머신러닝

3차 과제

RESNET USING TENSORFLOW

담당 교수님: 이혁준

강의 시간: 월3, 수4

학 과: 컴퓨터정보공학부

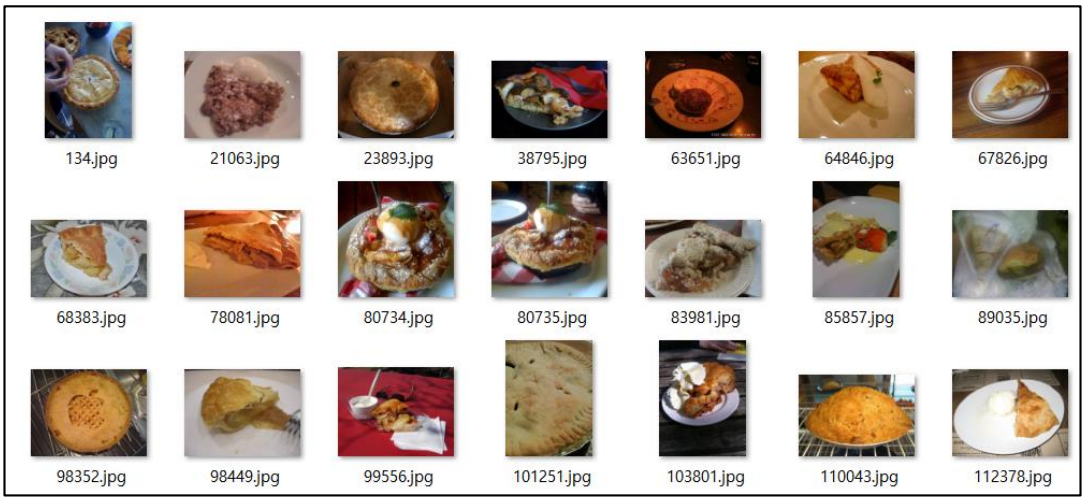
학 번: 2020202055

성 명: 최소윤

1. 과제 목적

Tensorflow를 사용하여 ResNet20을 구현하며 kaggle에 있는 오픈 데이터 셋인 Food-101 데이터를 학습에 사용한다. ResNet의 구조를 이해하고 어떤 과정을 통해 학습이 이루어지는지에 대해 배운다. 또한 ResNet을 통해 이미지를 학습하여 classification을 하면서 정확도를 높이기 위해 각 파라미터를 어떻게 설정하는 것이 최적의 결과를 도출할 지 고민해보고 비교하며 이를 구현한다.

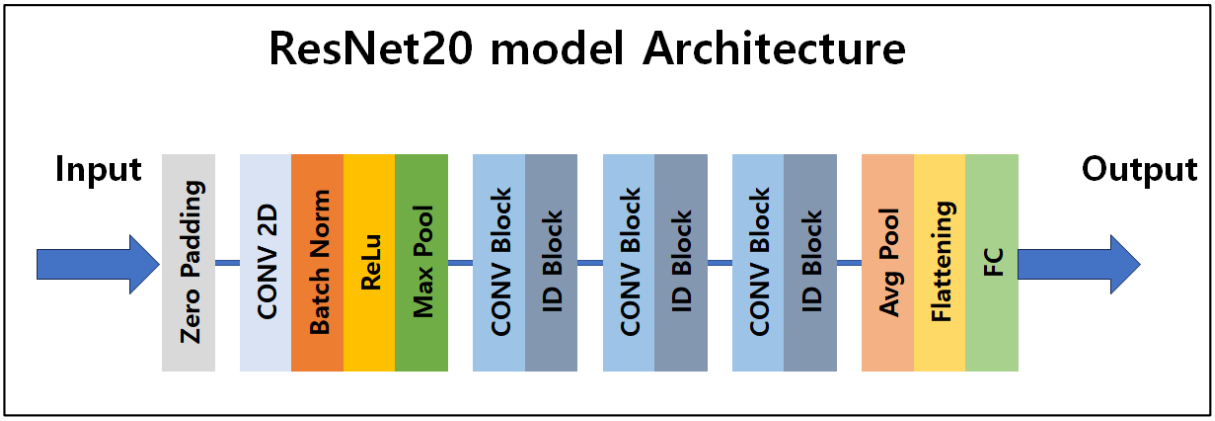
2. 데이터



[그림1] Food-101 dataset (apple pies)

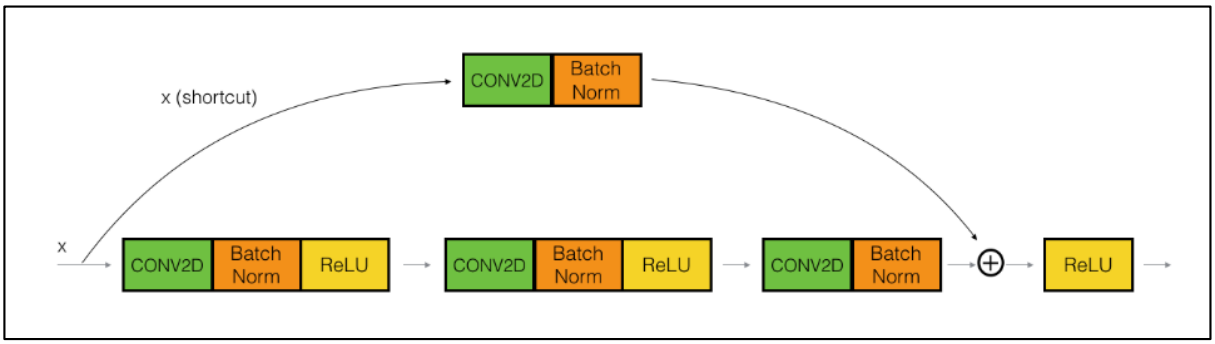
apple pies부터 waffles까지 101개 카테고리의 라벨이 지정된 음식 이미지 데이터 셋이다. 최상위 폴더 food101 폴더 하위에 images 폴더와 meta 폴더가 있고, images 폴더 밑에는 101개의 이미지 클래스 폴더들이 있다. 각 폴더 안에는 각각 1,000개의 이미지들이 있다. meta 폴더 안에 meta 폴더 안에는 train.txt와 test.txt등을 포함한 메타 파일들이 존재한다. 이미지의 크기는 다 다르며, 대부분의 이미지는 약 300 x 300 픽셀 크기로 조정되어 있다.

3. 네트워크 구조



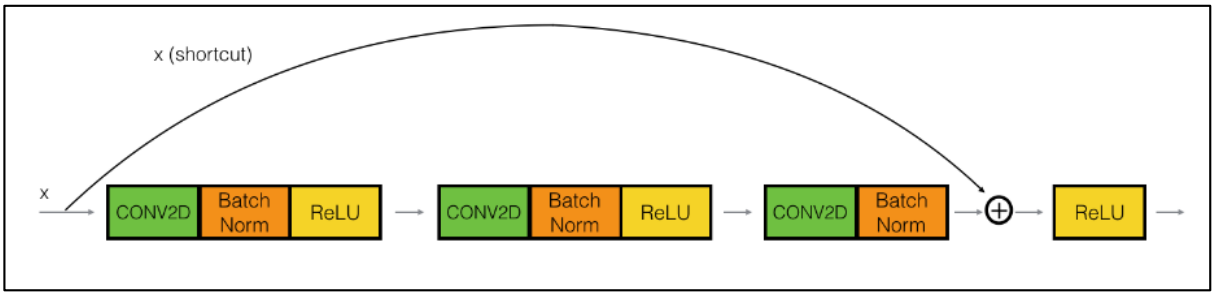
[그림2] ResNet20 model Architecture

위 그림은 구현한 ResNet20 모델의 구조이며 input으로 32 x 32 이미지가 들어온다. Zero padding이 진행된 후 convolution연산을 진행하고 Batch Normalization을 통해 각 미니배치의 입력 데이터의 분포를 정규화하여 학습 과정을 안정화 시킨다. 각 입력 특성의 분포가 평균이 0이고 분산이 1이 되도록 조정한다. 그런 후 활성화 함수인 Relu를 통해 입력값이 0보다 큰 경우에는 그 값을 그대로 출력하고, 0 이하인 경우에는 0을 출력하도록 한다. 그리고 Max Pooling을 하여 사이즈를 줄여준다. 여기서는 3 x 3 filter를 사용하여 stride 2만큼 Pooling을 진행하였다. 그 이후에는 CONV Block과 ID Block이 3번 반복된다. 그리고 Average Pooling을 진행한 후 결과 값을 Flattening을 하여 Fully Connected layer에 넣어 output을 얻는다.



[그림3] Convolutional Block

CONV Block은 ResNet의 기본 구성 요소 중 하나로, 주어진 입력을 더 작은 차원으로 변환하는 convolution 연산을 포함한다. 구성으로는 3 x 3 크기의 필터로 구성된 convolution layer가 있고 Batch Normalizaion layer와 활성화 함수인 ReLU layer가 있다.



[그림4] Identity Block

ID Block은 ResNet의 또 다른 구성 요소 중 하나로, 입력을 그대로 출력하는 역할을 한다. 구성으로는 3 x 3 크기의 필터로 구성된 convolution layer가 있고 Batch Normalizaion layer와 활성화 함수인 ReLU layer가 있다. Identity block은 입력과 출력의 차원이 동일하므로 residual learning을 가능하게 한다.

Convolutional block과 Identity block을 조합하여 ResNet은 깊은 신경망을 구성하며, residual learning을 통해 vanishing gradient 문제를 완화하고 더 나은 학습 성능을 얻을 수 있다.

layer name	output size	20-layer
conv1	16 x 16	3 x 3, 64, stride 2
conv2_x	8 x 8	$\begin{matrix}3\times3 & 128\\ 3\times3 & 128\end{matrix}$ X 2
conv3_x	4 x 4	$\begin{matrix}3\times3 & 256\\ 3\times3 & 256\end{matrix}$ X 2
conv4_x	2 x 2	$\begin{matrix}3\times3 & 512\\ 3\times3 & 512\end{matrix}$ X 2
	1 x 1	Average pool, 1000-d fc, softmax

[표1] ResNet20 layers

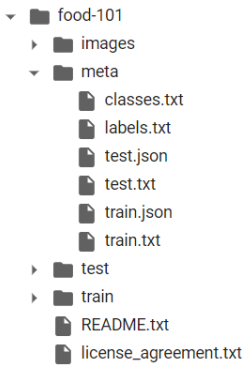
ResNet20 구현 시 각 layer 정보들이다. Input 사이즈는 (32, 32, 3)로 설정하였다. 학습을 시킬 때 input 사이즈가 크면 학습속도가 매우 느려지기 때문이다. Conv1에서는 conv2d layer와 max pooling을 통해 (16, 16, 64) 사이즈의 output을 낸다. Con2_v는 (8, 8, 128), Con3_v는 (4, 4, 256), Con4_v는 (2, 2, 512)의 output을 가진다. 마지막 layer에서는 flattening을 하여 Dense레이어에 최종적으로 101개의 클래스에 대한 확률을 출력한다.

4. 소스코드

제공된 CNN 예제 코드

CNN을 사용하여 food-101 data를 classification하는 코드이다. prepare_date 함수를 사용하여 학습 세트와 테스트 세트를 준비하여 데이터를 전처리하고 MobileNetV2 모델을 사용하여 전이 학습을 수행하는 CNN 모델을 구축한다. MobileNetV2는 사전 훈련된 가중치를 사용하며, 마지막 레이어를 수정하여 원하는 클래스 수에 맞게 조정한다. 데이터 generator를 사용하여 학습 및 검증 데이터를 생성하고 이를 사용하여 모델을 훈련한다. predict_class함수를 사용하여 새로운 이미지에 대한 클래스 예측을 수행한다. 이 예제 소스코드를 활용하여 ResNet20 모델을 구축한다.

```
import os
import urllib.request
import tarfile

# download datasets as a tar.gz file
url = 'http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz'
filename = 'food-101.tar.gz'
urllib.request.urlretrieve(url, filename)

# Decompression
tar = tarfile.open(filename, 'r:gz')
tar.extractall()
tar.close()

# Delete compressed files
os.remove(filename)
```



Food-101 데이터 셋을 Kaggle에서 다운 받아서 사용해도 되지만 upload하는데 오래 걸리므로 url을 이용하여 tar.gz로 다운받고 압축을 풀어 사용한다. 압축을 풀면 food-101 폴더가 생기며 내부 파일과 폴더는 위 그림과 같다.

```
import os
# Controls the output of the TensorFlow log
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# Import Keras Module
import keras.backend as K
from keras.layers import Dense, Dropout
from keras.layers import GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, CSVLogger, ReduceLROnPlateau
from keras.optimizers import SGD

# Importing modules for copying files and other operations
from shutil import copy
from collections import defaultdict
import os
```

모델 학습에 필요한 Keras 모듈을 import한다.

```
# Function to prepare training set and test set
def prepare_data(filepath, src, dest):
    # Create a dictionary to store the images for each class
    classes_images = defaultdict(list)
    with open(filepath, 'r') as txt:
        # Read the file paths and store them in a list
        paths = [read.strip() for read in txt.readlines()]
        for p in paths:
            # Split the path into class and image name
            food = p.split('/')
            # Append the image name with '.jpg' extension to the corresponding class in the dictionary
            classes_images[food[0]].append(food[1] + '.jpg')

    for food in classes_images.keys():
        print("\nCopying images into", food)
        if not os.path.exists(os.path.join(dest, food)):
            # Create a directory for the class if it doesn't exist
            os.makedirs(os.path.join(dest, food))
        for i in classes_images[food]:
            # Copy the images from the source directory to the destination directory
            copy(os.path.join(src, food, i), os.path.join(dest, food, i))
    print("Copying Done!")

# Prepare the training and test sets
prepare_data('food-101/meta/train.txt', 'food-101/images', 'food-101/train')
prepare_data('food-101/meta/test.txt', 'food-101/images', 'food-101/test')
```

Training set과 Test set data를 준비하는 함수로 filepath로 전달된 텍스트 파일에서 이미지 경로를 읽어서 'src' 폴더에서 'dest' 폴더로 이미지를 복사한다.

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, Add, Input, MaxPooling2D, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
```

```
# Define the identity block of ResNet
def identity_block(X, filters, kernel_size):
    # Store the input value to add it later to the output
    X_shortcut = X

    # Convolution layer, Batch Normalization layer, Activation function layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    # Convolution layer, Batch Normalization layer, Activation function layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    # Convolution layer, Batch Normalization layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)

    # Add the output of the shortcut path to the main path
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
```

ResNet 블록의 핵심인 Identity Block을 정의한다. 주어진 입력 X에 Conv2D, BatchNormalization, Activation 등의 레이어를 적용하여 출력을 생성한다.

```
# Define the convolutional block of ResNet
def convolutional_block(X, filters, kernel_size):
    # Store the input value to add it later to the output
    X_shortcut = X

    # Convolution layer, Batch Normalization layer, Activation function layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    # Convolution layer, Batch Normalization layer, Activation function layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    # Convolution layer, Batch Normalization layer
    X = Conv2D(filters, kernel_size, padding='same')(X)
    X = BatchNormalization()(X)

    # Shortcut connection - Perform convolution on the shortcut path
    X_shortcut = Conv2D(filters, kernel_size, padding='same')(X_shortcut)
    X_shortcut = BatchNormalization()(X_shortcut)

    # Add the output of the shortcut path to the main path
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
```

ResNet 블록의 핵심인 Convolutional Block을 정의한다. 주어진 입력 X에 Conv2D, BatchNormalization, Activation function 등의 레이어를 적용하여 출력을 생성한다.

```
# Define the ResNet20 model
def ResNet20(input_shape=(32, 32, 3), classes=101):
    # Input tensor
    X_input = Input(input_shape)
    X = X_input

    # Convolution layer, Batch Normalization layer, Activation function layer, Max pooling layer
    X = Conv2D(64, (3, 3), padding='same')(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)
    X = MaxPooling2D(2, 2, padding='same')(X) # output size: (16, 16, 64)

    # convolutional_block, identity_block, Max pooling layer
    X = convolutional_block(X, 128, (3,3))
    X = identity_block(X, 128, (3,3))
    X = MaxPooling2D(2, 2, padding='same')(X) # output size: (8, 8, 128)

    # convolutional_block, identity_block, Max pooling layer
    X = convolutional_block(X, 256, (3,3))
    X = identity_block(X, 256, (3,3))
    X = MaxPooling2D(2, 2, padding='same')(X) # output size: (4, 4, 256)

    # convolutional_block, identity_block, Max pooling layer
    X = convolutional_block(X, 512, (3,3))
    X = identity_block(X, 512, (3,3))
    X = MaxPooling2D(2, 2, padding='same')(X) # output size: (2, 2, 512)

    # global average pooling
    X = GlobalAveragePooling2D()(X)
    # Fully connected layer with softmax activation
    X = Dense(classes, activation='softmax')(X) # output size: (1, 1)

    # Create the model
    model = Model(inputs=X_input, outputs=X, name='ResNet20')

    return model
```

ResNet-20 모델을 정의하는 함수다. 주어진 입력 모양에 맞게 Convolutional Block과 Identity Block을 적용하고, 최종적으로 global average pooling과 fully connected layer를 추가한다. Input size는 (32, 32, 3)이며, classes는 카테고리가 101개이므로 101로 설정하였다. Convolution 연산 시 3 x 3 filter로 연산하고 입력 이미지에 padding을 적용하여 출력 특성 맵의 공간적인 차원이 입력 이미지와 동일하게 유지되도록 한다. 그 후 배치 정규화와 활성화 함수 relu를 사용한 후 Max pooling을 진행한다. 그 후에는 convolutional block과 identity block과 max pooling을 반복한다. 마지막으로 global average pooling을 사용하여 평균값으로 pooling을 하고 softmax 활성화 함수를 사용하여 101개의 클래스에 대한 확률을 구한다.

```
K.clear_session()

n_classes = 101 # Number of output classes
img_width, img_height = 32, 32 # Input image dimensions
train_data_dir = 'food-101/train' # Directory for training data
validation_data_dir = 'food-101/test' # Directory for validation data
nb_train_samples = 75750 # Number of training samples
nb_validation_samples = 25250 # Number of validation samples
batch_size = 32 # Batch size for training

# Data augmentation and preprocessing for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=10,  # Image rotation
    width_shift_range=0.1,  # Horizontal image shift
    height_shift_range=0.1,  # Vertical image shift
    fill_mode='nearest'  # Image filling mode
    )

# Data preprocessing for validation
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

101개의 class로 분류할 것이고 input 이미지의 차원은 32 x 32 이며, 학습 데이터 샘플의 수는 75,750, 검증 데이터 샘플의 수는 25,250이다. 학습 배치 사이즈는 32로 설정하였다. 학습을 위한 파라미터로 rescale은 1/255로 설정하였고, shear_range는 이미지를 변형할 때 전단 변환을 적용하는 범위를 지정하는 파라미터이며, zoom_range는 이미지를 확대 또는 축소하는 범위를 지정한다. horizontal_filip은 이미지를 수평으로 뒤집는 뒤집기 변환을 적용하는 것을 의미하며, True로 설정하여 학습 데이터의 이미지가 무작위로 수평으로 뒤집힐 수 있도록 한다. rotation_range, width_shift_range, height_shift_range, fill_mode 등의 파라미터를 사용하여 이미지를 회전하거나 수평, 수직으로 이동시키거나 이미지 변환 시에 새롭게 생성된 픽셀을 가장 가까운 픽셀의 값을 사용하여 채우도록 한다. 이와 같이 데이터 증강 관련 매개변수들을 설정하여 모델 성능을 향상시킬 수 있도록 한다.

```
# Generate training data batches
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

# Generate validation data batches
validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
```

Training data batch와 validation data batch를 생성하는 디렉토리에서 이미지를 로드하고 크기를 조정하고, one-hot encoding레이블을 생성한다.

```
# Create ResNet20 model
model = ResNet20(input_shape=(img_height, img_width, 3), classes=n_classes)

# Compile the model with optimizer, loss function, and metrics
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define callbacks for saving the best model, logging, and learning rate scheduling
checkpointer = ModelCheckpoint(filepath='best_model_resnet20.hdf5', verbose=1, save_best_only=True)
csv_logger = CSVLogger('history_resnet20.log')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1, min_lr=0.00001)  # Learning rate scheduling

# Train the model
history = model.fit(train_generator,
                            steps_per_epoch=nb_train_samples // batch_size,
                            validation_data=validation_generator,
                            validation_steps=nb_validation_samples // batch_size,
                            epochs=20,
                            verbose=1,
                            callbacks=[csv_logger, checkpointer, reduce_lr])

# Save the trained model
model.save('model_trained_resnet20.h5')
```

ResNet-20 모델을 생성하고 optimizer는 adam을 사용하여 loss function으로는 categorical_crossentropy를 사용한다. 평가 지표는 accuracy로 한다. 모델의 체크 포인트, csv로거, learning rate 감소 콜백을 설정한다. 모델을 학습시키며 generator에서 생성된 배치를 사용하고 epoch는 20으로 모델 학습을 진행한다. 마지막에는 모델을 파일로 저장한다.

```
%matplotlib inline
import matplotlib.pyplot as plt

# Create subplots for loss and accuracy
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

# Plot training and validation loss
loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

# Plot training and validation accuracy
acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

# Set labels for the axes
loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

# Set legends for the plots
loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

# Display the plot
plt.show()
```

Matplotlib을 사용하여 loss와 accuracy의 그래프를 그리고 출력하는 코드이다. train_loss, train_acc, val_loss, val_acc를 그려 하나의 그래프로 볼 수 있다. 가로축은 epoch이고 세로 왼쪽 축은 loss, 세로 오른쪽 축은 accuracy이다.

Forward propagation

입력된 이미지는 train_generator와 validation_generator를 통해 배치 단위로 제공이 되며, 입력 이미지는 ResNet20 모델의 입력층 X_input으로 전달된다. 입력된 이미지는 Con2D, BatchNormalization, Activation, MaxPooling2D 등의 layer를 거쳐 feature map을 추출하고 줄여나간다. 마지막으로 GlobalAveragePooling2D를 통해 feature map을 평균화한다. 평균화된 특성 맵은 Dense layer를 거쳐 각 클래스에 대한 확률값을 출력한다.

Back propagation

모델의 손실 함수로 categorical_crossentropy를 사용하여 손실함수의 값을 최소화하기 위해 Adam 옵티마이저를 사용하여 가중치를 업데이트 한다. 역전파 알고리즘을 통해 기울기를 계산하고 이를 사용하여 각 레이어의 가중치를 업데이트한다. 이 과정은 주어진 epoch수인 20번 반복되며 fit 함수를 통해 수행된다. 훈련 중에는 steps_per_epoch 및 validation_steps 매개변수에 지정된 수만큼 배치를 통해 훈련 및 검증이 수행된다. 훈련 중에는 csv_logger, checkpointer, reduce_lr 등의 콜백이 사용되어 모델의 훈련 과정을 기록하고 최적의 모델을 저장하며 학습률을 조정한다.

Image is transformed when forwarding

입력된 이미지는 ResNet20 모델을 거치며 convolution연산을 통해 feature map을 추출하게 되고 max pooling을 통해 값이 큰 값을 뽑아 행렬 연산을 진행한다.
이미지 전처리와 데이터 증강은 train_datagen 및 test_datagen을 통해 이루어지며, forwarding될 때 아래과 같은 변형이 적용된다.

Rescaling (정규화): train_datagen 및 test_datagen에서 rescale=1. / 255로 설정하였고, 입력 이미지의 픽셀값은 [0, 255] 범위에서 [0, 1] 범위로 정규화된다.
Shear Range (전단 변환): train_datagen에서 shear_range=0.2로 설정하였고, 이미지는 최대 0.2 라디안까지 전단 변환될 수 있다. 전단 변환은 이미지를 수평 방향으로 비틀어서 형태를 왜곡시킨다.
Zoom Range (확대/축소 변환): train_datagen에서 zoom_range=0.2로 설정하였고, 이미지는 최대 0.2만큼 확대 또는 축소될 수 있다. 확대 및 축소 변환은 이미지의 크기를 조정한다.
Horizontal Flip (수평 뒤집기): train_datagen에서 horizontal_flip=True로 설정되어 있으므로, 이미지는 50%의 확률로 수평으로 뒤집힐 수 있다.
Rotation Range (회전 변환): train_datagen에서 rotation_range=10으로 설정되어 있으므로, 이미지는 최대 10도까지 회전될 수 있다. 회전 변환은 이미지를 회전시킨다.
Width Shift Range (수평 이동): train_datagen에서 width_shift_range=0.1로 설정되어 있으므로, 이미지는 최대 10%까지 수평으로 이동될 수 있다.
Height Shift Range (수직 이동): train_datagen에서 height_shift_range=0.1로 설정되어 있으므로, 이미지는 최대 10%까지 수직으로 이동될 수 있다.
Fill Mode (이미지 채우기): train_datagen에서 fill_mode='nearest'로 설정되어 있으므로, 이미지 변환 시 발생하는 빈 픽셀은 주변 픽셀로 채워진다.

위의 변환들은 데이터 증강을 통해 학습 데이터의 다양성을 증가시키고, 모델의 일반화 성능을 향상시킨다. 이러한 변환은 학습 데이터에만 적용되며, 검증 데이터는 정규화 (rescale=1. / 255) 이외에는 추가적인 변환을 받지 않는다.

How and Why chose mini-batch, epoch, loss function, optimization function, and so on

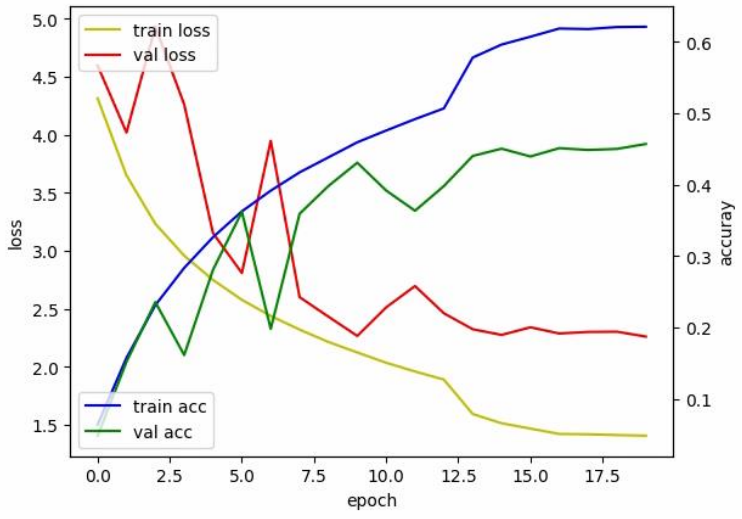
학습 입력 이미지는 32 X 32로 설정하였고, 학습 배치 사이즈는 32로 설정하였고 epoch는 20, loss function은 categorical_crossentropy, optimization function은 adam을 사용하였다. 학습 이미지 픽셀 값과 배치 사이즈를 위와 같이 설정한 이유는 값을 크게 설정하면

학습 시간이 오래 걸리고 또한 메모리 할당 문제로 오류가 나서 사이즈를 줄였다. Loss function인 categorical_crossentropy는 여러 카테고리로 분류하는 문제에 최적화 되어 있으며 이번 과제인 101개의 food 이미지를 분류하는 문제에서 사용하면 좋을 것 같아 손실 함수로 이용하였다. Optimizer로 adam은 성능이 잘 나오는 최적화 함수로 알려져 있으며 모멘텀이 있어 local minimum에 잘 빠지지 않도록 하여 성능이 잘 나오므로 adam을 선택하였다.

5. 실행결과 + PLOT

[first layer에 7 x 7 filter, Max Pooling 추가하고, epoch는 20]

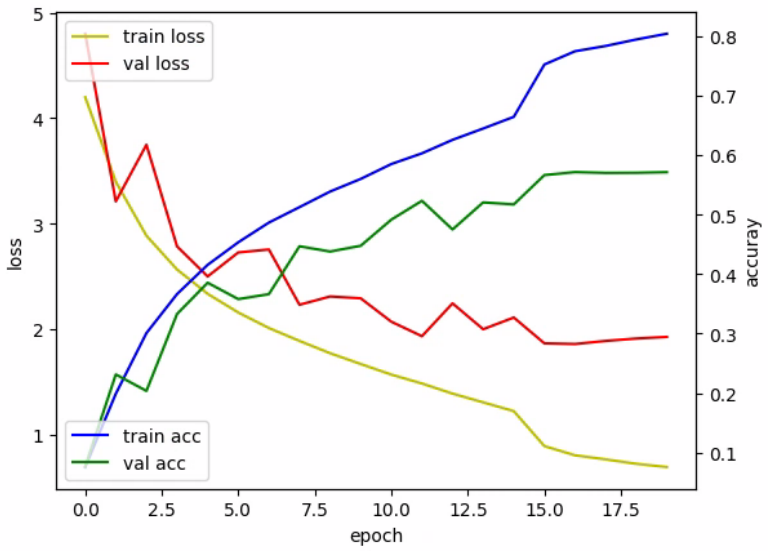
```
Epoch 15/20
2367/2367 [==============================] - ETA: 0s - loss: 1.5127 - accuracy: 0.5959
Epoch 15: val_loss did not improve from 2.26523
2367/2367 [==============================] - 193s 81ms/step - loss: 1.5127 - accuracy: 0.5959 - val_loss: 2.2739 - val_accuracy: 0.4501 - lr: 1.0000e-04
Epoch 16/20
2367/2367 [==============================] - ETA: 0s - loss: 1.4669 - accuracy: 0.6069
Epoch 16: val_loss did not improve from 2.26523

Epoch 16: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
2367/2367 [==============================] - 191s 81ms/step - loss: 1.4669 - accuracy: 0.6069 - val_loss: 2.3396 - val_accuracy: 0.4392 - lr: 1.0000e-04
Epoch 17/20
2367/2367 [==============================] - ETA: 0s - loss: 1.4206 - accuracy: 0.6183
Epoch 17: val_loss did not improve from 2.26523
2367/2367 [==============================] - 194s 82ms/step - loss: 1.4206 - accuracy: 0.6183 - val_loss: 2.2862 - val_accuracy: 0.4508 - lr: 1.0000e-05
Epoch 18/20
2367/2367 [==============================] - ETA: 0s - loss: 1.4179 - accuracy: 0.6176
Epoch 18: val_loss did not improve from 2.26523
2367/2367 [==============================] - 189s 80ms/step - loss: 1.4179 - accuracy: 0.6176 - val_loss: 2.2997 - val_accuracy: 0.4483 - lr: 1.0000e-05
Epoch 19/20
2367/2367 [==============================] - ETA: 0s - loss: 1.4108 - accuracy: 0.6204
Epoch 19: val_loss did not improve from 2.26523

Epoch 19: ReduceLROnPlateau reducing learning rate to 1e-05.
2367/2367 [==============================] - 195s 83ms/step - loss: 1.4108 - accuracy: 0.6204 - val_loss: 2.3015 - val_accuracy: 0.4498 - lr: 1.0000e-05
Epoch 20/20
2367/2367 [==============================] - ETA: 0s - loss: 1.4055 - accuracy: 0.6208
Epoch 20: val_loss improved from 2.26523 to 2.25945, saving model to best_model_resnet20.hdf5
2367/2367 [==============================] - 194s 82ms/step - loss: 1.4055 - accuracy: 0.6208 - val_loss: 2.2595 - val_accuracy: 0.4567 - lr: 1.0000e-05
```



위 결과를 확인해보면 학습 accuracy는 62%가 나왔으며 val_accuracy는 45%가 나왔다. ResNet20의 첫 번째 레이어에 3 x 3 filter로 convolution을 진행하는 것이 아닌 7 x 7로 filter 사이즈를 설정하여 진행한 결과이다. 또한 첫 번째 layer에 Max pooling 추가로 쌓았으며 epoch는 20이다.

[first layer에 3 x 3 filter를 적용하고 rotation_range, width_shift_range, height_shift_range, fill_model 파라미터를 추가, epoch는 20]

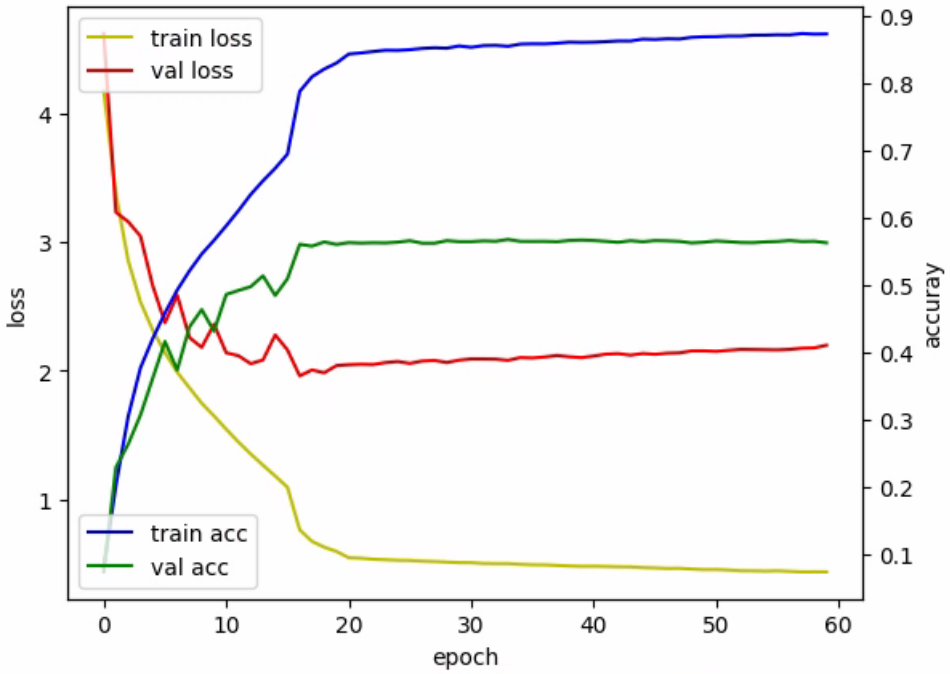
```
Epoch 15: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
2367/2367 [==============================] - 193s 82ms/step - loss: 1.2226 - accuracy: 0.6643 - val_loss: 2.1101 - val_accuracy: 0.5170 - lr: 0.0010
Epoch 16/20
2367/2367 [==============================] - ETA: 0s - loss: 0.8910 - accuracy: 0.7523
Epoch 16: val_loss improved from 1.93289 to 1.86527, saving model to best_model_resnet20.hdf5
2367/2367 [==============================] - 197s 83ms/step - loss: 0.8910 - accuracy: 0.7523 - val_loss: 1.8653 - val_accuracy: 0.5664 - lr: 1.0000e-04
Epoch 17/20
2367/2367 [==============================] - ETA: 0s - loss: 0.8036 - accuracy: 0.7746
Epoch 17: val_loss improved from 1.86527 to 1.85881, saving model to best_model_resnet20.hdf5
2367/2367 [==============================] - 196s 83ms/step - loss: 0.8036 - accuracy: 0.7746 - val_loss: 1.8588 - val_accuracy: 0.5714 - lr: 1.0000e-04
Epoch 18/20
2367/2367 [==============================] - ETA: 0s - loss: 0.7650 - accuracy: 0.7834
Epoch 18: val_loss did not improve from 1.85881
2367/2367 [==============================] - 194s 82ms/step - loss: 0.7650 - accuracy: 0.7834 - val_loss: 1.8881 - val_accuracy: 0.5700 - lr: 1.0000e-04
Epoch 19/20
2367/2367 [==============================] - ETA: 0s - loss: 0.7230 - accuracy: 0.7944
Epoch 19: val_loss did not improve from 1.85881
2367/2367 [==============================] - 195s 82ms/step - loss: 0.7230 - accuracy: 0.7944 - val_loss: 1.9109 - val_accuracy: 0.5702 - lr: 1.0000e-04
Epoch 20/20
2367/2367 [==============================] - ETA: 0s - loss: 0.6921 - accuracy: 0.8040
Epoch 20: val_loss did not improve from 1.85881

Epoch 20: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
2367/2367 [==============================] - 194s 82ms/step - loss: 0.6921 - accuracy: 0.8040 - val_loss: 1.9262 - val_accuracy: 0.5713 - lr: 1.0000e-04
```



위 결과를 확인해보면 학습 accuracy는 80%가 나왔으며 val_accuracy는 57%가 나왔다. ResNet20의 첫 번째 레이어에 3 x 3 filter로 convolution을 진행했으며 epoch는 20이다. rotation_range, width_shift_range, height_shift_range, fill_model 파라미터를 추가하여 데이터 증강 및 변형을 통해 조금씩 다른 이미지를 학습을 시켰으며 이러한 과정을 통해 모델의 학습이 더 잘 이루어지는 효과를 가져오게 했으며 학습 성능을 개선시켰다.

[first layer에 3 x 3 filter를 적용하고 rotation_range, width_shift_range, height_shift_range, fill_model 파라미터를 추가, epoch는 60]

```
Epoch 55: val_loss did not improve from 1.95822
2367/2367 [==============================] - 194s 82ms/step - loss: 0.4413 - accuracy: 0.8714 - val_loss: 2.1614 - val_accuracy: 0.5642 - lr: 1.0000e-05
Epoch 56/60
2367/2367 [==============================] - ETA: 0s - loss: 0.4433 - accuracy: 0.8719
Epoch 56: val_loss did not improve from 1.95822
2367/2367 [==============================] - 192s 81ms/step - loss: 0.4433 - accuracy: 0.8719 - val_loss: 2.1600 - val_accuracy: 0.5648 - lr: 1.0000e-05
Epoch 57/60
2367/2367 [==============================] - ETA: 0s - loss: 0.4393 - accuracy: 0.8717
Epoch 57: val_loss did not improve from 1.95822
2367/2367 [==============================] - 194s 82ms/step - loss: 0.4393 - accuracy: 0.8717 - val_loss: 2.1640 - val_accuracy: 0.5662 - lr: 1.0000e-05
Epoch 58/60
2367/2367 [==============================] - ETA: 0s - loss: 0.4353 - accuracy: 0.8738
Epoch 58: val_loss did not improve from 1.95822
2367/2367 [==============================] - 194s 82ms/step - loss: 0.4353 - accuracy: 0.8738 - val_loss: 2.1732 - val_accuracy: 0.5647 - lr: 1.0000e-05
Epoch 59/60
2367/2367 [==============================] - ETA: 0s - loss: 0.4361 - accuracy: 0.8729
Epoch 59: val_loss did not improve from 1.95822
2367/2367 [==============================] - 193s 82ms/step - loss: 0.4361 - accuracy: 0.8729 - val_loss: 2.1761 - val_accuracy: 0.5651 - lr: 1.0000e-05
Epoch 60/60
2367/2367 [==============================] - ETA: 0s - loss: 0.4353 - accuracy: 0.8732
Epoch 60: val_loss did not improve from 1.95822
2367/2367 [==============================] - 193s 82ms/step - loss: 0.4353 - accuracy: 0.8732 - val_loss: 2.1961 - val_accuracy: 0.5629 - lr: 1.0000e-05
```



위 결과를 확인해보면 학습 accuracy는 87%가 나왔으며 val_accuracy는 56%가 나왔다. ResNet20의 첫 번째 레이어에 3 x 3 filter로 convolution을 진행했으며 epoch는 60이다. rotation_range, width_shift_range, height_shift_range, fill_model 파라미터를 추가하여 데이터 증강 및 변형을 통해 조금씩 다른 이미지를 학습을 시켰으며 이러한 과정을 통해 모델의 학습이 더 잘 이루어지는 효과를 가져오게 했으며 학습 성능을 개선시켰다.

6. 참고문헌

Food101 데이터셋을 이용한 음식 이미지 분류기 만들기 /
https://velog.io/@vector13/Food101-%EB%8D%B0%EC%9D%B4%ED%84%B0%EC%85%8B%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%9D%8C%EC%8B%9D-%EC%9D%B4%EB%AF%B8%EC%A7%80-%EB%B6%84%EB%A5%98%EA%B8%B0-%EB%A7%8C%EB%93%A4%EA%B8%B0

ResNet 라이브러리 사용해보기 / https://codelist.tistory.com/33

Residual Network / https://velog.io/@jaegoo1199/Residual-Networks