Computer Vision – Spring 2023
Final Project

Team 5

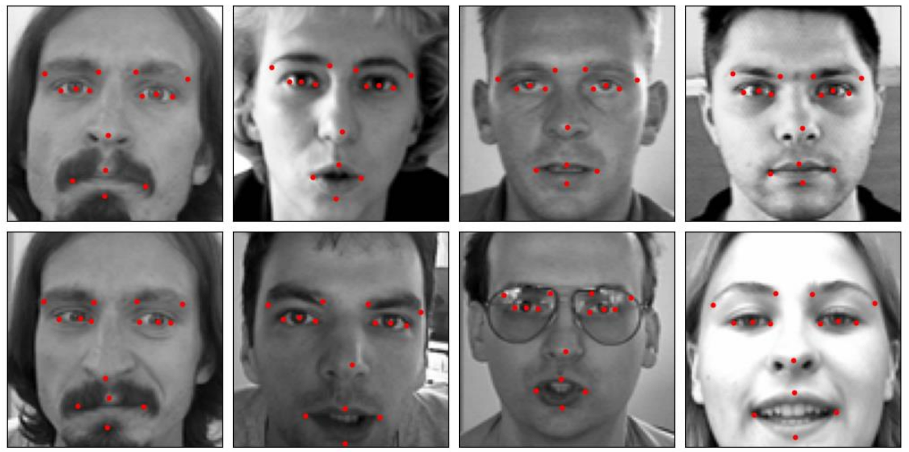
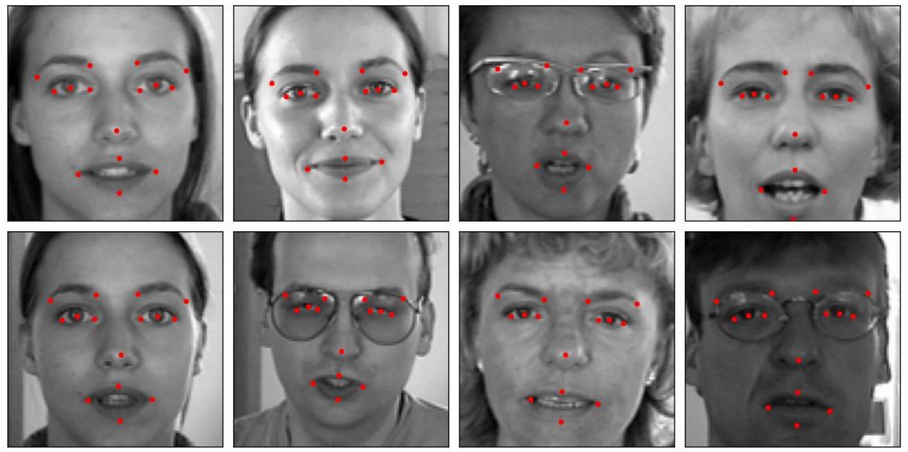
컴퓨터전자시스템공학부 김희찬
컴퓨터전자시스템공학부 이민우
컴퓨터전자시스템공학부 이동인

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Contents

(1) Dataset visualization

- Goal : Facial landmark detection
- Random images

(2) Preprocessing

- Split

The provided code defines a function that splits data into training, validation, and testing sets. The function takes two parameters: data, representing the input data array, and size, indicating the size of the data. The default split ratios are typically 60% for training, 20% for validation, and 20% for testing.

```
def split_data(data, size):
    np.random.seed(2023)

    idx = np.random.permutation(size)

    train_size = int(size * 0.6)
    val_size = int(size * 0.2)
    test_size = int(size * 0.2)

    train_ds = data[idx[: train_size]]
    val_ds = data[idx[train_size: train_size + val_size]]
    test_ds = data[idx[size - test_size:]]

    return train_ds, val_ds, test_ds
```

```
train_img, val_img, test_img = split_data(images, images.shape[0])
```

```
train_lmrk, val_lmrk, test_lmrk = split_data(landmarks, landmarks.shape[0])
```

(2) Preprocessing

- Reshape

The given code performs the task of splitting landmark data into training, validation, and testing sets and reshaping the landmark data into a more manageable format.

```
train_lmrk = train_lmrk.reshape((1284, -1))
val_lmrk = val_lmrk.reshape((428, -1))
test_lmrk = test_lmrk.reshape((428, -1))
```

The landmark data is divided into training, validation, and testing sets. The training set consists of 1284 data points, where each data point contains 15 landmarks represented by 2D coordinates. The validation and testing sets each contain 428 data points, following the same structure of 15 landmarks with 2D coordinates, resulting in a shape of (428, 15, 2) for both sets. Furthermore, the code reshapes the landmark data into a 2D array format, enabling more efficient processing. After the reshaping, the training set has a shape of (1284, 30), while the validation and testing sets have shapes of (428, 30) each.

```
print(train_lmrk.shape)
print(val_lmrk.shape)
print(test_lmrk.shape)
```

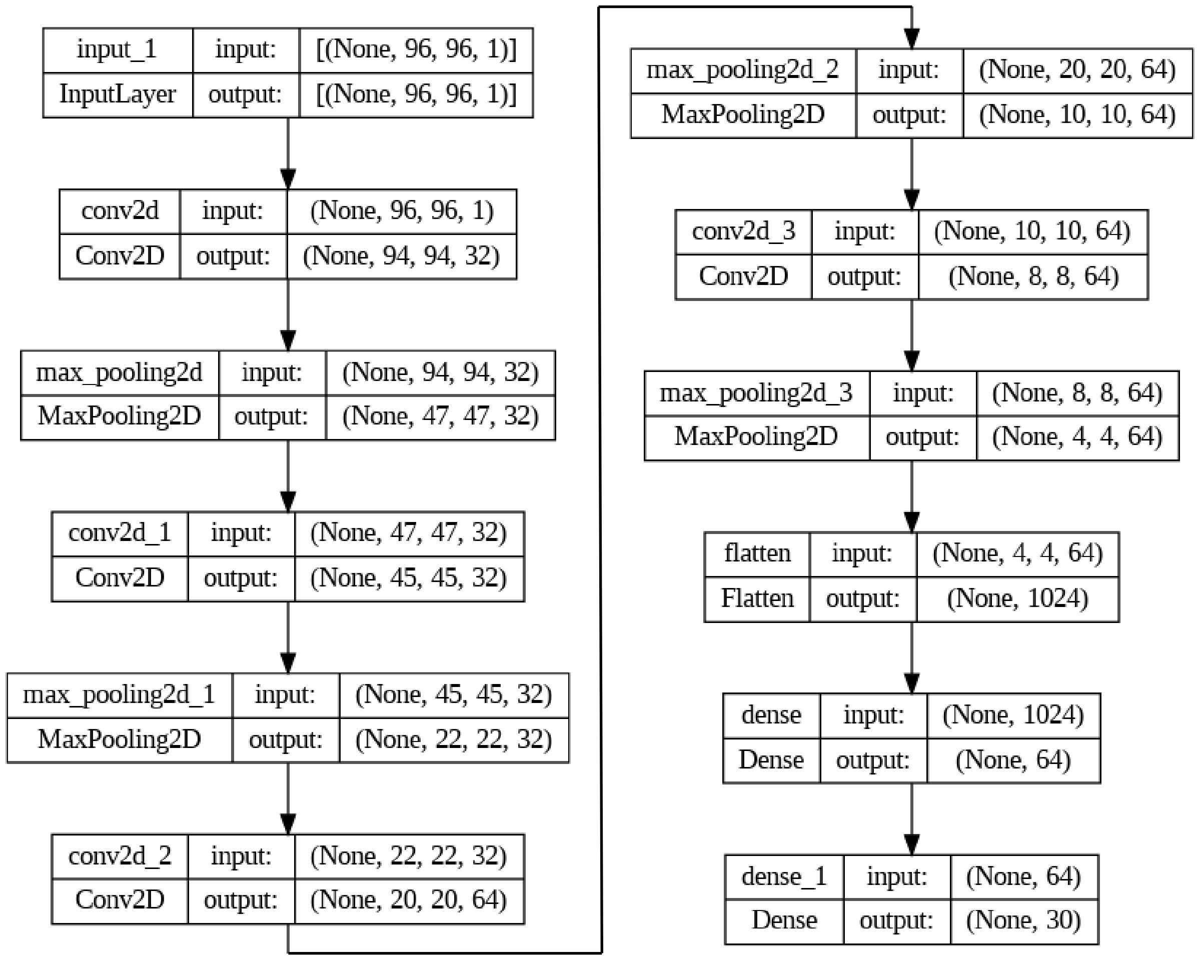
```
(1284, 15, 2)
(428, 15, 2)
(428, 15, 2)
```

$\Rightarrow$

```
(1284, 30)
(428, 30)
(428, 30)
```

(3) Network structure

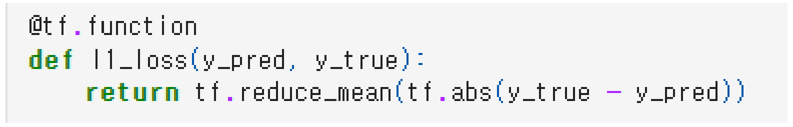
- **Input Layer** : It receives a single-channel image of size 96x96 as input.
- **Convolution Layers** : These layers are stacked alternately. Each convolution layer uses a 3x3 filter and a ReLU activation function to extract features from the input image
- **Pooling Layers** : These layers reduce the size of the feature maps by half using a 2x2 pooling window, preserving spatial information while reducing dimensions.
- **Dropout Layers** : These layers randomly deactivate neurons to prevent overfitting.
- **Flatten Layer** : This layer flattens the tensor output into a 1-dimensional form. It transforms the extracted feature maps from the previous step into a 1D shape.
- **Fully Connected Layers** : The fully connected layer consists of 64 neurons with a ReLU activation function. It uses the extracted features to learn more complex patterns.
- **Output Layer** : It consists of 30 neurons, representing the x and y coordinates of 15 landmarks to be predicted. The output layer does not have an activation function and provides linear outputs.



input_1	InputLayer	input:	[(None, 96, 96, 1)]
		output:	[(None, 96, 96, 1)]
conv2d	Conv2D	input:	(None, 96, 96, 1)
		output:	(None, 94, 94, 32)
max_pooling2d	MaxPooling2D	input:	(None, 94, 94, 32)
		output:	(None, 47, 47, 32)
conv2d_1	Conv2D	input:	(None, 47, 47, 32)
		output:	(None, 45, 45, 32)
max_pooling2d_1	MaxPooling2D	input:	(None, 45, 45, 32)
		output:	(None, 22, 22, 32)
conv2d_2	Conv2D	input:	(None, 22, 22, 32)
		output:	(None, 20, 20, 64)
max_pooling2d_2	MaxPooling2D	input:	(None, 20, 20, 64)
		output:	(None, 10, 10, 64)
conv2d_3	Conv2D	input:	(None, 10, 10, 64)
		output:	(None, 8, 8, 64)
max_pooling2d_3	MaxPooling2D	input:	(None, 8, 8, 64)
		output:	(None, 4, 4, 64)
flatten	Flatten	input:	(None, 4, 4, 64)
		output:	(None, 1024)
dense	Dense	input:	(None, 1024)
		output:	(None, 64)
dense_1	Dense	input:	(None, 64)
		output:	(None, 30)

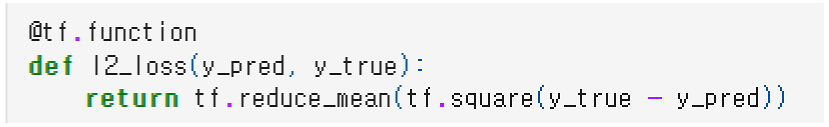
(4) Loss functions

- L1-loss



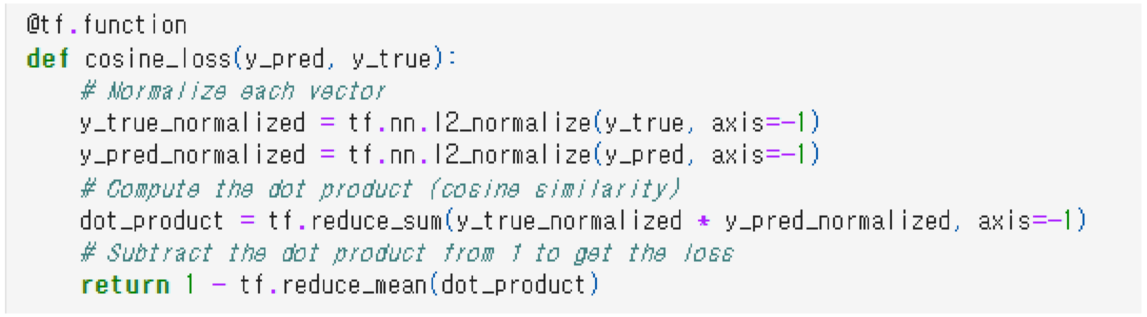
```
@tf.function
def l1_loss(y_pred, y_true):
    return tf.reduce_mean(tf.abs(y_true - y_pred))
```

- L2-loss



```
@tf.function
def l2_loss(y_pred, y_true):
    return tf.reduce_mean(tf.square(y_true - y_pred))
```

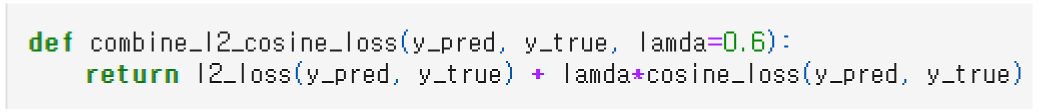
- cosine-loss



```
@tf.function
def cosine_loss(y_pred, y_true):
    # Normalize each vector
    y_true_normalized = tf.nn.l2_normalize(y_true, axis=-1)
    y_pred_normalized = tf.nn.l2_normalize(y_pred, axis=-1)
    # Compute the dot product (cosine similarity)
    dot_product = tf.reduce_sum(y_true_normalized * y_pred_normalized, axis=-1)
    # Subtract the dot product from 1 to get the loss
    return 1 - tf.reduce_mean(dot_product)
```

- Combined

  (L2-loss+cosine_loss)



```
def combine_l2_cosine_loss(y_pred, y_true, lamda=0.6):
    return l2_loss(y_pred, y_true) + lamda*cosine_loss(y_pred, y_true)
```

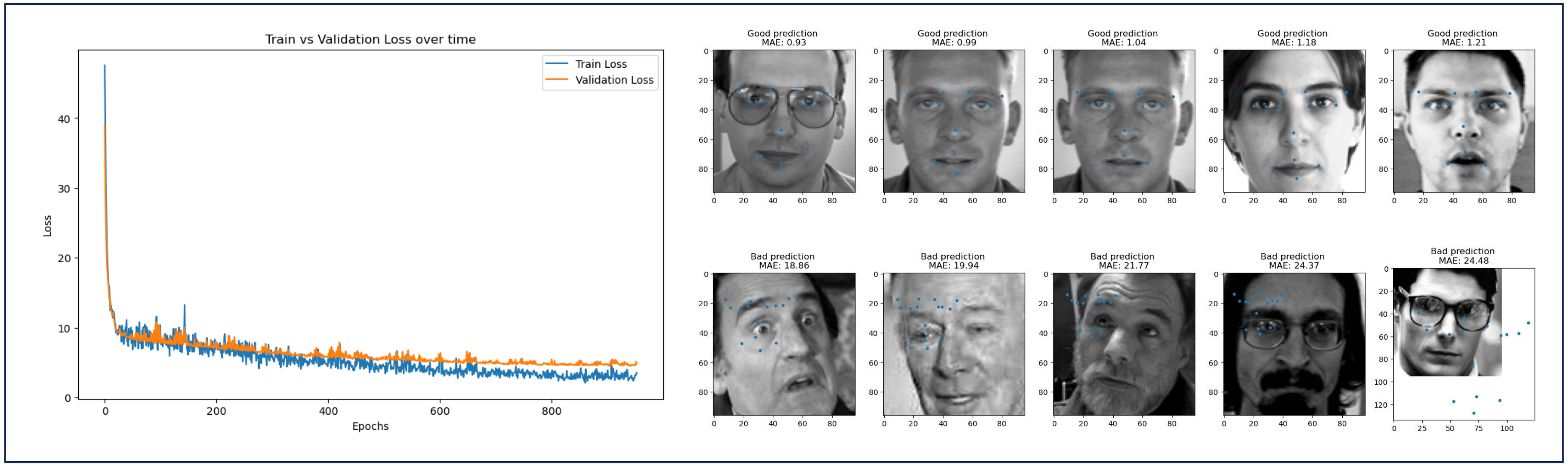
(5) Results

- Validation performance comparison (Loss functions)
  1. L1-loss
     - Validation Loss : 4.54167
     - Early Stopping : Training ended after 953 epochs
  2. L2-loss
     - Validation Loss : 46.915615
     - Early Stopping : Training ended after 757 epochs
  3. cosine-loss
     - Validation Loss : 0.0010575056
     - Early Stopping : Training ended after 1000 epochs (no early stopping)
  4. combined
     - Validation Loss : 43.097847
     - Early Stopping : Training ended after 810 epochs

(5) Results

- Validation performance comparison (Loss functions)

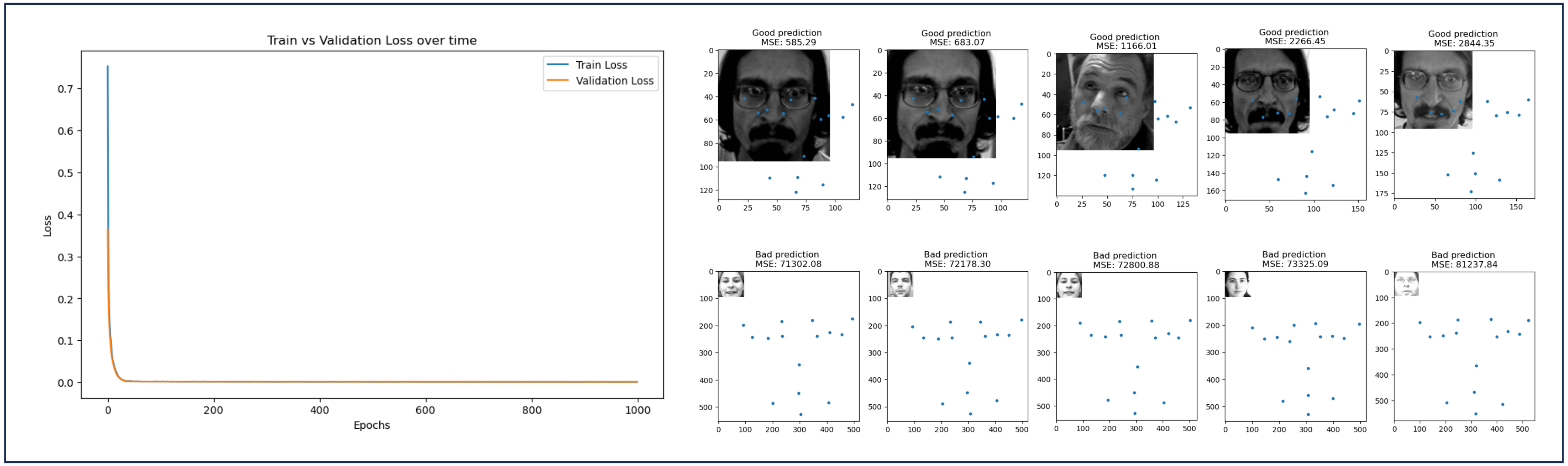
The performance appears to be better with cosine-loss, L1-loss, combined loss, and L2-loss in that order. However, we identified issues when applying L2-loss and cosine-loss. **L1-loss** calculates the absolute difference between predicted values and actual values, which makes it non-differentiable at every point. This can be problematic when using optimization algorithms like gradient descent to train the model.



(5) Results

- Validation performance comparison (Loss functions)

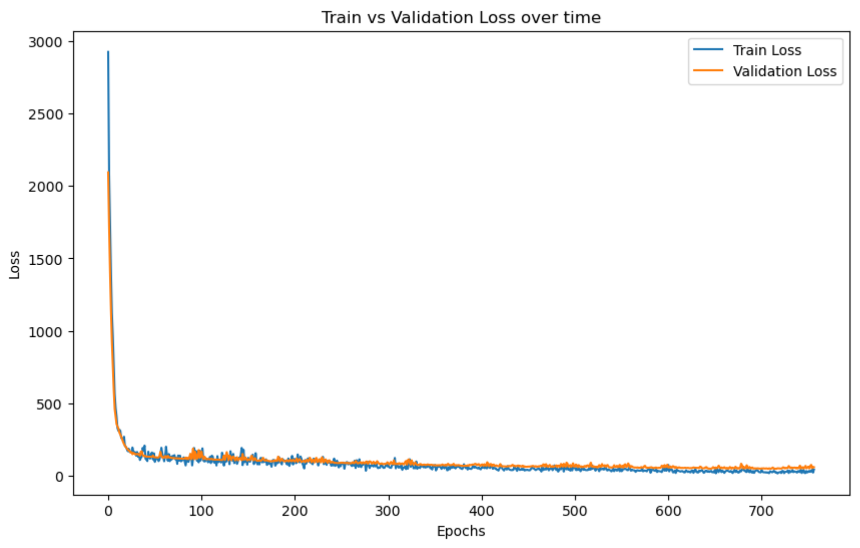
Also, when **cosine-loss** was applied, it showed excellent performance in terms of test and validation results. However, it exhibited poor performance on the test dataset, leading us to conclude that the model was overfitting.

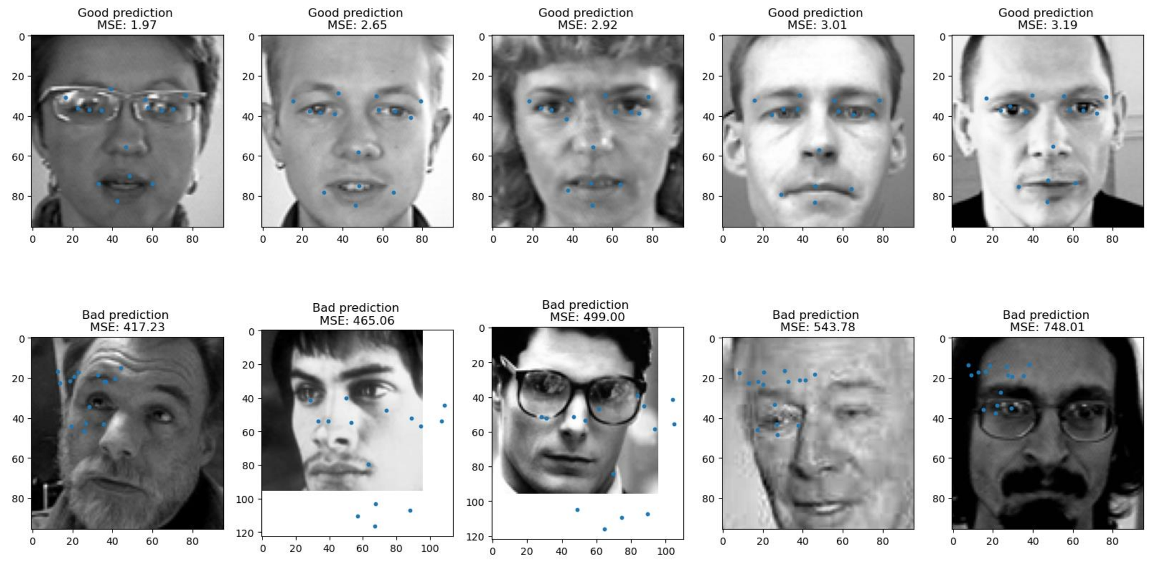


(5) Results

- Validation performance comparison (Loss functions)

When comparing the remaining **L2-loss** and combined loss, they showed similar losses. However, the combined loss exhibited better predictions during testing, leading us to choose it as the best-performing loss function.



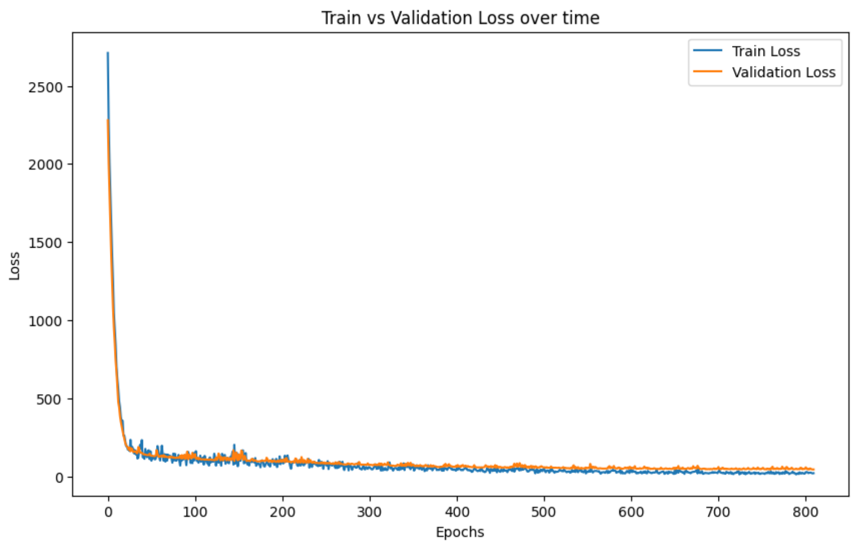


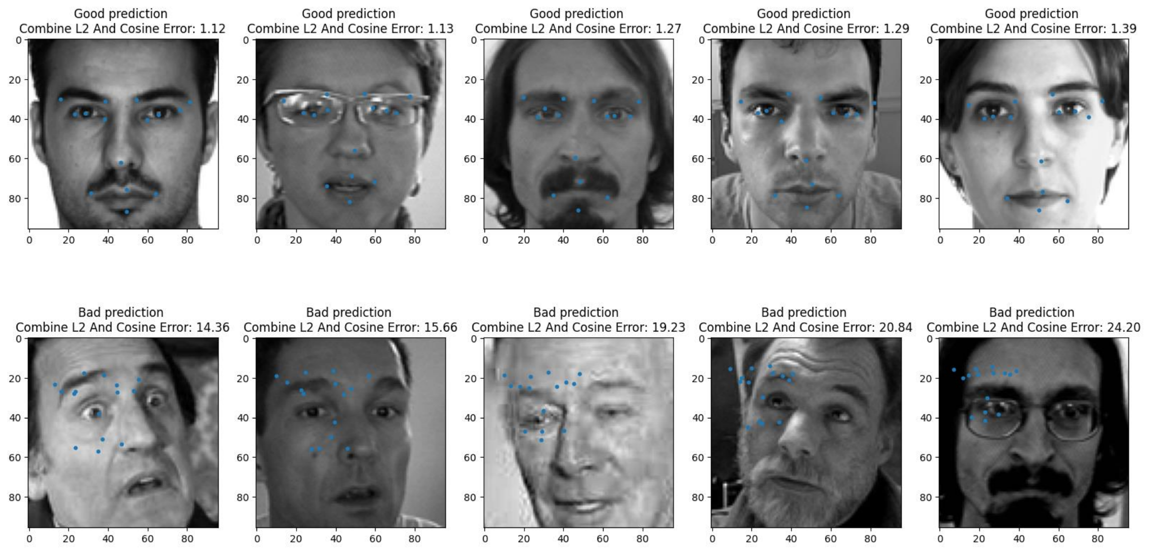
Good prediction MSE: 1.97
Good prediction MSE: 2.65
Good prediction MSE: 2.92
Good prediction MSE: 3.01
Good prediction MSE: 3.19
Bad prediction MSE: 417.23
Bad prediction MSE: 465.06
Bad prediction MSE: 499.00
Bad prediction MSE: 543.78
Bad prediction MSE: 748.01

(5) Results

- Validation performance comparison (Loss functions)

When comparing the remaining L2-loss and **combined loss**, they showed similar losses. However, the combined loss exhibited better predictions during testing, leading us to choose it as the best-performing loss function.




Good prediction
Combine L2 And Cosine Error: 1.12
Good prediction
Combine L2 And Cosine Error: 1.13
Good prediction
Combine L2 And Cosine Error: 1.27
Good prediction
Combine L2 And Cosine Error: 1.29
Good prediction
Combine L2 And Cosine Error: 1.39
Bad prediction
Combine L2 And Cosine Error: 14.36
Bad prediction
Combine L2 And Cosine Error: 15.66
Bad prediction
Combine L2 And Cosine Error: 19.23
Bad prediction
Combine L2 And Cosine Error: 20.84
Bad prediction
Combine L2 And Cosine Error: 24.20

(5) Results

- Validation performance comparison (Lamda values)

Lamda value	Validation loss	Early stopping
0.1	48.217827	879
0.2	47.032726	640
0.4	45.947998	839
0.6	43.097847	810
0.8	43.745644	732
1.0	46.398018	788
1.2	45.825542	706
1.4	41.44529	796

As the lamda value increases, we observe a decreasing trend in the validation loss, indicating an improvement in the model's performance. However, when the lamda value becomes too large, the validation loss starts to increase again, suggesting the possibility of overfitting. Additionally, we observe that as the lamda value increases, the number of epochs for early stopping decreases. This indicates that the model converges faster and terminates training earlier with larger lamda values. However, excessively large lamda values may lead to early stopping occurring too soon, limiting the model's generalization ability.

It can be concluded that among the lamda values provided, 1.4 appears to be the optimal choice. It demonstrates improved performance based on the validation loss and achieves early stopping at a reasonable epoch.

(5) Results

- Validation performance comparison (Regularization)

1. L1 Regularization(0.01)
   - Validation Loss : 45.98259
   - Early Stopping : Training ended after 799 epochs
2. L2 Regularization(0.01)
   - Validation Loss : 51.54513
   - Early Stopping : Training ended after 700 epochs

In the given results, the model with L1 regularization showed approximately 11.6% lower Validation Loss compared to the model with L2 regularization, indicating better performance. This can be interpreted as the effect of L1 regularization in improving the model's generalization performance. The reason behind this result is that L1 regularization imposes stronger constraints on the model's weights. By limiting the absolute values of the weights and removing unnecessary features, L1 regularization helps the model become more concise and focus on important features.

(5) Results

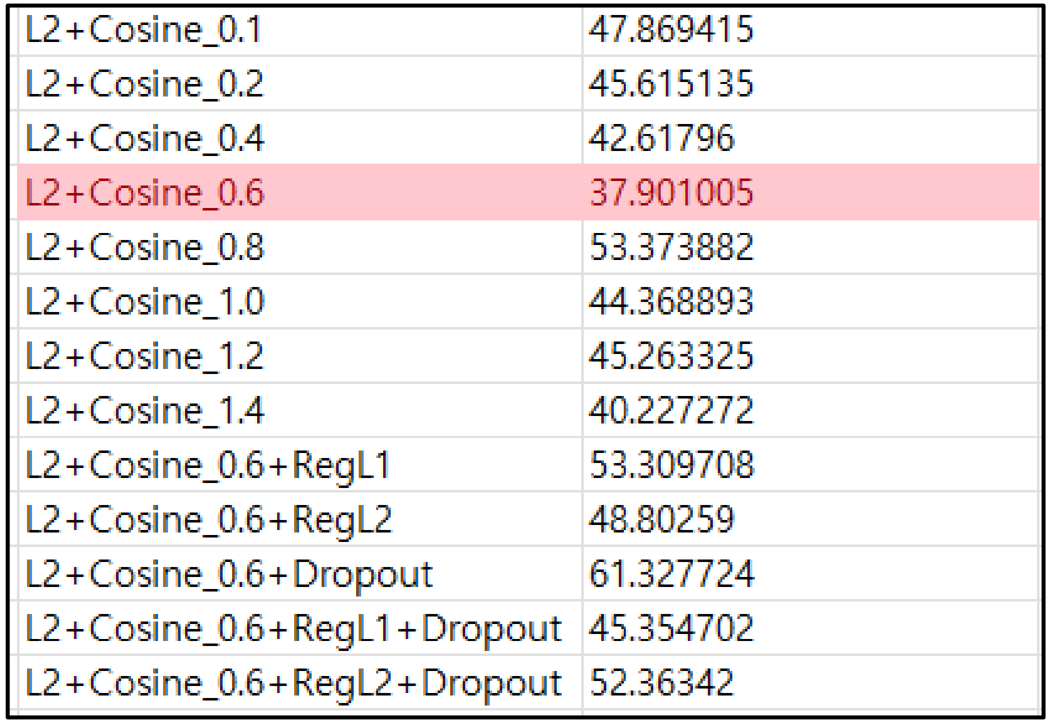
- Validation performance comparison (Dropout)
    1. Model with Dropout
        - Validation Loss : 43.097847
        - Early Stopping : Training ended after 723 epochs
    2. Model without Dropout
        - Validation Loss : 42.062702
        - Early Stopping : Training ended after 810 epochs

In terms of validation loss, the model without Dropout showed a lower value. However, when Early Stopping was applied, the model with Dropout finished training after 723 epochs, while the model without Dropout finished training after 810 epochs. This indicates that the model with Dropout converged faster and obtained the optimal model more quickly with the help of Early Stopping.

(5) Results

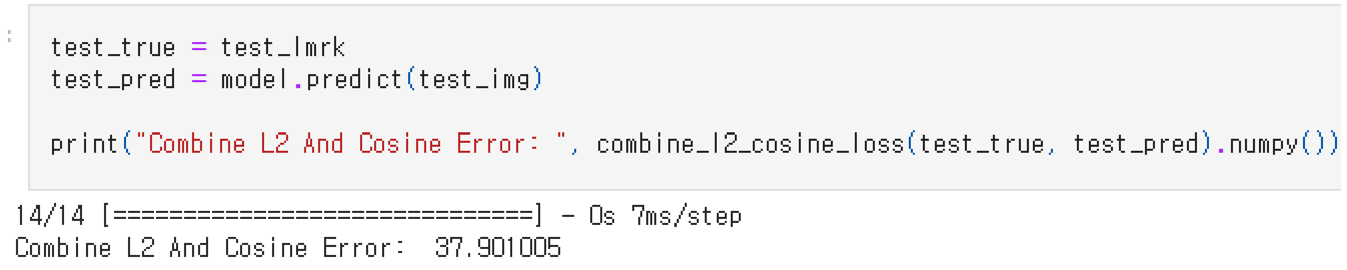
- Final performance on test set (Prediction error)

Combined loss (with a cosine lamda value of 0.6) were used to evaluate several models, and among them, the model with the lowest error was selected as the best model.



L2+Cosine_0.1	47.869415
L2+Cosine_0.2	45.615135
L2+Cosine_0.4	42.61796
L2+Cosine_0.6	37.901005
L2+Cosine_0.8	53.373882
L2+Cosine_1.0	44.368893
L2+Cosine_1.2	45.263325
L2+Cosine_1.4	40.227272
L2+Cosine_0.6+RegL1	53.309708
L2+Cosine_0.6+RegL2	48.80259
L2+Cosine_0.6+Dropout	61.327724
L2+Cosine_0.6+RegL1+Dropout	45.354702
L2+Cosine_0.6+RegL2+Dropout	52.36342

The evaluation of the best model resulted in an error of **37.901005**. This error measurement takes into account both the L2 distance and cosine similarity between the predicted landmarks and the ground truth landmarks. The lower the error value, the better the performance of the model, indicating that the predictions of the best model are closer to the ground truth landmarks compared to other models.



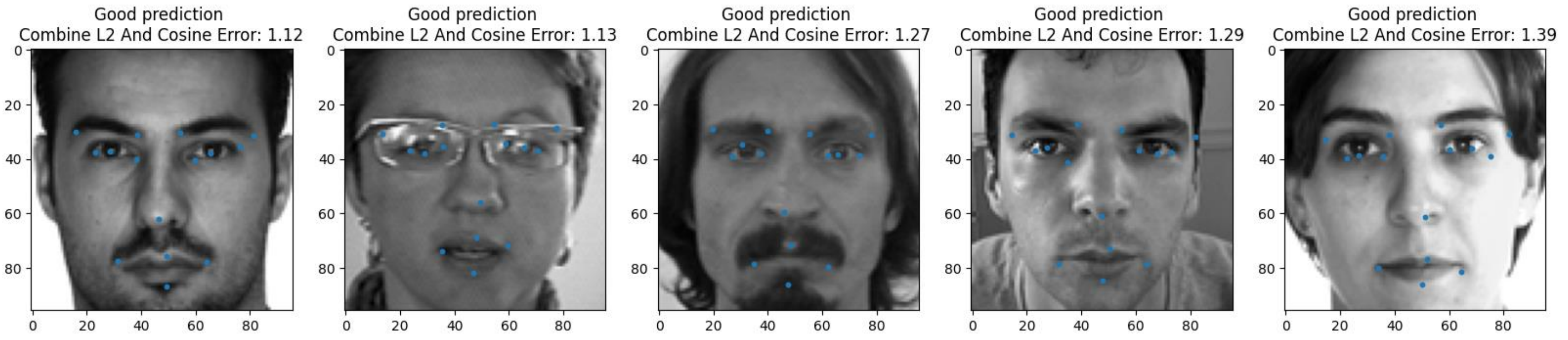
```
test_true = test_lmrk
test_pred = model.predict(test_img)

print("Combine L2 And Cosine Error: ", combine_l2_cosine_loss(test_true, test_pred).numpy())
```

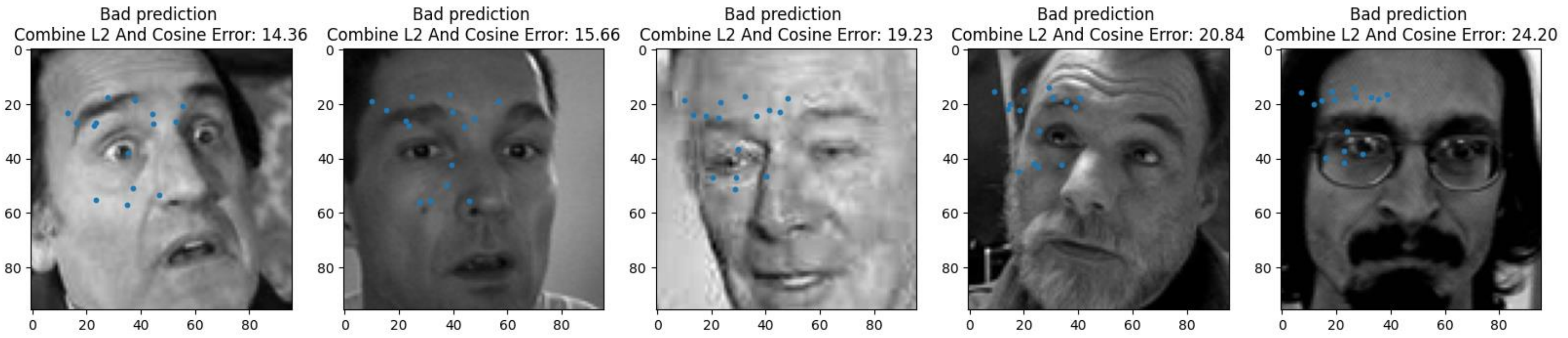
```
14/14 [==============================] - 0s 7ms/step
Combine L2 And Cosine Error:  37.901005
```

(5) Results

- Good prediction


Good prediction
Combine L2 And Cosine Error: 1.12
Good prediction
Combine L2 And Cosine Error: 1.13
Good prediction
Combine L2 And Cosine Error: 1.27
Good prediction
Combine L2 And Cosine Error: 1.29
Good prediction
Combine L2 And Cosine Error: 1.39

- Bad prediction


Bad prediction
Combine L2 And Cosine Error: 14.36
Bad prediction
Combine L2 And Cosine Error: 15.66
Bad prediction
Combine L2 And Cosine Error: 19.23
Bad prediction
Combine L2 And Cosine Error: 20.84
Bad prediction
Combine L2 And Cosine Error: 24.20

(5) Results

- Discussion

Due to the presence of numerous unadjusted hyperparameters, We believe that better results can be achieved by exploring a wider range of hyperparameter configurations during modeling. By adjusting additional hyperparameters, we can fine-tune the model and potentially improve its performance. This approach allows for more flexibility and optimization in capturing the intricacies of the data and the model's architecture. Therefore, by carefully tuning a diverse set of hyperparameters, we can increase the chances of obtaining superior results in our modeling endeavors.

(6) Code

- GitHub : https://github.com/DI-LEE/HUFS_CV/tree/main