머신러닝과 딥러닝 기말 과제

교수명: 손영두 교수님
학번: 2013313217
이름: 조상연

1. Introduction

본 과제는 X-ray 검사에 대한 실제 경험이 바탕이 되어 선정하게 되었다. 20살 때, 보건소에서 X-ray 검사를 받았을 때 한쪽 폐에 대한 이상 소견이 나와 큰 병원으로 가게 됐고, 가서 보니 양쪽 폐 모두가 문제가 있어 바로 다음날 수술을 받았다. 보건소 의사는 다른 쪽 폐의 문제를 찾지 못했던 것이다. 그 이후로 여러 번 X-ray 촬영을 하고 소견을 들어보니 의사마다 X-ray 사진을 보고 진단하는 능력이 천차만별이라는 것을 깨닫게 되었다. 실제로 해외 연구 사례를 보면, 영상의학과의 전문의 진단율이 높았고, 그 이후론 판독 경험에 따라 진단율이 달라진다고 한다.[1] 결핵, 폐렴 진단 같은 것에 있어 전문가끼리도 같은 사진에 대해서 판독 소견에 차이를 보인다는 것이다. 이러한 문제는 국민 생명과 직결되는 문제이고, 요즘과 같이 호흡기 질환의 위험성이 대두되고 있기 때문에 머신러닝과 딥러닝을 이용해 이러한 X-ray 판독에 보조장치로서 활용할 수 있다면 큰 도움이 될 것이다. 이와 관련된 데이터셋으로 Kaggle에서 흉부 X-Ray 데이터셋을 찾을 수 있었다. [2]

2. Data

2.1. Data Description

사진 1 X-ray 데이터 분류 및 샘플

Kaggle의 흉부 X-ray 데이터는 폐렴 진단을 위한 이미지 데이터로 전체 데이터는 5863개로 정상

[1] http://www.newsmp.com/news/articleView.html?idxno=160167
[2] https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

1590개, 바이러스성 폐렴 1493개, 박테리아성 폐렴 2780개로 구성되어 있다. 파일은 대략 200~400KB의 JPEG파일로 해상도는 각 파일별로 약간의 차이가 있어 통일해주는 리사이징 해주는 작업이 필요했다. 처음 베이스라인 모델을 위해서 128x128으로 리사이징을 해주었다.

2.2. Data Split

모델 검증을 위해 데이터셋을 Train, Val, Test로 나누고 이를 기반으로한 실험을 진행하였다. Test Size가 20%로, Validation Set을 전체 16% 정도로 비중을 주어 Overfitting을 잘 막을 수 있도록 하였다. 학습 데이터 비중은 전체 64%이다.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_new,
onehot_encoded, test_size=0.2, random_state=22)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=32)
```

2.3. Data Augmentation

Data Augmentation은 Out-of-Sample 데이터에 더 강건한 모델을 만들 수 있도록 해주는 중요한 기법이다. Keras의 ImageDataGenerator 를 이용하여 rotation (사진 회전), zoom (확대, 축소), shift (좌우 이동) 등의 옵션을 주어 다양한 데이터를 생성할 수 있도록 하였다.

```
datagen = ImageDataGenerator(
  rotation_range = 30,  # randomly rotate images in the range
  zoom_range = 0.2, # Randomly zoom image
  width_shift_range=0.1,
  height_shift_range=0.1,
  horizontal_flip = False,
  vertical_flip=False)
```

3. Model & Parameter

3.1. Baseline Model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 16)      448
_________________________________________________________________
activation (Activation)      (None, 126, 126, 16)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        4640
_________________________________________________________________
activation_1 (Activation)    (None, 61, 61, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 28800)             0
_________________________________________________________________
dense (Dense)                (None, 64)                1843264
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 195
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,848,547
Trainable params: 1,848,547
Non-trainable params: 0
_________________________________________________________________
```

그림 1 Baseline Model Summary

Baseline 모델은 이미지 데이터를 처리하는데 유용한 CNN(Convolutional Neural Network) 을 사용하였다. CNN Layer와 MaxPooling Layer를 각각 두 개씩 쌓고 이를 Flatten하게 해준 후 마지막에 softmax activation function을 통해 삼진 분류를 해주는 모델이다. 이러한 Baseline 기준으로 위 데이터 전처리를 따랐을 때 **정확도 65.18%** 가 나오는 것으로 확인되었다.

3.2. Image Resizing

기존의 이미지는 128*128로 리사이징하여 사용하였으나 원본이미지에서 너무 작은 이미지로 변환하다보니 중요한 정보가 사라질 수 있다. 그러므로 이를 방지하기 위해 기존의 이미지 크기를 224px로 늘려 좀 더 많은 정보를 담으면서도 학습의 부담을 크게 주지 않도록 하였다. 실제로 이를 적용하여 Baseline Model을 똑같이 학습시켰을 때 **약 5%p의 큰 성능 향상**을 보였다.

3.3. Epoch & Early Stopping

실제 모델의 학습 그래프를 살펴보면 Val Acc도 안정적으로 증가하고 있고 Training ACC도 증가하는 추세로 보아 아직 Fitting 이 부족하다는 것을 알 수 있다. 즉 Epoch를 늘려 더 많은 학습을 수행해야하지만 Overfitting의 위험성을 안고 있다. 그러므로 Validation Set을 통한 검증을 통해 Overfitting이 되기 전 학습을 중단시킬 수 있는 Early Stopping을 도입하여야 한다. Early Stopping은 Keras 내에서 Callback 형식으로 제공하고 있다. 이를 이용해 Validation Set에 대한 loss가 내려가다가 다시 올라갈 경우 몇 epoch까지 참을 것인지 (patience) 정하고 해당 patience 초과로 loss 가 계속 올라갈 경우 학습을 중단하고 가장 낮았던 loss였을 때의 weight로 모델을 반환할 수 있다.

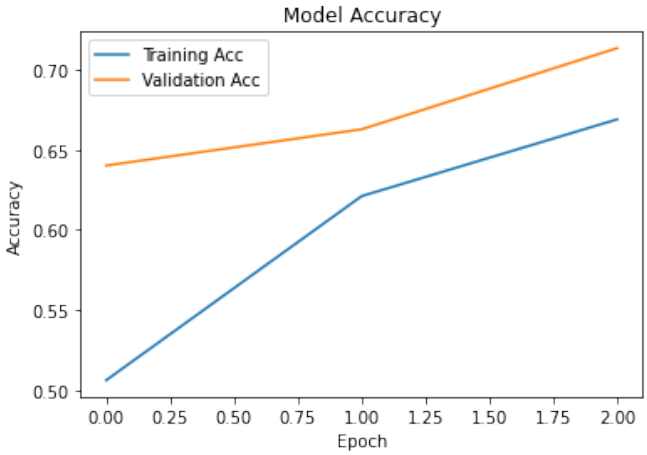

그림 2 Baseline 모델 학습 중 정확도 그래프

EarlyStopping Callback 함수

```
EarlyStopping_CallBack = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=10, verbose=0,
mode='auto', restore_best_weights=True)
```

Callback 함수 적용

```
history = model.fit(datagen.flow(X_train,y_train, batch_size = 32),
          epochs=100,
          callbacks=EarlyStopping_CallBack,
          validation_data = datagen.flow(X_val, y_val))
```

3.4. Batch Normalization & Dropout

주석 처리된 Batch Normalization을 적용해보고 그 효과를 본 뒤 Dropout도 적용해보고 Dropout의 Drop Rate를 0.2, 0.5, 0.8로 각각 조정하여 어떤 효과를 가지는지 실험해보고자 한다. 하지만 Batch Normalization 적용했을 때 정확도가 7%p 정도 감소하였다. 이는 정규화를 진행하면서 입력 분포를 일정하게 만들었지만 이것이 오히려 성능에 방해 요소로 작용한 것으로 보인다. 특히 Loss가 학습 중 매우 불규칙한 모습을 보이며 학습이 제대로 안되는 모습을 보여 생각보다 빠른 18 epoch만에 학습이 종료되었다.

그렇기에 Dropout은 Batch Normalization을 제외하고 아래 코드와 같이 적용해보았다. Dropout rate 0.2를 마지막 레이어 전에 추가하였고 이를 통해 Overfitting 문제를 방지하고자 하였다. 학습 중 Loss가 안정적으로 하락하며 특히 Validation Loss가 이전 BatchNorm 적용 모델에서 0.65가 가장 낮은 수치였다면 이번 Dropout 0.2 모델에선 0.51 까지 낮아지고 정확도는 72.95%까지 높아졌다. Dropout 0.5에선 정확도 73.03%로 소폭 증가했지만 Dropout 0.8에서 정확도가 64.33%로 대폭 낮아져 적당한 수준의 Dropout이 필요함을 알 수 있었다.

```
model.add(Dropout(0.2))
model.add(Dropout(0.5))
model.add(Dropout(0.8))
```

3.5. Transfer Learning

관련된 논문을 탐색하던 중 같은 문제에 대해 Transfer Learning 을 적용하여 기존 모델보다 훨씬 더 좋은 성능을 보인 논문을 찾을 수 있었다. [3] 해당 논문에선 기존의 CNN 모델의 Pre-trained된 Weight를 이용하여 마지막 FC레이어만 해당 도메인으로 바꾸는 Transfer Learning을 적용하여 이진분류 정확도 99%, 삼진분류 정확도 96%의 높은 성능을 달성하였다. 논문의 핵심은 여러 모델(AlexNet, ResNet, DenseNet, SqueezeNet) 중 DenseNet201에서 가장 성능이 좋았으며 이때 설정한 하이퍼 파라미터는 Optimization: SGD, Momentum: 0.9, Learning Rate: 0.0003, Mini-batch: 16 이다.

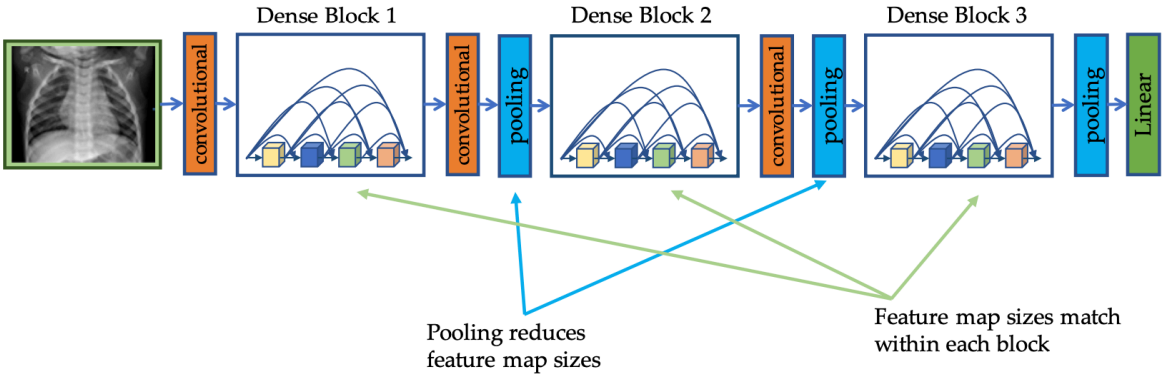

그림 3 DenseNet201 구조도[4]

DenseNet의 가장 큰 특징은 CNN 이전 레이어의 Feature Map을 계속해서 이어준다는 것으로 이를 통해 초반 레이어의 Feature를 재사용하고 이와 함께 Vanishing Gradient 문제를 해소할 수 있다는 장점이 있다. Keras에선 이 DenseNet의 학습된 모델을 제공하고 있으며 ImageNet기반의 Weight를 폐렴 진단에도 사용한다는 것이 핵심이다. 자칫 도메인이 크게 달라 효율이 의심될 수 있지만 실제 적용해보니 큰 성능 향상과 빠른 학습이 가능하다는 점에서 매우 효율적인 학습 방법이다.

[3] Rahman, Tawsifur, et al. "Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection using Chest X-ray." Applied Sciences 10.9 (2020): 3233.

[4] 위와 같음

Densenet 모델 불러오기 및 컴파일

```
import tensorflow as tf
from tensorflow import keras

def get_transfer_model(dropout=False):
  base_model = tf.keras.applications.DenseNet201(
      include_top=False,
      weights="imagenet"
  ) # Densenet 불러오기
  base_model.trainable = True # DenseNet을 학습 가능하도록 함
  inputs = keras.Input(shape=(224,224,3))
  x = base_model(inputs, training=True)
  x = keras.layers.GlobalAveragePooling2D()(x)
  if dropout:
    x = keras.layers.Dropout(dropout)(x)
  outputs = keras.layers.Dense(3, activation = 'softmax')(x)
  return keras.Model(inputs, outputs)
```

```
densenet_model = get_transfer_model(dropout=0.2)
densenet_model.compile(optimizer = "adam",
              loss = 'categorical_crossentropy' ,
              metrics = ['accuracy'])
```

위 코드를 통해 DenseNet Transfer Learning을 진행하였고 Dropout, Batch Size, Optimization, Learning Rate를 조절해가며 하이퍼 파라미터 튜닝을 진행하였다. 결과는 아래와 같다.

4. Experiment

Model	Epoch	Acc	Recall	F1-Score
Baseline	3	65.18%	0.65	0.66
Baseline + Image Resizing (128 to 224)	3	70.13%	0.69	0.71
Baseline + Image Resizing + Batch Normalization	18	63.56%	0.58	0.52
Baseline + Image Resizing + Dropout (0.2)	86	72.95%	0.70	0.72
Baseline + Image Resizing + Dropout (0.5)	52	73.03%	0.70	0.73
Baseline + Image Resizing + Dropout (0.8)	46	64.33%	0.60	0.59
Densenet 201 + Adam + Lr 0.001 + Batch size 32	29	80.71%	0.80	0.80
Densenet 201 + Adam + Lr 0.001 + Batch size 32 + Dropout (0.2)	32	81.14%	0.80	0.81
Densenet 201 + SGD + Lr 0.001 + Batch size 32 + Dropout (0.2)	**16**	**82.67%**	**0.82**	**0.82**
Densenet 201 + SGD + Lr 0.001 + Batch size 32 + Dropout (0.5)	13	81.14%	0.79	0.80
Densenet 201 + SGD + Lr 0.0003 + Batch size 32 + Dropout (0.2)	24	81.99%	0.82	0.82
Densenet 201 + SGD + Lr 0.001 + Batch size 16 + Dropout (0.2)	**19**	**83.70%**	**0.84**	**0.84**

*** Recall, F1-Score Macro Average 기준**

우선 Transfer Learning을 적용했을 때 기존 모델보다 훨씬 빠른 학습이 가능했다 Epoch 기준으로 절반 이하 수준으로 시간이 걸렸으며 동시에 더 나은 성능을 보였다. **Dropout Rate 0.2를 적용했을 때 더 나은 성능**을 보였고 0.5을 적용했을 때 소폭 감소하는 것을 알 수 있다. 이는 이미 학습된 모델이기 때문에 Dropout Rate를 더 높였을 때 모델이 유실된 정보를 더 추측하기 보단 기존에 학습된 Weight에 더 영향을 받은 것이 아닐까라고 추측해볼 수 있다. 또한 **Adam보다 SGD에서 더 안정적이고 빠른 학습**을 보여주었고 Learning Rate를 논문 기준인 0.0003까지 낮추었지만 오히려 학습이 더 느리고 정확도는 낮아지는 모습을 보였다. 이는 논문에선 적용하지 않은 Dropout으로 인한 것으로 보인다. 마지막으로 **Batch Size를 16으로 줄였을 때 정확도 약 1%p 소폭의 성능 향상**을 보였다. 이는 좀 더 가중치 업데이트를 자주 일어나도록 하여 이미 학습된 가중치들이 현재 데이터에 맞게 더 자주 바뀔 수 있도록 한 점이 유효한 것으로 보인다.

5. Conclusion

기존 CNN모델의 정확도 65.18%에서 정확도 83.70%까지 끌어올리며 여러 파라미터 튜닝을 경험해보았다. 특히 Densenet을 이용한 Transfer Learning은 성능과 속도를 모두 잡는 가장 좋은 선택이였으며 세부적으로는 적절한 Dropout을 통한 Overfitting 방지, Optimization과 Learning Rate를 이용한 학습 최적화, 마지막으로 Batch Size 조절을 통한 더 많은 가중치 업데이트를 진행하여 마지막 1%p까지 성능을 끌어올릴 수 있었다. 아쉬운 점은 좀 더 다양한 모델을 테스트 해보지 못한 점과 논문에서 나온 96% 성능까진 구현하지 못했다는 점이다. 논문에선 Test Size가 190정도로 매우 작았다는 점을 감안하더라도 83.7% 이란 성능은 96%와 괴리가 있다. 추후엔 Transformer나 Auto Encoder 등의 최신 모델들을 적용해보면 더 의미 있는 결과가 나올 수 있을 것이라 생각한다. 그리고 실제 문제를 해결하는데 있어 83.7% 란 정확도는 아직 아쉬운 수치라 생각한다. 하지만 데이터가 더 많아지고 더 정밀한 모델이 나온다면 충분히 극복이 가능한 수치라고도 생각한다.