인공지능1

보고서명 : 인공지능1 기말 프로젝트 보고서

학과명 : 융합소프트웨어학부

학번 : 60161606

성명 : 박준현

목차

1. 배경 및 서론 (Backgrounds)

1.1 배경

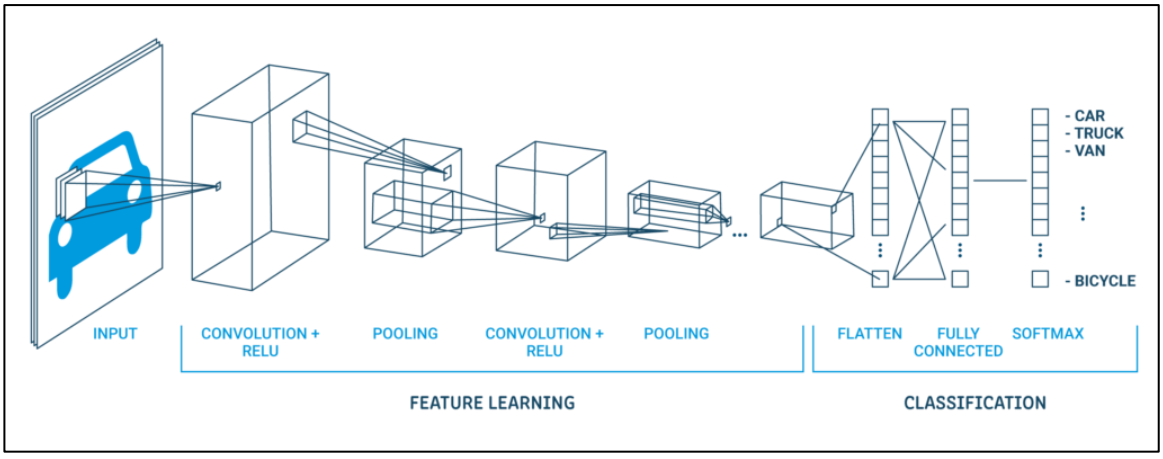
오늘날 가게에서는 흔히 주문을 받는 직원 대신 키오스크를 더 자주 볼 수 있다. 키오스크는 아주 짧은 시간만에 전 세계의 많은 가게들에게서 도입되었고 아주 훌륭한 업무 효율을 내고 있다. 그리고 최근 서빙을 하는 직원 대신 인공지능 서빙 로봇의 도입이 여러 가게에서 시범적으로 일어나고 있다. 서빙이란 단순히 주문을 받는 것 보다는 조금 더 복잡한 업무라고 생각한다. 고객과의 의사소통을 통해 고객이 주문한 음식이 제대로 나왔는지 확인도 해야 한다. 하지만 이런 기능에 있어서 아직 많이 부족한 현실이다. 만약 직원이 사람이라면 즉각적으로 잘못 나온 음식에 대한 의사소통이 이루어질 수 있을 것이지만 서빙 로봇이라면 그렇지 않다. 그래서 나는 서빙 로봇이 고객의 주문과 음식이 제대로 나왔는지 확인할 수 있어야 한다고 생각했다. 이를 구현하기 위해서는 영상 입력을 통해 음식이 어떤 음식인지 알아야 하고 그것이 고객이 주문한 음식과 동일한 지 판별하여야 한다. 이를 해결하기 위해 음식 이미지를 학습하고 영상의 음식이 어떤 음식인지 분류할 수 있는 능력을 딥러닝을 통해 가질 수 있도록 할 것이다.

1.2 서론

본 프로젝트의 목적은 입력 받은 영상을 보고 해당 영상에 있는 음식이 어떤 음식인지 판단하는 인공지능을 개발하는 것이다. 음식을 제대로 분류할 수 있는 분류기 로봇을 만들기 위해 어떤 기술을 사용할 것인지 또 어떻게 인공지능을 학습할 수 있을지에 대해 고민해보았다.

2. 핵심 아이디어 (Key Ideas)

2.1 CNN (Convolutional Neural Network)



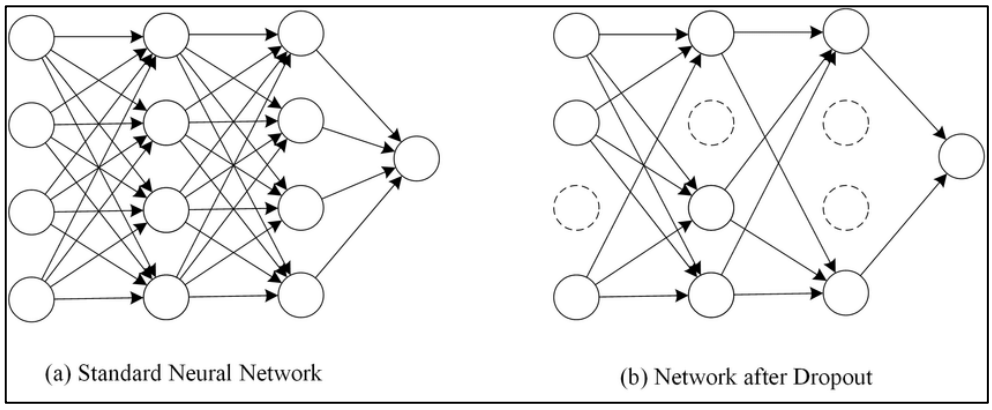
CNN은 인간의 시각 기능을 모방하여 만들었으며 컨볼루션층, 풀링층, 완전연결층으로 구성되어 있는 신경망 모델이다. 컨볼루션 커널을 통해 모델의 복잡도를 줄이고 특징을 추출한다. 풀링층으로 공간적 해상도와 차원을 줄이고 완전연결층은 분류를 담당한다.

➔ CNN을 통하여 라벨링 된 이미지를 학습시키고 학습이 완료되면 한국 음식(10가지)에 대해 분류할 수 있도록 한다.

2.2 규제 (Reguralization)

CNN을 통해서 모델의 복잡도를 줄일 수 있지만 여전히 과잉적합이나 성능부족 같은 단점이 존재하므로 이를 규제기법을 통해서 해결하고자 한다. 규제 기법에는 데이터 증대, 가중치 감쇠, 드롭아웃 등 여러가지 방법이 있지만 이 프로젝트에서는 드롭아웃을 주로 사용하여서 규제를 진행할 예정이다. 드롭아웃이란 학습 중에 일정 비율의 가중치를 임의로 선택하여 불능으로 만들고 학습하는 규제 기법을 말한다.

<Dropout>



프로젝트 설계 초기에는 각 음식별로 300장의 샘플 이미지만 가지고 있어서 영상 데이터의 양이 적다고 판단하였고 이미지 증대를 통한 규제도 생각하였다. 하지만 클래스별 약 1천장의 충분한 학습 데이터셋을 사용할 수 있는 허가를 AIHub(https://aihub.or.kr)로부터 얻었으므로 이미지 증대를 통한 규제는 사용하지 않을 예정이다.

3. 모델 / 방법 결정 전략 (Methods)

3.1 모델 결정 요소

- 모델들은 대중적인 모델 2가지와 직접 제작한 모델을 사용할 예정이며 그 중 가장 정확도가 높은 모델을 찾아내는 방법으로 실험이 진행될 예정이다. keras의 Sequential을 사용하여 모델을 제작하고 이에 여러 레이어(Layer)들을 쌓아서 구조를 만들 것이다. 모델 내에서 사용할 레이어들은 다음과 같다.

- **Conv2D**

➢ Convolution 층으로 커널을 사용하여 컨볼루션 연산을 수행하고 특징 맵을 추출한다.

➢ 커널에 보폭을 주어서 다운 샘플링으로 복잡도를 줄인다.

- **MaxPool2D**

➢ 최대치를 구하는 최대 풀링 연산을 통해 공간적 해상도를 줄인다.

➢ 자기 자신의 값들 중 가장 큰 값을 선택하여 서브샘플링을 진행한다.

- **Dense**

➢ 분류를 위한 완전 연결층 (Fully Connected)

- **Dropout**

➢ 일정 비율의 가중치를 임의로 선택하여 불능으로 만들고 학습하게 하는 규제 기법.

➢ Dropout(0.25) : 전체 가중치에서 25%만큼 학습시에 불능 상태로 만든다.

- **Flatten**

➢ 다차원의 구조를 1차원의 구조로 직렬화 시켜주는 레이어이다.

➢ 일반적으로 Conv2D, MaxPool2D 층과 Dense 층 사이에 위치하고 있다.

3.2 방법 결정 전략

프로젝트의 실험에 대한 방법은 규제가 없는 CNN모델을 제작하고 학습시켜 모델의 정확률과 손실률을 파악해 보고자 한다. 규제가 없는 모델의 대조군으로 규제를 추가한 CNN모델을 동일한 방식으로 제작, 학습하고 모델의 정확률과 손실률을 파악하고 두 모델이 어떤 차이점이 있고 어떻게 변화하는지 알아볼 것이다.

규제의 경우에는 주로 Dropout 층의 유무에 따른 CNN모델의 차이를 알아보고자 한다. 또한 Dropout 층의 계수를 조정하거나 FC층의 노드 수 변화에 따라서 성능이 어떻게 달라질 수 있는지 공부해보고자 한다. 앞서 2-2장에서 말했듯이 데이터를 증대하는 Data Augmentation 기법 사용하게 되면 좀 더 좋은 성능의 딥러닝 모델을 구축할 수 있겠지만 클래스별 약 1천장의 영상 데이터셋을 통해 학습을 진행할 예정이기에 따로 사용하지 않을 예정이다.

대중적으로 유명한 모델들을 만들거나 나만의 모델을 새로 설계, 제작한 이후 위의 방법을 통해학습을 하고 성능과 손실을 그래프를 통해 비교해보고자 한다.

각 모델마다 동일한 사진으로 학습을 진행하며 batch_size는 32로 고정한다. Train_test_split을 통해 학습, 검증집합을 나누고 100세대 동안 학습을 진행한다. 활성함수는 마지막 출력층을 제외하고는 모두 ReLU함수를 사용하고, 출력층에서는 Softmax 활성함수를 사용한다. 사진은 컬러 사진이므로 입력 값은 사진별로 3장씩(R, G, B) 들어간다. 모든 모델에서 옵티마이저는 Adam을 사용할 예정이다.

CNN모델을 개발하는 툴은 Spyder를 통하여 진행하고 Tensorflow와 Keras를 라이브러리를 이용할 예정이다. 본격적으로 모델을 학습하기 전 데이터 전처리는 데스크탑의 CPU 연산을 통해 전처리를 진행한 이후, 영상을 학습할 때에는 구글에서 제공하는 Colab의 GPU 환경을 통해 학습할 예정이다.

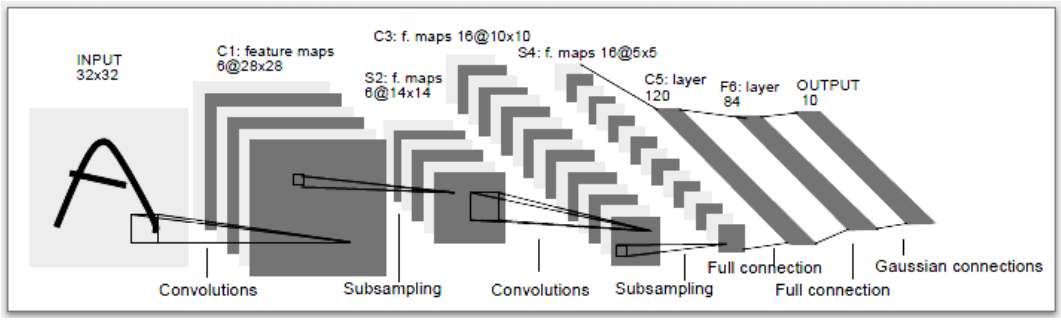
학습을 마치고 나면 얼마나 학습이 잘 되었는지 확인하기 위해서 테스트 데이터셋을 만들어 해당 모델을 통해 예측을 진행할 것이다. 학습을 할 때 사용하였던 사진, 인터넷에서 새로 검색한 사진, 직접 찍은 음식사진 등을 집어넣고 예측을 진행하여 어느 정도의 정확률을 가지는지 측정해보고자 한다. 또한 학습한 모델을 이용하여서 학습하지 않았던 전혀 새로운 테스트셋 데이터를 가지고 어떤 음식인지 분류 예측을 진행할 것이다.

연산을 진행할 하드웨어의 정보는 아래와 같다.

- 데스크탑 : CPU (AMD 라이젠5 3600XT), GPU(GTX-1660 SUPER), RAM(16G)
- Colab : CPU(Intel Xeon 2.2GHz), GPU(TESLA K80), RAM(13G)

4. 실험 결과 (Experimental Results)

4.1 LeNet-5 모델

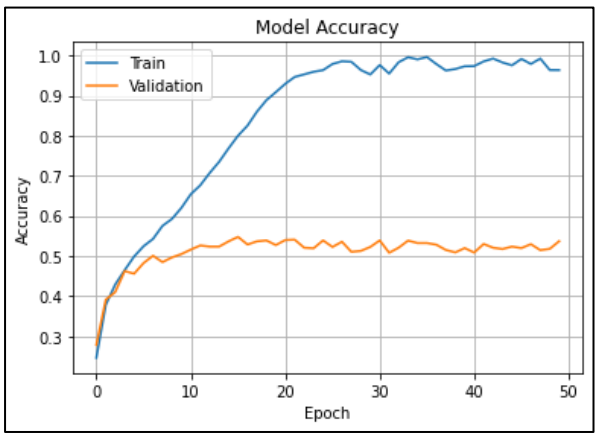


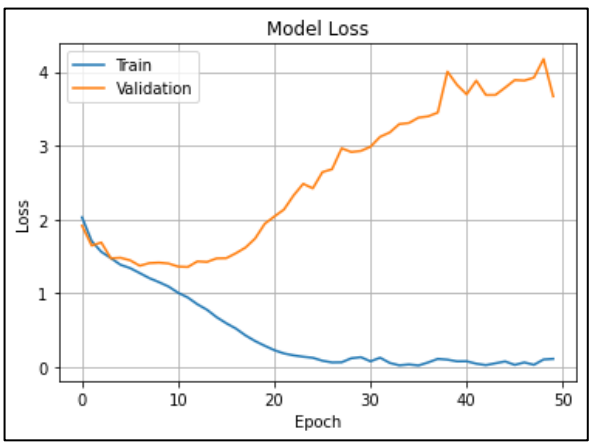
가장 처음엔 CNN의 성공적인 초기 모델인 LeNet-5를 통하여 실험해보았다.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_38 (Conv2D)           (None, 28, 28, 6)         456
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 14, 14, 16)        2416
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 7, 7, 120)         48120
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 3, 3, 120)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 1080)              0
_________________________________________________________________
dense_19 (Dense)             (None, 84)                90804
_________________________________________________________________
dense_20 (Dense)             (None, 10)                850
=================================================================
Total params: 142,646
Trainable params: 142,646
Non-trainable params: 0
```

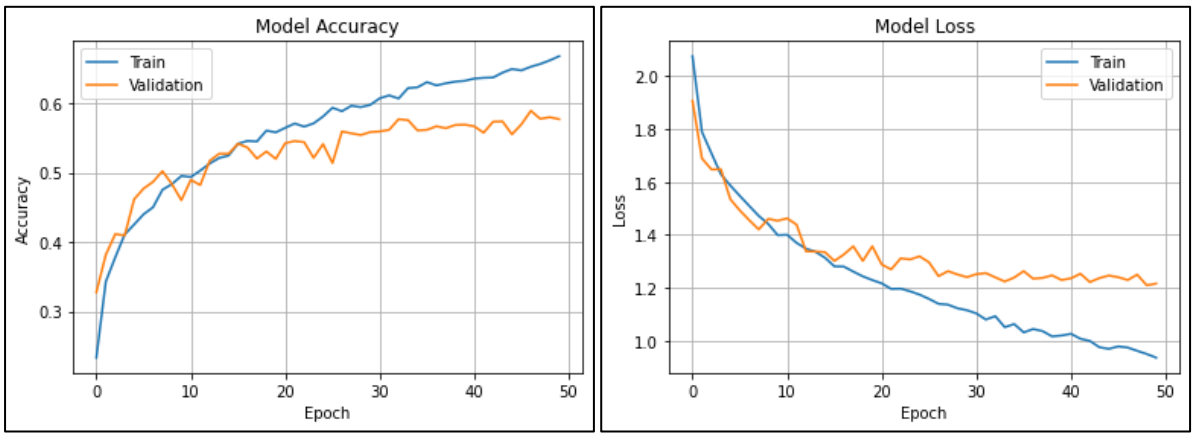
← 구현한 기존LeNet5 구조

기존 LeNet5 모델을 도입하여 음식 이미지에 대한 학습을 시작했다. 입력은 (28,28,3) 으로 넣어주었고 이외의 커널이나 풀링은 동일하게 적용하였다. 학습 결과는 아래와 같다.

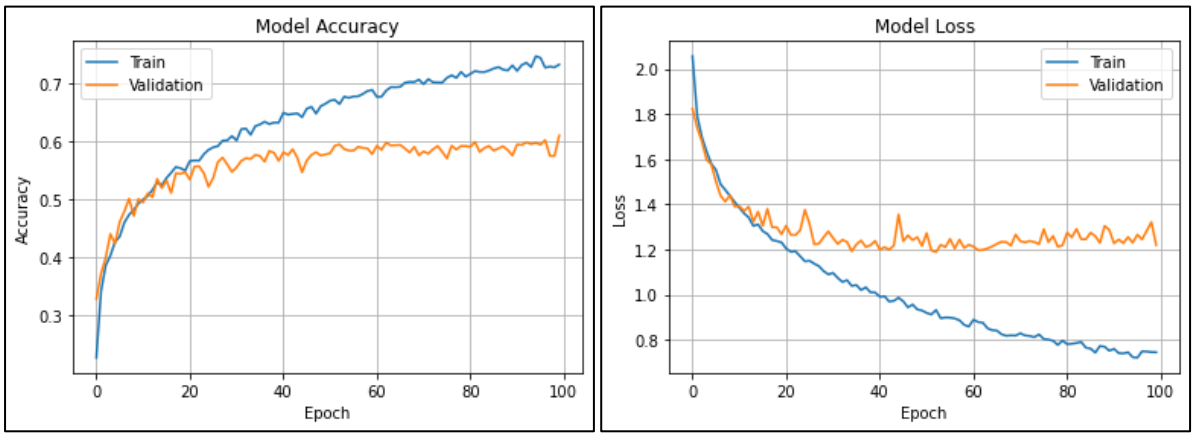




우선적으로 정확률평가에서 약 53%를 기록하였고 손실함수를 보면 epoch 10 정도에서부터 val_loss가 점점 증가함을 알 수 있었다. 이는 과잉적합이 진행될 때 나타나는 양상인데 과잉적합을 방지하기 위한 규제로 기존의 LeNet5모델에 Dropout 레이어를 추가하여서 학습해보기로 하였다. 결과는 아래와 같다.



풀링층 뒤마다 Dropout을 진행하였고 1,2 번째 Dropout은 25%, 3번째는 50%로 진행하였다. 아무 규제도 적용하지 않았을 때 보다는 정확률이 57.7%로 약 4.3%정도 향상하였다. 또한 손실함수 그래프를 통해 과잉적합이 일어나지 않고 정상적으로 우하향하는 그래프를 그리는 것을 볼 수 있다. 이를 통해 Dropout 레이어를 모델에 추가하는 것은 과잉적합 방지에 도움이 된다는 것을 알 수 있었다. Epoch을 50으로 진행하였을 때는 수렴이 덜 된 것 같아 100까지 늘려서 돌려보기로 했다.



세대를 100세대까지 늘리자 기존의 학습보다 더 안정적인 수렴을 하는 것처럼 보였다. 정확률 또한 61%로 증가한 것을 볼 수 있었고, 손실함수 곡선 또한 안정적으로 감소 혹은 수렴하는 것이 보였다. 하지만 LeNet5의 경우 CNN의 아주 초창기 모델이고 현대시대의 고화질 사진들의 특징을 추출하기에는 부적합하지 않을까? 라는 생각을 했다. MNIST 데이터셋과 같은 숫자들은 잘 분류해내지만 영상인식에는 한계가 있을 것이라고 생각하여 좀 더 성능이 좋은 영상인식 모델을 찾아서 적용해보고 싶었다.

테스트 데이터셋은 총 12장으로 아래와 같이 구성했다.
학습했던 이미지 5장[갈비탕, 간장게장, 고등어구이, 계란말이, 계란찜],
인터넷에서 가져온 사진 3장[고등어구이, 간장게장, 갈비탕],
직접 찍은 사진 4장[갈비탕, 계란말이, 계란말이, 갈비찜]

- 학습했던 이미지 5장

- 인터넷에서 가져온 사진 3장

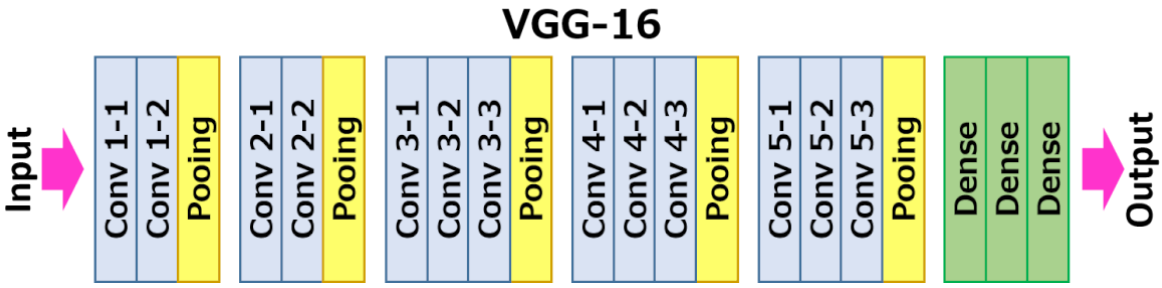
- 직접 찍은 사진 4장

위의 테스트 데이터셋을 통해 학습이 끝난 모델을 통해 예측을 해보았다. 이미 학습했던 데이터, 인터넷에서 가져온 데이터, 그리고 직접 찍은 사진들을 입력으로 넣어보았다. 결과는 아래와 같다.

```
internet_galbiTang.jpg 사진은 갈비탕 (으)로 판별됩니다.
Internet_ganzangGaezang.jpg 사진은 간장게장 (으)로 판별됩니다.
Internet_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_eggRoll.jpg 사진은 계란말이 (으)로 판별됩니다.
learned_eggSteamed.jpg 사진은 계란찜 (으)로 판별됩니다.
learned_galbiTang.jpg 사진은 갈비탕 (으)로 판별됩니다.
learned_ganzangGaezang.jpg 사진은 간장게장 (으)로 판별됩니다.
learned_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_eggRoll.jpg 사진은 계란말이 (으)로 판별됩니다.
photo_eggRoll2.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_galbiTang.jpg 사진은 감자탕 (으)로 판별됩니다.
photo_galbiZzim.jpg 사진은 계란말이 (으)로 판별됩니다.
```

예측결과를 보면 이미 학습했던 데이터에 대해서는 상당히 정확률이 높은 것을 알 수 있었다. 또한 인터넷에서 가져온 데이터에 대해서도 높은 정확률을 보였다. 하지만 직접 찍은 사진에 대해서는 인식률이 많이 떨어졌는데, 일반적으로 색감이나 이미지의 모양이 일반적인 영상들과는 달라 그런 것 같다. 학습한 이미지들은 대체로 전체적인 음식의 모습이 보이는 반면 직접 촬영한 영상 같은 경우에는 음식이 확대되어 있거나 조명이 밝지 않아서 색이 어둡게 나왔었다.

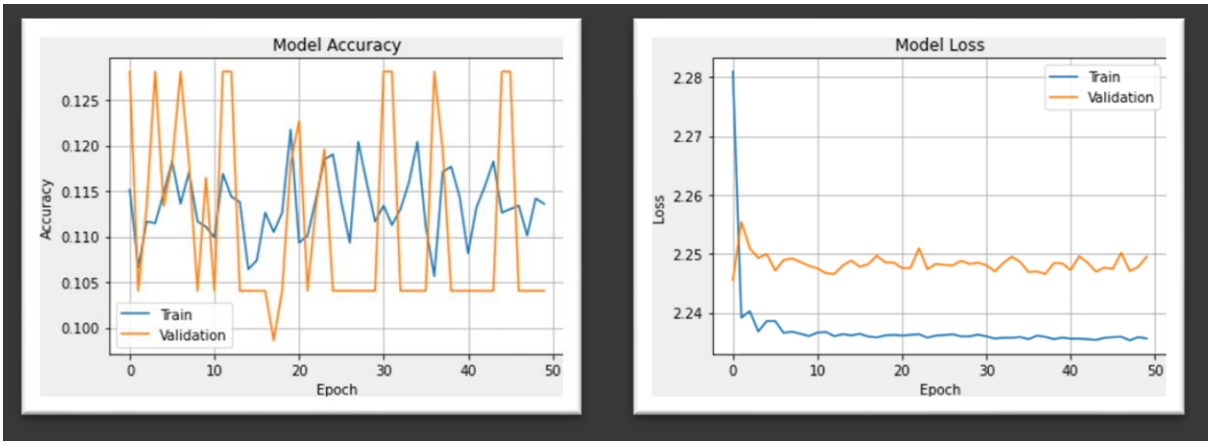
4.2 VGG16 모델



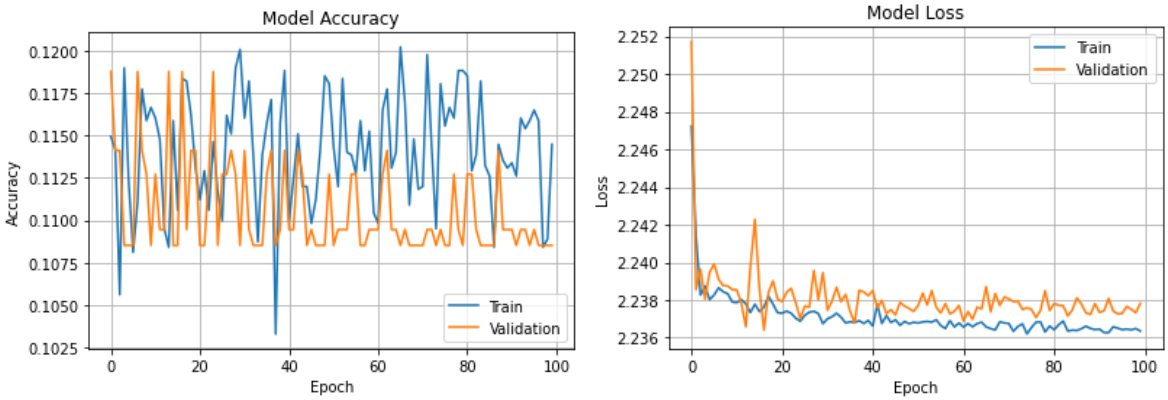
VGG16은 옥스포드 대학의 연구팀 VGG에 의해 개발된 모델로, 2014년 이미지넷 이미지 인식 대회에서 준우승을 한 모델이다. 총 16개 층으로 구성된 모델이다. 3 x 3의 작은 사이즈의 필터 커널을 설정하여서 네트워크의 깊이를 깊게 만들었다. VGG16 모델이 이미지 인식에 있어 좋은 성능을 낸다고 생각하여서 정확률을 올리기 위해 도입해보기로 결정했다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2, 2, 512)         2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2, 2, 512)         2359808
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 2, 2, 512)         2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              2101248
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 10)                40970
=================================================================
Total params: 33,638,218
Trainable params: 33,638,218
Non-trainable params: 0
```

← VGG16 모델의 구조



VGG16의 구조상 깊은 CNN이다 보니 학습에서 굉장히 많은 시간이 소요되었다. 또한 영상 인식에서 뛰어난 성능을 거두었다고 하기 부끄러울 만큼의 정확도를 보여주었다. 정확한 이유는 모르겠지만 정확률은 약 10~11% 사이가 번갈아 가면 나왔고 들이는 학습시간에 비해 실망스러운 결과였다. 다행히도 과잉적합과 관련된 문제는 없었지만 애초에 손실함수에 값이 너무 높아 실제 값과의 오차가 많이 나는 상황이었다. 'VGG16이 Dropout이 없어서일까?' LeNet5에 적용한 것과 같이 Dropout을 풀링층 이후에 모두 추가하여서 학습해보기로 하였다.



VGG16 에 Dropout 층을 추가하여 보아도 이렇다 할 효과는 없었다. 이후 원하는 결과가 나오지 않아서 VGG16 모델과 성능, 학습에 대한 조사를 추가로 진행해 보았고 VGG16같은 딥CNN의 경우 큰 데이터의 트레이닝에 효과적이란 말을 들었다. 아무래도 클래스당 약 1천장의 사진밖에 학습시키지 않아서인지 수십만장을 학습하기 위해 설계된 VGG16은 적절하지 않았다는 판단을 했다. 테스트 데이터셋에 대한 결과를 통해서도 학습 및 예측이 안된다는 것을 알 수 있었다.

```
internet_galbiTang.jpg 사진은 고등어구이 (으)로 판별됩니다.
Internet_ganzangGaezang.jpg 사진은 고등어구이 (으)로 판별됩니다.
Internet_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_eggRoll.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_eggSteamed.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_galbiTang.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_ganzangGaezang.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_eggRoll.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_eggRoll2.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_galbiTang.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_galbiZzim.jpg 사진은 고등어구이 (으)로 판별됩니다.
```

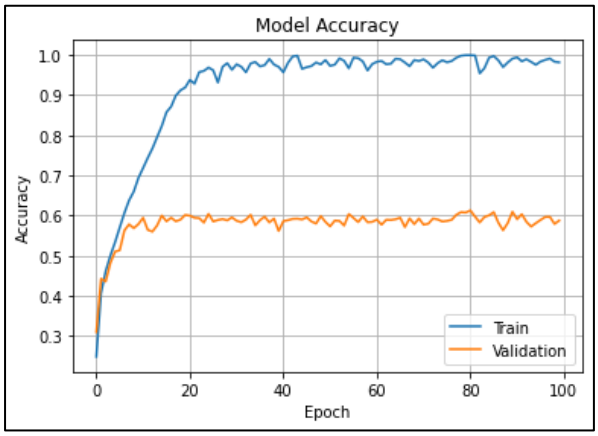
4.3 직접 설계한 모델

Layer (type)	Output Shape	Param #
conv2d_60 (Conv2D)	(None, 30, 30, 32)	896
conv2d_61 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_34 (MaxPooling	(None, 14, 14, 32)	0
conv2d_62 (Conv2D)	(None, 12, 12, 64)	18496
max_pooling2d_35 (MaxPooling	(None, 6, 6, 64)	0
conv2d_63 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_36 (MaxPooling	(None, 2, 2, 64)	0
flatten_10 (Flatten)	(None, 256)	0
dense_28 (Dense)	(None, 256)	65792
dense_29 (Dense)	(None, 10)	2570

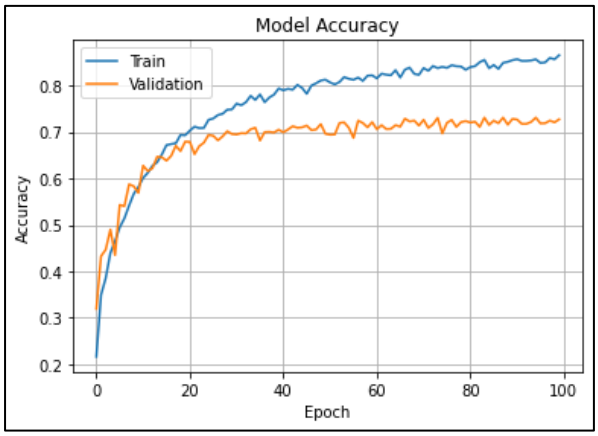
Total params: 133,930
Trainable params: 133,930
Non-trainable params: 0

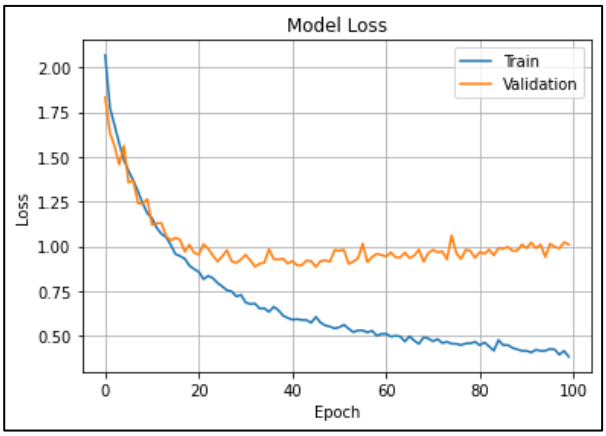
← 설계한 모델 구조

Keras를 사용하여 직접 CNN모델을 설계해보기로 하였다. 제작한 모델은 C-C-P-C-P-C-P-FC-FC 의 구조로 설계하였고 dropout은 우선 빼고 실험해보기로 하였다. 직접 제작한 모델에 100세대 동안 학습했을 때의 결과는 아래와 같다.



정확률은 58%정도로 학습이 진행되었지만 손실함수를 보면 LeNet5 모델을 통해 학습시켰을 때와 비슷하게 과잉적합이 일어나는 것을 볼 수 있다. 앞선 예시들과 마찬가지로 과잉적합을 규제하기 위해 Dropout층을 넣고나서 새로 학습을 해보기로 하였다.



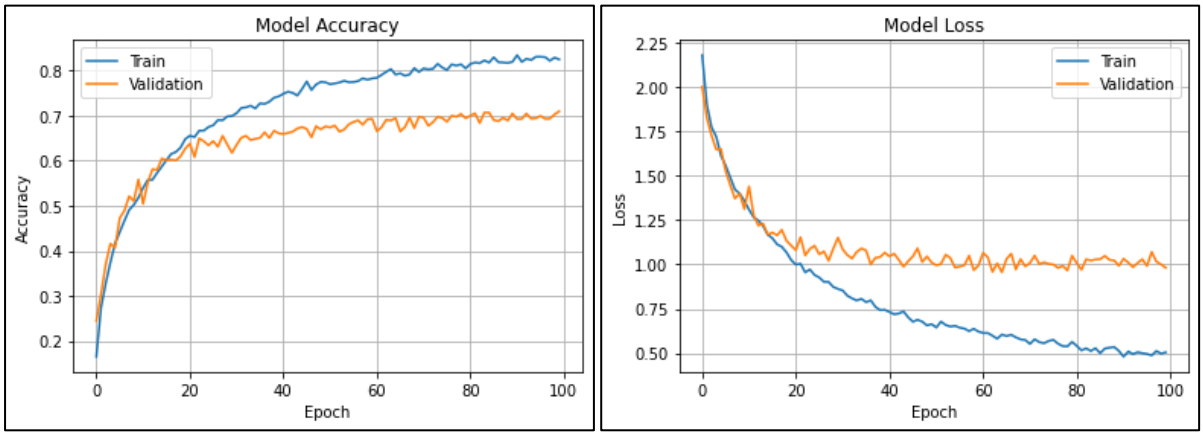


Dropout층을 추가하여 규제를 진행하니 과잉적합이 어느정도 잡히는 모습을 볼 수 있고, 정확률 또한 72%정도로 약 15%정도 증가한 것을 볼 수 있다. 지금까지 진행했던 실험들 중 가장 높은 정확률을 가지고 있었고 손실률도 제일 낮았다. 하지만 과잉적합을 모두 잡은 것은 아니기에 추가적인 규제도 함께 시도해보기로 하였다. 과잉적합을 효과적으로 해결할 수 있는 방안을 찾아보다 출력층 직전의 노드 수를 줄여 불필요한 최종 매개변수를 제거하면 문제를 완화할 수 있다는 설명을 보고 바로 실험해보기로 생각하고 모델을 약간 수정했다. 모델은 C-C-P-D-C-P-D-C-P-D-FC-D-FC-FC-FC 의 순서로 레이어를 쌓았다. 기존의 모델보다 Dropout 계수를 약간 상향 조정하였고 완전 연결층을 2개 추가하여 출력층에서 노드 수를 급감하지 않고 조금씩 줄여 출력층 직전의 노드를 16개 까지 줄였다.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_19 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_20 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_21 (Dropout)         (None, 2, 2, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_15 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_22 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 32)                8224
_________________________________________________________________
dense_17 (Dense)             (None, 16)                528
_________________________________________________________________
dense_18 (Dense)             (None, 10)                170
=================================================================
Total params: 140,282
Trainable params: 140,282
Non-trainable params: 0
```

← 규제를 추가한 모델 구조

변화한 모델의 정확률과 손실함수 그래프는 아래와 같다.



규제를 조정하기 전의 그래프보다 정확률은 약 1% 떨어진 71%를 기록하였지만 과잉적합은 해결한 모습을 보인다. 약간 떨어진 정확률의 경우 과잉적합으로 인한 잘못된 정확률이 아닐까 라는 생각도 들었기에 이 문제를 해결한 것에 큰 의의를 두었다. 그리고 이어서 앞서 진행했던 동일한 테스트 데이터셋을 통해 모델이 음식을 분류 예측해보도록 하였다.

```
internet_galbiTang.jpg 사진은 갈비탕 (으)로 판별됩니다.
Internet_ganzangGaezang.jpg 사진은 고등어구이 (으)로 판별됩니다.
Internet_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_eggRoll.jpg 사진은 계란말이 (으)로 판별됩니다.
learned_eggSteamed.jpg 사진은 갈비탕 (으)로 판별됩니다.
learned_galbiTang.jpg 사진은 갈비탕 (으)로 판별됩니다.
learned_ganzangGaezang.jpg 사진은 고등어구이 (으)로 판별됩니다.
learned_godeunguh.jpg 사진은 고등어구이 (으)로 판별됩니다.
photo_eggRoll.jpg 사진은 계란말이 (으)로 판별됩니다.
photo_eggRoll2.jpg 사진은 계란말이 (으)로 판별됩니다.
photo_galbiTang.jpg 사진은 갈비탕 (으)로 판별됩니다.
photo_galbiZzim.jpg 사진은 계란말이 (으)로 판별됩니다.
```

해당 모델 같은 경우에는 인터넷에서 가져온 사진, 학습했던 사진, 직접 찍은 사진들 에서도 꽤 높은 예측 정확률을 가진 것으로 나타났다.

LeNet5, VGG16보다 직접 설계한 모델이 가장 괜찮은 성능을 내는 모습을 통해서 층이 많다고 (깊다고) 다 좋은 것은 아니며 각각의 데이터셋에 적합한 모델을 찾아내고 설계해야 한다는 것을 알게 되었다.

5. 결론 및 느낀점 (Conclusion)

5.1 충분한 데이터셋이 필요하다.

본격적인 프로젝트를 시작하기 전 작은 파일럿 프로젝트의 개념으로 각 클래스마다 300장의 영상 데이터를 가지고 학습을 진행한 적이 있다. 딥러닝에 문외한이었던 나로서는 "300장의 이미지정도면 충분히 가능하지 않을까?" 라고 생각 했지만 현실은 그렇지 못했다. 그래서 더 풍부한 영상 데이터를 찾기 위해 노력했고 각 클래스별 1000여장이 되는 데이터셋을 찾아 학습에 사용하게 되었다. 충분한 데이터셋이 있어야 모델 학습에서의 안정성이나 성능을 보장할 수 있다고 생각을 가질 수 있는 계기가 되었다. 다음에는 좀 더 풍부한 데이터셋을 기반으로 클래스당 3천~1만장 정도의 이미지를 학습시키는 프로젝트를 진행해 보거나 적은 데이터를 이미지 증대를 통해 학습해보고 싶다고 느꼈다.

5.2 모델의 선택 혹은 설계는 신중해야 한다.

해결하고자 하는 문제가 있을 때, 해당 문제를 풀기에 가장 적절한 CNN 구조를 설계하는 것은 많은 시행착오가 필요했다. 예를 들자면, 모델에서 레이어를 몇 개를 사용할 것인가, 각 은닉층에서의 노드 수는 몇 개로 정할 것인가, 필터 크기를 어느 정도로 할 것인가 등 수많은 변수들을 고려해야 했었다. 또한 성능이 좋다고 평가된 모델이라고 무작정 도입해서 사용한다고 나의 데이터셋을 정확하게 분류할 수 있는 것은 아니었다. 해결하고자 하는 문제, 데이터의 양, 사용할 규제의 종류에 따라서 유연하게 모델을 결정해야 한다는 것을 느꼈다. ILSVRC 같은 대회에서 왜 많은 팀들이 그들의 모델을 개선하기 위해 노력하는 지 잘 느낄 수 있는 프로젝트였다.

5.3 규제의 유무는 모델의 성능에 큰 영향을 미친다.

이번 프로젝트를 통해 실험을 진행하면서 규제가 없는 모델에서는 과잉적합이 빈번하게 발생했었고 규제를 추가했을 때 이런 문제들이 잡히는 것을 확인할 수 있었다. CNN에서 규제할 수 있는 방법으로는 여러가지 방법이 존재하지만 그 중 Dropout이라는 쉬우면서도 강력한 규제를 사용하면서 규제가 모델의 성능이나 과잉적합에 이로운 영향을 준다는 것을 알게 되었다. 하지만 너무 과한 규제를 가했을 때는 (예. Dropout 계수를 너무 높게 하는 것) 성능의 저하를 가져온다는 것도 알게 되었다. 또한 출력층 직전의 은닉층 노드 수를 줄여서 과대적합 문제를 어느 정도 완화할 수 있다는 것도 새로 알게 되었다. 수업에서 자세하게 배우지 못한 규제들도 조금 더 공부해보았는데 흥미로운 내용이 많아서 CNN의 전체적 이해에 많은 도움이 된 것 같다.

참고 문헌

<도서>

1. 박유성,「텐서플로 케라스를 이용한 딥러닝」, 자유아카데미, 2020년

2. 솔라리스,「텐서플로로 배우는 딥러닝」, 영진닷컴, 2018년

<논문>

1. 김준섭, "딥러닝을 활용한 지문 이미지 분류에 관한 연구.", 순천향대학교 석사학위 논문, 2021

<웹사이트>

1. KerasKorea, "작은 데이터셋으로 강력한 이미지 분류 모델 설계하기",

https://keraskorea.github.io/posts/2018-10-24-little_data_powerful_model/, (2021.06.09)

2. 유리상자 속 이야기, "CNN을 이용하여, 얼굴을 분류해보자",

https://crystalcube.co.kr/192, (2021.06.09)

3. 흰고래의꿈, "자신만의 이미지 데이터로 CNN 적용해보기",

https://twinw.tistory.com/252, (2021.06.12)

4. Codinfox, "케라스와 함께하는 쉬운 딥러닝 - CNN모델 개선하기",

https://buomsoo-kim.github.io/keras/2018/05/04/Easy-deep-learning-with-Keras-10.md/, (2021.06.13)

5. 한 번 해보겠습니다., "VGGNet 개념정리",

https://daechu.tistory.com/10, (2021.06.13)