2022-1 패턴인식과 머신러닝

Classification of k-fashion images

fashion image를 입력받아 자동으로 Resort / Retro / Sporty 로 분류하는 convolutional neural network (CNN) 개발

1976037 김나윤
1970030 박근아
2260054 이하은
1970081 정민주

1. Purpose

본 프로젝트의 목적은 fashion image를 입력받아 자동으로 Resort / Retro / Sporty 로 분류하는 Convolutional Neural Network (CNN)을 개발하는 것으로 해당 프로젝트에서는 AlexNet, VGGNet, GoogleNet, ResNet, Densenet의 모델 구조를 참고하여 학습을 진행하였다.

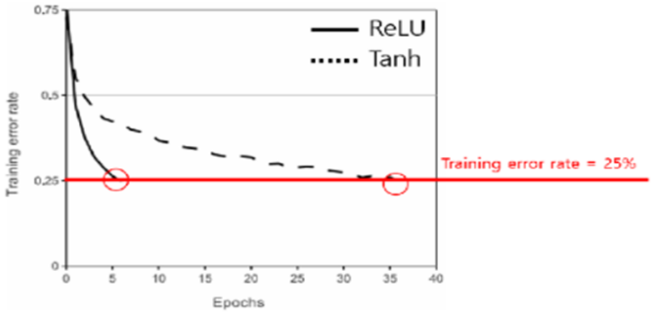
2. Background

Lecture note6의 tuning scenario를 참고하여 각 뉴럴넷의 parameter tuning을 수행해보고자 하였다. 실행결과를 위해 정리한 배경은 다음과 같다.

1) 각 뉴럴넷을 선택한 이유 (AlexNet, VGGNet, GoogleNet, ResNet, Densenet)

a) AlexNet

AlexNet은 활성화 함수로 ReLU를 사용하는데 이를 통해 정확도를 유지하면서 학습 및 예측 속도를 증가시킬 수 있다. 실제로 AlexNet 개발자들이 CIFAR-10 데이터셋과 4개의 Convolutional network를 사용해 Tanh과 ReLU의 학습속도를 비교하는 실험을 진행 했을 때, training error rate가 25%까지 내려가는데 걸리는 epoch가 ReLU이 Tanh보다 약 7배 가까이 빠른 것을 확인할 수 있었다.



[그림1] ReLU

b) VGGNet

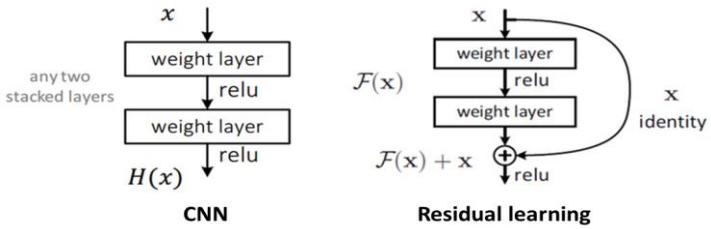
VGGNet은 ReLU non-linear를 여러 개 사용할 수 있기 때문에 decision function이 잘 학습될 수 있다. 또한, 작은 컨볼루션 필터(3x3, 1 strides)로 깊은 레이어(16-19 weight layers)를 만들어 좋은 성능을 낸다. 즉, 학습해야 할 파라미터(weight) 수를 줄여 정규화(regularization) 문제를 줄인다는 장점이 있다.

c) GoogleNet

22개의 layer로 구성되어 있으며 inception module을 사용해 parameter 수는 줄어들지만 망의 깊이는 더한 model이다. 특히, 1x1 convolution을 이용한 9개의 inception module을 이용하여 module 입력값에 대해 4가지 종류의 convolution, pooling을 수행하고 각 결과를 channel 방향으로 concatenate하여 output을 구성하여 feature추출과정에서는 spare한 연결을, matrix는 dense한 구성을 가능하게 한다. 결과적으로 parameter 수는 줄어들지만 망은 깊어지는 효과를 얻는다.

d) ResNet

ResNet은 기존의 AlexNet, VGG, GoogleNet가 convolution을 통해 tensor(Feature maps)를 시키는 것과 달리, input에 additive한 값을 Conv로 학습시킨다. 기존의 VGG 네트워크보다 더 깊지만 residual block을 활용하여 복잡도와 성능을 개선하였다. 이때 구현도 복잡하지 않고 학습 난이도가 매우 낮다는 이점이 있다.



[그림2] Residual Learning

e) Densenet

DenseNet은 ResNet과 Pre-Activation ResNet보다 적은 파라미터 수로 더 높은 성능을 가진 모델이다. DensNet은 이전 레이어의 피쳐맵을 그 이후의 모든 레이어의 피쳐맵에 연결하는데 이때 ResNet과 다르게 덧셈이 아니라 concatenate(연결)을 수행한다.

2) Data Augmentation

기존의 training dataset만으로는 validation error가 줄어들 것 같지않아서 data augmentation을 수행해주었다. 색 역시 패션을 분류하는데 고려되는 요소가 될 수 있다고 생각하여 랜덤하게 이미지의 좌우반전과 이미지 기울이기(-30도~30도)만을 수행해주었다. 다른 option인 crop의 경우 이미지마다 객체의 위치가 달라 수행하지 않았고 blur의 경우에도 이미지의 원본 화질이 좋지 않아 고려하지 않았다. 또한 데이터 증대에 따른 에러감소가 크지 않아서 기존 7200장 → 14400장 으로 data를 2배 까지만 늘려주었다.

3) Tuning

a) Training Dataset Batch Size

메모리의 한계와 속도 저하 때문에 전체 training dataset을 여러 작은 그룹으로 나누었고, 하나의 소그룹에 속하는 data의 수가 batch size이다. Batch size가 큰 경우 training set 분포를 좀 더 근사해서 추정할 수 있다. 즉, noise를 감소시켜 모델의 convergence를 향상시킬 수 있다. 반면, batch size를 작게 하는 경우 noise가 많아 모델의 불안정한 convergence를 유발할 수 있다. 하지만, 약간의 noise는 overfitting을 방지하여 모델의 성능을 향상시키는 효과가 있다. 즉, batch size를 작게하면 noise가 많아지지만 regularization 효과를 줄 수 있다. 본 프로젝트에서는 batch size를 조정하며 그 결과를 비교해보았다.

b) Optimizer 조정

(a) L2 regularization과 weight decay

Weight decay는 모델의 weight의 제곱합을 패널티 텀으로 주어 (=제약을 걸어) loss를 최소화하는 것을 말한다. 이는 L2 regularization과 동일하며 L2 penalty라고도 부른다.

$$Loss(w,x) = DataLoss(w,x) + \frac{1}{2}\lambda\|w\|^2$$

$$w \leftarrow w - \eta\left(\frac{\partial DataLoss}{\partial w} + \lambda w\right)$$

$$= w(1-\eta\lambda) - \eta\frac{\partial DataLoss}{\partial w}$$

Loss를 미분했을 때. 기본 dataloss에 w의 lambda배만큼을 더하게 되므로 가중치값이 그만큼 보정된다. 따라서 gradient descent에서 weight를 업데이트 할 때, 이전 weight의 크기를 일정 비율 감소시켜줌으로써 오버피팅을 방지한다. λ값을 weight decay라고 하며, 이는 hyper-parameter로 실험적으로 적절한 값을 찾아주면 된다.

(b) adam, adamW

Adam은 gradient의 1차 모먼트 mt와 2차 모먼트 vt를 사용하여 모멘텀 효과와 weight마다 다른 learning rate를 적용하는 adaptive learning rate 효과를 동시에 보는 최적화 알고리즘이다. 그런데, adam의 구조상
L2 regularization을 적용할 때, weight decay의 효과가 떨어진다. 따라서 regularzation에 의한 weight decay 효과뿐만 아니라 weight 업데이트 식에 직접적으로 weight decay 텀을 추가하여 이 문제를 해결한 것이 adamW이다.

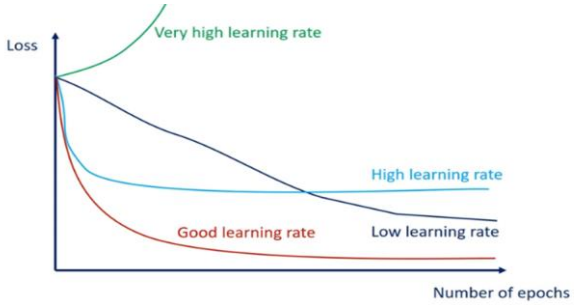
Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda\boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1\boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2\boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1-\beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1-\beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t\left(\alpha\hat{\boldsymbol{m}}_t/(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1}\right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

그림에서 초록색으로 표시된 부분이 없다면 L2 regularization을 포함한 손실함수에 Adam을 적용한 것과 똑같다. 하지만 초록색 부분을 직접적으로 weight 업데이트 식에 추가시켜줌으로써 weight decay 효과를 볼 수 있는 optimizer이다. 따라서 weight decay가 강하게 필요한 경우 adamW를, 그렇지 않은 경우 adam을 optimizer로 사용하여 경험적으로 결과값을 비교해보며 학습을 진행하였다.

c) Learning Rate Tuning

Learning rate는 어느 정도의 크기로 기울기가 줄어드는 지점으로 이동하겠는가를 나타내는 지표이다.



[그림4] Learning rate and Loss

최적의 훈련 모델을 구현하기 위해 다양한 hyperparameter를 조절하였는데 그중, learning rate가 클 때(1e-3), data loss가 커지는 문제가 발생했다. 적당한 수준의 learning rate를 설정하여 손실을 최소화할 필요가 있었기에 시간이 좀 더 소요되더라도 learning rate를 줄여(1e-4) 학습의 손실을 줄이는 방향으로 목표를 설정하였다.

4) High Accuracy High Loss Case

Training 과정에서 validation accuracy가 높아지면서 동시에 loss 역시 증가하는 상황들이 발생하였다. 일반적으로 accuracy가 높아질때는 loss 역시 낮아져야하지만 그렇지 않은경우 overfitting이 발생했다고 생각하였고 loss가 증가하더라도 그 폭이 적은 모델 또는 전반적인 loss가 작은 모델을 best model로 선정하고자하였다.

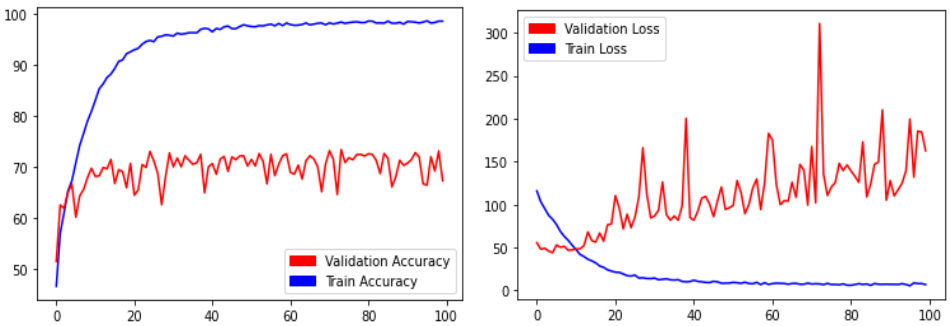
3. Modeling

1) AlexNet

본 네트워크에서는 이미지 사이즈가 227x227이어야하므로 이를 조정해주었다. 또한 Batch Normalization을 추가하여 Regularization Effect를 주고자하였다.

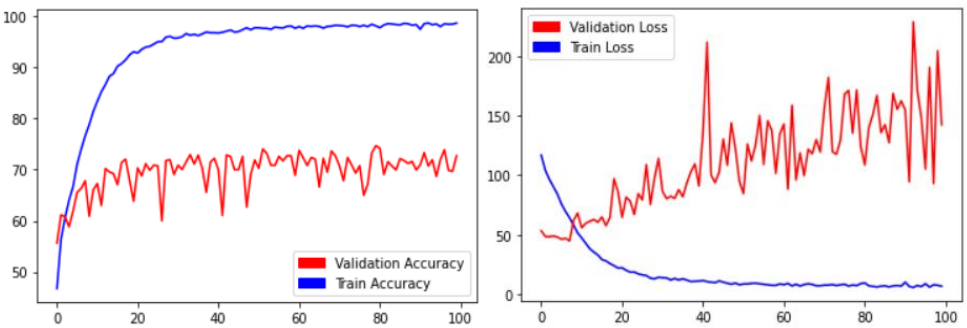
a) AlexNet (Dropout: 0.6, λ: 1e-4)

AlexNet 모델은 Dropout과 Regularization을 주 변수로 설정하고 해당 변수의 값을 달리하면서 결과값의 변화를 살펴보았다. 먼저, Dropout을 0.6, Regularization을 1e-4로 설정하고 모델을 실행시킨 결과,



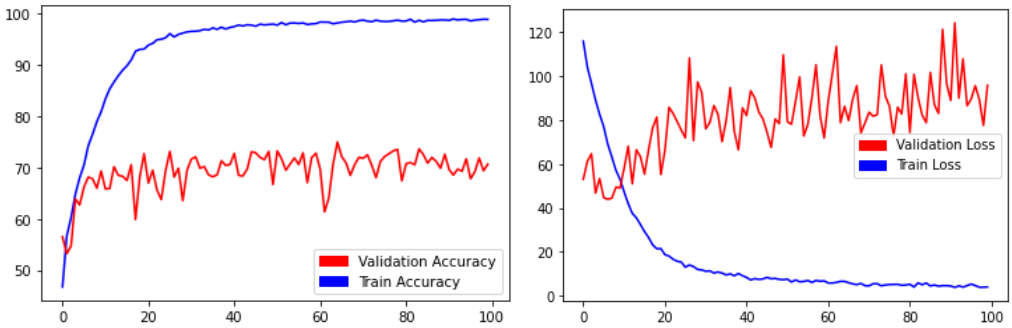
train accuray는 99.7569%, validation accuracy는 74.1667% 의 수치를 얻을 수 있었다.

b) AlexNet (Dropout: 0.8, λ: 1e-4)



마찬가지로 실험을 진행하되, Dropout은 0.8로 증가시키고 Regularization을 1e-4로 유지하여 모델을 실행시킨 결과, train accuray는 99.2498%, validation accuracy는 74.6667%의 수치를 얻을 수 있었다.

c) AlexNet (Dropout: 0.8, λ: 1e-3)



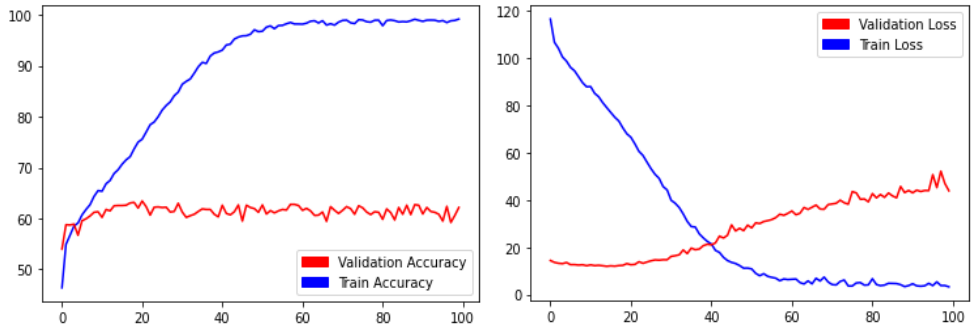
weight decay 값을 1e-4에서 1e-3으로 증가시킨 결과 가장 높은 validation accuracy에서의 모델은 train accuracy가 99.6597%, validation accuracy 75%로 나온 것을 확인할 수 있었다. weight decay 값이 1e-4일 때보다는 loss 값이 줄어든 것을 확인할 수 있었다.

2) VGGNet

VGGNet은 "Very deep convolutional networks for large-scale image recognition"으로, network의 깊이를 깊게 만드는 것이 성능에 어떤 영향을 미치는지를 확인하는 목적을 가진다.

a) VGGNet (ConvLayer:8)

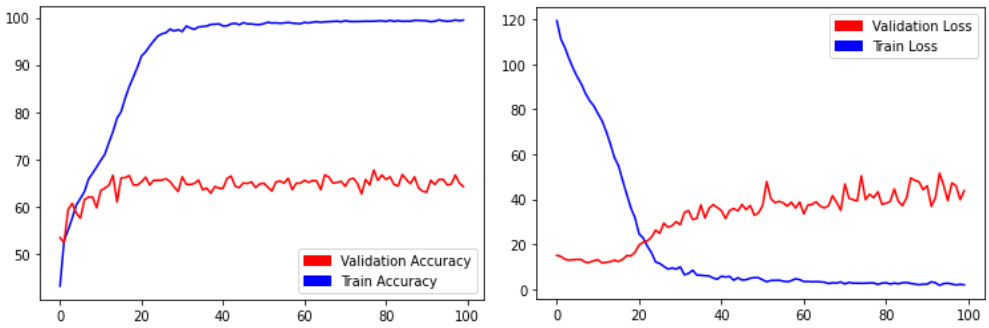
Baseline으로 주어진 Convolutional layer가 8개로 이루어진 VGGNet의 결과는 다음과 같다.



Best score은 epoch 20에서 나왔으며, Train Accuracy가 78.0649%, Validation Accuracy가 63.44이다.

b) VGGNet (ConvLayer:10)

Convolution layer의 수를 추가하여 기존 8개에서 10개로 바꾸었고, 이때 학습해야 할 parameter의 수를 299,667에서 7,649,091로 증가시켰다. Training한 결과는 다음과 같다.

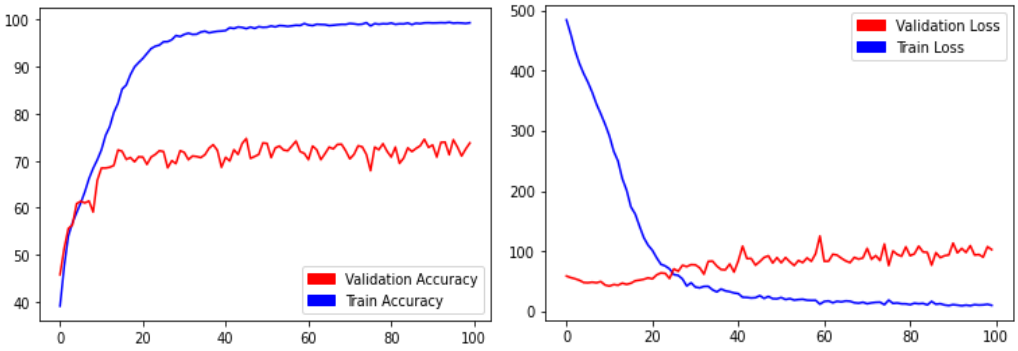


Best score은 Train Accuracy가 99.38%, Validation accuracy가 71.28%인 epoch이었다.

3) GoogleNet

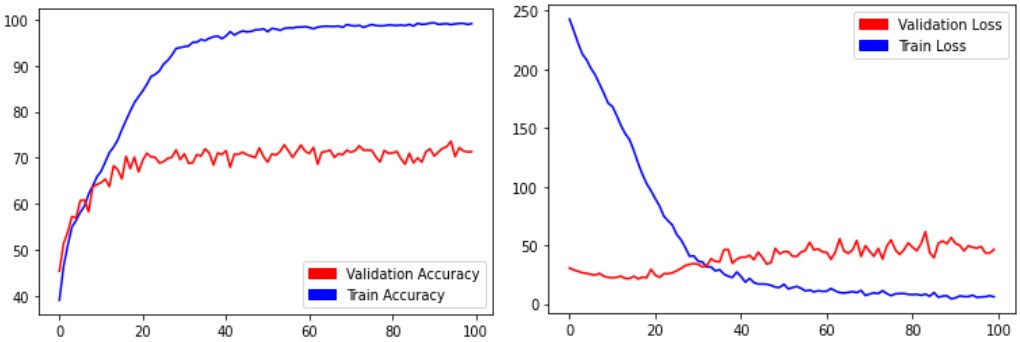
구글넷에서는 주요하게 batch size를 조정하여 모델을 비교해보았다. 이외에 구글넷에서 정의하고 있고 튜닝을 통해 정한 파라미터는 dropout = 0.4, learning rate = 0.00005, weight decay = 0.005 이다. 그리고 optimizer로는 AdamW를 이용해주었다.

a) batch size = 32



best score는 45번째 epoch에서 validation accuracy가 74.72%, validation loss가 82.74로 나왔다. weight decay를 높게 잡아줬더니, accuracy의 진폭이 크지 않은 채로 안정적으로 수렴할 수 있었다.

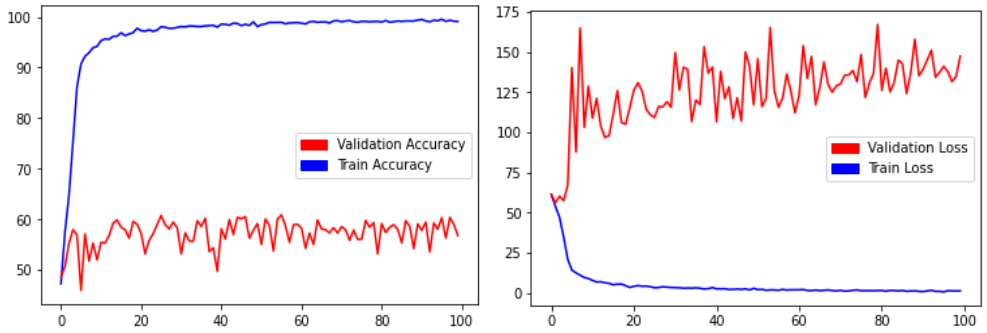
b) batch size = 64



best score는 94번째 epoch에서 validation accuracy가 73.61%, validation loss가 48.30로 나왔다. batch size를 늘려주었을 때의 validation loss와 accuracy 분포가 조금 더 안정적인 것을 확인할 수 있었다.

4) ResNet34

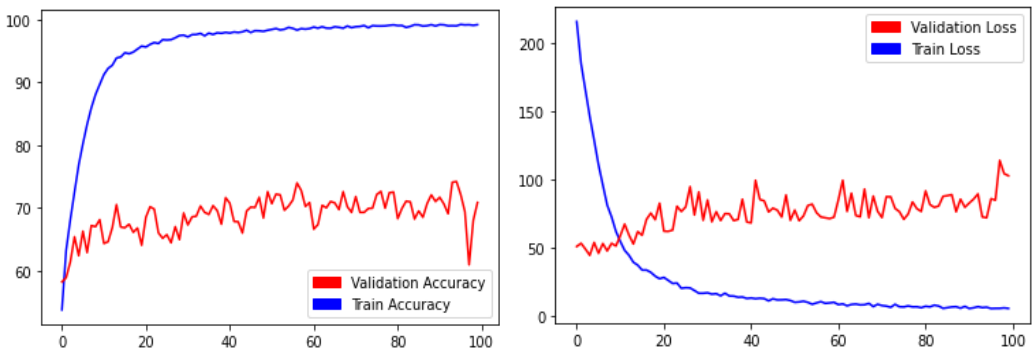
본 프로젝트에서는 34층의 Resnet을 이용해주었다.



Resnet34 모델에서는 초반의 validation accuracy이후에 큰 상승이 보이지 않았고 validation loss 역시 초반 이후 overfitting이 발생하는 것을 확인할 수 있었다. best score는 55번째 epoch에서의 Train accuracy가 98.6528% 그리고 Validation accuracy가 60.7778이었다.

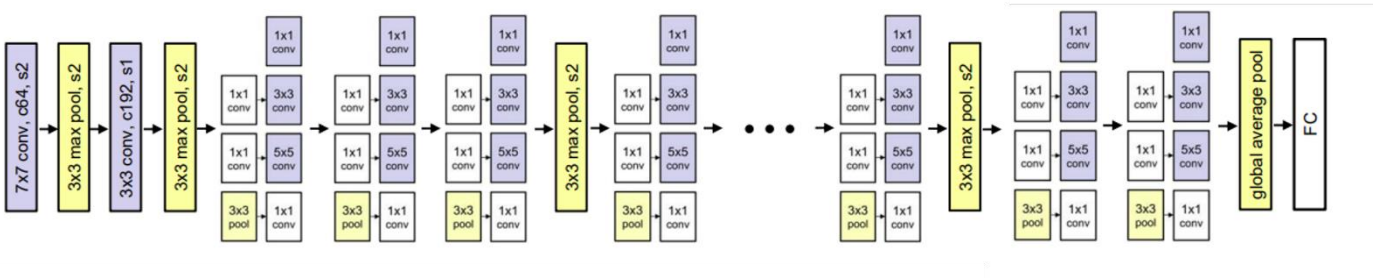
5) DenseNet

DenseNet 모델에서는 best model에서의 train accuracy가 99.7152% validation accuracy가 74.22%였다. 전체적인 validation accuary와 loss의 양상이 Alexnet과 비슷하며 크게 진동하는 것을 볼 수 있었다.



4. Conclusion

AlexNet, VGGNet, GoogleNet, ResNet, Densenet의 구조 및 parameter를 변경해가며 학습을 진행한 결과 가장 높은 검증 데이터의 정확도를 보여준 학습 결과는 Alexnet에서 λ를 1e-3일 때로 설정하고 실행했을 때였다. 하지만 GoogleNet의 validation loss가 더 낮게, 안정적으로 나온다는 점, Validation accuracy도 다른 모델에 비해 안정적으로 증가하고 그 값이 충분히 잘 나온다는 점을 들어 Googlenet에서 batch size를 64로 설정한 모델을 최종 모델로 선정하게 되었다. 따라서 구글넷에서 필요한 224x224의 fashion image를 입력으로 받고 class를 예측할 수 있도록 코드를 작성하였다. 이때의 block diagram은 아래 그림과 같다.



	precision	recall	f1-score	support
0	0.70	0.68	0.69	600
1	0.77	0.78	0.77	600
2	0.74	0.75	0.75	600
accuracy			0.74	1800
macro avg	0.74	0.74	0.74	1800
weighted avg	0.74	0.74	0.74	1800

최종적으로 사이킷런의 classification_report를 이용해 validation 결과를 출력한 것은 왼쪽의 표와 같다. 0번 클래스에 대해 다른 클래스보다 예측율이 떨어진다는 아쉬움은 있지만 이는 다른 모델에서도 공통적으로 발생하는 문제였다. 그 점을 제외하면 f1-score 역시 잘 나온 것을 확인할 수 있다.

5. Code Reference

data augmentation : https://hipolarbear.tistory.com/19

vggnet:https://minjoos.tistory.com/6 , https://deep-learning-study.tistory.com/521

densenet :https://deep-learning-study.tistory.com/545

resnet:https://pseudo-lab.github.io/pytorch-guide/docs/ch03-1.html

alexnet:https://medium.com/analytics-vidhya/alexnet-a-simple-implementation-using-pytorch-30c14e8b6db2

googlenet : https://deep-learning-study.tistory.com/523 , https://url.kr/bhkc65

classification report : https://gaussian37.github.io/ml-concept-ml-evaluation/#accuracy-1

<표. 모델 선정과정>

Model	Hyper parameter	Epochs (saved ep)	Accuracy	F1-Score	Val Loss	#Params
AlexNet	batch size=128 dropout=0.6 lr, λ =0.0001	100(79)	Train: 99.7569 Val:74.1667	0.74	98.6404	58,294,339
	batch size=128 dropout=0.8 lr, λ=0.0001	100(92)	Train: 99.2498 Val: 74.6667	0.75	157.9	58,294,339
	batch size=128 dropout=0.8 λ=0.001 lr=0.0001	100(64)	Train: 99.6597 Val: 75.0000	0.75	86.21	58,294,339
VGGNet (layer 8,10)	batch size=128 λ=0 lr=0.0001	100(20)	Train: 78.06 Val: 63.44	0.63	19.36	299,667
	batch size=128 λ=0 lr=0.0001	100(82)	Train: 99.38 Val: 71.28	0.71	44.65	7,649,091
GoogleNet	batch size=32 dropout=0.4 adamW: learning rate=0.0005 λ=0.005	100(45)	Train: 99.4165 Val: 74.72	0.75	82.74	5,972,467
	batch size=64 dropout=0.4 adamW: learning rate=0.0005 λ = 0.005	100(94)	Train: 99.6735 Val: 73.61	0.74	48.30	5,972,467
Resnet34	batch size=128 lr=0.0001 λ=0.0001	100(55)	Train: 98.6528 Val: 60.7778	0.61	115.585	21,286,211
Densenet	batch size=64 lr=0.0001 λ=0.0001	100(94)	Train: 99.7152 Val: 74.22	0.74	71.89	1,000,827