인공지능프로그래밍

Lab 04: AlexNet for ImageNet Challenge (ILSVRC)

학 번 : 2021202058

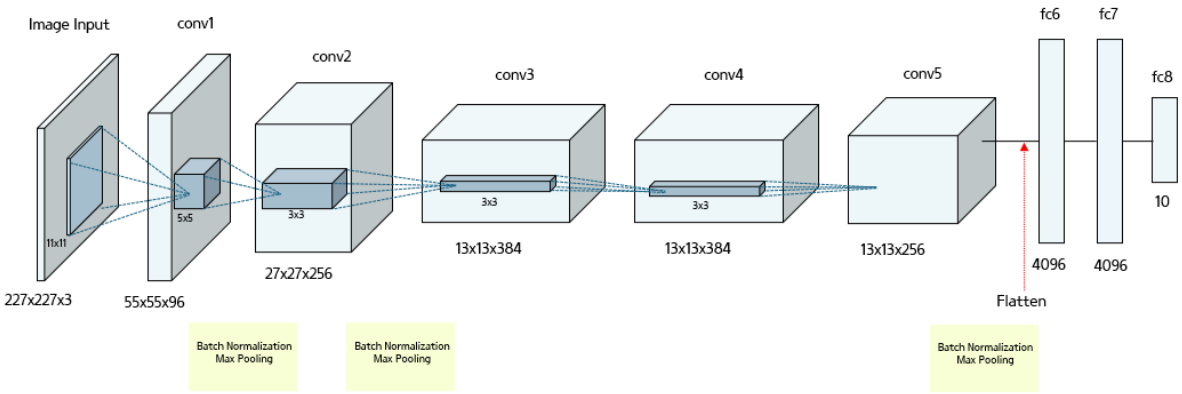
성 명 : 송채영

날 짜 : 2024.10.12

Lab Objective

이번 과제에서는 AlexNet의 구조를 이해하고 구현해보는 것이다. CIFAR10 데이터셋과 imagenette 데이터셋을 활용하여 Image Classification을 진행해본다. Keras의 sequential API를 사용해 AlexNet의 Network를 구현하고 구현한 모델을 바탕으로 학습을 진행한 후 랜덤하게 이미지를 뽑아 실제 이미지와 예측 값이 같은 지 비교해본다.

Program flow



전체적인 코드의 흐름이다. AlexNet은 5개의 convolution layer와 3개의 fully-connected layer, 총 8개의 layer로 구성되어 있다. 먼저 Input으로는 227x227x3, 즉 227x227 사이즈의 RGB 컬러 이미지가 입력된다. 이후 1st layer로 96개의 11x11x3 사이즈 커널로 입력을 convolution한다. 그 후 Batch Normalization과 Max pooling을 적용하였다. 이어서 2nd layer로 256개의 5x5x96 사이즈 커널을 사용하여 전 단계의 특성맵을 convolution한다. 역시 Batch Normalization과 Max pooling을 적용하였다. 3rd layer는 384개의 3x3x256 사이즈 커널을 사용하여 전 단계의 특성맵을 convoulution한다. 4th layer로 384개의 3x3x384 사이즈 커널을 convolution한 후 5th layer로 256개의 3x3x384 사이즈 커널을 convolution한다. 그 후 Batch Normalization과 Max pooling을 적용해주었다. 이후 Flatten을 하여 9216이 되었고, 이를 2번의 Dense layer와 dropout을 적용한 후 마지막 최종 10개의 class로 출력하였다. 자세한 설명은 코드에서 하겠다.

Result

Load Libraries

```
[1]  import os
     # Set the backend of Keras to TensorFlow
     os.environ["KERAS_BACKEND"] = "tensorflow"

     import numpy as np
     import tensorflow as tf
     import tensorflow_datasets as tfds
     import keras
     import matplotlib.pyplot as plt

     # List available GPUs and print the number of GPUs
     gpus = tf.config.list_physical_devices('GPU')
     print("Num GPUs Available: ", len(gpus))
```

```
Num GPUs Available:  1
```

필요한 모듈을 import 하고 사용 가능한 GPU를 확인하는 코드이다. os.environ["KERAS_BACKEND"] = "tensorflow"를 설정해주며 keras의 백엔드를 명시적으로 TensorFlow로 설정한 것이다. 또한 사용가능한 GPU의 개수를 출력하며 1개 사용할 수 있음을 알 수 있다.

Prepare Datasets

```
[2]  dataset = 'cifar10'

     if dataset == 'cifar10':
         # Load the original CIFAR10 dataset
         # CIFAR10 dataset contains 50000 training images and 10000 test images of 32x32x3 pixels
         # Each image contains a small object such as bird, truck, etc...
         (ds_train, ds_test, ds_val), ds_info = tfds.load('cifar10', split=['train[:80%]', 'test', 'train[80%:]'],
                                                          batch_size=None, shuffle_files=True, as_supervised=True,
                                                          with_info=True)
     elif dataset == 'imagenette':
         # Imagenette is a subset of 10 easily classified classes from Imagenet
         # (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute).
         (ds_train, ds_test, ds_val), ds_info = tfds.load('imagenette/320px-v2', split=['train', 'validation[:50%]', 'validation[50%:]'],
                                                          batch_size=None, shuffle_files=True, as_supervised=True,
                                                          with_info=True)
     else:
         print('Dataset Error')

     print(ds_info.features)
     print(ds_info.splits)
     print(ds_info.splits['train'].num_examples)
```

```
FeaturesDict({
    'id': Text(shape=(), dtype=string),
    'image': Image(shape=(32, 32, 3), dtype=uint8),
    'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
})
{'train': <SplitInfo num_examples=50000, num_shards=1>, 'test': <SplitInfo num_examples=10000, num_shards=1>}
50000
```

TensorFlow Datasets를 사용하여 CIFAR-10 데이터셋과 Imagenette 데이터셋을 로드하는 코드이다. CIFAR-10의 경우 데이터셋은 32x32 크기의 3채널 RGB 이미지로 구성되었고 10개의 클래스로 이루어져 있다. 학습 데이터셋의 80%를 train으로, 나머지 20%를 valid로 전체 테스트 데이터셋을 test로 사용함을 설정해주었다. Imagenette은 Imagenet에서 10개의 클래스를 뽑아 쉽게 분류할 수 있도록 만든 데이터셋인데, 데이터 셋의 크기는 320x320 크기의 이미지이다. 나는 CIFER-10을 사용하였으며, 데이터셋의 각 특징, train, test, validation으로 나눈 정보, 학습 데이터셋에 있는 예제가 출력된 것을 확인할 수 있다.

```
[3]  # Get dataset information
     n_channels = ds_info.features['image'].shape[-1]

     if dataset == 'imagenette':
         classes = ['tench', 'English springer', 'cassette player', 'chain saw',
                    'church', 'French horn', 'garbage truck', 'gas pump',
                    'golf ball', 'parachute']
     else:
         classes = ds_info.features['label'].names
     n_classes = ds_info.features['label'].num_classes

     n_train = len(ds_train)
     n_test = len(ds_test)
     n_val = len(ds_val)

     print(n_train,n_test,n_val)
```

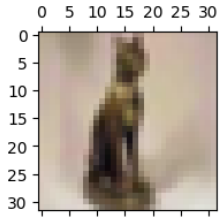
```
40000 10000 10000
```

먼저 채널 수를 가져온 후, 위에서 선택한 데이터셋에 따라 클래스의 이름을 설정해준다. CIFER-10을 선택했으므로 class는 ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']이다. 이후 데이터셋의 크기를 출력하여 확인해보았다. CIFER-10은 학습 데이터 셋이 50,000개의 이미지 이므로 40,000장을 train, 10,000장을 valid, 테스트 데이터 셋 전체인 10,000장을 test로 사용한다.

Show a Sample Data

```
# Select a random image from the training dataset, visualize the image, and print a class label
idx = np.random.randint(n_train-1)

for element in ds_train.skip(idx).take(1):
    image, label = element

print('Image demension:', image.shape, ', label:',label.numpy())

dimage = tf.reshape(image, image.shape)
plt.figure(figsize=(2,2))
plt.matshow(dimage, 1)
plt.show()
print('The picture is', classes[label])
```

```
Image demension: (32, 32, 3) , label: 3
```




```
The picture is cat
```

CIFER-10 데이터셋에서 무작위로 선택된 이미지를 시각화한 후 해당 이미지의 class label을 출력하는 코드이다. Image의 크기로 (32,32,3)이 잘 출력된 것을 알 수 있다.

Building Input Data Pipelines

```
[5]  # Resize the image and normalize it, and convert the label to a one-hot encoded format
     def tfds_4_NET(image, label):

         image = tf.image.resize((image / 255), [227,227], method='bilinear')
         label = tf.one_hot(label, n_classes)

         return image, label
```

```
[6]  # Prepare the training, validation, and test datasets and batching them into batches of size 64
     n_batch = 64

     dataset = ds_train.map(tfds_4_NET, num_parallel_calls=tf.data.AUTOTUNE)
     dataset = dataset.shuffle(buffer_size = 256).batch(batch_size=n_batch)

     valiset = ds_val.map(tfds_4_NET, num_parallel_calls=tf.data.AUTOTUNE)
     valiset = valiset.shuffle(buffer_size = 256).batch(batch_size=n_batch)

     testset = ds_test.map(tfds_4_NET, num_parallel_calls=tf.data.AUTOTUNE)
     testset = testset.shuffle(buffer_size = 256).batch(batch_size=n_batch)
```

TensorFlow Dataset에서 불러온 이미지의 크기를 조정하고 정규화한후, label을 one-hot 인코딩 형식으로 변환하여 train, test, validation 데이터셋을 만드는 과정의 코드이다.

Network Definition of AlexNet with Keras Sequential API

```
# Define the AlexNet model
AlexNet = keras.Sequential([
    ### START CODE HERE ###

    # Input Layer
    keras.layers.InputLayer(shape=(227,227,3)), # Input layer for 227x227 RGB images

    # Layer 1
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=4, activation='relu'), # Convolution with 96 filters, ReLU activation
    keras.layers.BatchNormalization(),                                                  # Normalize activations
    keras.layers.MaxPool2D(pool_size=(3,3), strides=2),                                 # Downsampling the feature maps

    # Layer 2
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=1, activation='relu', padding='same'),   # Convolution with 256 filters, ReLU activation, same padding
    keras.layers.BatchNormalization(),                                                                   # Normalize activations
    keras.layers.MaxPool2D(pool_size=(3,3), strides=2),                                                  # Downsampling

    # Layer 3 to 5
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=1, activation='relu', padding='same'),  # First convolutional layer
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=1, activation='relu', padding='same'),  # Second convolutional layer
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=1, activation='relu', padding='same'),  # Third convolutional layer
    keras.layers.BatchNormalization(),                                                                  # Normalize activations
    keras.layers.MaxPool2D(pool_size=(3,3), strides=2),                                                 # Downsampling

    keras.layers.Flatten(), # Flatten the output

    # Layer 6 to 8
    keras.layers.Dense(4096, activation='relu'),                                                        # First dense layer
    keras.layers.Dropout(0.5),                                                                          # Dropout for regularization
    keras.layers.Dense(4096, activation='relu'),                                                        # Second dense layer
    keras.layers.Dropout(0.5),                                                                          # Dropout for regularization
    keras.layers.Dense(10, activation='softmax')                                                        # Output layer

    ### END CODE HERE ###
])

AlexNet.summary()
```

AlexNet 모델을 TensorFlow의 keras API를 사용하여 정의한 후 모델의 구조를 출력하는 코드이다. keras의 sequential 모델을 사용하여 AlexNet을 정의하였다. 입력 레이어는 입력 이미지의 크기와 채널 수를 정의하였다. 227x227 픽셀 크기와 3채널 RGB를 의미한다. 이어서 Layer 1은 convolution layer, Batch Normalization, MaxPool2D로 이루어져있다. 먼저 96개의 필터를 가진 합성곱 층으로 커널 크기는 11x11이고 stride는 4이다. 활성화 함수로는 ReLU가 사용되었다. 이어서 Batch Normalization을 적용한 후 3x3 크기의 Max pooling을 통해 피처 맵의 차원을 줄였다. Layer 2는 convolution layer, Batch

Normalization, MaxPool2D로 이루어져있다. 256개의 필터를 가진 합성곱 층으로 커널 크기는 5x5, stride는 1, 활성화는 ReLU, 패딩은 same으로 설정하였다. 이후는 위와 같다. 3~5 Layer는 세개의 합성곱 층을 연속적으로 쌓았다. 각 합성곱 층은 384, 384, 256개의 필터를 가지며 각 층 후 Batch Normalization과 Max Pooling을 적용하였다. 합성곱 층을 거쳐 나온 출력을 1D 벡터로 변환하는 Flatten을 적용하였다. 6~8 Layer는 Fully Connected layer에 해당하며 두 개의 층이 Dense로 4096개의 뉴런을 가지며 각 층은 ReLU 활성화 함수를 적용하였다. 이후 Dropout을 적용하였는데 50%를 랜덤으로 생략하여 과적합을 방지하였다. 마지막 Dense 층은 클래스 수에 해당하는 10개의 뉴런을 가지며 softmax 활성화 함수를 적용하여 클래스 확률을 출력해주었다.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34,944
batch_normalization (BatchNormalization)	(None, 55, 55, 96)	384
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614,656
batch_normalization_1 (BatchNormalization)	(None, 27, 27, 256)	1,024
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885,120
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884,992
batch_normalization_2 (BatchNormalization)	(None, 13, 13, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37,752,832
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 10)	40,970

Total params: 58,324,746 (222.49 MB)
Trainable params: 58,323,530 (222.49 MB)
Non-trainable params: 1,216 (4.75 KB)

위에서 구현한 모델의 구조를 출력한 결과로, 각 layer의 출력 형태와 파라미터 수가 expected output과 동일한 것을 확인할 수 있다.

Training the Model

```
# Compile AlexNet
opt = keras.optimizers.Adam(learning_rate=0.001)
AlexNet.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['acc'], jit_compile=True)
```

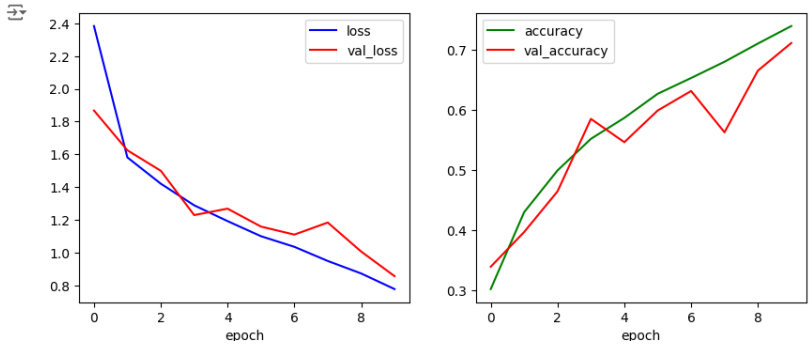
```
[9]  # Using the training dataset, train the AlexNet model
     n_epochs = 10

     results = AlexNet.fit(dataset, epochs=n_epochs, batch_size=n_batch,
                           validation_data=valiset, validation_batch_size=n_batch,
                           verbose=1)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 103s 134ms/step - acc: 0.2403 - loss: 4.3133 - val_acc: 0.3396 - val_loss: 1.8678
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 121s 120ms/step - acc: 0.4127 - loss: 1.6336 - val_acc: 0.3972 - val_loss: 1.6250
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 69s 110ms/step - acc: 0.4802 - loss: 1.4731 - val_acc: 0.4647 - val_loss: 1.5000
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 81s 109ms/step - acc: 0.5399 - loss: 1.3282 - val_acc: 0.5850 - val_loss: 1.2293
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 85s 114ms/step - acc: 0.5794 - loss: 1.2130 - val_acc: 0.5463 - val_loss: 1.2686
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - acc: 0.6220 - loss: 1.1161 - val_acc: 0.5989 - val_loss: 1.1591
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 77s 106ms/step - acc: 0.6418 - loss: 1.0699 - val_acc: 0.6313 - val_loss: 1.1100
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 82s 106ms/step - acc: 0.6705 - loss: 0.9674 - val_acc: 0.5627 - val_loss: 1.1838
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 84s 110ms/step - acc: 0.6994 - loss: 0.9129 - val_acc: 0.6651 - val_loss: 1.0067
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 79s 105ms/step - acc: 0.7327 - loss: 0.7983 - val_acc: 0.7109 - val_loss: 0.8563
```

AlexNet 모델을 컴파일하고 학습시키는 과정의 코드이다. Adam optimizer를 사용했으며 loss 함수는 categorical crossentropy, learning rate는 0.001이다. 모델을 학습한 결과를 살펴보면, 전체적으로 train 정확도는 지속적으로 증가하고 train loss는 감소하는 것을 통해 학습이 잘 되고 있음을 알 수 있다. Validation의 경우 정확도는 처음에는 점점 증가하다가 epoch 8에서 정확도가 살짝 떨어졌지만 나머지 에폭에서 다시 증가하는 추세다.

Plot Convergence Graph

```
[10] # plot loss and accuracy
     plt.figure(figsize=(10,4))
     plt.subplot(1,2,1)
     plt.plot(results.history['loss'], 'b-', label='loss')
     plt.plot(results.history['val_loss'], 'r-', label='val_loss')
     plt.xlabel('epoch')
     plt.legend()

     plt.subplot(1,2,2)
     plt.plot(results.history['acc'], 'g-', label='accuracy')
     plt.plot(results.history['val_acc'], 'r-', label='val_accuracy')
     plt.xlabel('epoch')
     plt.legend()

     plt.show()
```



모델의 훈련 결과를 loss와 정확도를 그래프로 시각화하는 코드이다. Loss plot의 경우, train loss는 지속적으로 감소하고 있으며, validation loss는 초기에는 비슷한 추세를 따르다가 후반부에는 약간 증가하는 것을 확인할 수 있다. 정확도 plot의 경우 train은 약 73%에 도달하며, validation의 경우 34%에서 시작하여 약 71%를 뽑아낸 것을 확인할 수 있다.

Evaluate Model Performance

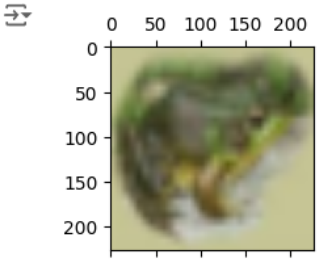
```
[11] # Evaluate the performance of the AlexNet model
     AlexNet.evaluate(testset)
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 7s 43ms/step - acc: 0.7093 - loss: 0.8684
[0.8756893277168274, 0.7081000208854675]
```

Test dataset에서 훈련된 AlexNet 모델의 성능을 평가하는 코드로, 157개의 batch가 있으며 test dataset의 모델 정확도는 70.93%, loss는 0.8684임을 확인할 수 있다. 이어서 출력된 것은 [손실, 정확도] 배열로 최종 평가 지표이다.

Test Model with a Random Sample

```
[12] # Images are randomly selected to compare real class labels with predictions using AlexNet
     idx = np.random.randint(n_test-1)

     for element in ds_test.skip(idx).take(1):
         img, lbl = element
         X_test, y_test = tfds_4_NET(img, lbl)

     X_test = tf.expand_dims(X_test, axis=0)

     dimage = np.array(X_test[0])
     plt.figure(figsize=(2,2))
     plt.matshow(dimage, fignum=1)
     plt.show()

     outt_4 = AlexNet.predict(X_test)
     p_pred = np.argmax(outt_4, axis=-1)

     print('My prediction is ' + classes[p_pred[0]])
     print('Actual image is  ' + classes[tf.argmax(y_test, -1)])
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
My prediction is frog
Actual image is  frog
```

위 코드는 test dataset에서 이미지를 무작위로 선택한 후 AlexNet 모델의 예측 class label을 실제 class label과 비교하는 코드이다. 예측 값과 실제 값이 개구리로 일치하는 것을 확인할 수 있다.

Discussion

이번 과제를 통해 AlexNet 모델에 대해 처음 알게 되었다. 모델을 구현하는데 있어 CNN과 크게 다르지 않아 코드를 구현하는데 있어 어려움은 없었다. 하지만 과제를 진행할 때 계속해서 jupyter notebook을 활용하였는데, tensorflow_dataset 모듈이 import되지 않아 그 부분에 시간을 많이 썼던 것 같다. 결론적으론 버전에 맞게 깔 경우 GPU를 지원하지 않아 결국 이번 과제에서는 구글 코랩을 활용하였다. 또한 맨 처음 conv layer을 제외하고는 모두 padding을 주어 이미지 사이즈를 줄어들지 않게 하였는데, same으로 함으로써 출력 기능 맵이 입력 기능 맵과 동일한 높이 및 너비를 갖도록 보장하기 위해서이다. Conv1에는 padding을 사용하지 않았는데 검색해보니 valid로 해주는 것과 같은 것을 알게 되었다. 여기서 첫 번째 layer에는 적용하지 않는 이유는 첫 layer에서 큰 특징을 캡처하기 위해서인 것 같다. 실제로 AlexNet에서는 다음 layer에서 처리할 공간 특징이 더 적도록 입력 이미지가 더 작은 차원으로 축소하는 것을 바탕으로 생각한 것이다.