Seoul National University
College of Engineering
Department of Naval Architecture and Ocean Engineering
1, Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea

Fall 2020

딥러닝

PA # 4

Instructor name	김태완 교수님
Student name	이용준
Department	조선해양공학과
Student ID	2015-19595
Submission date	2020.11.20

Contents

1. Problem Definition

Class	Label
Airplane	0
Automobile	1
Bird	2
Cat	3
Deer	4
Dog	5
Frog	6
Horse	7
Ship	8
Truck	9

- CIFAR-10 데이터를 VGG Network로 학습시키고, 그 결과를 시각화 및 분석하라.
- VGG 네트워크의 마지막 fully connected layer에 dropout을 추가해보고 그 결과를 비교하라. 성능이 좋아졌거나 나빠졌다면 그 결과와 원인을 분석하라.
- VGG 네트워크의 layer들 사이에 batch normalization을 추가해보고 그 결과를 비교하라. 성능이 좋아졌거나 나빠졌다면 그 결과와 원인을 분석하라.

2. Problem Analysis

1. 본 과제의 수행을 위하여 VGG 모델을 만들어야 한다. 딥러닝을 쉽게 구현할 수 있도록 도와주는 framework인 Tensorflow와 keras를 활용하여 모델을 만들었다.

2. 학습 데이터로 3000개의 이미지와 라벨이, 테스트 데이터로 100개의 이미지와 라벨이 주어졌다. 이 데이터들은 전처리를 거친 후에 코드 상에서 사용할 수 있다. 이미지들을 데이터화(preprocessing)시킬 때에도 keras의 패키지를 이용하였다.

3. 성능 보완을 위해 Dropout 기법과 Batch normalization 기법을 사용해보고, 그 결과를 기존과 비교해본다.

4. 쉽게 학습 결과를 알아볼 수 있도록 console 창에 예측 라벨과 실제 라벨을 출력한다.

3. Code Explanation

프로젝트에 사용된 함수에 관한 설명이다.

vgg_block(in_layer, n_conv, n_filter, filter_size=(3, 3), reduce_size=True)		
Parameter	Type	Explanation
in_layer	ff.keras.layers	Input layer에 대한 정보를 담고 있다.
n_conv	int	한 블록 속 Convolution의 개수
n_filter	int	필터의 개수
filter_size	( )	필터의 사이즈를 뜻하며, 3X3으로 고정한다.
reduce_size	bool	Max pool을 위한 bool값

4. Result & Conclusion

4.1 Result

1번 문제 : VGG Network를 구성하고 3000개의 데이터로 학습, 100개의 데이터로 테스트를 해보았다. 학습은 epoch = 30으로 했다. 학습하는 동안의 loss는 계속 줄어들었기 때문에 정상적으로 학습이 진행되고 있음을 확인하였다. 하지만 test 과정에서의 accuracy는 커졌다 작아졌다 약간의 변동이 있었고 epoch=30의 학습 후의 최종 테스트에서의 정확도는 41%였다.

```
model.fit(x_train, y_train, batch_size=100, epochs=30,
          validation_data=(x_test, y_test))
```

```
Epoch 20/30
30/30 [==============================] - 9s 286ms/step - loss: 0.2147 - accuracy: 0.9243 - val_loss: 3.0519 - val_accuracy: 0.4300
Epoch 21/30
30/30 [==============================] - 9s 285ms/step - loss: 0.1460 - accuracy: 0.9520 - val_loss: 3.4563 - val_accuracy: 0.4800
Epoch 22/30
30/30 [==============================] - 9s 286ms/step - loss: 0.1619 - accuracy: 0.9457 - val_loss: 3.3748 - val_accuracy: 0.4400
Epoch 23/30
30/30 [==============================] - 9s 285ms/step - loss: 0.1179 - accuracy: 0.9650 - val_loss: 4.2943 - val_accuracy: 0.4400
Epoch 24/30
30/30 [==============================] - 9s 289ms/step - loss: 0.0756 - accuracy: 0.9763 - val_loss: 4.0703 - val_accuracy: 0.4300
Epoch 25/30
30/30 [==============================] - 9s 286ms/step - loss: 0.0604 - accuracy: 0.9840 - val_loss: 3.7133 - val_accuracy: 0.5200
Epoch 26/30
30/30 [==============================] - 9s 285ms/step - loss: 0.0991 - accuracy: 0.9707 - val_loss: 3.6934 - val_accuracy: 0.4800
Epoch 27/30
30/30 [==============================] - 9s 290ms/step - loss: 0.1280 - accuracy: 0.9587 - val_loss: 3.4602 - val_accuracy: 0.4700
Epoch 28/30
30/30 [==============================] - 9s 286ms/step - loss: 0.0578 - accuracy: 0.9840 - val_loss: 4.5502 - val_accuracy: 0.5000
Epoch 29/30
30/30 [==============================] - 9s 309ms/step - loss: 0.0435 - accuracy: 0.9870 - val_loss: 4.6268 - val_accuracy: 0.4500
Epoch 30/30
30/30 [==============================] - 10s 325ms/step - loss: 0.0248 - accuracy: 0.9933 - val_loss: 4.8718 - val_accuracy: 0.4100
```

```
predict= 6, label= 6
predict= 7, label= 5
predict= 2, label= 6
predict= 1, label= 1
predict= 9, label= 9
predict= 2, label= 3
predict= 7, label= 8
predict= 0, label= 0
predict= 3, label= 6
predict= 0, label= 8
predict= 1, label= 9
predict= 7, label= 7
predict= 8, label= 8
predict= 6, label= 6
predict= 3, label= 3
predict= 0, label= 0
predict= 8, label= 9
predict= 1, label= 0
predict= 8, label= 8
predict= 1, label= 1
predict= 0, label= 2
predict= 9, label= 3
predict= 3, label= 5
predict= 9, label= 9
predict= 7, label= 2
predict= 9, label= 1
predict= 1, label= 9
predict= 6, label= 3
predict= 8, label= 8
predict= 8, label= 8
predict= 2, label= 6
predict= 6, label= 6
predict= 1, label= 9
predict= 3, label= 7
predict= 2, label= 5
predict= 8, label= 8
predict= 0, label= 4
predict= 2, label= 2
predict= 2, label= 4
predict= 8, label= 7
predict= 9, label= 9
```

위의 사진은 test 결과의 일부분이다.

2번 문제 : 성능 향상을 위해 fully connected layer에 dropout을 적용하였다.

```
flatten = tf.keras.layers.Flatten()(vgg_block03) # 2048
dense01 = tf.keras.layers.Dense(512, activation='relu')(flatten)
dropout = tf.keras.layers.Dropout(rate = 0.2)(dense01) ## dropout
output = tf.keras.layers.Dense(10, activation='softmax')(dropout)
```

```
Epoch 20/30
30/30 [==============================] - 9s 293ms/step - loss: 0.2758 - accuracy: 0.9057 - val_loss: 2.5989 - val_accuracy: 0.4900
Epoch 21/30
30/30 [==============================] - 9s 293ms/step - loss: 0.2325 - accuracy: 0.9213 - val_loss: 3.0703 - val_accuracy: 0.4600
Epoch 22/30
30/30 [==============================] - 9s 290ms/step - loss: 0.1977 - accuracy: 0.9300 - val_loss: 3.7508 - val_accuracy: 0.4400
Epoch 23/30
30/30 [==============================] - 9s 289ms/step - loss: 0.1676 - accuracy: 0.9380 - val_loss: 3.2877 - val_accuracy: 0.4100
Epoch 24/30
30/30 [==============================] - 9s 288ms/step - loss: 0.1161 - accuracy: 0.9607 - val_loss: 3.8585 - val_accuracy: 0.4200
Epoch 25/30
30/30 [==============================] - 9s 290ms/step - loss: 0.1178 - accuracy: 0.9583 - val_loss: 3.5214 - val_accuracy: 0.4900
Epoch 26/30
30/30 [==============================] - 9s 288ms/step - loss: 0.1320 - accuracy: 0.9563 - val_loss: 3.9894 - val_accuracy: 0.4000
Epoch 27/30
30/30 [==============================] - 9s 290ms/step - loss: 0.0980 - accuracy: 0.9670 - val_loss: 4.1548 - val_accuracy: 0.4100
Epoch 28/30
30/30 [==============================] - 9s 287ms/step - loss: 0.1199 - accuracy: 0.9633 - val_loss: 3.6413 - val_accuracy: 0.4600
Epoch 29/30
30/30 [==============================] - 9s 288ms/step - loss: 0.1043 - accuracy: 0.9660 - val_loss: 4.6302 - val_accuracy: 0.4000
Epoch 30/30
30/30 [==============================] - 9s 288ms/step - loss: 0.0906 - accuracy: 0.9710 - val_loss: 4.1864 - val_accuracy: 0.3900
```

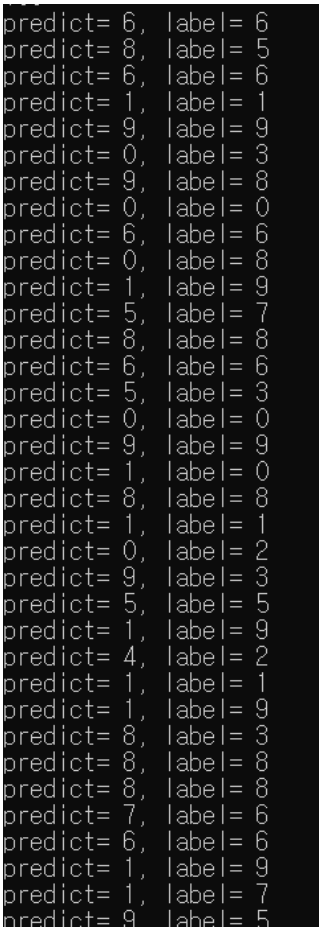
```
predict= 6, label= 6
predict= 0, label= 5
predict= 2, label= 6
predict= 9, label= 1
predict= 9, label= 9
predict= 0, label= 3
predict= 2, label= 8
predict= 0, label= 0
predict= 3, label= 6
predict= 8, label= 8
predict= 1, label= 9
predict= 4, label= 7
predict= 0, label= 8
predict= 3, label= 6
predict= 3, label= 3
predict= 0, label= 0
predict= 8, label= 9
predict= 9, label= 0
predict= 8, label= 8
predict= 1, label= 1
predict= 2, label= 2
predict= 9, label= 3
predict= 5, label= 5
predict= 9, label= 9
predict= 0, label= 2
predict= 1, label= 1
predict= 9, label= 9
predict= 6, label= 3
predict= 8, label= 8
predict= 8, label= 8
predict= 2, label= 6
predict= 3, label= 6
predict= 9, label= 9
predict= 2, label= 7
predict= 3, label= 5
predict= 8, label= 8
predict= 0, label= 4
predict= 1, label= 2
```

Dropout을 적용하여 동일하게 epoch=30으로 학습했다. 성능 향상을 기대하였지만 학습하는 동안 epoch 당 val_accuracy는 40~50%을 웃돌았다. 최종 학습 후의 정확도는 39%로 dropout을 적용하기 전보다 오히려 성능이 감소하였음을 확인했다. Dropout의 목적은 과적합을 막는 것인데 이번 과제에서는 과적합이 뚜렷하게 일어나지 않았기 때문에 별로 도움이 되지 않았다고 생각해볼 수 있다.

3번 문제 : Batch Normalization을 각 layer에 적용하여 학습시켜 보았다.

```
layer = tf.keras.layers.Conv2D(n_filter, filter_size, padding='SAME')(layer)
layer = tf.keras.layers.BatchNormalization()(layer) ##batch normalization
layer = tf.keras.layers.Activation(activation='relu')(layer)
```

```
Epoch 20/30
30/30 [==============================] - 14s 478ms/step - loss: 0.0331 - accuracy: 0.9927 - val_loss: 1.9812 - val_accuracy: 0.5900
Epoch 21/30
30/30 [==============================] - 14s 478ms/step - loss: 0.0301 - accuracy: 0.9913 - val_loss: 2.1358 - val_accuracy: 0.5200
Epoch 22/30
30/30 [==============================] - 15s 483ms/step - loss: 0.0405 - accuracy: 0.9873 - val_loss: 2.0631 - val_accuracy: 0.5500
Epoch 23/30
30/30 [==============================] - 14s 481ms/step - loss: 0.0621 - accuracy: 0.9833 - val_loss: 2.5530 - val_accuracy: 0.4800
Epoch 24/30
30/30 [==============================] - 14s 477ms/step - loss: 0.1237 - accuracy: 0.9597 - val_loss: 3.3832 - val_accuracy: 0.4700
Epoch 25/30
30/30 [==============================] - 14s 474ms/step - loss: 0.1456 - accuracy: 0.9517 - val_loss: 2.8125 - val_accuracy: 0.4700
Epoch 26/30
30/30 [==============================] - 14s 473ms/step - loss: 0.1075 - accuracy: 0.9640 - val_loss: 2.4039 - val_accuracy: 0.5200
Epoch 27/30
30/30 [==============================] - 14s 473ms/step - loss: 0.1539 - accuracy: 0.9470 - val_loss: 1.9703 - val_accuracy: 0.5400
Epoch 28/30
30/30 [==============================] - 14s 474ms/step - loss: 0.0953 - accuracy: 0.9690 - val_loss: 2.1284 - val_accuracy: 0.5700
Epoch 29/30
30/30 [==============================] - 14s 473ms/step - loss: 0.0490 - accuracy: 0.9870 - val_loss: 2.3158 - val_accuracy: 0.5500
Epoch 30/30
30/30 [==============================] - 14s 478ms/step - loss: 0.0715 - accuracy: 0.9767 - val_loss: 2.5601 - val_accuracy: 0.5200
```



Batch Normalization을 추가한 결과, 큰 성능 향상이 있었다고는 보기 어렵지만 최종 학습 후의 정확도가 52%로 추가 전보다 정확도가 소폭 상승(11%p)했음을 확인하였다. Batch Normalization은 정규화를 통해 parameter들의 분포를 고르게 해주어 학습 성과를 올려주었다고 생각해볼 수 있다.

4.2 Conclusion

1. Dropout과 Batch Normalization을 추가했을 때는 성능 향상이 되었음을 확인할 수 있었지만 dropout만 추가하였을 때는 성능이 오히려 소폭 감소하였다.

2. 이번 과제를 통해 이미지파일을 input으로 받을 때 전처리하는 방법에 대해 공부할 수 있었다. 또 VGG 모델을 직접 구성하여 이미지 분류를 해볼 수 있었다. 모델의 층을 다양하게 바꿔보면 더 좋은 성능의 모델을 만들 수 있을 것으로 기대된다.