바이오 인공지능 프로젝트

Make Breas Cancer(BC) Classifier using Wisconsin Brest Cancer data

1. Data Info

```
In [4]: data.info()

<class 'pandas.core.frame.DataFrame'>
Float64Index: 569 entries, nan to nan
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      569 non-null    int64
 1   diagnosis               569 non-null    object
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
 14  perimeter_se            569 non-null    float64
 15  area_se                 569 non-null    float64
 16  smoothness_se           569 non-null    float64
 17  compactness_se          569 non-null    float64
 18  concavity_se            569 non-null    float64
 19  concave points_se       569 non-null    float64
 20  symmetry_se             569 non-null    float64
 21  fractal_dimension_se    569 non-null    float64
 22  radius_worst            569 non-null    float64
 23  texture_worst           569 non-null    float64
 24  perimeter_worst         569 non-null    float64
 25  area_worst              569 non-null    float64
 26  smoothness_worst        569 non-null    float64
 27  compactness_worst       569 non-null    float64
 28  concavity_worst         569 non-null    float64
 29  concave points_worst    569 non-null    float64
 30  symmetry_worst          569 non-null    float64
 31  fractal_dimension_worst 569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 146.7+ KB
```

Wisconsin Breast Cancer data는 기본적으로 32개의 column으로 이루어져있으며 1번 column에는 sample ID, 두 번째는 BC에 대한 진단명이 나와있고, 3~32 column에는 유방 조직에서 각 세포의 핵의 feature에 따른 mean, median, worst 값이 나타나있다.

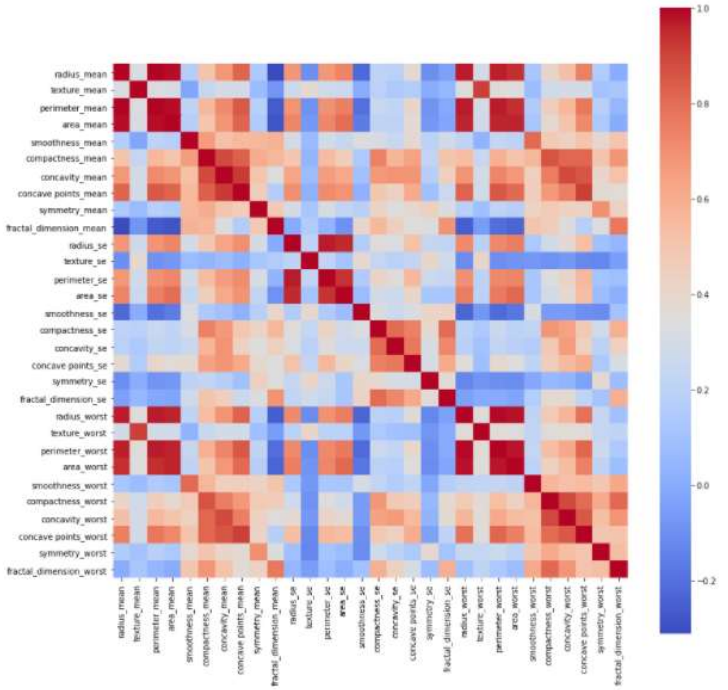
2. Data preprocessing

해당 feature들을 확인 해 보면 핵의 지름, 면적 등 각 feature간의 correlation이 어느정도 존재 하는 것을 알 수 있다. 따라서 Filter method의 filter로서 correlation을 계산 한 후에 적당한 feature selection 과정을 거쳐 적당한 feature만 골라 학습을 하였다.

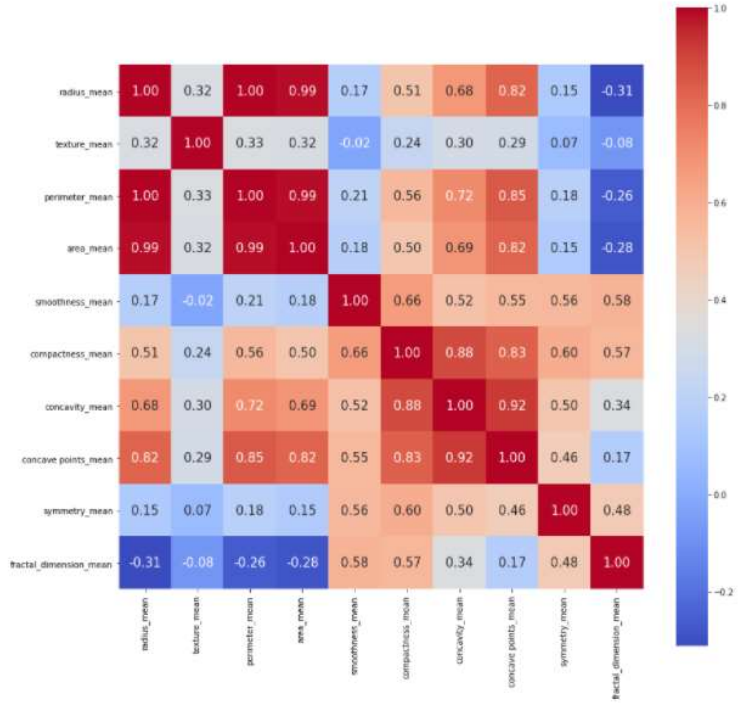
diagnosis

In [11]:
```
corr = data[features].corr()
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True,
        xticklabels= features, yticklabels= features,
        cmap= 'coolwarm')
```

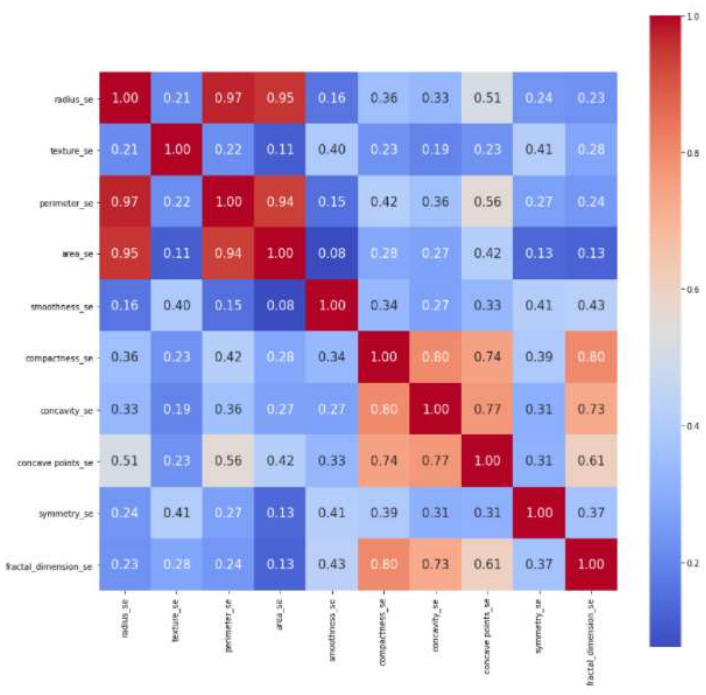
Out[11]: <AxesSubplot:>



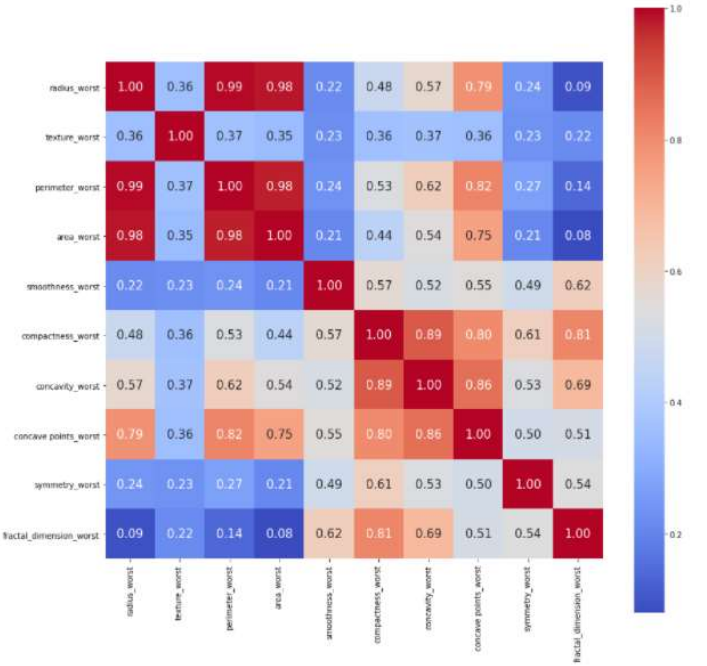
Out[12]: <AxesSubplot:>



Out[13]: <AxesSubplot:>



Out[37]: <AxesSubplot:>



각 feature간의 correlation을 계산해 heatmap으로 나타내어 보면 다음과 같이 mean, se, worst 별로 상관관계가 높은 feature들이 존재하는 것을 확인 하였고, feature가 의미하는 바와 실제 값으로 나온 correlation을 통해 적당히 판단해서 각 6개의 feature를 선택하였다. (correlation이 높은 feature가 다수 존재하면 model이 해당 feature에 너무 영향을 많이 받기 때문에 다른 feature들의 영향력을 감소 시킬 수 있음.)

prediction_features

['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'smoothness_se', 'compactness_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst', 'smoothness_worst', 'compactness_worst', 'symmetry_worst', 'fractal_dimension_worst']

mean_features

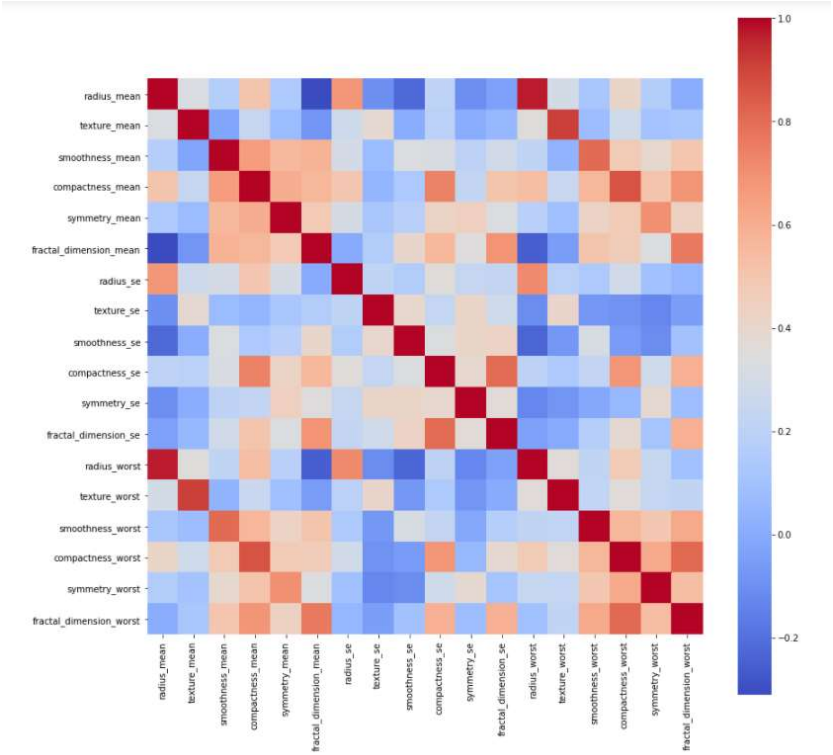
['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean']

se_features

['radius_se', 'texture_se', 'smoothness_se', 'compactness_se', 'symmetry_se', 'fractal_dimension_se']

se_features

['radius_se', 'texture_se', 'smoothness_se', 'compactness_se', 'symmetry_se', 'fractal_dimension_se']



Correlation이 높은 feature들을 하나의 대표 feature로 줄여서 correlation을 계산해보면 위 와 같은 plot이 그려지는 것을 확인 할 수 있다. Plot을 보면 correlation이 있는 feature가 줄어 든 것을 확인 할 수 있었다.

3. Model training/ Check Model Accuracy

Data에서 Training/Test set을 0.75:0.25 비율로 나눠주기

```
In [32]: #Data를 train/ test set으로 나눈다.
         train, test = train_test_split(data, test_size = 0.25)# 0.75 : 0.25 비율로 train/test set을 나눠 주었다.

         # 나누어진 set의 shape 확인
         print(train.shape)
         print(test.shape)

         (426, 31)
         (143, 31)

In [33]: train_X = train[prediction_features]# training set에서 사전에 선정한 feature에 대해서만 training set을 만들어준다.
         train_y=train.diagnosis# 각 sample에 대한 진단 label 저장
         # 위와 같은 과정을 test set에도 적용한다.
         test_X= test[prediction_features]
         test_y =test.diagnosis
```

만들어진 Training/Test set을 가지고 Random forest, SVM algorithm을 이용해 Classifier model 만들어 성능 비교

일반적으로 classfication performance가 좋은 Random forest와 Support Vector Machine Algorithms을 이용해 Breast Cancer Classifier model 만들기

```
In [41]: model=RandomForestClassifier(n_estimators=100)# 일반적으로 높은 성능을 보이는 RF 알고리즘으로 모델 생성

In [42]: model.fit(train_X,train_y)# RF 알고리즘을 이용해 model fit
Out[42]: RandomForestClassifier()

In [43]: prediction=model.predict(test_X)# 학습된 모델을 이용해 Test set의 Accuracy 측정

In [44]: metrics.accuracy_score(prediction,test_y) # test set에 대한 Accuracy
Out[44]: 0.9790209790209791

In [45]: model = svm.SVC()
         model.fit(train_X,train_y)
         prediction=model.predict(test_X)
         metrics.accuracy_score(prediction,test_y)
Out[45]: 0.9230769230769231
```

선택한 feature들의 mean, se, worst 값을 포함하는 feature set을 가지고 RF, SVM 알고리즘을 이용해 classifier를 만들어 test set에 대한 예측의 정확도를 살펴보면 Random forest의 경우 0.98의 분류 성능을 보였고, SVM의 경우 0.92정도의 분류 성능을 보이는 것을 확인 할 수 있었다. 다른 feature set을 가지고 분류 성능을 확인했을 때에도 대부분 90% 이상의 분류성능을 보이는 것을 확인 할 수 있었다.

Wisconsin Breast Cancer data를 이용했을 때 대부분 분류 성능이 높게 나온 것을 확인

Image에서 각 feature에 해당하는 값을 실수 값으로 변환한 data에서는 대부분 높은 분류 성능을 가지는 것을 확인 할 수 있었고, 이후에는 그냥 Image data 자체를 model learning에서 input으로 하여 분류를 진행하는 model을 만들어서 성능을 확인해 볼 예정이다.

IDC(major type of Breast Cancer) patch image를 이용한 Classifier

1. Data

Loading Data

```
# image 데이터 load
# image shape(5547, 50, 50, 3))
X = np.load('X.npy')

# image label 데이터 shape(5547,1) : (0 = no cancer, 1 = cancer)
Y = np.load('Y.npy')

# 이미지 데이터들을 무작위로 섞음 - 데이터의 label이 순서대로 나열되어있기 때문에

perm_array = np.arange(len(X))
np.random.shuffle(perm_array)
X = X[perm_array]
Y = Y[perm_array]

print("X image Shape :", X.shape, ", Y label Shape =", Y.shape)
```

```
X image Shape : (5547, 50, 50, 3) , Y label Shape = (5547,)
```

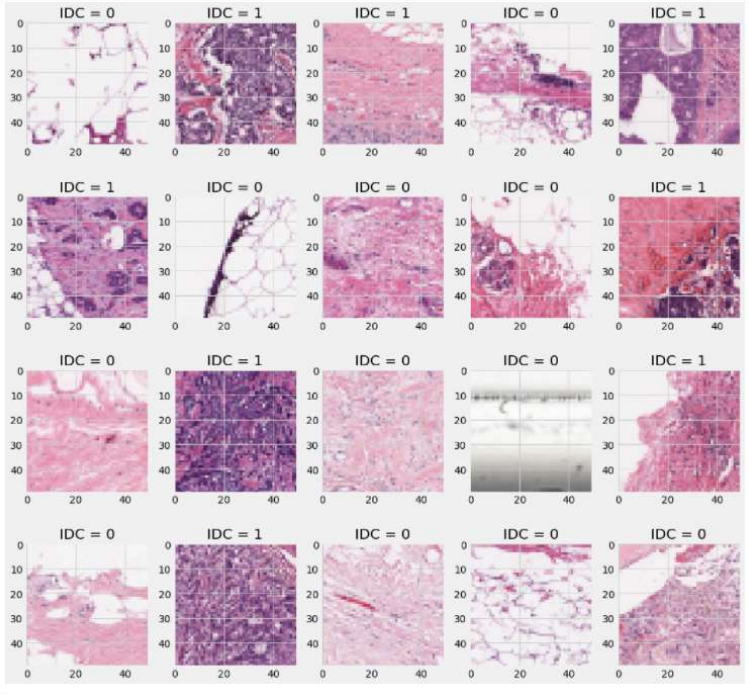
```
# Data에 Cancer label과 Non_Cancer label이 몇개 씩 있는지 확인

print('There are', len(X[Y==0]), 'negetive samples. There are', len(X[Y==1]), 'positive samples')
```

```
There are 2759 negetive samples. There are 2788 positive samples
```

```
# 데이터 미리보기

plt.figure(figsize=(16, 16))
for i in range(20):
    plt.subplot(4, 5, i + 1)
    plt.title('IDC = %d' %Y[i])
    plt.imshow(X[i])
```



다음과 같은 IDC patch image 준비, 각 patch image는 IDC positive의 경우 label이 1이고, negative의 경우에는 label이 0이다.

2. Data 준비 for Classical ML algorithms

```
#데이터 정규화
X = X/255.0

#Data를 ML 알고리즘에 사용하기 위해서 1D 형태로 바꾸어준다.

X_reshaped = X.reshape(X.shape[0], X.shape[1]*X.shape[2]*X.shape[3])


#print(X_reshaped.shape)

#overfitting을 방지하기 위해 데이터를 Training Set, Test Set으로 나누어준다. (7.5 : 2.5)

X_train, X_test, Y_train, Y_test = train_test_split(X_reshaped, Y, test_size=0.25)

print('There are', len(X_train[Y_train==0]), 'non Cancer samples and ' , len(X_train[Y_train==1]), 'Cancer samples in training Set ')
print('There are', len(X_test[Y_test==0]),'non Cancer samples and ', len(X_test[Y_test==1]), 'Cancer samples in test Set ')
```

```
There are 2063 non Cancer samples and  2097 Cancer samples in training Set
There are 696 non Cancer samples and  691 Cancer samples in test Set
```

Classical ML algorithms을 이용해 classifier를 학습시키기 위해 data를 다음과 같이 1D data로 만들어주고, overfitting을 방지하기 위해서 data를 train/test 0.75:0.25 비율로 나누어주었다.

3. Make Classifier Using Classical ML algorithms & Check Test Accuracy

```
# model Fit + Evaluation 함수 만들기

def fit_evaluate(model):

  # 모델훈련
  model.fit(X_train, Y_train)

  # 훈련된 모델을 가지고 X_test set을 예측 하고, 예측한 것과, 실제를 값을 비교하여 Accuracy 측정
  model_pred = model.predict(X_test)
  model_accuracy = accuracy_score(Y_test, model_pred)


  # 모델 정확도 return
  return model_accuracy
```

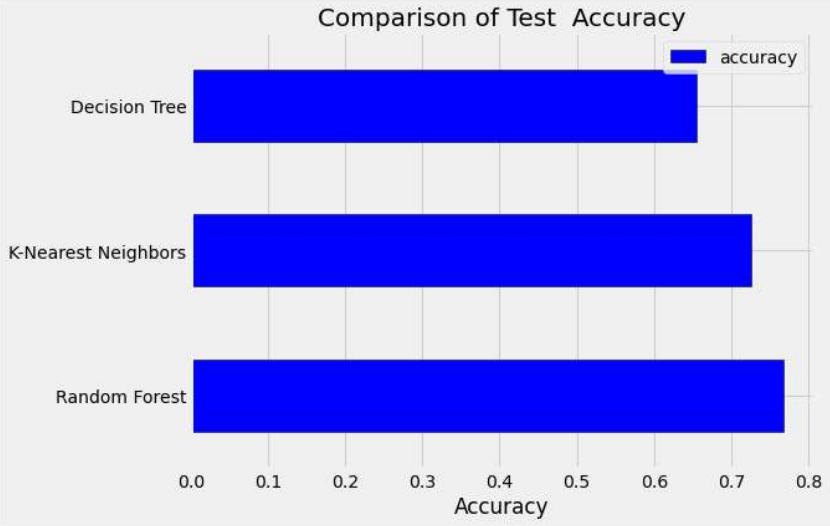
```
#Decision Tree Classifier
DTR = DecisionTreeClassifier(random_state=42)
DTR_accuracy = fit_evaluate(DTR)
print('DTR accuracy:', DTR_accuracy)

#Random Forest Classifier
RFC = RandomForestClassifier(random_state=42)
RFC_accuracy = fit_evaluate(RFC)
print('RFC accuracy:', RFC_accuracy)

#KNN
KNN = KNeighborsClassifier(10)
KNN_accuracy = fit_evaluate(KNN)
print('KNN accuracy:', KNN_accuracy)
```

```
DTR accuracy: 0.6553713049747657
RFC accuracy: 0.7678442682047585
KNN accuracy: 0.7253064167267483
```

모델 훈련을 위해서 다음과 같은 code를 이용해서 1D 형식으로 변환된 IDC patch image data를 이용해 Random forest, Decision Tree, KNN 알고리즘을 이용해서 classifier learning후 test Accuracy 측정.



Test Accuracy를 비교해보면, 다음과 같이 RF 알고리즘을 이용한 model의 경우에 가장 높은 0.76의 accuracy를 가지는 것을 확인 할 수 있었다.

4. CNN Model을 이용하여 Classifier 생성

```
# 1D 로 바꾸기 전의 기존 이미지 데이터를 training, test set으로 나누어 주었다.
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size = 0.25)
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape

# Y set은 현재 value data set인데 이를 categorical Value로 바꾸어 주었다.
ytrain = to_categorical(ytrain,2)
ytest = to_categorical(ytest,2)
```

Classical ML algorithms을 이용할 때와 다르게 CNN에 input으로 넣을 data의 경우에는 1D data로 변환을 하지 않고 image data를 그대로 이용했다.

또한 classical ML algorithms을 이용할 때와 동일하게 Training/Test set을 0.75:0.25 비율로 나누어주었다.

```
#Validation의 Accuracy가 80%에 도달하면 학습 종료
import tensorflow as tf
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self,epoch,logs={}):
    if(logs.get("val_accuracy") >= 0.800):
      print("Reached 80% accuracy!")
      self.model.stop_training = True
callbacks=myCallback()
```

CNN을 이용한 classifier는 classical ML algorithms 보다 높은 정확도가 기대되기 때문에 76%보다 4% 높은 80%의 정확도에 도달하면 model의 learning을 종료 시키도록 설정하였다.

```
tf.keras.backend.clear_session()
K.clear_session() #clear session to make sure there is no overtraining
tf.random.set_seed(51)
np.random.seed(51)

input_shape = (xtrain.shape[1], xtrain.shape[2], 3)
batch_size = 64
num_classes = 2
epochs = 30
channelDim=-1
#시작할 때 learning rate 0.02
INIT_LR=1e-2

model = tf.keras.Sequential()

#첫번째 layer
model.add(SeparableConv2D(32, (3,3), padding="same",input_shape=input_shape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=channelDim))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(SpatialDropout2D(0.2))

#두번째 layer
model.add(SeparableConv2D(64, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=channelDim))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(SpatialDropout2D(0.3))

#세번째 layer
model.add(SeparableConv2D(128, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=channelDim))

#네번째 layer
model.add(SeparableConv2D(256, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=channelDim))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))

#다섯번째 layer
model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(num_classes))
model.add(Activation("softmax"))


opt=Adagrad(lr=INIT_LR,decay=INIT_LR/epochs)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

기본적으로 batch size는 64로 지정해주었고, epoch는 30으로 initial learning rate는 0.02으로 설정 한 후 learning을 진행하였다.

각 layer의 convolution으로는 Seperable Conv2D를 이용했고, 활성 함수는 relu를 이용했다. 추가적으로 학습 속도와, Local minimum에 빠지는 gradient vanishing 문제를 방지하기 위해 각 layer 별로 Batch Normalization을 해주었다.

Pooling부분에서는 최댓값을 뽑아내는 Maxpooling method중에서 MaxPooling2D를 이용했다.

추가적으로 Overfitting 문제를 방지하기 위해서 모델에서 네트워크의 일부를 생략하고 학습을 진행하는 Dropout을 이용해 주었다.

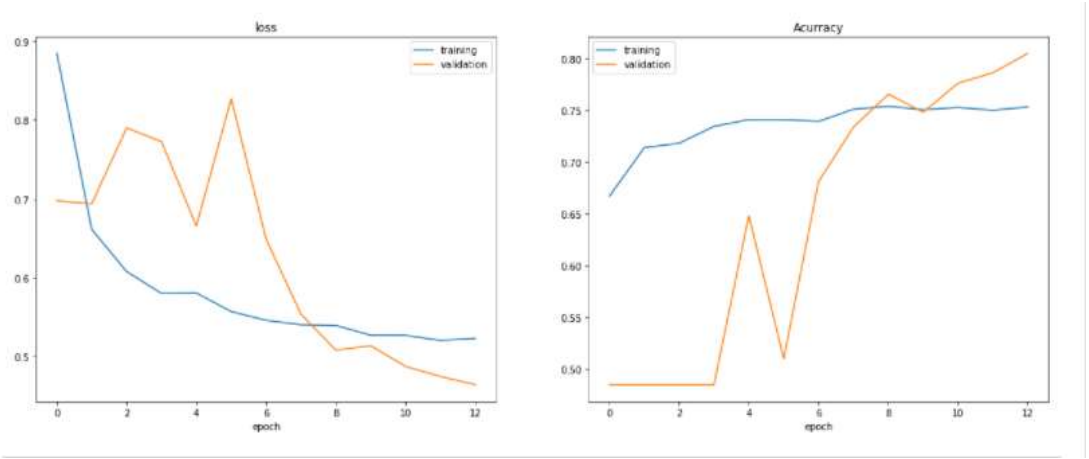
마지막 layer에서는 Activation 함수로 softmax를 이용했다.

```
datagen = ImageDataGenerator(
    rotation_range=30,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=True,
    zoom_range=0.05,
    #shear_range=0.05,
    fill_mode='nearest'
 )
datagen.fit(xtrain)
history_1=model.fit(datagen.flow(xtrain,ytrain, batch_size=batch_size),
            epochs=epochs,
            validation_data = (xtest, ytest),
            verbose = 2,
            steps_per_epoch=len(xtrain)/batch_size,
            shuffle=True, #shuffle the data
            callbacks=[myCallback()]) #Valiadtion set에 대한 Accuracy 80% 이상이 되면 학습 종료
score_1 = model.evaluate(xtest, ytest, verbose=0)
print(score_1)
```

Model을 training하기 전에 부족한 data에 따른 문제를 해결하기 위해서 ImageDataGenerator를 이용하여 data의 수를 늘려주었다.

```
Epoch 1/30
65/65 - 47s - loss: 0.8846 - accuracy: 0.6668 - val_loss: 0.6976 - val_accuracy: 0.4845
Epoch 2/30
65/65 - 38s - loss: 0.6615 - accuracy: 0.7139 - val_loss: 0.6938 - val_accuracy: 0.4845
Epoch 3/30
65/65 - 39s - loss: 0.6080 - accuracy: 0.7183 - val_loss: 0.7900 - val_accuracy: 0.4845
Epoch 4/30
65/65 - 38s - loss: 0.5802 - accuracy: 0.7346 - val_loss: 0.7727 - val_accuracy: 0.4845
Epoch 5/30
65/65 - 39s - loss: 0.5808 - accuracy: 0.7411 - val_loss: 0.6657 - val_accuracy: 0.6482
Epoch 6/30
65/65 - 40s - loss: 0.5572 - accuracy: 0.7411 - val_loss: 0.8271 - val_accuracy: 0.5097
Epoch 7/30
65/65 - 39s - loss: 0.5461 - accuracy: 0.7394 - val_loss: 0.6492 - val_accuracy: 0.6813
Epoch 8/30
65/65 - 39s - loss: 0.5406 - accuracy: 0.7512 - val_loss: 0.5537 - val_accuracy: 0.7340
Epoch 9/30
65/65 - 38s - loss: 0.5396 - accuracy: 0.7541 - val_loss: 0.5083 - val_accuracy: 0.7657
Epoch 10/30
65/65 - 39s - loss: 0.5272 - accuracy: 0.7507 - val_loss: 0.5139 - val_accuracy: 0.7484
Epoch 11/30
65/65 - 39s - loss: 0.5272 - accuracy: 0.7529 - val_loss: 0.4877 - val_accuracy: 0.7765
Epoch 12/30
65/65 - 39s - loss: 0.5206 - accuracy: 0.7502 - val_loss: 0.4749 - val_accuracy: 0.7866
Epoch 13/30
65/65 - 39s - loss: 0.5233 - accuracy: 0.7534 - val_loss: 0.4646 - val_accuracy: 0.8053
Reached 80% accuracy!
[0.464586466550827, 0.805335283279419]
```

13번째 epoch 만에 Test Set의 Accuracy가 0.8053에 도달한 것을 확인 할 수 있었다.



학습 진행에 따른 Training Set, Test Set의 Loss, Accuracy를 plot으로 나타내면 위와 같이 loss는 점점 줄어들고, Accuracy는 증가하는 모습을 볼 수 있었고, 성공적으로 학습이 이루어진 것으로 판단 할 수 있었다.

최종 프로젝트 발표 이후의 개선 할 점

Patch image를 이용한 모델 생성시 classical ML algorithms에 비해서 CNN의 경우에 더 높은 성능을 보이는 model 생성이 가능했다. 그러나 학습이 진행됨에 따라 training Set보다 test set의 성능이 더 높게 나오는 것 또한 확인 할 수 있었다. 결국 좋은 classifier는 training, test set 모두에서 좋은 성능을 보여야하기 때문에 이러한 문제를 해결하기 위해서 K-fold Cross validation을 이용해 model을 평가하면서 학습을 진행해보려고 한다.

비록 80% 정도의 정확도를 얻긴 했지만 이를, 실제 진단에 이용하기에는 낮은 수치라고 생각된다. 모델의 성능을 늘릴 수 있는 다른 방식들을 더 고안해보는 노력을 해봐야한다.

프로젝트에서 느낀 점

이론적으로 배운 것을 생각하며 모델을 만든다고 생각했을 때에는 손쉬울 줄만 알았는데, 우선 프로젝트의 초반 단계인 데이터를 구하는 것부터 어려움을 겪었다. 또한 생각을 실제 컴퓨터 프로그래밍 언어로 옮기는 것 또한 쉽지않았던 것 같다. 이후에 실제적으로 이용 가능한 모델을 만들기 위해서는 더 많은 노력과 공부가 필요할 것 같다는 생각을 가지게 되었다.