머신러닝

학번: 2016722074

학과: 컴퓨터정보공학과

이름: 김영태

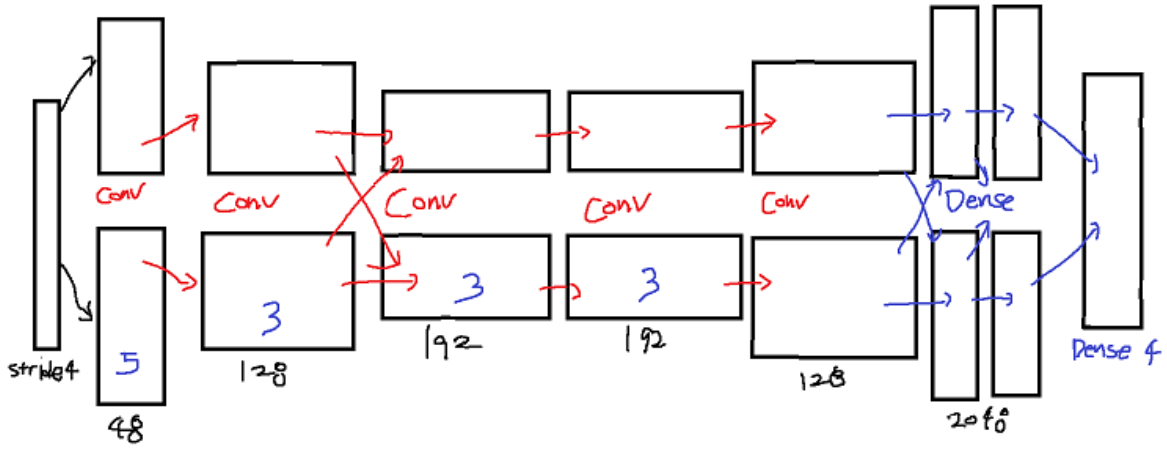
데이터 설명 및 목적

이번 과제에서 사용한 데이터셋은 kaggle에서 얻은 데이터 셋이다. 이미지 분류를 위한 다중 클래스 날씨 데이터 세트이다. 이 데이터셋은 총 4개의 클래스로 나뉘어져 있다. 각각 Cloudy, Rain, Shine, Sunrise이다. 각각 300, 215, 253, 357개의 데이터가 있으며 총 1125개의 이미지 파일이다.

이번 과제에서는 Convolutional Neural Network를 이용하여 이 4가지 클래스의 데이터를 분류하는 작업을 수행한다. 먼저 데이터셋을 train dataset과 test데이터 셋으로 분리한다. 이 train 데이터 셋으로 모델을 훈련시키고, test 데이터셋으로 훈련시킨 모델의 성능을 측정한다. 모델의 적절한 파라미터를 찾고 모델의 성능을 측정한다.

네트워크 구조 설명

이번 과제는 CNN을 사용한 Classification 문제이다. 해당 과제에서 사용한 모델은 AlexNet이다. AlexNet은 2012년에 개최된 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 대회의 우승을 차지한 컨볼루션 신경망(CNN) 구조이다. AlexNet의 구조는 다음과 같다. AlexNet은 총 8개의 Layer로 구성되어 있다. 5개의 Convolution Layer와 3개의 fully-connected 레이어로 구성되어 있다. 두번째, 네번째, 다섯번째 Convolution Layer들은 전 단계의 같은 채널의 feature맵들과만 연결되어 있는 반면, 세번째 Convolution Layer는 전 단계의 두 채널의 feature맵들과 모두 연결되어 있다.



먼저 첫번째 레이어에서는 입력 이미지를 컨볼루션 연산한다. Stride는 4이고 총 96개의 11*11*3 사이즈의 필터 커널로 연산한다. 활성화 함수는 ReLU함수를 사용했다.

ReLU함수는 rectified linear unit의 약자로서 같은 정확도를 유지하면서 tanh함수를 사용하는것보다 속도가 6배 빠르다는 장점이 있다.

이어서 3 x 3 overlapping max pooling이 stride 2로 시행된다. 그 다음에는 수렴 속도를 높이기 위해 local response normalization이 시행된다. 여기서 Max Pooling은 Convolution을 거쳐 나온 activation map을 resizing하여 새로운 Layer를 얻는다. 이 Max Pooling은 overfitting을 방지하기 위해서 존재한다. Pooling을 통해 feature map의 크기를 줄이고, 이는 overfitting을 방지한다. 다음으로 Batch Normalization Layer는 각 layer에 들어가는 input을 normalize 시킴으로써 layer의 학습을 가속한다.

두번째 레이어는 256개의 5 x 5 x 48 커널을 사용하여 전 단계의 feature map을 Convolution해준다. stride는 1로, zero-padding은 2로 설정했다. 따라서 27 x 27 x 256 feature map (256장의 27 x 27 사이즈 feature maps)을 얻게 된다. 역시 ReLU 함수로 활성화한다. 그 다음에 3 x 3 overlapping max pooling을 stride 2로 시행한다. 그 결과 13 x 13 x 256 feature maps을 얻게 된다. 그 후 local response normalization이 시행되고, feature map의 크기는 13 x 13 x 256으로 그대로 유지된다.

세번째 Convolution Layer는 384개의 3 x 3 x 256 커널을 사용하여 전 단계의 feature map을 Convolution 연산 해준다. stride와 zero-padding 모두 1로 설정한다. 따라서 13 x 13 x 384 feature map (384장의 13 x 13 사이즈 feature maps)을 얻게 된다. 역시 ReLU 함수로 활성화한다.

네번째와 다섯 번째 Convolution Layer까지 연산을 진행하고 ReLU 함수로 활성화한다.

여섯번째와 Layer는 6 x 6 x 256 특성맵을 flatten해줘서 6 x 6 x 256 = 9216차원의 벡터로 만들어준다. 4096개의 뉴런과 fully connected 해준다. 그 결과를 ReLU 함수로 활성화한다.

일곱번째 레이어(Fully connected layer)는 4096개의 뉴런으로 구성되어 있다. 전 단계의 4096개 뉴런과 fully connected되어 있다. 출력 값은 ReLU 함수로 활성화된다.

마지막 레이어는 softmax 함수를 적용해서 총 4개 클래스에 속할 확률을 나타낸다.

이 모델에서는 over fitting을 막기 위해 dropout을 사용했다. Drop out이란 fully-connected layer의 뉴런 중 일부를 생략하면서 학습을 진행하는 것을 의미한다. 일부 뉴런의 값을 0으로 바꿔 버리기 때문에 0으로 바뀐 뉴런들은 forward pass와 back propagation에 아무런 영향을 미치지 않는다.

소스코드 설명

```
train_datagen = ImageDataGenerator(rescale = 1.0/255.,

                                   rotation_range=90,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.3,

                                   validation_split=0.2,
                                   vertical_flip=True)


train_generator = train_datagen.flow_from_directory(dataset,
                                                    batch_size=16,
                                                    shuffle=True,
                                                    class_mode='categorical',
                                                    subset='training',
                                                    target_size=(150,150))

# valid_datagen = ImageDataGenerator(rescale = 1.0/255.)

valid_generator = train_datagen.flow_from_directory(dataset,
                                                    batch_size = 16,
                                                    shuffle=True,
                                                    class_mode='categorical',
                                                    subset='validation',
                                                    target_size=(150,150))

test_datagen = ImageDataGenerator(rescale = 1.0/255.)

test_generator = test_datagen.flow_from_directory(dataset,
                                                    batch_size = 16,
                                                    shuffle=True,
                                                    target_size=(150,150))
```

먼저 데이터셋을 train data, validation data, test data로 나눈다. 이때 ImageDataGenerator 라이브러리를 임포트해 사용한다.

```
81
82  model=tf.keras.models.Sequential([
83      tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(150,150,3)),
84      tf.keras.layers.BatchNormalization(),
85      tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
86      tf.keras.layers.Conv2D(filters=128, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
87      tf.keras.layers.BatchNormalization(),
88      tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
89      tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
90      tf.keras.layers.BatchNormalization(),
91      tf.keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
92      tf.keras.layers.BatchNormalization(),
93      tf.keras.layers.Conv2D(filters=128, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
94      tf.keras.layers.BatchNormalization(),
95      tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
96      tf.keras.layers.Flatten(),
97      tf.keras.layers.Dense(1024,activation='relu'),
98      tf.keras.layers.Dropout(0.5),
99      tf.keras.layers.Dense(1024,activation='relu'),
100     tf.keras.layers.Dropout(0.5),
101     tf.keras.layers.Dense(4,activation='softmax')
102 ])
103
104
105 model.compile(optimizer = 'Adam', loss= tf.keras.losses.CategoricalCrossentropy() ,metrics=['accuracy'])
106 model.summary()
107
108 history = model.fit(train_generator, validation_data = valid_generator,
109                     epochs=30
110                     )
111
```

다음으로 모델을 작성한다. 이번 과제에 사용한 모델은 AlexNet으로 총 8개의 Layer로 구성 되어 있다. Tf.keras.models.Sequential에 모델에 필요한 레이어들을 작성한다. 이때 기존의 AlexNet과는

다르게 하이퍼파라미터 값을 변경했다. 두번째 레이어는 128개의 filter, 세번째, 네번째는 256개, 다섯번째는 128개의 filter로 개수를 줄였다. 또 마지막 레이어의 Dense도 클래스가 4개이므로 4개의 출력으로 파라미터를 바꿨다. 이미지는 해당 모델의 각 레이어를 통과하면서 조금식 변형된다.

Layer (type)	Output Shape	Param #
conv2d_87 (Conv2D)	(None, 35, 35, 96)	34944
batch_normalization_75 (BatchNormalization)	(None, 35, 35, 96)	384
max_pooling2d_57 (MaxPooling2D)	(None, 17, 17, 96)	0
conv2d_88 (Conv2D)	(None, 17, 17, 128)	307328
batch_normalization_76 (BatchNormalization)	(None, 17, 17, 128)	512
max_pooling2d_58 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_89 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_77 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_90 (Conv2D)	(None, 8, 8, 256)	65792
batch_normalization_78 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_91 (Conv2D)	(None, 8, 8, 128)	32896
batch_normalization_79 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_59 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_19 (Flatten)	(None, 1152)	0
dense_53 (Dense)	(None, 1024)	1180672
dropout_30 (Dropout)	(None, 1024)	0
dense_54 (Dense)	(None, 1024)	1049600
dropout_31 (Dropout)	(None, 1024)	0
dense_55 (Dense)	(None, 4)	4100

먼저 input layer에서 150*150이었던 이미지는 35*35로 변형되고 필터의 개수인 96개에 맞게 35*35*96으로 변형된다. Batch normalization에서는 사이즈가 변형되지 않고 유지된다. Max

pooling layer에서 over fitting을 방지하기 위해 특성의 크기를 줄인다. 여기서 feature map은 17*17*96으로 변형된다. 두번째 Conv레이어 에서는 필터가 128이므로 17*17*128으로 변형된다. 두번째 Conv후 batch normalization에서는 사이즈가 유지되고 이어서 Max pooling에서 8*8*128으로 feature map의 크기가 바뀐다. 이어 5번째 Conv 레이어까지 feature map의 크기는 변하지 않는다. 원래의 AlexNet에서는 feature map의 크기가 더 컸는데, 해당과제에서 원래의 AlexNet의 파라미터를 그대로 적용하니 최종 accuracy가 좋지 못했다. 따라서 전체적인 파라미터를 줄이고 성능을 향상시켰다. 이어 Max pooling에서 feature map의 사이즈는 3*3*128까지 줄어든다. 6번째 레이어에서 3*3*128의 feature map을 flatten해서 벡터로 만들어준다. 이를 6번째 레이어의 1024개의 뉴런과 Fully connected하고 이 결과를 ReLU로 활성화한다. 7번째 레이어도 6번재 레이어와 동일하고 마지막 레이어는 4개의 뉴런으로 이루어지고 이전의 1024개의 뉴런과 Fully connected되어있다. 이 4개의 뉴런의 출력값에 Softmax함수를 적용해서 4개의 클래스에 속할 확률을 구한다.

해당 모델을 컴파일한 후 에포크 30으로 모델을 훈련시켰다.

```
112  import matplotlib.pyplot as plt
113
114  train_acc = history.history['accuracy']
115  train_loss = history.history['loss']
116  val_acc = history.history['val_accuracy']
117  val_loss = history.history['val_loss']
118
119  plt.figure(figsize=(12, 6))
120
121  plt.subplot(1, 2, 1) # jumlah row, jumlah column, column/row berapa
122  plt.plot(train_acc, label='Training')
123  plt.plot(val_acc, label='Validation')
124  plt.xlabel('Epochs')
125  plt.ylabel('Accuracy')
126  plt.legend()
127
128  plt.subplot(1, 2, 2) # jumlah row, jumlah column, column/row berapa
129  plt.plot(train_loss, label='Training')
130  plt.plot(val_loss, label='Validation')
131  plt.xlabel('Epochs')
132  plt.ylabel('Loss')
133  plt.legend()
134
135
136  plt.show()
```

모델을 훈련시킨 후 plt를 임포트해서 훈련 결과를 출력한다. 훈련의 결과는 train accuracy와 validation accuracy, train loss와 validation loss를 그래프로 표시한다.

```
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
new_model = model

probabilities = new_model.predict(valid_generator)


predictions = []
for prob in probabilities:
   best_index = np.argmax(prob)
   predictions.append(best_index)

labels = valid_generator.classes
# tn,fp,fn,tp = confusion_matrix(labels, predictions).ravel()
cm = confusion_matrix(labels, predictions).ravel()
#cm = [[tp, fp],
#      [fn, tn]]

cr = classification_report(labels, predictions)
print(cm)
print(cr)


loss, accuracy = model.evaluate(test_generator)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))


```

그 후 validation 데이터셋을 통해 모델의 probability를 얻는다. 이 확률 값들을 통해 Confusion matrix을 구한다.

```
              precision    recall  f1-score   support

           0       0.26      0.20      0.22        60
           1       0.20      0.02      0.04        43
           2       0.23      0.48      0.31        50
           3       0.24      0.23      0.23        71

    accuracy                           0.24       224
   macro avg       0.23      0.23      0.20       224
weighted avg       0.23      0.24      0.21       224
```

위와 같이 Classification_report 모듈을 사용하여 평가지표를 출력했다. 각각 Precision, Recall, F1 score, Accuracy등이 있다. Precision은 정확도를 나타내는 지표로 데이터를 얼마나 정확하게 분류했는지를 평가하는 척도이다. Precision은 정밀도이다. 모델을 통해 분류한 그룹이 있을 때 이 모델이 얼마나 믿을 정도로 해당 그룹을 분류했는지를 평가하는 척도이다. Recall은 전체 예측 중 얼마나 TP가 많은지에 대한 척도이다.

- **Accuracy(정확도)** = $\frac{TP + TN}{TP + TN + FP + FN}$
- **Precision(정밀도)** = $\frac{TP}{TP + FP}$
- **Recall(재현도)** = $\frac{TP}{TP + FN}$

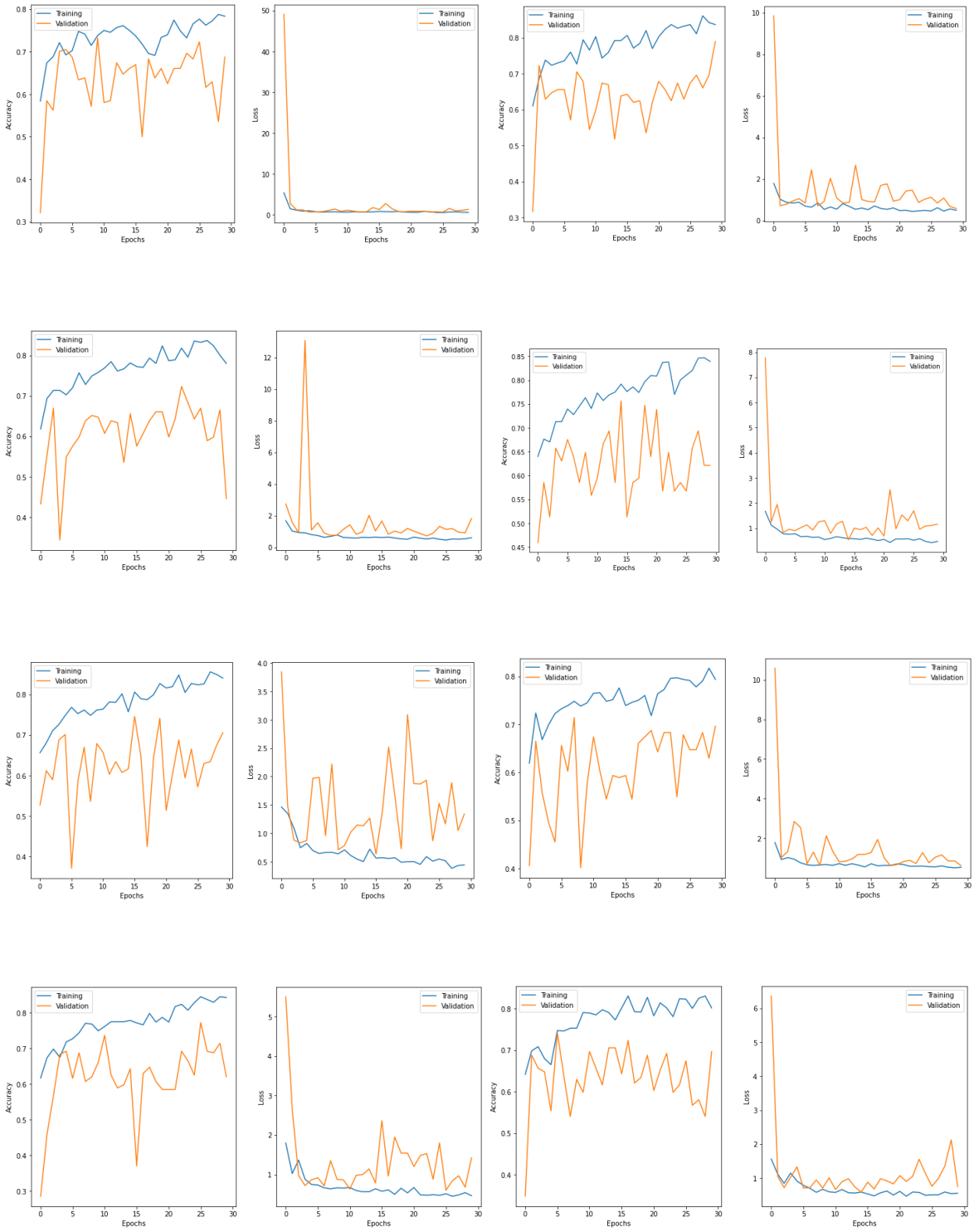
F1 score = 2*(Recall * Precision) / (Recall + Precision)

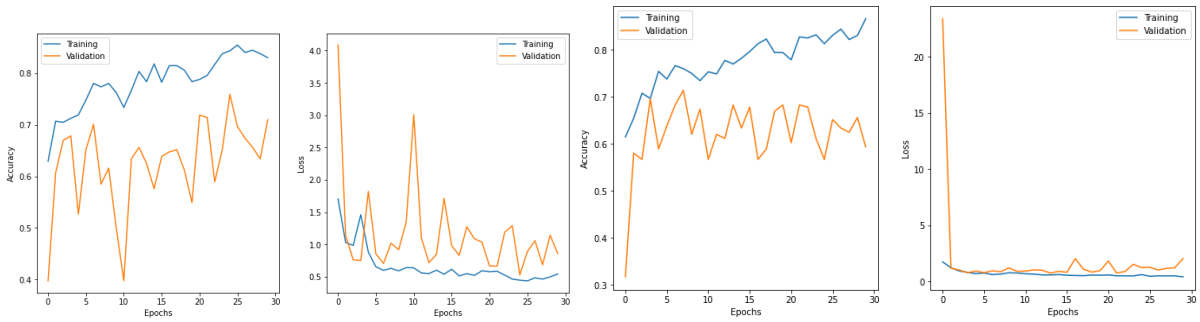
macro avg = (normal+abnormal) /2 * precision or recall or f1 score

weighted avg = normal/(normal+abnormal) * precision or recall or f1 score

마지막으로 test 데이터셋을 통해 evaluate함수를 사용해 모델의 최종 Loss값과 Accuracy를 측정한다.

실행결과






```
Loss = 0.97475
Accuracy = 0.68978
```

```
Loss = 0.49885
Accuracy = 0.80711
```

```
Loss = 0.58477
Accuracy = 0.75200
```

```
Loss = 1.07138
Accuracy = 0.72178
```

```
Loss = 0.99452
Accuracy = 0.74133
```

```
Loss = 0.63887
Accuracy = 0.75022
```

```
Loss = 1.45579
Accuracy = 0.73244
```

```
Loss = 1.94092
Accuracy = 0.61956
```

```
Loss = 0.51424
Accuracy = 0.78933
```

```
Loss = 0.75867
Accuracy = 0.78133
```

참고문헌

https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160

https://www.learnopencv.com/understanding-alexnet/

https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637