기계학습 프로젝트 보고서

emotion detection (FER)

20101220 김준표
20101241 소보길
22102482 김서영
21100801 현준혁

1. Problem Definition

1.1 Context

얼굴 표정 인식 (Facial Expression Recognition, FER)은 얼굴 표정을 분석하여 다양한 감정(행복, 슬픔, 분노, 놀람, 두려움, 혐오 등)을 분류하는 기술이다. FER은 심리학, 사람과 컴퓨터의 상호작용, 엔터테인먼트 등 다양한 분야에 사용될 수 있다. 예를 들어, 사람의 감정을 인식하여 그에 대하여 적절히 반응해 맞춤형 추천과 조정을 제공할 수 있고, 사람의 심리 진단에 유용하게 사용될 수 있다.

1.2 Problem Statement

label이 있는 사람의 표정 사진 데이터를 총 7가지 감정, 행복, 혐오, 슬픔, 분노, 놀람, 두려움, 무표정으로 분류한다.

1.3 Approach

데이터 불균형 문제를 해결하기 위해 데이터 증강(data augmentation)과 오버샘플링(oversampling)을 활용한다. 다양한 모델을 학습한 뒤, 가장 성능이 우수한 3개의 모델을 선정하여 앙상블 기법을 적용할 예정이다. 이를 통해 감정 분류의 정확도를 최대화하고, 모델의 일반화 성능을 향상시킨다.

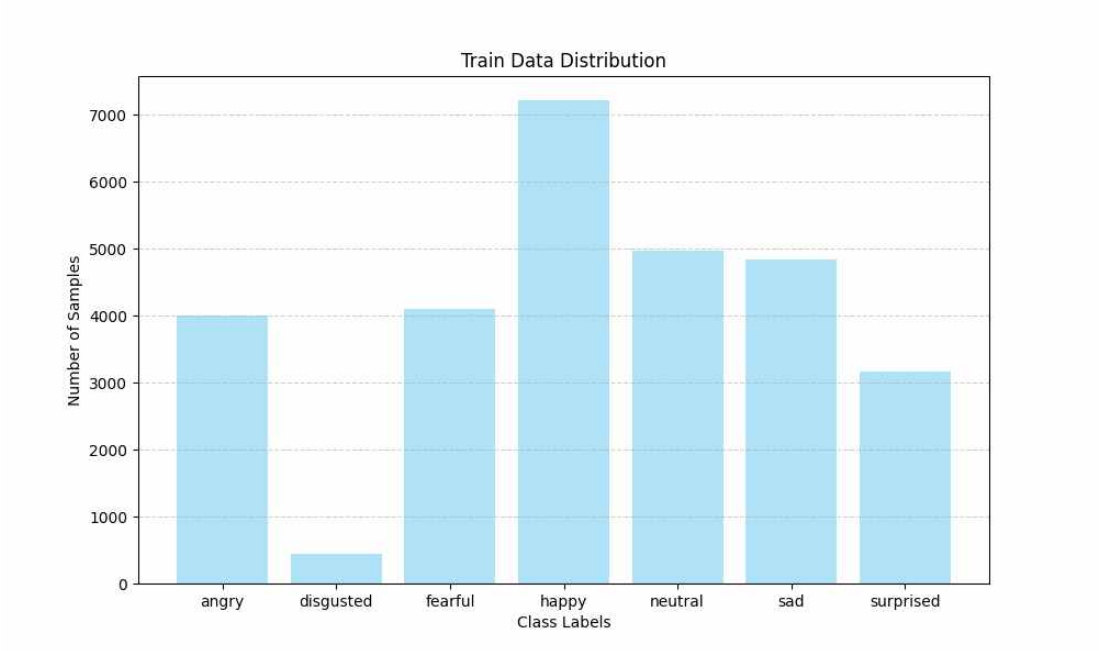
2. Data Explanation

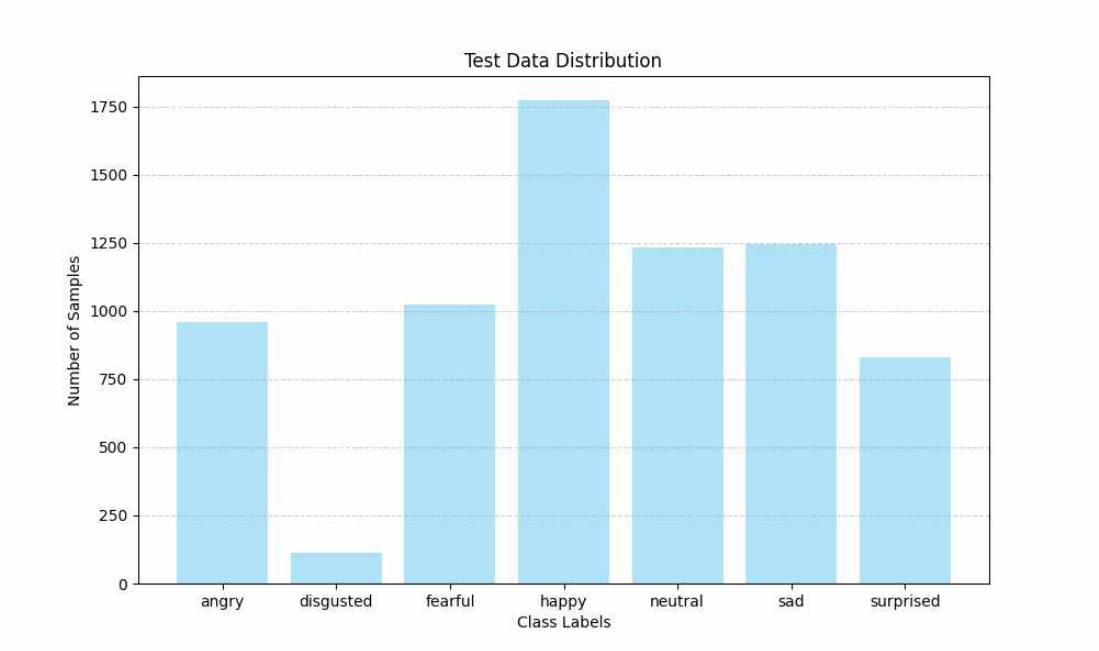
2.1 Data Information

48 x 48 grayscale 이미지로 7가지 label (angry, disgusted, fearful, happy, neutral, sad, surpirsed)이 달려있다. training dataset은 28709개의 이미지, test dataset은 7178개의 이미지로, 약 1:4의 비율로 나누어져 있다.

2.2 Data Example

2.3 Data Distribution





2.4 Data Problem

- 데이터의 질 : 실제 사람의 얼굴이 아닌 경우, 손으로 얼굴을 가린 경우, 워터마크가 남아 있는 경우가 있어 데이터의 질이 좋지 않다.

- 클래스 불균형 : “happy” data의 경우가 다른 데이터 대비 많은 것을 볼 수 있고, “disgusted” data가 다른 클래스에 비해 한참 부족하다.

3. Data preprocessing

3.1 Data Normalization

grayscale 이미지이므로 각 pixel 값은 [0, 255] 범위를 가진다. 이를 [0, 1] 범위로 normalization

```
image = image / 255.0
```

3.2 Resizing and RGB conversion

모델의 입력 크기에 따라 이미지 데이터의 크기와 채널 수 변경

```
image_size=(img_size, img_size),
color_mode="grayscale" if gray else "rgb",
```

3.2 Train-Validation Split

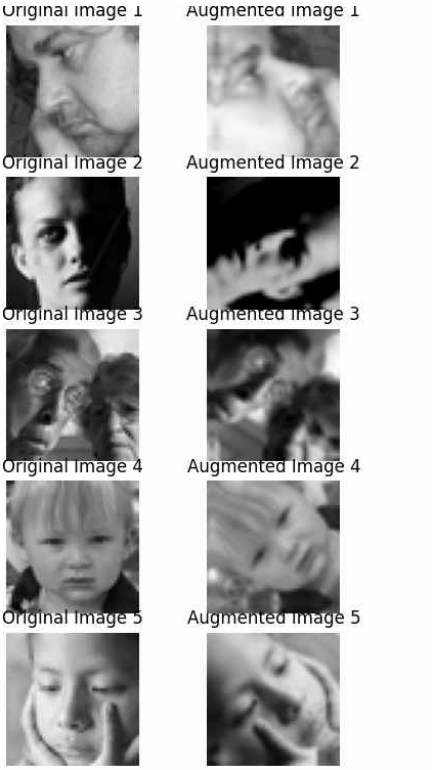
Stratified Sampling을 사용하여 전체 training data의 8:2의 비율로 training set과 validation set으로 나누었다.

```
X_train, X_val, y_train, y_val = train_test_split(
    all_images, all_labels, test_size=VALIDATION_SIZE, stratify=all_labels, random_state=RANDOM_STATE
)
```

3.3 Data Augmentation

뒤집기, 회전, 이동, 밝기, 대비에 변화를 주어 데이터를 추가하였다.

```
def augment_image(images):
    augmentation_layers = tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomTranslation(0.1, 0.1),
        tf.keras.layers.RandomBrightness(0.2),
        tf.keras.layers.RandomContrast(0.2)
    ])

    # 증강 적용
    augmented = augmentation_layers(images)
    return augmented
```



3.4 Oversampling

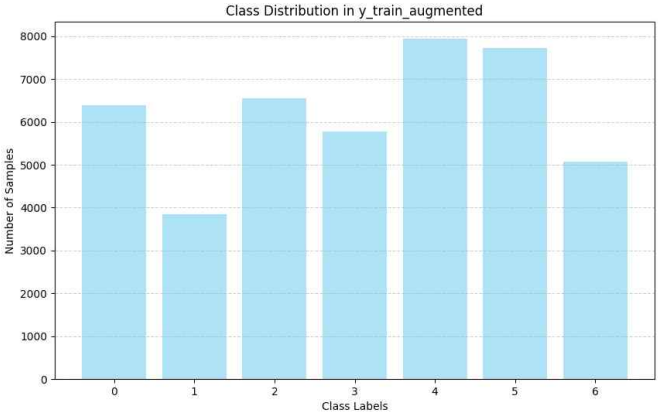
클래스 불균형을 해결하기 위해 “disgusted” class의 데이터를 강화하여 oversampling

4. Handling Class Imbalance

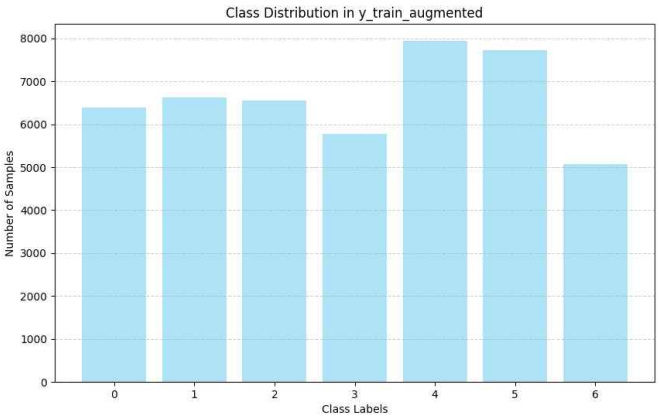
클래스 불균형을 해결하기 위해 oversampling과 손실함수 조합을 테스트하여 손실함수와 sampling 방식을 결정하였다.

4.1 Oversampling

- Mild Oversampling : disgusted class를 10회 oversampling



- Oversampling : 다른 class들과 instance 수가 비슷하게 18회 sampling



4.2 Loss Function

- Categorical Crossentropy
- Categorical Crossentropy + Weighted train
- Focal Loss

4.2.1 Focal Loss

focal loss는 클래스 불균형 문제를 해결하기 위해 설계된 loss function으로, 쉽게 예측된 샘플의 가중치를 낮추고, 어려운 샘플에 더 높은 가중치를 부여하여, 클래스 불균형 해결을 도와준다. focal loss 자체가 드문 클래스에 가중치를 주고 있으므로 weighted train을 고려하지 않았다.

4.3 Train

Oversampling 방식과 Loss function을 조합하여 총 6가지 조합을 사용하여 모델을 훈련시켰다. 모델은 간단한 CNN 모델을 사용하여 15 epoch 동안 학습을 진행했다.

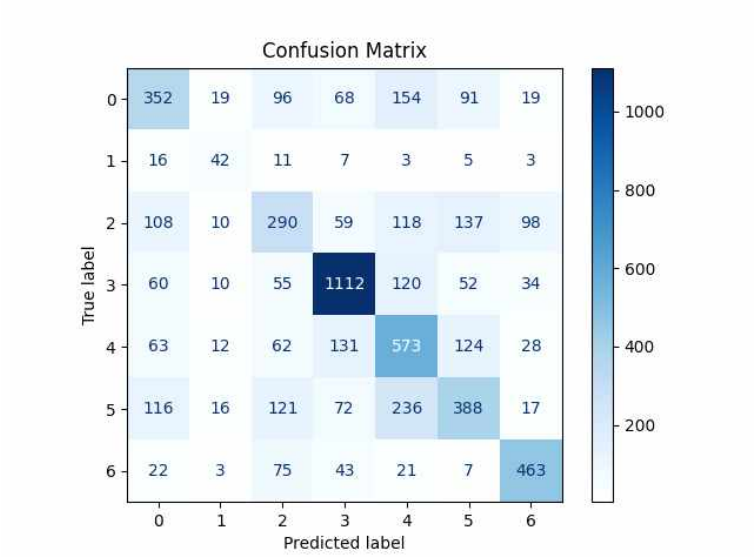
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(INPUT_SIZE, INPUT_SIZE, 1), kernel_initializer='he_normal'  ),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

4.4 Result

Oversampling + Categorical Crossentropy가 가장 좋은 성능을 보였다.

	Categorical Crossentropy	Categorical Crossentropy + Weighted Train	Focal Loss
Mild Oversampling	0.5502	0.5486	0.5406
Oversampling	0.5608	0.5578	0.5557

metric : validation accuracy



5. Base Models Training

학습해 볼 모델은 총 5가지로, CNN, DenseNet, VGG, ResNet, AlexNet을 학습한다. 모델은 대표적인 CNN 모델들과, kaggle에서 좋은 결과를 보였던 모델을 고려하여 선정하였다.

5.1 Hyperparameter Tuning

- 학습은 hyperparameter tuning과 함께 진행되며, 다음 hyperparameter를 조정한다.
- Grid Search로 진행되며 epoch은 10, early stopping을 적용하였다.
- batch size도 tuning을 시도하였으나, 컴퓨팅 리소스의 한계로 32로 고정하여 진행

```
# 하이퍼파라미터 그리드
learning_rates = [1e-4, 1e-3, 1e-2]
optimizers = ['adam', "SGD_momentum", "RMSprop"]
activation_functions = ['relu', 'elu']
```

5.2 Metric

- accurcy만으로 평가하기 어려운 부분이 있어, Top-2 accuracy도 함께 고려하여 평가

5.3 Training

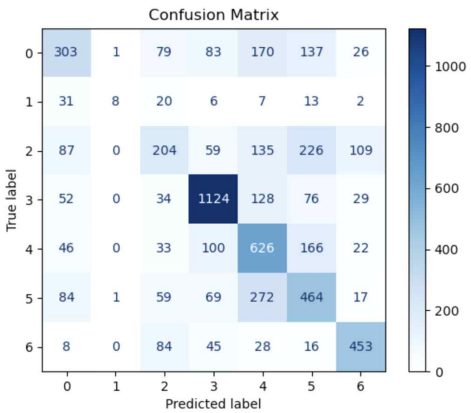
5.3.1 CNN

```
#CNN [48, 48, 1]
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(INPUT_SIZE, INPUT_SIZE, 1), kernel_initializer='he_normal'  ),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

- Conv2D, Pooling 을 3겹으로 쌓은 간단한 CNN

- Accuracy Learning Curve

- Top-2 Accuracy Learning Curve



- confusion matrix

Result

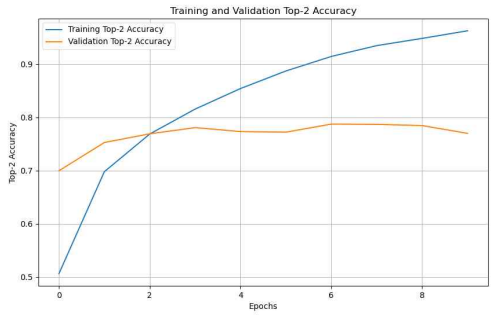
- Optimal: {'learning_rate': 0.001, 'optimizer': 'adam', 'activation': 'relu'}
- Validation Accuracy: 0.5542
- Validation Top-2 Accuracy: 0.7485

5.3.2 DenseNet

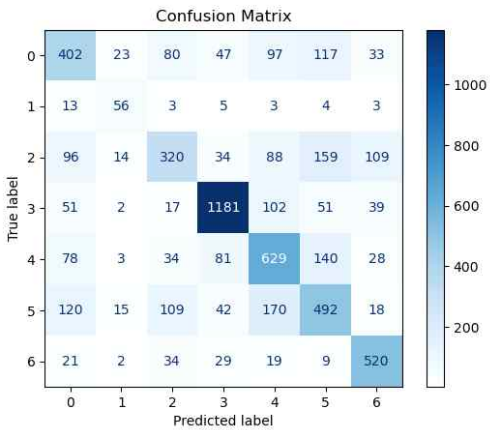
```
model.summary()

#DenseNet [48, 48, 3]
model = tf.keras.models.Sequential([
    tf.keras.applications.DenseNet169(
                                      input_shape=(INPUT_SIZE,INPUT_SIZE, 3),
                                      include_top=False,
                                      weights="imagenet"
                                     ),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.05)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1024, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.05)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation="relu", kernel_regularizer = tf.keras.regularizers.l2(0.05)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(NUM_CLASSES, activation="softmax", name="classification"),
    ])
```

- imagenet을 학습한 DenseNet을 reuse한 모델
- overfitting 방지 위해 l2 regularizer 사용

- Accuracy Learning Curve



- Top-2 Accuracy Learning Curve



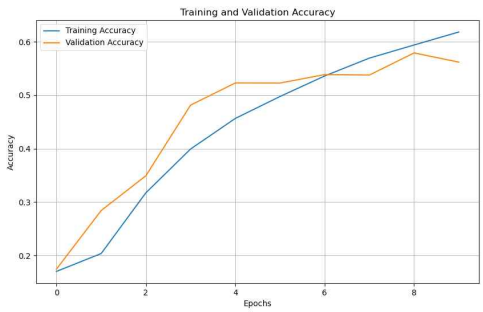
- confusion matrix

Result

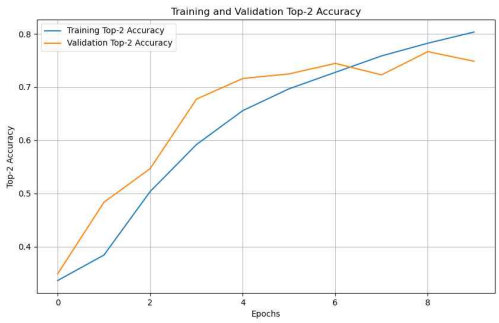
- Optimal: {'learning_rate': 0.001, 'optimizer': 'adam', 'activation': 'relu'}
- Validation Accuracy: 0.6272
- Validation Top-2 Accuracy: 0.7886

5.3.3 VGG

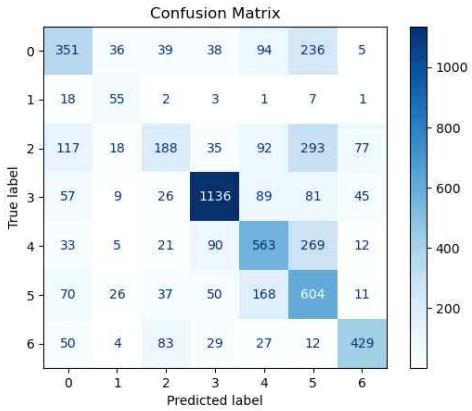
```
#VGG
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(INPUT_SIZE, INPUT_SIZE, 1), padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # Block 2
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # Block 3
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # Block 4
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # Block 5
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same', kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    # Fully Connected Layers
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```



- Accuracy Learning Curve



- Top-2 Accuracy Learning Curve



Result

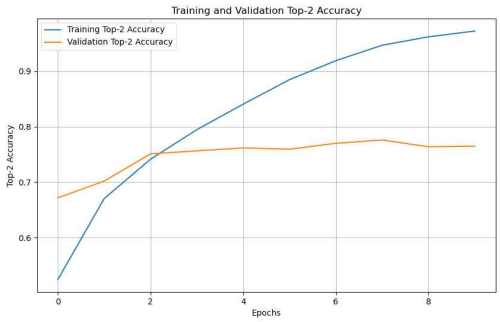
- Optimal: {'learning_rate': 0.01, 'optimizer': 'sgd_momentum', 'activation': 'elu'}
- Validation Accuracy: 0.5792
- Validation Top-2 Accuracy : 0.7521

5.3.4 ResNet

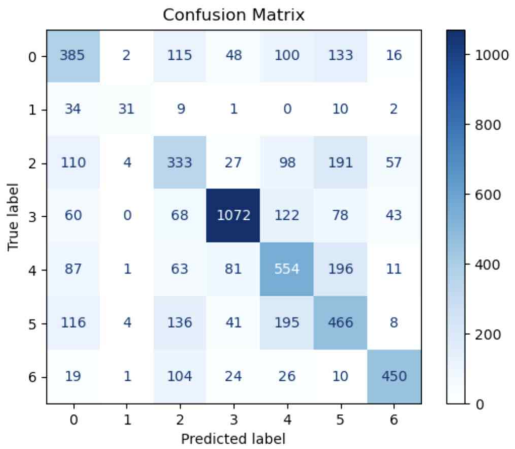
```
# ResNet [224, 224, 3]
model = tf.keras.models.Sequential([
    tf.keras.applications.ResNet50V2(include_top=False, weights='imagenet', input_shape=[INPUT_SIZE, INPUT_SIZE, 3]),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(512, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(7, activation='softmax')
])
```

- imagenet을 학습한 DenseNet을 reuse한 모델

- Accuracy Learning Curve



- Top-2 Accuracy Learning Curve



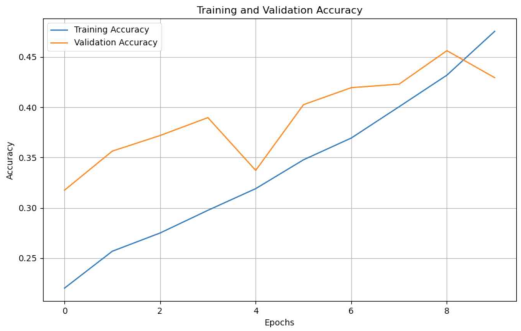
- Confusion Matrix

Result

- Optimal: {'learning_rate': 0.0001, 'optimizer': 'adam', 'activation': 'elu'}
- Validation Accuracy: 0.5947
- Validation Top-2 Accuracy : 0.7886

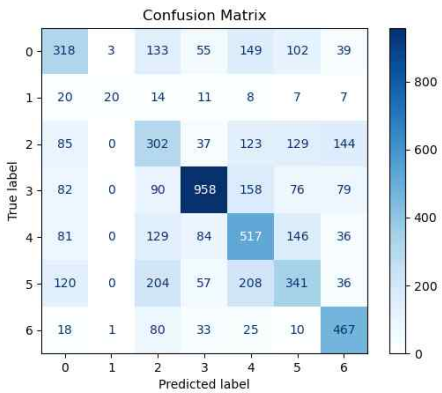
5.3.5 AlexNet

```python
# AlexNet
alexnet_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu', input_shape=(227, 227, 3)),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),


    tf.keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),

    tf.keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),

    tf.keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),

    tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1000, activation='relu'),  # Adjust this to the number of classes in your dataset
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(7, activation='softmax')  # Assuming 7 classes for FER
])
```



- Accuracy Learning Curve

- Top-2 Accuracy Learning Curve



- Confusion Matrix

Result

- Optimal: {'learning_rate': 0.0001, 'optimizer': 'adam', 'activation': 'elu'}
- Validation Accuracy: 0.4561
- Validation Top-2 Accuracy : 0.6491

5.1 Base Model Training Result

- DenseNet > ResNet > VGG > CNN > AlexNet 순으로 결과가 좋았다.
- ResNet, DenseNet은 overfit 되었다.
- AlexNet은 학습이 제대로 이루어지지 못했다.
- 가장 결과가 좋은 세 모델을 ensemble에 사용하기로 결정

6. Ensemble

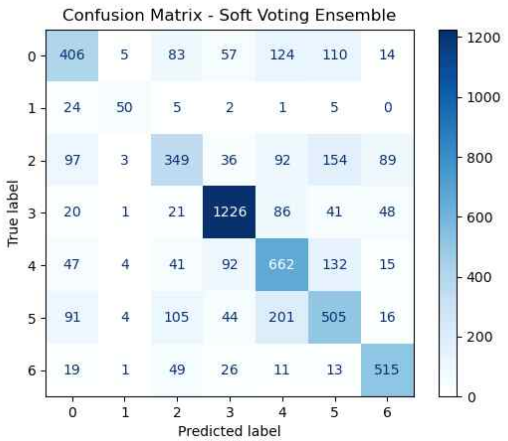
6.1 Approach

- 전에서 찾은 hyperparameter를 사용하여 DenseNet, ResNet, VGG를 재학습 시켜 저장
- 다양성을 위해 학습 시 사용할 training set의 seed를 다르게 설정
- Soft Voting 방식 사용

```
for images, labels in dataset:
    # 각 모델의 예측 확률 수집
    predictions = [model.predict(images, verbose=0) for model in models]

    # 평균 확률 계산 (Soft Voting)
    avg_predictions = np.mean(predictions, axis=0)
    y_pred_probs.extend(avg_predictions)

    # 실제 레이블 수집
    y_true.extend(labels.numpy())

# 최종 예측값: 확률에서 argmax로 클래스 결정
y_pred = np.argmax(y_pred_probs, axis=1)
return y_true, y_pred, np.array(y_pred_probs)
```

6.2 Validation Result

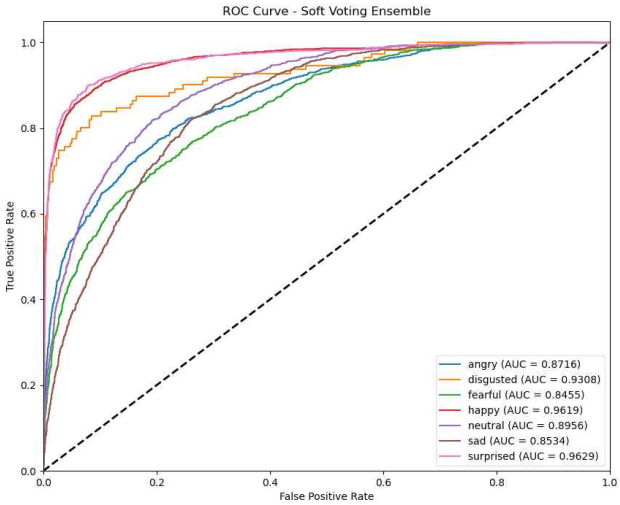
- Validation Accuracy : 64.91



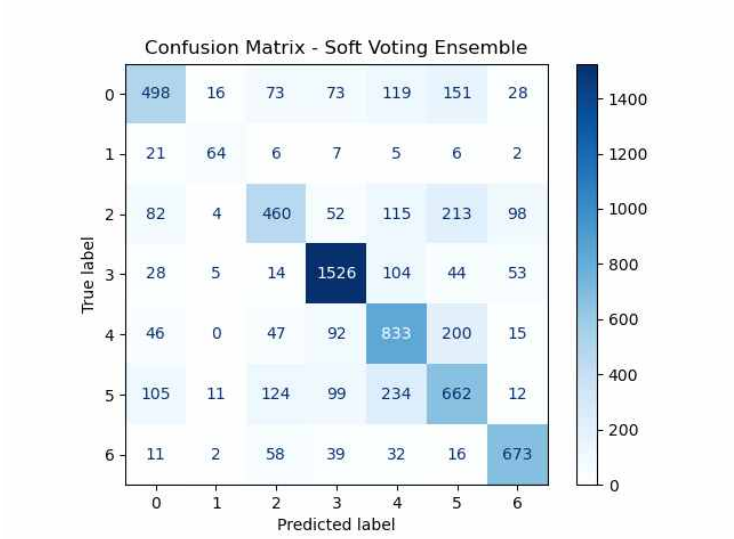
- Confusion Matrix

7. Test Result

- Accuracy : 0.6570
- Top-2 Accuracy : 0.8183
- Precision : 0.6541
- Recall : 0.6570
- F1 Score : 0.6525
- Mean AUC :0.9031



- ROC Curve



- Confusion Matrix

8. Conclusion

8.1 Analysis

- 하나의 모델로 학습하였을 때 보다, 앙상블 하였을 때의 결과가 눈에 띄게 좋아짐
- 전반적인 AUC score가 높음, 특히 happy와 surprised가 굉장히 잘 분류되고 있음
- Fearful과 Sad는 상대적으로 분류 성능이 좋지 못함
- top-2 accuracy가 80% 이상으로 만족스러운 결과가 나옴

8.2 Areas for Improvements

- DenseNet과 ResNet은 overfitting 되는 경향이 있었음, 이를 해결하기 위해 l2 regularizaiton을 적용하기도 했지만, 더 많은 regularization 전략을 적용해 볼 필요가 있었음
- pre-trained model을 reuse할 때, model을 freeze 하지 않고 바로 학습을 진행했음. 학습이 다 완료된 후에, 관련 내용을 수업에서 더 자세히 배워 아쉬움이 남음
- imagenet을 학습한 모델만 reuse 했지만, 다른 데이터(FER)를 학습한 모델을 적용 해보지 못했음.
- 더 다양한 모델을 학습하지 못하였고, 앙상블 시 많은 모델을 적용하지 못했음.

9. Code Explanation

9.1 Data Processing

```
def load_train_data(
                    img_size=DEFAULT_SIZE,
                    gray=False,
                    normalization=True,
                    batch_size = 64
                                        ):

    # 전체 데이터셋을 로드
    train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
        train_path,
        labels="inferred",
        label_mode="int",
        image_size=(img_size, img_size),
        color_mode="grayscale" if gray else "rgb",
        batch_size=batch_size,
        shuffle=False,
        seed=RANDOM_STATE,
        class_names=classes
    )
```

- training data를 불러옴
- 각 모델의 입력에 따라, img_size와 channel 변경

```
# 한 번에 numpy 배열로 변환
all_images = np.vstack([images.numpy() for images, _ in train_dataset])
all_labels = np.hstack([labels.numpy() for _, labels in train_dataset])


# Stratified split: 데이터를 훈련 세트와 검증 세트로 나누기
X_train, X_val, y_train, y_val = train_test_split(
    all_images, all_labels, test_size=VALIDATION_SIZE, stratify=all_labels, random_state=RANDOM_STATE
)
```

- training dataset을 stratified sampling으로 8:2 비율로 나눔

```
def augment_image(images):
    augmentation_layers = tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomTranslation(0.1, 0.1),
        tf.keras.layers.RandomBrightness(0.2),
        tf.keras.layers.RandomContrast(0.2)
    ])

    # 증강 적용
    augmented = augmentation_layers(images)
    return augmented
```

- data augment
- 회전, 이동, 밝기, 대비, 뒤집기 사용

```
if normalization:
    image = image / 255.0
```

- 데이터에 normalization 적용

```
# disgusted 클래스의 데이터
X_train_disgusted = X_train[disgusted_indices]
y_train_disgusted = y_train[disgusted_indices]

# 나머지 클래스의 데이터 (happy 제외)
X_train_other = X_train[other_indices]
y_train_other = y_train[other_indices]

# disgusted 클래스는 18배로 증강
augmented_disgusted = np.concatenate(
    [augment_image(X_train_disgusted) for _ in range(18)]
)

y_train_disgusted_aug = np.repeat(y_train_disgusted, 18)

# 다른 클래스들은 1번만 증강
augmented_other = augment_image(X_train_other)
```

- disgusted는 augment 적용하여 18번 oversampling
- happy와 disgusted를 제외한 나머지는 한번씩 augment

9.2 Training

```
# 하이퍼파라미터 그리드
learning_rates = [1e-4, 1e-3, 1e-2]
optimizers = ['adam', "SGD_momentum", "RMSprop"]
activation_functions = ['relu', 'elu']
```

```
# Grid Search
for lr in learning_rates:
    for opt_name in optimizers:
        for activation_name in activation_functions:
```

- GridSearchCV가 아닌 for문을 사용하여 gridsearch 진행

```
# 각 반복마다 새로운 모델 생성
model = tf.keras.models.Sequential([

    # TODO 모델 집어 넣기

    tf.keras.layers.Conv2D(32, (3, 3), activation=activation_name, input_shape=(INPUT_SIZE, INPUT_SIZE, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation=activation_name, kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation=activation_name, kernel_initializer='he_normal'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation=activation_name, kernel_initializer='he_normal'),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])

# optimizer 설정
if opt_name == 'adam':
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
elif opt_name == 'RMSprop':
    optimizer = tf.keras.optimizers.RMSprop(learning_rate=lr)
else:
    optimizer = tf.keras.optimizers.SGD(learning_rate=lr, momentum=0.9)
```

- 모델에 hyperparameter 적용

```
# 모델 컴파일, 손실함수 설정
model.compile(
    optimizer=optimizer,
    loss = "sparse_categorical_crossentropy",
    metrics=['accuracy', tf.keras.metrics.SparseTopKCategoricalAccuracy(k=2)]
)

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    restore_best_weights=True
)

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    './models/DenseNet.keras',  # 모델 저장 경로, model 이름으로
    monitor='val_accuracy',
    save_best_only=True,  # 가장 좋은 모델만 저장
    mode='max',  # val_accuracy가 최대일 때 저장
    verbose=1
)


history = model.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=val_ds,
    callbacks=[early_stopping],
    verbose=1,
)
```

- “sparse_categorical_crossentropy”를 loss function으로 사용
- early_stopping으로 overfitting 방지
- model_checkpoint callback으로 필요한 경우 모델 저장

9.3 Ensemble

```
dense_model = tf.keras.models.load_model('./models/DenseNet.keras')
vgg_model = tf.keras.models.load_model('./models/VGG.keras')
res_model = tf.keras.models.load_model('./models/ResNet.keras')

# Soft Voting 앙상블 함수
def soft_voting_ensemble(models, dataset):
    """
    Soft Voting 앙상블 함수
    Args:
        models (list): 앙상블에 사용할 모델 리스트
        dataset (tf.data.Dataset): 예측에 사용할 데이터셋
    Returns:
        y_true (list): 실제 레이블
        y_pred (list): 앙상블 예측 결과
    """
    y_true = []
    y_pred_probs = []

    for images, labels in dataset:
        # 각 모델의 예측 확률 수집
        predictions = [model.predict(images, verbose=0) for model in models]

        # 평균 확률 계산 (Soft Voting)
        avg_predictions = np.mean(predictions, axis=0)
        y_pred_probs.extend(avg_predictions)

        # 실제 레이블 수집
        y_true.extend(labels.numpy())

    # 최종 예측값: 확률에서 argmax로 클래스 결정
    y_pred = np.argmax(y_pred_probs, axis=1)
    return y_true, y_pred, np.array(y_pred_probs)

# Soft Voting 실행
models = [dense_model, vgg_model, res_model]
y_true, y_pred, y_pred_probs  = soft_voting_ensemble(models, test_ds)
```

- 모델을 불러와 soft ensemble로 예측

9.4 Training Result Visualization

```
# 결과 시각화
plt.figure(figsize=(10, 6))
plt.plot(best_history['accuracy'], label='Training Accuracy')
plt.plot(best_history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Top-2 Accuracy 그래프
plt.figure(figsize=(10, 6))
plt.plot(best_history['sparse_top_k_categorical_accuracy'], label='Training Top-2 Accuracy')
plt.plot(best_history['val_sparse_top_k_categorical_accuracy'], label='Validation Top-2 Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Top-2 Accuracy')
plt.title('Training and Validation Top-2 Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

- accuracy와 top-2 accuracy의 learning curve plot

```
# Confusion Matrix 계산을 위한 수정
y_true = []
y_pred = []


# 전체 validation 데이터셋에서 예측값과 실제값 수집
for images, labels in val_ds:
    predictions = best_model.predict(images)
    y_pred.extend(np.argmax(predictions, axis=1))
    y_true.extend(labels.numpy())

cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(NUM_CLASSES))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

- confusion matrix plot

```
# 정확도, Precision, Recall, F1 Score 계산
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')
```

- precision, recall, f1 score 측정

```
y_true_one_hot = tf.keras.utils.to_categorical(y_true, num_classes=NUM_CLASSES)
auc_scores = []
plt.figure(figsize=(10, 8))

for i, class_name in enumerate(classes):
    # 각 클래스에 대한 ROC Curve
    fpr, tpr, _ = roc_curve(y_true_one_hot[:, i], y_pred_probs[:, i])
    roc_auc = auc(fpr, tpr)
    auc_scores.append(roc_auc)

    plt.plot(fpr, tpr, label=f'{class_name} (AUC = {roc_auc:.4f})')
```

- AUC score 측정, ROC curve plot