[Machine Learning]

[2023-1]

Homework 4

[Due Date] 2023.06.14

Student ID : 2018112007

Name : 이승현

Professor : Juntae Kim

1. What is the *Adam* optimization algorithm? Explain how it is different from the normal Stochastic Gradient Descent. (10 pts)

Your Answer
Adam 최적화 알고리즘은 모멘텀과 RMSprop 을 섞어 놓은 최적화 알고리즘으로, 딥러닝에서 널리 사용되는 알고리즘이다. Adam 은 "Adaptive Moment Estimation"의 약자로, adaptive learning rate 를 사용하여 Gradient Descent 의 성능을 향상시키는 방법이다. Adam 알고리즘은 Gradient Descent 의 단점 중 하나인 learning rate 를 정하는 것에 대한 어려움을 해결하기 위해 개발되었다. 기존의 Gradient Descent 에서는 모든 가중치에 동일한 learning rate 를 적용하는 반면, Adam 은 가중치마다 다른 learning rate 를 적용하여 learning rate 를 개별적으로 조절한다. Adam 알고리즘의 주요 아이디어는 두 가지 추정치, 즉 일차 모멘트 추정(First Moment Estimation)과 이차 모멘트 추정(Second Moment Estimation)을 유지하면서 가중치 업데이트를 수행하는 것이다. 이러한 추정치는 이전의 gradient 정보를 사용하여 계산된다. 일차 모멘트 추정은 gradient 의 지수 이동 평균(Exponential Moving Average)을 계산하여 Gradient 의 평균을 추정하고, 이차 모멘트 추정은 gradient 제곱의 지수 이동 평균을 계산하여 gradient 의 분산을 추정한다. Adam 은 이러한 추정치를 사용하여 가중치를 업데이트합니다. 각 가중치에 대해 learning rate 를 개별적으로 조정하고, 모멘트 추정치를 사용하여 Gradient 의 크기와 방향을 결정한다. 따라서 Adam 은 Gradient Descent 에 비해 빠르고 효율적인 학습을 할 수 있다.

2. Describe following activation functions and their characteristics: *sigmoid, tanh, ReLU, Leaky ReLU, ELU*. Refer to the TensorFlow API and show how it is defined. (10 pts)

Your Answer
1. sigmoid • sigmoid 함수는 입력값을 0 과 1 사이의 값으로 압축하는 S 모양의 곡선이다. • 입력값이 크게 양수이면 1 에 가까운 값을 출력하고, 크게 음수이면 0 에 가까운 값을 출력한다. • 이진 분류 문제에서 출력층의 활성화 함수로 자주 사용된다. • TensorFlow 에서의 sigmoid 함수 정의: tf.math.sigmoid(x, name=None) 2. tanh • tanh 함수는 입력값을 -1 과 1 사이의 값으로 압축한다. • 입력값이 양수일 때는 1 에 가까운 값을, 음수일 때는 -1 에 가까운 값을 출력한다.

- sigmoid 함수와 마찬가지로 비선형 함수이며, 중심이 0 인 대칭 함수이다.
- TensorFlow 에서의 tanh 함수 정의:

  tf.math.tanh(x, name=None)

3. ReLU
   - ReLU 함수는 입력값이 0 보다 작으면 0 을 출력하고, 0 보다 크면 입력값을 그대로 출력한다.
   - 음수 입력값에 대해 비선형적으로 대응하며, 계산 비용이 낮고 효과적으로 작동하는 특징이 있다.
   - ReLU 함수는 다층 신경망에서 가장 많이 사용되는 활성화 함수 중 하나이다.
   - TensorFlow 에서의 ReLU 함수 정의:

     tf.nn.relu(features, name=None)

4. Leaky ReLU
   - Leaky ReLU 함수는 ReLU 함수의 변형이다.
   - 입력값이 0 보다 작을 때 일정한 작은 기울기를 가진 선형 함수를 적용하여, 음수 영역에서도 작은 값이 흐를 수 있도록 한다.
   - 이로 인해 ReLU 의 dead neuron 문제를 완화하고, 음수 영역에서도 정보를 전달할 수 있다.
   - TensorFlow 에서의 Leaky ReLU 함수 정의:

     tf.nn.leaky_relu(features, alpha=0.2, name=None)

5. ELU
   - ELU 함수는 입력값이 양수일 경우는 그대로 출력하고, 음수일 경우 지수 함수를 사용하여 출력값을 조정한다.
   - 음수 영역에서도 작은 값을 출력하기 때문에 ReLU 의 dead neuron 문제를 완화하고, 더 강한 표현력을 가질 수 있다.
   - ELU 함수는 음수 영역에서 ReLU 보다 부드럽게 변화하므로, gradient 소실 문제를 완화하는 효과가 있다.
   - TensorFlow 에서의 ELU 함수 정의:

     tf.nn.elu(features, name=None)

3. Describe following losses and where they are used: *Mean Squared Error, Categorcal Cross Entropy, Hinge*. Refer to the TensorFlow API and show how it is defined. (10 pts)

Your Answer
1. Mean Squared Error • Mean Squared Error 는 예측값과 실제값 사이의 제곱 오차를 계산하고, 이를 평균화한 값이다.

- 회귀(Regression) 문제에서 주로 사용되며, 예측값과 실제값 사이의 거리를 제곱하여 오차를 계산한다.
- 예측값과 실제값의 차이가 클수록 오차가 커지므로, 모델이 정확한 예측을 하도록 학습된다.
- TensorFlow 에서의 Mean Squared Error 의 loss 함수 정의:

tf.compat.v1.mean_squared_error(labels, predictions, weights=1.0, scope=None, loss_collection=ops.GraphKeys.LOSSES, reduction=Reduction.SUM_BY_NONZERO_WEIGHTS)

2. Categorical Cross Entropy
   - Categorical Cross Entropy 는 Multi-class Classification 문제에서 사용되는 손실 함수이다.
   - 실제값과 예측값 사이의 차이를 측정하기 위해 정보 이론의 개념인 엔트로피를 활용한다.
   - 정답 클래스에 대한 예측값의 확률을 최대화하도록 모델을 학습한다.
   - TensorFlow 에서의 Categorical Cross Entropy 의 loss 함수 정의:

tf.keras.metrics.categorical_crossentropy(y_true, y_pred, from_logits=False, label_smoothing=0.0, axis=-1**)**

3. Hinge
   - Hinge loss 는 이진 분류(Binary Classification) 문제에서 사용되는 손실 함수이다.
   - 서포트 벡터 머신(Support Vector Machine)에서 사용되며, 분류 경계선과의 마진을 최대화하는 것을 목표로 한다.
   - 실제값과 예측값 사이의 차이가 일정 값 이상이면 오차가 발생하며, 그 이하는 오차가 발생하지 않는다.
   - TensorFlow 에서의 Hinge 의 loss 함수 정의:

tf.compat.v1.hinge_loss(labels, logits, weights=1.0, scope=None, loss_collection=ops.GraphKeys.LOSSES, reduction=Reduction.SUM_BY_NONZERO_WEIGHTS)

4. Compute the total number of parameters in CNN with following architecture for 100 x 100 x 3 image input. Stride=1, no padding. Show each layer's output shape and # of parameters. Show how you calculate them. (10 pts)

- conv layer 1: 16 filters of 5 x 5 size + 2 x 2 max pooling
- conv layer 2: 32 filters of 3 x 3 size + 2 x 2 max pooling
- dense layer: 256 outputs
- dense layer: 10 outputs

Your Answer

1. Convolutional Layer 1

- 입력 이미지 크기: 100 x 100 x 3
- 필터 수: 16
- 필터 크기: 5 x 5
- 출력 형태 계산: (입력 크기 - 필터 크기) / 스트라이드 + 1 = (100 - 5) / 1 + 1 = 96 x 96 x 16
- 파라미터 개수 계산: (필터 크기 x 입력 채널 + 1) x 필터 수 = (5 x 5 x 3 + 1) x 16 = 1216

Max Pooling

- 입력 형태: 96 x 96 x 16
- 풀링 크기: 2 x 2
- 출력 형태 계산: (입력 크기 - 풀링 크기) / 스트라이드 + 1 = (96 - 2) / 2 + 1 = 48 x 48 x 16 (각 채널마다 풀링됨)
- 파라미터 개수: 0 (풀링 레이어는 학습 가능한 파라미터가 없음)

2. Convolutional Layer 2

- 입력 형태: 48 x 48 x 16
- 필터 수: 32
- 필터 크기: 3 x 3
- 출력 형태 계산: (입력 크기 - 필터 크기) / 스트라이드 + 1 = (48 - 3) / 1 + 1 = 46 x 46 x 32
- 파라미터 개수 계산: (필터 크기 x 입력 채널 + 1) x 필터 수 = (3 x 3 x 16 + 1) x 32 = 4640

Max Pooling

- 입력 형태: 46 x 46 x 32
- 풀링 크기: 2 x 2
- 출력 형태 계산: (입력 크기 - 풀링 크기) / 스트라이드 + 1 = (46 - 2) / 2 + 1 = 23 x 23 x 32 (각 채널마다 풀링됨)
- 파라미터 개수: 0 (풀링 레이어는 학습 가능한 파라미터가 없음)

3. Dense Layer 1

- 입력 형태: 23 x 23 x 32 = 16928 (1 차원으로 펼쳐짐)
- 출력 개수: 256
- 파라미터 개수 계산: (입력 개수 + 1) x 출력 개수 = (16928 + 1) x 256 = 4333824

4. Dense Layer 2

- 입력 개수: 256
- 출력 개수: 10 (클래스 수에 따라 다름)
- 파라미터 개수 계산: (입력 개수 + 1) x 출력 개수 = (256 + 1) x 10 = 2570

따라서, 주어진 CNN 모델의 총 파라미터 개수는 각 레이어의 파라미터 개수를 합산한 값이다.
1216 + 4640 + 4333824 + 2570 = 4342250

이와 같이 총 4342250 개의 파라미터가 CNN 모델에 존재한다..

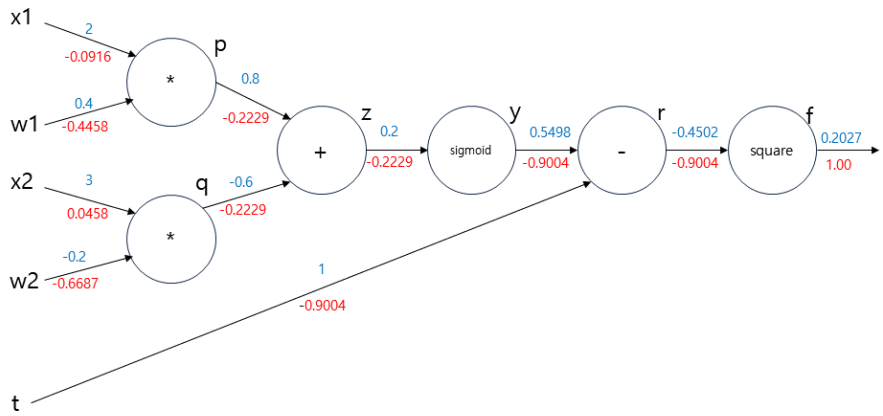
5. Briefly suggest a deep learning model for the machine translation. Describe how you can train the model. (10 pts)

Your Answer
기계 번역을 위한 간단한 딥러닝 모델로. LSTM (Long Short-Term Memory) 기반의 인코더-디코더 모델을 사용한다. 1. 인코더 (Encoder) - 입력 문장을 고정된 길이의 벡터 표현으로 변환한다. - 다층 LSTM 레이어를 사용하여 문장의 시퀀스를 처리하고, 마지막 스텝의 은닉 상태를 출력한다. 2. 디코더 (Decoder) - 인코더의 출력 벡터를 입력으로 받아 번역된 문장을 생성한다. - 다층 LSTM 레이어를 사용하여 번역 문장을 한 단어씩 생성한다. - 초기 입력으로는 시작 토큰을 사용하고, 이전 스텝에서 예측한 단어를 현재 스텝의 입력으로 사용한다. - 번역 문장의 끝을 나타내는 종료 토큰이 생성될 때까지 반복한다. 3. 훈련 (Training) - 훈련 데이터는 입력 문장과 해당 번역 문장 쌍으로 구성된다. - 인코더에 입력 문장을 주어 인코더의 출력 벡터를 얻는다. - 디코더의 초기 입력은 시작 토큰이고, 실제 번역 문장을 디코더의 목표 출력으로 사용한다. - 디코더는 입력과 목표 출력을 사용하여 다음 단어를 예측하고, 손실 함수를 통해 예측과 실제 값을 비교하여 오차를 계산한다. - back propagation 알고리즘을 사용하여 오차를 역전파하고, gradient descent 를 통해 모델의 가중치를 조정한다. 4. 추론 (Inference) - 훈련된 모델을 사용하여 실제 번역을 수행한다. - 입력 문장을 인코더에 주어 인코더의 출력 벡터를 얻는다. - 디코더의 초기 입력은 시작 토큰이고, 디코더를 통해 다음 단어를 예측한다. - 예측된 단어를 다음 입력으로 사용하여 번역 문장을 점진적으로 생성한다. - 종료 토큰이 생성되거나 최대 길이에 도달할 때까지 반복한다.

6. Represent following equations as a computational graph, and compute $\frac{\partial f}{\partial w_1}$ for $x_1 = 2$, $x_2 = 3$, $w_1 = 0.4$, $w_2 = -0.2$, $t = 1$, as automatic differentiation does. Regard sigmoid as one operation and use $\frac{\partial y}{\partial z} = y(1-y)$. (20 pts)

$$z = x_1 w_1 + x_2 w_2$$
$$y = sigmoid(z)$$
$$f = (y - t)^2$$

Your Answer



P = x1 * w1

q = x2 * w2

z = p + q

y = sigmoid(z)

r = y – t

f = r ^ 2

$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial \mathrm{r}} * \frac{\partial \mathrm{r}}{\partial y} * \frac{\partial y}{\partial \mathrm{z}} * \frac{\partial \mathrm{z}}{\partial \mathrm{p}} * \frac{\partial \mathrm{p}}{\partial w_1}$

$\frac{\partial f}{\partial \mathrm{r}} = 1 * 2 * (-0.4502) = -0.9004$

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial \mathrm{r}} * \frac{\partial \mathrm{r}}{\partial y} = -0.9004 * 1 = -0.9004$

$\frac{\partial f}{\partial \mathrm{z}} = \frac{\partial f}{\partial y} * \frac{\partial y}{\partial \mathrm{z}} = -0.9004 * 0.5498 * (1 - 0.5498) = -0.2229$

$\frac{\partial f}{\partial \mathrm{p}} = \frac{\partial f}{\partial \mathrm{z}} * \frac{\partial \mathrm{z}}{\partial \mathrm{p}} = -0.2229 * 1 = -0.2229$
$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial \mathrm{p}} * \frac{\partial \mathrm{p}}{w_1} = -0.2229 * 2 = -0.4458$
따라서, $\frac{\partial f}{\partial w_1} = -0.4458$ 이다.

7. Design a CNN model for image classification on Olivetti Faces Dataset to identify persons from images. You may design your own model with arbitrary number of layers and filters. Train your model with 60% of the data, and check the accuracy on the remaining 40% data. Describe the learned model, show the model accuracy, and show examples of correct and incorrect classification results. (30 pts)

- Sample model architecture :
  3 convolution layers with 32 filters of size (3, 3), 64 filters of size (3, 3),
  64 filters of size (3, 3) each. Max pooling with size (2, 2) after 1st and 3rd layer.
  2 fully connected layers with 512, 40 outputs each.
  Use dropout for some layers.

- Dataset :
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_olivetti_faces.html

The Olivetti Faces dataset contains a set of face images taken at AT&T Laboratories Cambridge. The sklearn.datasets.fetch_olivetti_faces is the data fetching function that downloads the data. There are 10 different images of each of 40 distinct persons. For some persons, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). The 64x64 pixels image is quantized to 256 grey levels and stored as unsigned 8-bit integers; the loader will convert these to floating point values on the interval [0, 1]. The target for this database is an integer from 0 to 39 indicating the identity of the person pictured.

- Plotting several images (person 0, 1, 2)

```
from sklearn import datasets
from matplotlib import pyplot as plt
```

```
face = datasets.fetch_olivetti_faces()
X = face.data
y = face.target

%matplotlib inline

fig = plt.figure(figsize=(10, 4))
for i in range(30):
    ax = fig.add_subplot(3, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(face.images[i], cmap=plt.cm.bone)
```

Code
import numpy as np import matplotlib.pyplot as plt from sklearn.datasets import fetch_olivetti_faces from sklearn.model_selection import train_test_split from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout face = fetch_olivetti_faces() X = face.data.reshape(-1, 64, 64, 1) y = face.target from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1, stratify=y) model = Sequential([

```
Conv2D(32, kernel_size=(3, 3), input_shape=(64, 64, 1), activation="relu"),
MaxPool2D(pool_size=2),
Conv2D(64, kernel_size=(3, 3), activation="relu"),
Conv2D(64, kernel_size=(3, 3), activation="relu"),
MaxPool2D(pool_size=2),
Flatten(),
Dense(512, activation="relu"),
Dropout(.5),
Dense(40, activation="softmax")
])

model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

plt.plot(history.epoch, history.history['loss'])

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

loss, acc = model.evaluate(X_train, y_train)
print('Train accuracy : %.4f' % acc)
loss, acc = model.evaluate(X_test, y_test)
print('Test accuracy : %.4f' % acc)

predictions = model.predict(X_test[:10])
predicted_labels = np.argmax(predictions, axis=1)
true_labels = y_test[:10]

print("Predicted Labels:", predicted_labels)
print("True Labels:", true_labels)

plt.imshow(face.images[predicted_labels[5]], cmap=plt.cm.bone)

plt.imshow(face.images[true_labels[5]], cmap=plt.cm.bone)
```

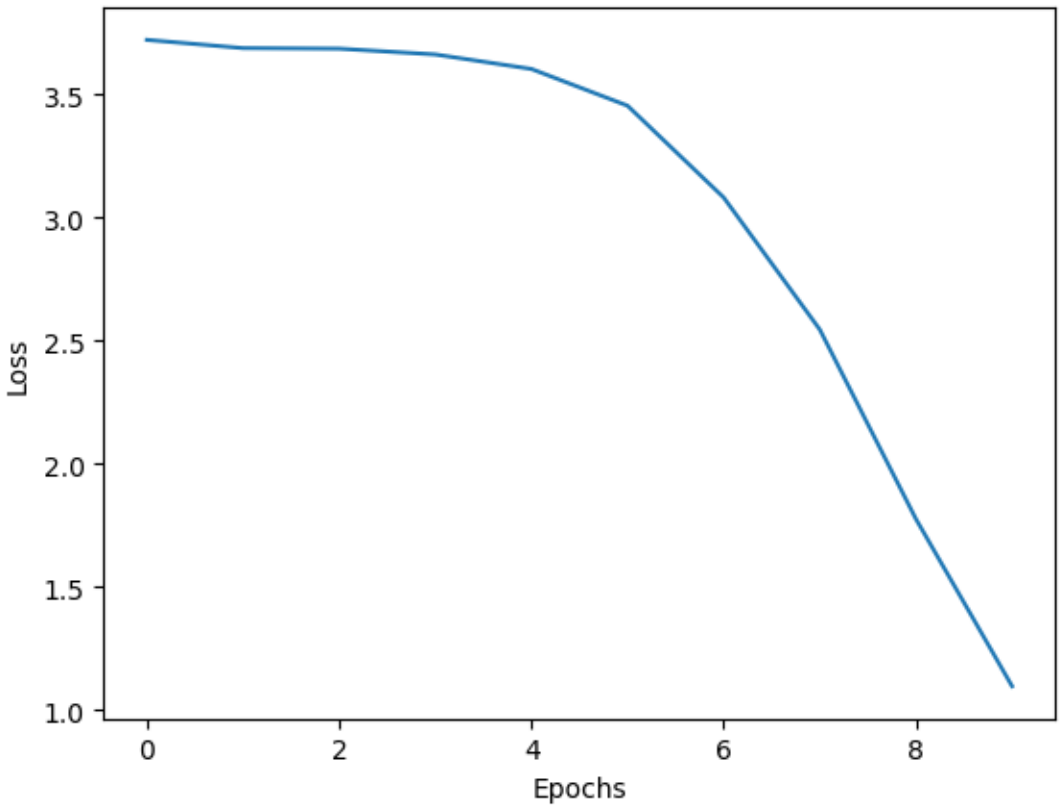
```
plt.imshow(face.images[predicted_labels[0]], cmap=plt.cm.bone)

plt.imshow(face.images[true_labels[0]], cmap=plt.cm.bone)
```

Result(Captured images)

```
Epoch 1/10
8/8 [==============================] - 5s 557ms/step - loss: 3.7215 - accuracy: 0.0042 - val_loss: 3.6868 - val_accuracy: 0.0250
Epoch 2/10
8/8 [==============================] - 4s 440ms/step - loss: 3.6880 - accuracy: 0.0375 - val_loss: 3.6826 - val_accuracy: 0.0500
Epoch 3/10
8/8 [==============================] - 3s 418ms/step - loss: 3.6858 - accuracy: 0.0417 - val_loss: 3.6691 - val_accuracy: 0.0875
Epoch 4/10
8/8 [==============================] - 3s 423ms/step - loss: 3.6623 - accuracy: 0.0708 - val_loss: 3.6300 - val_accuracy: 0.1375
Epoch 5/10
8/8 [==============================] - 5s 631ms/step - loss: 3.6034 - accuracy: 0.0875 - val_loss: 3.5395 - val_accuracy: 0.1437
Epoch 6/10
8/8 [==============================] - 3s 427ms/step - loss: 3.4542 - accuracy: 0.1417 - val_loss: 3.3019 - val_accuracy: 0.2125
Epoch 7/10
8/8 [==============================] - 3s 426ms/step - loss: 3.0815 - accuracy: 0.2375 - val_loss: 2.7263 - val_accuracy: 0.4625
Epoch 8/10
8/8 [==============================] - 4s 485ms/step - loss: 2.5451 - accuracy: 0.3958 - val_loss: 2.0080 - val_accuracy: 0.6875
Epoch 9/10
8/8 [==============================] - 4s 469ms/step - loss: 1.7759 - accuracy: 0.5583 - val_loss: 1.3739 - val_accuracy: 0.7625
Epoch 10/10
8/8 [==============================] - 3s 424ms/step - loss: 1.0969 - accuracy: 0.7083 - val_loss: 0.9795 - val_accuracy: 0.7563
```

- 10 번의 epoch 로 모델을 훈련하고 있다.
- 훈련을 거듭할수록 정확도가 높아지고, loss 는 줄어들고 있다.
- Validation data 에 대해서도 정확도는 높아지고, loss 가 줄어들고 있다.



- Epoch 에 따른 loss 감소율을 보여주고 있다.

- 처음에 완만하게 감소하다가 끝에서 급격하게 감소하는 모습을 확인할 수 있다.

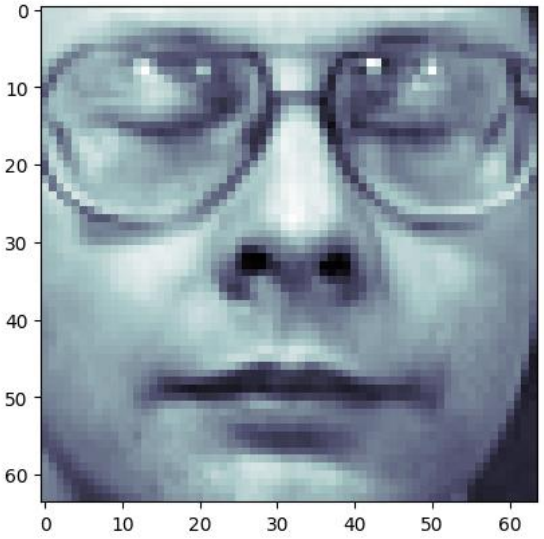
```
8/8 [==============================] - 1s 81ms/step - loss: 0.5122 - accuracy: 0.9500
Train accuracy : 0.9500
5/5 [==============================] - 0s 88ms/step - loss: 0.9795 - accuracy: 0.7563
Test accuracy : 0.7563
```

- 정확도를 출력한 모습이다.
- Training data 에 대한 정확도는 0.9500 로 높은 모습을 확인할 수 있다.
- Training data 에 대한 loss 는 0.5122 이다.
- Test data 에 대한 정확도는 0.7563 으로 training data 보다 다소 낮다.
- Test data 에 대한 정확도는 0.9795 로 training data 보다 다소 높다.

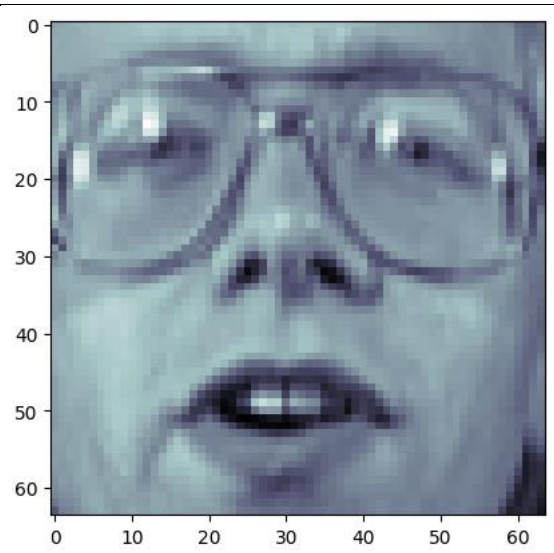
```
1/1 [==============================] - 0s 100ms/step
Predicted Labels: [19 19 17  2 27 38 13  9 37 21]
True Labels: [19 19 17  2 27 14 13  9 37 21]
```

- 예측된 클래스와 실제 클래스의 값을 처음부터 10 개만 출력한 모습이다.
- 대부분 값이 일치하지만 6 번째에서 클래스 값이 다르다.

<예측과 실제가 다른 경우>

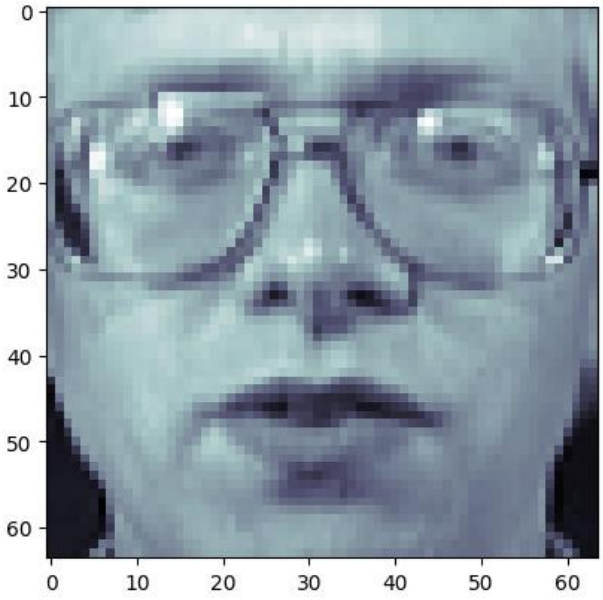


- 모델을 훈련한 후 prediction 을 수행하였을 때 원래 값과 다른 경우를 살펴본다.
- 위 이미지는 예측된 이미지 중 6 번째를 출력한 것이다.

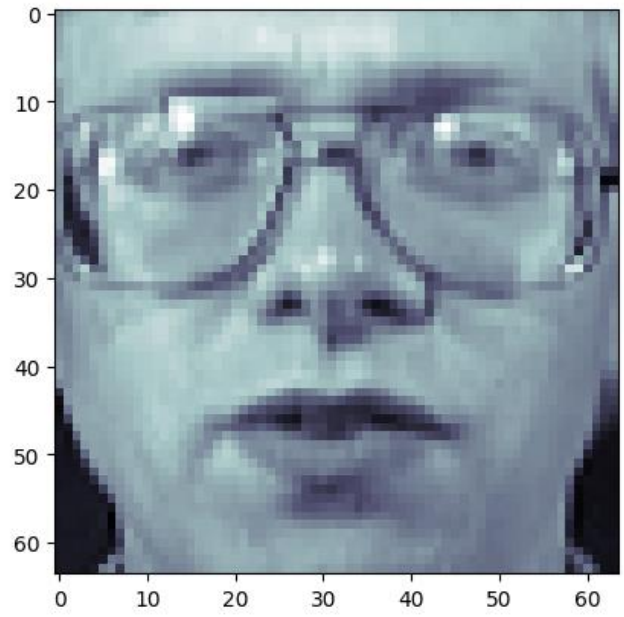


- 실제 이미지 중 6 번째를 출력한 것이다.
- 위의 예측된 이미지와 다른 이미지임을 확인할 수 있다.

<예측과 실제가 같은 경우>



- 반대로 실제 값과 예측 값이 같은 경우를 확인한다.
- 위 이미지는 예측된 이미지 중 첫번째 이미지이다.



- 실제 이미지 중 첫번째 이미지이다.
- 예측된 이미지와 같은 모습을 띈다.

Description

The Olivetti Faces dataset 을 가져와 training data 60%, test data 40%로 나누어 모델의 학습과 예측을 진행하였다. 이때 reshape 로 이미지 data 의 크기를 조정한 다음 모델에 전달하도록 했다.
모델의 구성으로는 32 * 3 * 3 크기의 convolutional layer 와 64 * 3 * 3 크기의 convolutional layer 2 개에 fully connected layer 2 개 구성된다. 이 때 첫번째와 세번째 레이어에 2 * 2 max pooling 을 진행한 후, 레이어 절반을 대상으로 drop 을 진행하였다.

모델의 최적화 알고리즘으로 adam, loss 함수로 sparse categorical cross entropy 를 선택하였고, 정확도를 측정하기 위해 metrics 를 지정하였다.

모델을 training data set 을 이용하여 학습을 하고, 이때 test data set 으로 validation 을 진행하였다. 10 번의 epoch 를 거치면서 최종 loss 는 1.0969, 정확도는 0.7083 이었으며, validation data 에 대해서 loss 는 0.9795, 정확도는 0.7563 으로 training data set 보다 다소 결과가 좋지 못했다. 그래도 훈련을 거듭할수록 loss 는 낮아지고, 정확도가 증가하는 모습을 확인할 수 있다. 위의 그래프를 살펴보면 처음에는 loss 가 완만히 감소하다가 후반부에 급격하게 감소하는 모습을 확인할 수 있다.

모델의 정확도를 training data 와 test data 를 이용해 측정해본 결과 training data 는 0.9500, test data 는 0.7563 이라는 결과가 나왔다.

Test data 에 대하여 예측을 진행하고 실제 값과 같은 지 확인해보기 위해 처음부터 10 개의 이미지에 대하여 어떤 클래스에 속해 있는지 출력해봤는데 대부분 실제 클래스와 예측 클래스가 같았지만 6 번째 data 에서 예측 값이 다른 모습을 확인할 수 있다.

이를 확인하기 위해 이미지를 출력해봤더니 실제로 이미지가 다른 모습을 확인할 수 있었다. 또한 실제 클래스와 예측 클래스가 같은 data 를 출력했더니 이미지가 일치하는 모습을 확인할 수 있었다.

Note

1. Summit the file to e-class as pdf.
2. Specify your pdf file name as “hw4_<StudentID>_<Name>.pdf”