기말고사 대체과제 보고서

과목명 : 딥러닝/클라우드
학번 : 32152339
주전공 : 응용통계학과
이름 : 송준영

목차

Ⅰ. 결과 보고

▶ Ⅰ-1) 작업 환경

Colab PRO (GPU ver.)

```
1  print('tensorflow version :',tf.__version__)
2  print('keras versoin :',tf.keras.__version__)
```

```
tensorflow version : 2.3.0
keras versoin : 2.4.0
```

▶ Ⅰ-2) Neural network structure

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 224, 224, 16)      448
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 224, 224, 16)      2320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
sequential_8 (Sequential)    (None, 56, 56, 32)        2160
_________________________________________________________________
sequential_9 (Sequential)    (None, 28, 28, 64)        7392
_________________________________________________________________
sequential_10 (Sequential)   (None, 14, 14, 128)       27072
_________________________________________________________________
dropout_5 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
sequential_11 (Sequential)   (None, 7, 7, 256)         103296
_________________________________________________________________
dropout_6 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
sequential_12 (Sequential)   (None, 512)               6425088
_________________________________________________________________
sequential_13 (Sequential)   (None, 128)               66176
_________________________________________________________________
sequential_14 (Sequential)   (None, 64)                8512
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 130
=================================================================
Total params: 6,642,594
Trainable params: 6,640,226
Non-trainable params: 2,368
_________________________________________________________________
```

▶ I -3) Test loss, accuracy

```
loss, acc = clf.evaluate(test_datagen.flow(X_test,  to_categorical(test_data['label'].values)), verbose=2)
print('test loss: {:0.4f}'.format(loss))
print('test accuracy: {:5.2f}%'.format(100*acc))
```

```
20/20 - 0s - loss: 0.2302 - accuracy: 0.9215
test loss: 0.2302
test accuracy: 92.15%
```

```
loss, acc = clf.evaluate(test_datagen.flow(X_test,  to_categorical(test_data['label'].values)), verbose=2)
print('test loss: {:0.4f}'.format(loss))
print('test accuracy: {:5.2f}%'.format(100*acc))
```

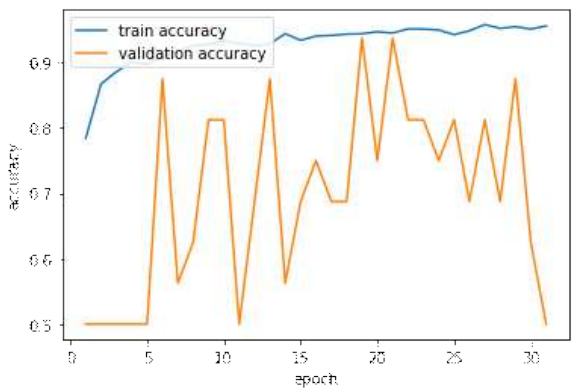
```
20/20 - 9s - loss: 0.2348 - accuracy: 0.9167
test loss: 0.2348
test accuracy: 91.67%
```

성능이 고정되지 않아 두 번 실행하였다.

▶ I -4) Training epoch loss/accuracy

```
Epoch 21/50
164/164 [==============================] - 55s 337ms/step - loss: 0.1469 - accuracy: 0.9451 - val_loss: 0.1236 - val_accuracy: 0.9375
Epoch 22/50
164/164 [==============================] - 55s 337ms/step - loss: 0.1274 - accuracy: 0.9514 - val_loss: 0.4636 - val_accuracy: 0.8125
Epoch 23/50
164/164 [==============================] - 55s 337ms/step - loss: 0.1265 - accuracy: 0.9512 - val_loss: 0.4035 - val_accuracy: 0.8125
Epoch 24/50
164/164 [==============================] - 55s 336ms/step - loss: 0.1323 - accuracy: 0.9497 - val_loss: 0.8008 - val_accuracy: 0.7500
Epoch 25/50
164/164 [==============================] - 55s 334ms/step - loss: 0.1448 - accuracy: 0.9426 - val_loss: 0.2995 - val_accuracy: 0.8125
Epoch 26/50
164/164 [==============================] - 55s 336ms/step - loss: 0.1327 - accuracy: 0.9485 - val_loss: 0.9536 - val_accuracy: 0.6875
Epoch 27/50
164/164 [==============================] - 55s 334ms/step - loss: 0.1093 - accuracy: 0.9579 - val_loss: 0.6737 - val_accuracy: 0.8125
Epoch 28/50
164/164 [==============================] - 55s 337ms/step - loss: 0.1190 - accuracy: 0.9522 - val_loss: 0.4750 - val_accuracy: 0.6875
Epoch 29/50
164/164 [==============================] - 55s 335ms/step - loss: 0.1109 - accuracy: 0.9546 - val_loss: 1.4364 - val_accuracy: 0.8750
Epoch 30/50
164/164 [==============================] - 55s 336ms/step - loss: 0.1167 - accuracy: 0.9512 - val_loss: 0.9701 - val_accuracy: 0.6250
Epoch 31/50
164/164 [==============================] - ETA: 0s - loss: 0.1262 - accuracy: 0.9558Restoring model weights from the end of the best epoch.
164/164 [==============================] - 55s 336ms/step - loss: 0.1262 - accuracy: 0.9558 - val_loss: 2.4519 - val_accuracy: 0.5000
Epoch 00031: early stopping
```

▶ I -5) 학습 곡선 그래프



Ⅱ. 과정 및 소감

▶ Ⅱ-1) Data Load

```
# train path 적재
normal_cases_dir = train_dir / 'NORMAL'
pneumonia_cases_dir = train_dir / 'PNEUMONIA'
normal_cases = normal_cases_dir.glob('*.jpeg')
pneumonia_cases = pneumonia_cases_dir.glob('*.jpeg')

# 적재할 리스트 생성
train_data = []

# 정상 이미지와 라벨 적재
for img in normal_cases:
    train_data.append((img,0))

# 폐렴 이미지와 라벨 적재
for img in pneumonia_cases:
    train_data.append((img, 1))

# 데이터 프레임으로 변환
train_data = pd.DataFrame(train_data, columns=['image', 'label'],index=None)

# 데이터 섞기
train_data = train_data.sample(frac=1.).reset_index(drop=True)

# 데이터 프레임 살펴보기
train_data.head()
```

```
img_size=224
X_train = np.zeros(shape=(len(train_data),img_size,img_size,3))

for idx,fname in enumerate(tqdm(train_data.image)):
    img = image.load_img(fname,target_size=(img_size,img_size))
    img_array_train = image.img_to_array(img)
    img_array_train = np.expand_dims(img_array_train,axis=0)
    X_train[idx] = img_array_train
```

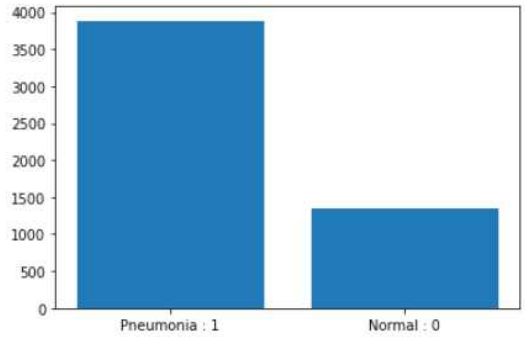
```
100% 5225/5225 [01:16<00:00, 68.68it/s]
```

train_data 리스트에 train이미지들의 path를 적재한다. keras의 image함수를 통해서 path을 이미지로 변환한 후, 이미지를 분석을 위해 array로 변환한다. 더불어 tqdm모듈을 활용하여 이미지를 변환할 때 상태바를 출력해 진행상황을 파악하도록 하였다.

▶ Ⅱ-2) EDA

```
plt.bar(["Pneumonia : 1","Normal : 0"],train_data['label'].value_counts())
```

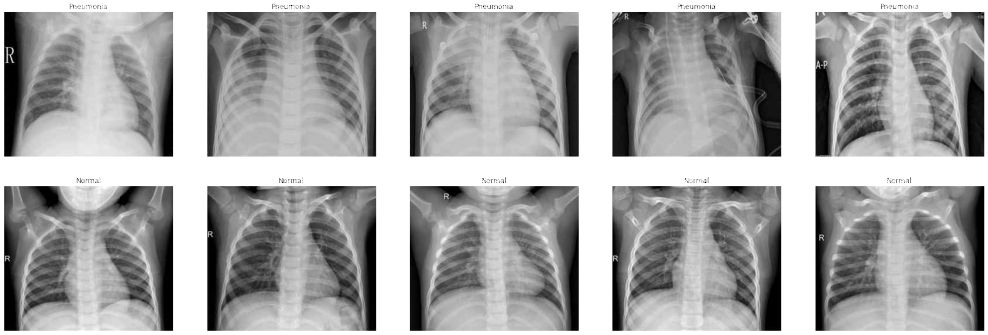
```
<BarContainer object of 2 artists>
```



폐렴이미지의 수가 3배정도 높은 빈도를 관찰할 수 있음

train set의 label 분포를 막대그래프를 통해서 관찰해본다. 폐렴이미지가 정상이미지의 3배 많은 빈도를 보인다. 모델링 과정에서 이 불균형을 조정하면 모델 성능 향상에 도움이 될 것으로 예상된다.

```
#train image 관찰하기
pneumonia_samples = (train_data[train_data['label']==1]['image'].iloc[:5]).tolist()
normal_samples = (train_data[train_data['label']==0]['image'].iloc[:5]).tolist()
# 폐렴과 정상 이미지 통합
samples = pneumonia_samples + normal_samples
del pneumonia_samples, normal_samples
# 시각화
f, ax = plt.subplots(2,5, figsize=(30,10))
for i in range(10):
    img = imread(samples[i])
    ax[i//5, i%5].imshow(img, cmap='gray')
    if i<5:
        ax[i//5, i%5].set_title("Pneumonia")
    else:
        ax[i//5, i%5].set_title("Normal")
    ax[i//5, i%5].axis('off')
    ax[i//5, i%5].set_aspect('auto')
plt.show()
```



육안상으로 정상과 폐렴을 구분하기 힘들다. 직접 관찰했을 때의 특징은 다음과 같다.

- 가운데로 이미지가 모두 정렬되어 가운데로 모두 정렬되어 있음.
- 이미지의 좌우상하가 명확함.

- 이미지의 각도가 일정함.
- 흑백이며 x-ray이기 때문에 밝기도 어느정도 일정함.

시각화를 통해서 얻어낸 정보를 바탕으로 Augmentation에 활용한다.

▶ Ⅱ-3) Augmentation

```
datagen = ImageDataGenerator(
          rescale = 1./255,
          featurewise_center=False,
          samplewise_center=False,
          featurewise_std_normalization=False,
          samplewise_std_normalization=False,
          zca_whitening=False,
          rotation_range = 30,
          zoom_range = 0.2,
          width_shift_range=0.1,
          height_shift_range=0.1,
          horizontal_flip = False,
          vertical_flip=False)
test_datagen = ImageDataGenerator(rescale=1./255)
```

30도씩 랜덤하게 회전
무작위로 20% 확대/축소
10%만큼 수평으로 이동
10%만큼 수직으로 이동

```
datagen.fit(X_train)
```

이미지 augmentation은 원래 이미지를 다양한 기법으로 변환하여 양(volume)을 증폭시키는 기법이다. 일반적으로 성능 향상과 과적합 방지에 효과적으로 알려져있다. 현재 dataset은 validation set이 총 16개이기 때문에 augmentation이 반드시 필요하다고 생각하였다. 그래서 keras framework의 ImageData을 사용하여 augmentation을 진행하였다.

EDA 과정에서 얻어낸 인사이트를 활용할 수 있었는데, 이미지의 밝기, 각도, 정렬상태, 상하좌우가 어느정도 자리가 잡혀있는 data이기 때문에 대부분 큰 변환 없이 augmentation을 진행하였다. 상하좌우로 뒤집거나, 밝기를 조정하거나 하는 등의 증폭은 행하지 않았다. 또한 추후 정확한 성능평가를 위하여 train set과 validation set에 적용한 rescale를 test set에도 적용하였다.

▶ Ⅱ-4) Bulid CNN architecture

```
def conv_block(filters):
    block = tf.keras.Sequential([
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D()
    ]
    )
    return block
```

최종 CNN Architecture 초반에 반복되어 사용되는 Convolution layers의 한 블록(Convolution layer*2 + Batch Normalization layer + MaxPooling layer)을 하나의 함수로 지정함으로써, 이후 구축되는 CNN Architecture의 code를 간소화시켰다.

Convolution layer에서 padding 옵션을 'same'으로 줌으로써 padding을 사용하여 출력이미지 사이즈가 입력이미지 사이즈와 동일하도록 하였다. 또한, Batch Normalization층을 매 층에 끼워 넣어줌으로써 Gradient Vanishing / Gradient Exploding 문제를 방지하였다. 추가적으로 Max pooling을 사용함으로써, 특징적인 정보가 손실되지 않도록 하였다.

```
def dense_block(units, dropout_rate):
    block = tf.keras.Sequential([
        tf.keras.layers.Dense(units, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(dropout_rate)
    ])
    return block
```

최종 CNN Architecture 반복되어 사용되는 DNN layers의 한 블록(Dense layer + Batch Normalization layer + Dropout layer)을 하나의 함수로 지정함으로써, 이후 구축되는 CNN Architecture의 code를 간소화시켰다.

깊은 모델을 생성할 것이기 때문에 Dense layer에서 activation function을 'ReLU'로 사용함으로써 Gradient Vanishing 문제를 방지하였다. 또한, Batch Normalization층을 매 층에 끼워 넣어줌으로써 Gradient Vanishing / Gradient Exploding 문제를 방지하였다. Dropout을 DNN 매 층 삽입함으로써 Overfitting을 방지하였다.

```
def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(img_size, img_size, 3)),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.MaxPool2D(),
        conv_block(32),
        conv_block(64),
        conv_block(128),
        tf.keras.layers.Dropout(0.2),
        conv_block(256),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),
        tf.keras.layers.Dense(2, activation='softmax')
    ])
    model.compile( loss='binary_crossentropy',optimizer='adam',metrics='accuracy')
    return model
```

최종 CNN Architecture을 위와 같이 설계하였다. 해당 과제의 평가 기준이 test accuracy

이므로 metric을 accuracy로 하였다. optimizer는 rmsprop을 시도했었으나 adam의 성능이 더 우수하여 adam을 선택하였다.

```
# 모델 밸런싱

train_data['label'].value_counts(normalize=True)[0]

initial_bias = np.log([train_data['label'].value_counts()[1]/train_data['label'].value_counts()[0]])
initial_bias

weight_for_0 = (1 / train_data['label'].value_counts()[0])*(len(train_data))/2.0
weight_for_1 = (1 / train_data['label'].value_counts()[1])*(len(train_data))/2.0

class_weight = {0: weight_for_0, 1: weight_for_1}

print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
```

```
Weight for class 0: 1.95
Weight for class 1: 0.67
```

EDA과정에서 training dataset의 label 빈도가 imbalanced한 것을 확인하였다. 이를 밸런싱하기 위하여 가중치 list를 생성한다. 정상과 폐렴의 빈도를 1:1로 맞춰주기 위하여 normal class(0)에는 1.95를, pneumonia class(1)에는 0.67을 할당하였다.

▶ Ⅱ-5) Training

```
model_dir = data_dir / 'model'
checkpoint_path = model_dir / "cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
batch_size=32

es = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=10,
                   verbose=1, mode='min', baseline=None, restore_best_weights=True)

# rlr = ReduceLROnPlateau(monitor='val_loss', patience = 8, verbose=1,factor=0.5, min_lr=0.000001)

chkpt = ModelCheckpoint(filepath=checkpoint_path, save_best_only=True, save_weights_only=True)

clf = build_model()
hist=clf.fit(datagen.flow(X_train,to_categorical(y_train),batch_size=batch_size),
             validation_data=datagen.flow(X_val, to_categorical(y_val),batch_size=batch_size),
             epochs=50,
             class_weight=class_weight,
             callbacks=[es, chkpt
                        #  ,rlr
                        ]
             )
```

최적의 training을 위하여 몇 개의 callback을 활용하였다. 먼저, EarlyStopping을 사용함으로써 최적의 Epoch(Validation set의 loss가 가장 낮은 epoch)를 찾으면 조기에 멈추도록 하였다. validation set의 양이 적기 때문에 validation accuracy로 조기멈춤을 한다면 다소 정확하지 않은 결과가 관찰돼 최종적으로는 validation loss를 사용하였다. 하지만 최적의

epoch를 찾더라도 patience가 10이므로 10 epoch만큼 뒤에 멈춘다. 그렇기에 최적의 epoch를 저장하도록 ModelCheckpoint를 사용하였다. 이로써 최적의 epoch를 찾을 준비를 마쳤다. ReduceLROnPlateau함수를 통하여 validation 성능이 개선되지 않으면 Learning rate를 낮아지도록 하였으나, 최종 성능에 도움을 주지 못하여 최종적으로는 사용하지 않았다.

위 코드를 보면 datagen.flow함수 내 batch_size를 설정한 것을 볼 수 있다. 이는 augmentation을 한 장당 32장만큼 한다는 의미이고, 이 하나의 증폭된 32장의 양이 한 mini batch로 훈련한다.

▶ Ⅱ-6) 소감

해당 경진대회에는 특히 아쉬운 점이 많이 남는다. 컴퓨터 자원의 한계가 제일 아쉬운 부분이었는데, colab pro에서도 부여된 RAM memory를 모두 소요하여 세션이 다운되는 문제가 반복되어 애를 먹었다. 특히 cross validation을 위해 3번정도 반복적으로 CNN을 훈련시키면 Colab pro 고성능 ram도 다운이 되어 결국엔 cross validation과 stacking을 사용하지 못하였다. 더불어 batch-size 역시 64이상 설정하였을 때 다운이 발생하는 경우도 빈번하였다.

더불어 valid accuracy가 변동이 심하여 데이터 수가 더 많았다면이라는 아쉬움도 남는다. 하지만 이러한 조건으로 인하여 augmentation에 대한 중요성을 알 수 있었고, 해당 방법론에 다양한 시도를 했던 것 같아 한편으로는 image 데이터를 다루는 역량이 조금은 발전하지 않았나 싶다. 실제 augmentation으로 인해 놀라운 성능 향상 효과를 경험하기도 하였다. 또한 전이학습을 사용하지 않고 직접 CNN architecture을 구축함으로써 CNN에서 필수적인 방법론에 대하여 더 집중력 있게 탐구할 수 있었다.

kaggle에서 image classification 경진대회를 통해 computer vision을 입문하니 많은 양을 빠르게 배울 수 있었던 것 같다. 타 참여자의 코드를 참고하면서, 해당 코드의 작동원리를 직접 공부하며 적용해보니 처음이지만 양호한 결과를 낼 수 있었다. 요즘 대부분의 경진대회는 computer vision 분야라고 해도 과언이 아니다. 그만큼 machine learning 분야에서의 computer vision의 중요성이 대두되고 있다. data를 다루는 장래를 희망하는 학생으로서 해당 분야 역시 연구와 학습에 최선을 다해야 한다는 것을 이번 경험을 통해 느낄 수 있었다.

또한 deep learning은 작업 환경의 자원이 굉장히 중요하다는 것을 깨달았다. 중간고사 경진대회와는 다르게 이미지 데이터를 훈련시키는 것이다 보니 물리적 환경의 아쉬움을 굉장히 많이 느꼈기 때문이다. local 환경으로 작업하니 out of memory 이슈가 자주 나타나 결국은 colab pro를 구입하였다. 물론 colab pro에서도 memory 이슈가 자주 발생하긴 하였으나, 로컬에 비해 굉장히 효율적으로 작업을 진행할 수 있었다. colab pro를 활용하여 앞으로도 machine learning 학습과 실습에 정진할 것이다.