에 따른 loss, accuracy 그래프〉

2. ResNet18(*아래 그림은 34Layer)



ResNet은 깊어질 수록(깊은 layer)가 얕은 Layer보다 잘 동작하고 training error또한 줄어들고, 일반화 성능 또한 다른모델에 비해 높습니다. ResNet은 Layer의 입력을 layer의 출력에 바로 연결시키는 'skip connection'을 사용합니다. 이 경우 입력이 그대로 skip connection이 되어 연산증가가 없으며 몇 개의 layer를 포함할지 선택도가능하며, fully connected layer와 convolution layer도 가능합니다. 입력단과 출력단이 같으면 그냥 identity mapping을 쓸 수 있습니다.

2.1 BasicBlock 정의

BasicBlock은 2개의 convolution layer로 구성되며, 하나의 [3x3,64] layer를 거친뒤 , BatchNormalization -> 'relu' 후 그다음 레이어로 돌아가고, 다음 레이어도 앞과 똑같은 과정을 거칩니다. 대신 입력으로 들어오는 input이 identity mapping 이용해 2번째 convolution을 거친 결과에 더해질 수 있도록 만들고, 그 위에 'relu'를 적용합니다.

$$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$$

2.2 ResNet Architecture

ResNet의 특징은 convolution layer는 3x3filter를 사용, 복잡도를 줄이기 위해 maxpooling,hidden FC, dropout을 사용하지 않습니다. 출력 featuremap크기를 줄일 땐 stride=2로 조정해줍니다. 또한 2개의 convolution layer마다 skip connection을 연결해줍니다.

```
# ResNet 정의
# 위의 BasicBlock을 여러번 연결하는 방식으로 구현됨.
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):    #block의 개수 입력받을 수 있음 ,CiFar10은 class개수 10개라
        super(ResNet, self).__init__()
        self.in_planes = 64

        # 64개의 3x3 필터(filter)를 사용
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512, num_classes)                    #Fullyconnected Layer

    def _make_layer(self, block, planes, num_blocks, stride):        # BasicBlock이 연속될 수 있게끔 만들어줌. 즉, 이어붙임.
        strides = [stride] + [1] * (num_blocks - 1)                  # 첫번째 convolution 연산에 의해서만 너비와 높이(Dimension)이 줄어들 수 있게 만들고
        layers = []                                                  # stride=2로 설정해서 줄어들게 만듦.
        for stride in strides:                                       #그 다음 이어붙히는 것은 다 같은 Dimension만 되게 한다.
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes # 다음 레이어를 위해 채널 수 변경
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)                                   #Maxpooling이 아닌 avgpooling을 사용한다.
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

optimizer='sgd', learning_rate=0.1, loss='crossentropy', momentum=0.9, weight_decay=0.0002으로 파라미터 조정, learning_rate가 100,150될 때 1/10으로 감소시키는 함수 적용시켰습니다. 다만 지금은 epoch=20이라 실제로 적용되진 않았으며. Batch Normalization도 적용한 그 결과값입니다.

average_loss	accuracy
0.0048977	0.8366

3. GoogLeNet

3.1 Inception Modul

googLeNet이 아주 딥한 신경망으로 구성되어 학습이 가능했던 것은 Inception Module 때문입니다.
Inception Module은 입력값에 대해 , 4가지 종류의 Convolution,Pooling을 수행하고, 4개 결과를 채널방향으로 합칩니다. 이러한 Inception Module이 모델에 총 9개가 있습니다.

1) 1x1 convolution
2) 1x1 convolution + 3x3 convolution
3) 3x3 convolution + 5x5 convolution
4) 3x3 MaxPooling + 1x1 convolution

이 4개의 연산결과를 featuremap을 쌓습니다. 3x3 MaxPooling에서 입력과 출력의 Dimension같아야하므로 Pooling에서 Padding을 추가해주는 방식입니다.

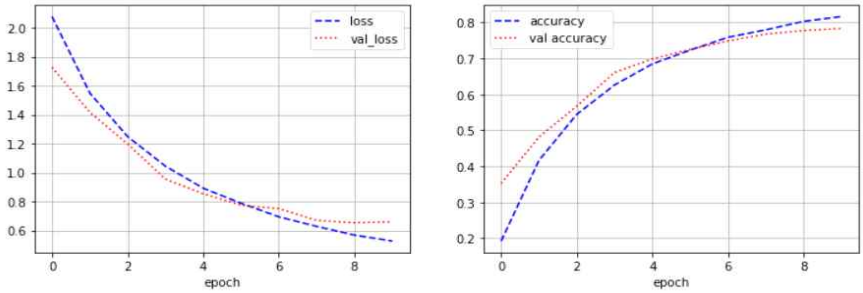
3.2 GoogLeNet Architecture

input layer 한 개와 레이어 초반에는 인셉션 모듈이 들어가지 않으며. 이걸 stem layer라 하는데, 레이어 초반에는 인셉션이 효과가 별로 없어서 이렇게 한다고 합니다. 이 후 위의 inception module이 9개 적용. 이미지 분류단계로 googLeNet은 Fully connected 방식대신에 global average pooling을 써서 1차원으로 만들어 softmax층과 연결해줍니다.

*optimizer='adam'

loss	accuracy
0.6608	0.7826

<모델 학습 후 결과값>



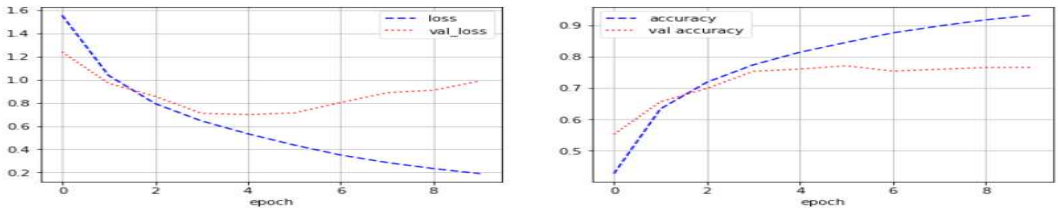
<epoch 증가 시 loss,accuracy 변화 그래프>

4.VGG

4.1 VGG Net Architecture

항상 convolution filter size는 가장작은 3x3으로 고정시키며, 2개의 convolution layer 후 maxpooling을 3번 반복합니다. 모델 optimizer로는 'adam'을 사용, batchsize는 64, epoch는 10번으로 조정하였습니다. 실행결과는

loss	accuracy
0.9889	0.7658



<epoch 증가 시 loss,accuracy 변화 그래프>

accuracy가 증가하다 감소하는 모습과 loss가 감소하다 증가하는 모습이 확인되어 early_stopping callback method의 필요성을 느꼈습니다.

5.AlexNet

AlexNet은 8개의 레이어로 구성되어 있습니다. 5개의 컨볼루션 레이어와 3개의 full-connected 레이어로 구성되며. 두번째, 네번째, 다섯번째 컨볼루션 레이어들은 전 단계의 같은 채널의 특성맵들과만 연결되어 있는 반면, 세번째 컨볼루션 레이어는 전 단계의 두 채널의 특성맵들과 모두 연결되어 있습니다

5.1 AlexNet Architecture

1) convolution layer: 96개의 11x11x3 필터로 입력데이터 convolution. stride를 4로 설정했고, zero-padding은 사용하지 않았습니다. 그 다음에 ReLU 함수로 활성화. 이어서 3x3 overlapping max pooling이 stride 2로 시행.

2) convolution layer:256개의 5x5x48필터 사용하여 전 단계의 featuremap을 convolution해준다. stride는 1로, zero-padding은 2로 설정,역시 ReLU 함수로 활성화. 그 후 3x3 overlapping max pooling을 stride 2로 시행.

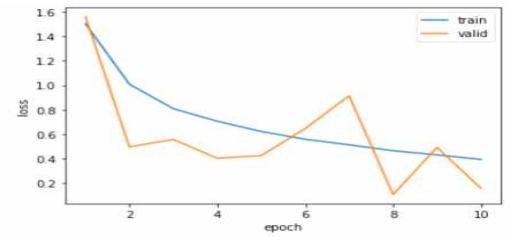
3) 세 번째 convolution layer: 384개의 3x3x256 커널을 사용하여 전 단계의 featuremap을 convolution해준다. stride와 zero-padding 모두 1로 설정. 역시 ReLU 함수로 활성화.

4) 네 번째 convolution layer: 384개의 3x3x192 커널을 사용해서 전 단계의 featuremap을 convolution해준다. stride와 zero-padding 모두 1로 설정, 역시 ReLU 함수로 활성화.

5) 다섯번째 convolution layer: 256개의 3x3x192 커널을 사용해서 전 단계의 featuremap을 convolution해준다. stride와 zero-padding 모두 1로 설정, 역시 ReLU 함수로 활성화. 그 다음에 3x3 overlapping max pooling진행.

6)이미지 분류단계로 average pooling을 써서 1차원으로 만들어 softmax층과 연결해줍니다.

optimizer는 adam, learning_rate=3e-4, dropout을 적용한 실행 결과는 아래와 같습니다.



loss	accuracy
0.1580	0.9059

<epoch 증가 시 loss 변화 그래프>

사용해본 5가지 모델의 accuracy값입니다. 물론 dropout, batchnormalization, learning_rate, momentum 등 epoch, parameter조정이 다르기에 결과값이 다른 것도 있겠지만, 감안하고 비교해서 보면 될 것 같습니다.

	CNN	ResNet	GoogLeNet	VGGNet	AlexNet
accuracy	0.6872	0.8366	0.7826	0.7658	0.9059