

REPORT

Kongju National University



인공지능 학기말 프로젝트 (텐서플로 딥러닝)

학과명 : 컴퓨터공학과

교과명 : 인공지능

분 반 : 01분반

학 번 : 201901822

이 름 : 이한결

제출일 : 2023-11-30



공주대학교
KONGJU NATIONAL UNIVERSITY

1. 삼목 게임 학습

서론

<https://github.com/datasets/tic-tac-toe> 사이트의 "tic-tac-toe.csv"의 삼목 게임을 텐서플로를 이용한 딥러닝으로 구현한다.

	A	B	C	D	E	F	G	H	I	J
1	TL	TM	TR	ML	MM	MR	BL	BM	BR	class
2	x	x	x	x	o	o	x	o	o	TRUE
3	x	x	x	x	o	o	o	x	o	TRUE
4	x	x	x	x	o	o	o	o	x	TRUE
5	x	x	x	x	o	o	o	b	b	TRUE
6	x	x	x	x	o	o	b	o	b	TRUE
7	x	x	x	x	o	o	b	b	o	TRUE
8	x	x	x	x	o	b	o	o	b	TRUE
9	x	x	x	x	o	b	o	b	o	TRUE
10	x	x	x	x	o	b	b	o	o	TRUE
11	x	x	x	x	b	o	o	o	b	TRUE
12	x	x	x	x	b	o	o	b	o	TRUE
13	x	x	x	x	b	o	b	o	o	TRUE
14	x	x	x	o	x	o	x	o	o	TRUE
15	x	x	x	o	x	o	o	x	o	TRUE
16	x	x	x	o	x	o	o	o	x	TRUE
17	x	x	x	o	x	o	o	b	b	TRUE
18	x	x	x	o	x	o	b	o	b	TRUE
19	x	x	x	o	x	o	b	b	o	TRUE
20	x	x	x	o	x	b	o	o	b	TRUE
21	x	x	x	o	x	b	o	b	o	TRUE
22	x	x	x	o	x	b	b	o	o	TRUE
23	x	x	x	o	o	x	x	o	o	TRUE
24	x	x	x	o	o	x	o	x	o	TRUE
25	x	x	x	o	o	x	o	o	x	TRUE
26	x	x	x	o	o	x	o	b	b	TRUE
27	x	x	x	o	o	x	b	o	b	TRUE
28	x	x	x	o	o	x	b	b	o	TRUE
29	x	x	x	o	o	b	x	o	b	TRUE
30	x	x	x	o	o	b	x	b	o	TRUE

-> tic-tac-toe.csv

tic-tac-toe.csv은 틱택토의 결과, 총 958번의 게임이 존재한다. (틱택토란 이렇게 생긴 9칸 위에 1P는 O, 2P는 X를 번갈아가며 그리고 먼저 O나 X를 3개가 직선으로 이어지게 만들면 승리하는 게임이다.) TL은 Top Left, TM은 Top Middle, TR은 Top Right. 이런식으로 9칸에 들어가는 O,X가 있다. 마지막에 class에 승패의 결과가 있다. TRUE면 X가 이긴 것, False면 X가 진것이다.

본론

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

텐서플로와 넘파이등 사용할 모듈을 import해준다.

```
def load_tictactoe(shuffle=False):
    label={'o':0, 'x':1, 'b':2}
    result = {'true':1, 'false':0}
    data = np.loadtxt("./Data/tic-tac-toe.csv", skiprows=1, delimiter=',',
                      converters={0: lambda name: label[name.decode()],
                                   1: lambda name: label[name.decode()],
                                   2: lambda name: label[name.decode()],
                                   3: lambda name: label[name.decode()],
                                   4: lambda name: label[name.decode()],
                                   5: lambda name: label[name.decode()],
                                   6: lambda name: label[name.decode()],
                                   7: lambda name: label[name.decode()],
                                   8: lambda name: label[name.decode()],
                                   9: lambda resultname: result[resultname.decode()]})

    if shuffle:
        np.random.shuffle(data)
    return data
```

tic-tac-toe.csv을 가져오는 함수이다. skiprows = 1로 첫 줄은 스킵하고 delimiter =',' 로 항목을 구분하며, tic-tac-toe.csv에서 문자인 o, x, true, false는 텐서플로로 학습을 시키지 못하니 converters을 이용하여 {'o':0, 'x':1, 'b':2} , {'true':1, 'false':0}의 정수 레이블로 변환한다. shuffle이 ture로 데이터 순서를 섞는다.

```
def train_test_data_set(tictactoe_data, test_rate=0.2):
    n = int(tictactoe_data.shape[0]*(1-test_rate))
    x_train = tictactoe_data[:n,:-1]
    y_train = tictactoe_data[:n, -1]

    x_test = tictactoe_data[n:,-1]
    y_test = tictactoe_data[n:,-1]
    return (x_train, y_train), (x_test, y_test)
```

n개의 샘플을 (1-test_rate)비율의 훈련데이터와 (test_rate)의 테스트데이터로 구분한다

디폴트값으로 0.2가 설정되어있다.

```
tictactoe_data = load_tictactoe(shuffle=True)
(x_train, y_train), (x_test, y_test) = train_test_data_set(tictactoe_data, test_rate=0.2)

print("x_train.shape:", x_train.shape)
print("y_train.shape:", y_train.shape)
print("x_test.shape:", x_test.shape)
print("y_test.shape:", y_test.shape)
```

tictactoe_data로 변환한 tic-tac-toe.csv데이터로 순서를 섞어서 로드한다.

확인을 위해 학습데이터와 테스트데이터의 모양을 출력한다.

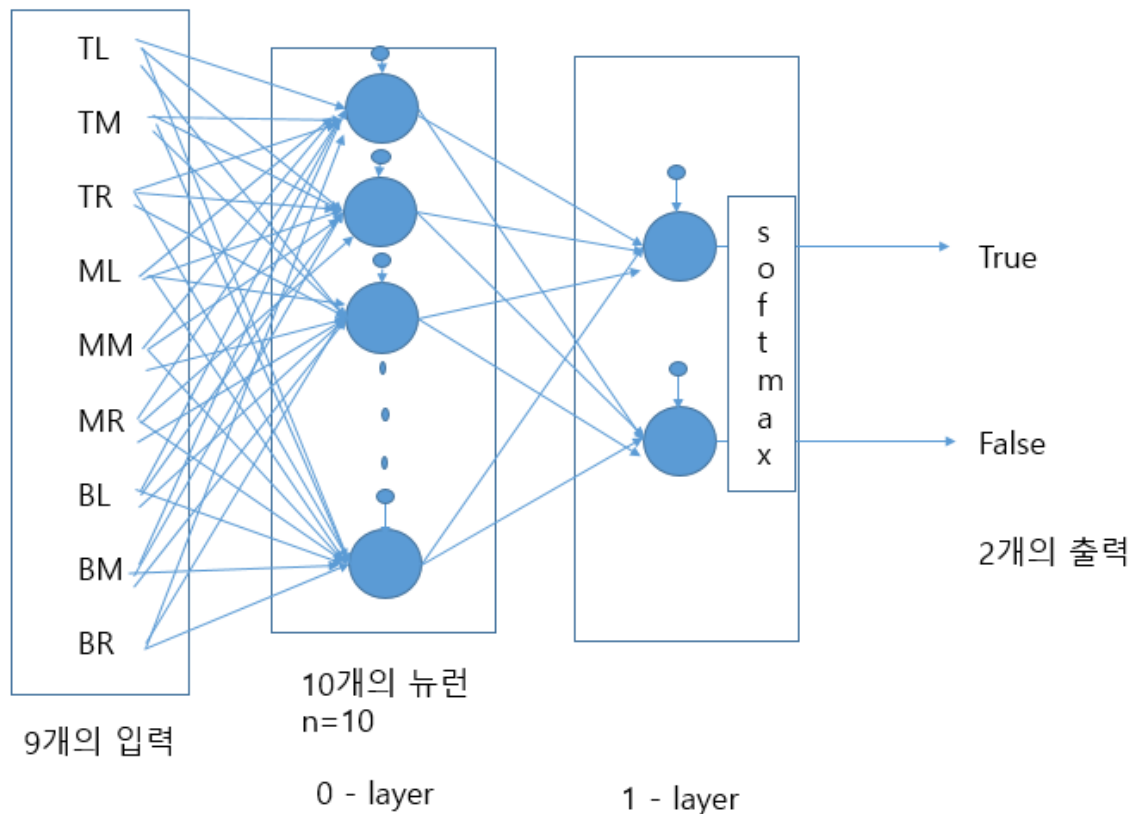
```
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

손실함수를 categorical_crossentropy를 사용할 예정이니 y_train과 y_test를 원-핫 인코딩으로 변환한다.

```
n = 10
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=n, input_dim=9, activation='sigmoid'))
model.add(tf.keras.layers.Dense(units=2, activation='softmax'))
model.summary()

opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
model.compile(optimizer=opt, loss= 'MSE', metrics=['accuracy'])
model.compile(optimizer=opt, loss='categorical_crossentropy',metrics=['accuracy'])
ret = model.fit(x_train, y_train, epochs=100, verbose=0)
```

출력층 활성화함수가 softmax인 신경망을 생성하고 손실함수를 categorical_crossentropy를 설정하여 학습한다.



가중치 0층은 90개(9×10), 1층은 20개(10×2)이며, 바이어스는 각각 10개, 2개이다.

즉, 파라미터는 각각 100개($9 \times 10 + 10$), 22개($10 \times 2 + 2$)로 총 122개이다.

```
print("len(model.layers):", len(model.layers))
loss = ret.history['loss']
plt.plot(loss)
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()
```

모델이 몇 층인지 출력하고 반복횟수를 x축, 손실값을 y축으로 하여 손실값을 보여준다

```
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

학습에서 얻은 모델을 test데이터로 정확도와 손실을 출력하여 평가한다.

```
y_pred = model.predict(x_train)
y_label = np.argmax(y_pred, axis = 1)
C = tf.math.confusion_matrix(np.argmax(y_train, axis = 1), y_label)
print("confusion_matrix(C):", C)
```

x_{train} 를 적용하여 출력 y_{pred} 를 계산하고 큰값위치 찾아서 y_{label} 에 저장한다.

`tf.math.confusion_matrix`로 컨퓨전 행렬 C 를 계산하고 출력한다.

실험 결과

```
===== RESTART: C:\Users\82106\Desktop\과제\3학년 2학기\4. 인공지능\2201.py =====
x_train.shape: (766, 9)
y_train.shape: (766,)
x_test.shape: (192, 9)
y_test.shape: (192,)
Model: "sequential"

Layer (type)                 Output Shape         Param #
=====
dense (Dense)                (None, 10)           100
dense_1 (Dense)              (None, 2)            22
=====
Total params: 122
Trainable params: 122
Non-trainable params: 0

len(model.layers): 2
24/24 - 0s - loss: 0.1845 - accuracy: 0.9321
6/6 - 0s - loss: 0.2701 - accuracy: 0.8646
confusion_matrix(C): tf.Tensor(
[[229  39]
 [ 13 485]], shape=(2, 2), dtype=int32)
```

958개의 샘플을 80%의 훈련데이터(766개) 와 20%의 테스트데이터(192개)로 구분됐다

summary로 보면 sequential모델에 0층은 10개의 뉴런으로 파라미터 100개, 1층은 2개의 뉴런으로 파라미터 22개로 Dense로 연결되어있다.

총 파라미터는 122개이고, 총 2층이다.

evaluate는 학습데이터의 정확도는 93.21%, 테스트데이터의 정확도는 86.46%이다.

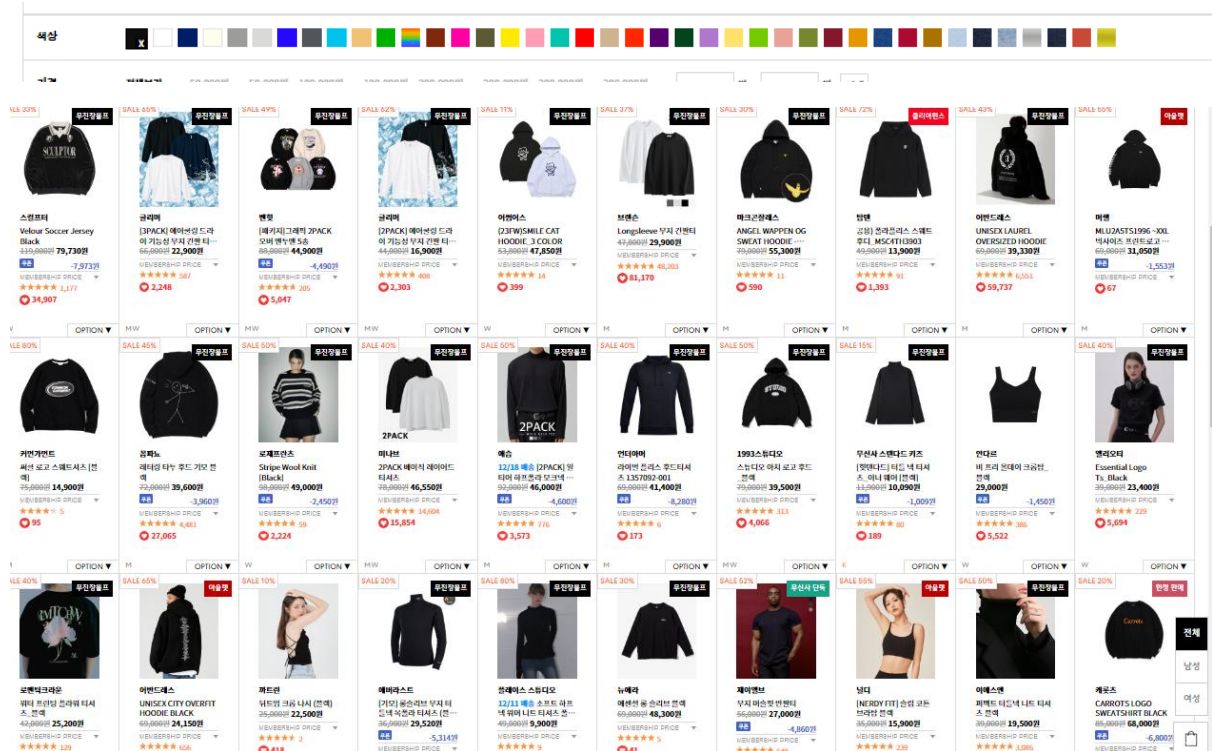
훈련 데이터에서 컨퓨전 행렬C에서 보면 766개중 52개 (39+13)를 잘못 예측했다.

2. 컬러(color) 분류

서론

카메라로 촬영한 컬러(R, G, B, Yellow, Pink 등) 식별, 색상 종류는 각자 정하기

데이터셋 폴더에 무신사사이트를 이용하여 옷의 컬러를 필터를 적용해 옷들을 100장씩 저장한다.

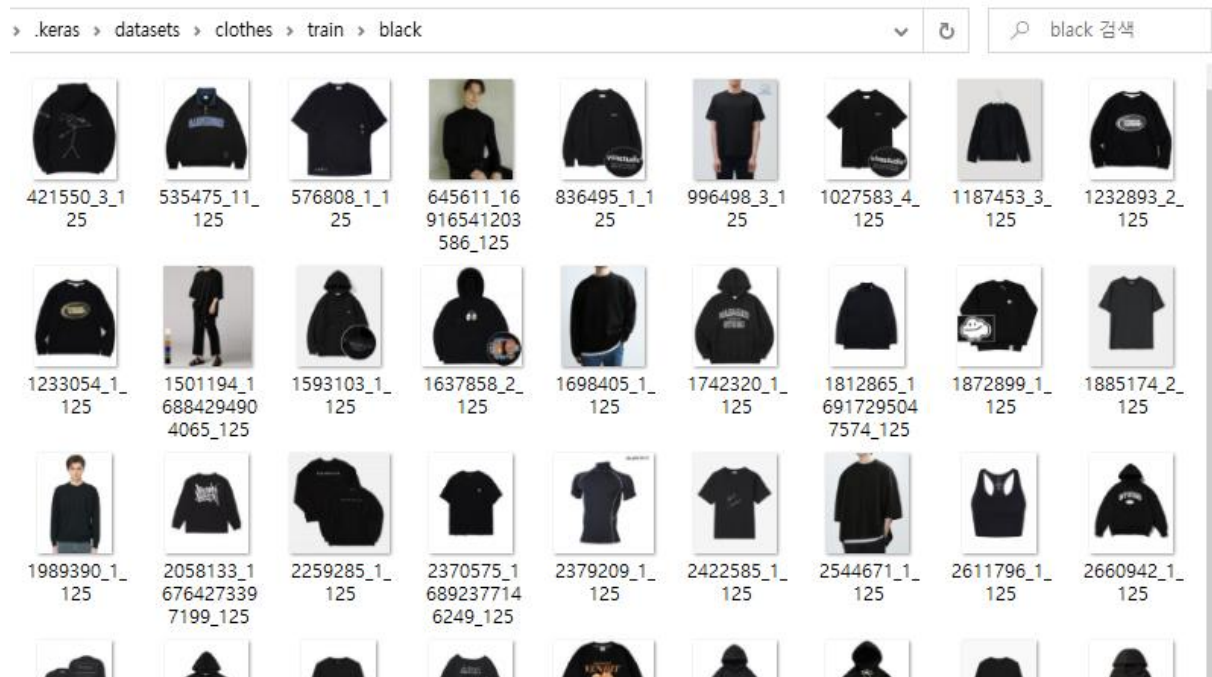


검정색필터를 이용한 검정색 옷 이미지 저장

색 분류는 Black(검정), Red(빨강), Blue(파랑)을 분류해서 각각의 파일에 넣어 학습할 데이터셋을 만들었다.

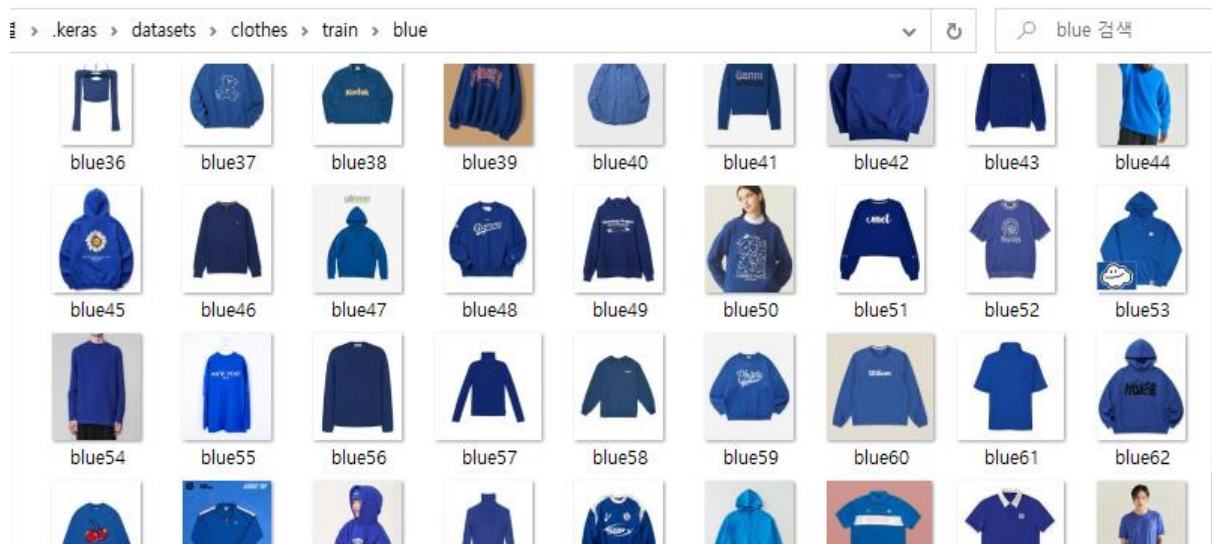
.keras > datasets > clothes > train		
이름	수정한 날짜	유형
black	2023-11-23 오후 7:27	파일 폴더
blue	2023-11-23 오후 7:17	파일 폴더
red	2023-11-23 오후 6:58	파일 폴더

검정색 폴더



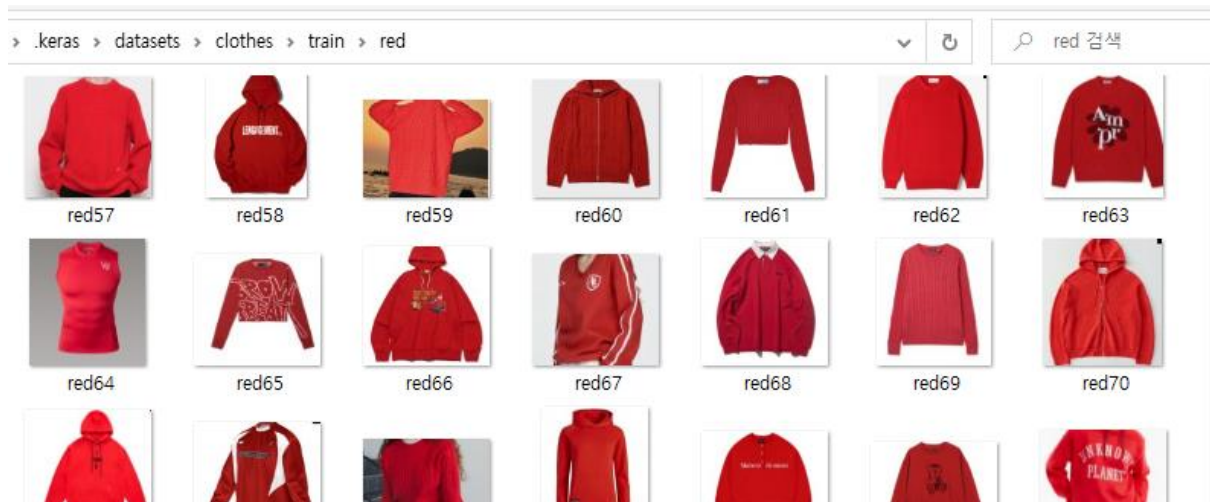
검정색 옷 100개의 사진

파란색 폴더



파란색 옷 100개의 사진

빨강색 폴더



빨강색 옷 100개의 사진

테스트데이터는 직접 촬영한 사진을 이용하였다.

> .keras > datasets > clothes > test			
이름	수정한 날짜	유형	크기
black	2023-11-28 오후 3:42	파일 폴더	
blue	2023-11-28 오후 3:45	파일 폴더	
red	2023-11-28 오후 3:46	파일 폴더	

테스트 폴더








허나, 테스트데이터를 직접 촬영한 사진으로 국한하다 보니 데이터(사진)이 별로 없어서 가족, 지인들의 사진들도 추가하였다.



그래도 30장씩밖에 구할 수 없었다.

검정색옷 폴더

> .keras > datasets > clothes > test > black

black 검색








검정색옷
 검정색옷2
 검정색옷3
 검정색옷4
 검정색옷5
 검정색옷6
 검정색옷7







검정색옷8
 검정색옷9

파란색옷 폴더

> .keras > datasets > clothes > test > blue

blue 검색

파란색옷
 파란색옷1
 파란색옷2
 파란색옷3
 파란색옷4
 파란색옷5
 파란색옷6

파란색옷7
 파란색옷8
 파란색옷9
 파란색옷10
 파란색옷11
 파란색옷12

빨강색옷 폴더

> .keras > datasets > clothes > test > red

red 검색

빨강색옷
 빨강색옷1
 빨강색옷2
 빨강색옷3
 빨강색옷4
 빨강색옷5
 빨강색옷6

빨강색옷7
 빨강색옷8
 빨강색옷9

본론

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dense
from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
```

텐서플로와 넘파이등 사용할 모듈을 import해준다.

```
def create_cnn2d(input_shape=(224, 224, 3), num_class = 3):
    inputs = Input(shape=input_shape)
    x=Conv2D(filters=16, kernel_size = (3,3), activation='relu')(inputs)
    x=BatchNormalization()(x)
    x=MaxPool2D()(x)

    x=Conv2D(filters=32, kernel_size=(3,3), activation='relu')(x)
    x=MaxPool2D()(x)
    x=Dropout(rate=0.5)(x)

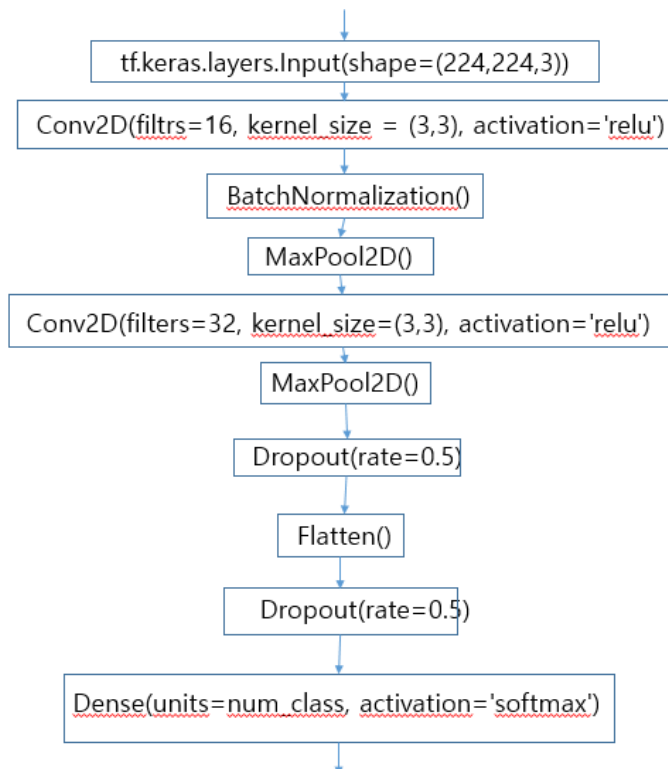
    x=Flatten()(x)
    x= Dense(units=256, activation='relu')(x)
    x= Dropout(rate=0.5)(x)
    outputs= Dense(units=num_class, activation='softmax')(x)
    model = tf.keras.Model(inputs, outputs)

    opt = RMSprop(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

model = create_cnn2d()
```

cnn2d함수는 사진을 224x224사이즈의 RGB 3개, 즉 224x224x3 모양의 입력을 받는 model을 생성한다. 검정,빨강,파랑을 분류할예정이니 num_class를 3개로 분류하는 함수형 API모델을 생성한다.

생성한 모델을 model에 저장한다.



>create_cnn2d 모델 구조

총 파라미터는 23,894,051개이다.

```

train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2,
    validation_split=0.2)
test_datagen = ImageDataGenerator(rescale= 1./255)
  
```

훈련데이터를 rescale = 1./255,로 RGB값을 0~1사이로 정규화하고, 실시간 확장을 위한 ImageDataGenerator 객체에 rotation_range=20로 0에서 20도 범위에서 랜덤하게 회전 확장하고, width_shift_range=0.1,height_shift_range=0.1, 로 수직,수평으로 랜덤하게 평행 이동시키는 범위를 0.1로 설정, 수평으로 horizontal_flip=True,로 이미지를 뒤집게도 하고 zoom_range=0.2로 확대/축소 범위를 0.2로 설정한다. 그리고 validation_split=0.2로 검증데이터 0.2비율로 분리한다.

테스트 데이터도 RGB값을 0~1사이로 정규화한다.

```
img_width, img_height = 224, 224
```

이미지 가로 세로 224x224로 할거니 변수에 넣어 저장

```
train_dir= "C:/Users/82106/.keras/datasets/clothes/train"
```

```
test_dir = "C:/Users/82106/.keras/datasets/clothes/test"
```

폴더를 정한다

```
train_generator= train_datagen.flow_from_directory(  
    train_dir, target_size=(img_width, img_height), batch_size=32,  
    class_mode="categorical", subset='training')
```

```
valid_generator= train_datagen.flow_from_directory(  
    train_dir, target_size=(img_width, img_height), batch_size=32,  
    class_mode="categorical", subset='validation')
```

```
test_generator= test_datagen.flow_from_directory(  
    test_dir, target_size=(img_width, img_height), batch_size=32,  
    class_mode="categorical")
```

train_datagen.flow_from_directory()로 훈련 데이터 폴더에서 subset=training, class_mode="categorical"로 훈련 영상의 확장 제너레이터를 생성한다. categorical인 이유는 출력 Dense층에서 units=3, activation='softmax' 이기때문이다. 배치크기는 32로 설정한다.

분리한 Validation과 테스트데이터도 확장 제너레이터를 생성한다

```
print("train_generator.class_indices=", train_generator.class_indices)
```

```
print("test_generator.class_indices=", test_generator.class_indices)
```

훈련과 테스트 확장 제너레이터의 분류를 출력한다

```
train_steps= int(np.ceil(train_generator.classes.shape[0]/train_generator.batch_size))
```

```
valid_steps= int(np.ceil(valid_generator.classes.shape[0]/valid_generator.batch_size))
```

```
test_steps= int(np.ceil(test_generator.classes.shape[0]/test_generator.batch_size))
```

세개의 steps를 계산하다. 각각 batch의 크기는 32이다.

```
ret = model.fit(train_generator, epochs=100,  
                validation_data=valid_generator,  
                steps_per_epoch= train_steps,  
                validation_steps=valid_steps,  
                verbose=2)
```

확장 제너레이터 된 학습데이터를 100번반복하여 학습시킨다. 검증데이터도 확장 제너레이터된 검증데이터로 설정해주고 위에서 계산한 steps들로 learning_rate=0.001로 학습시킨다. 반복마다 보기위해 verbose=2로 설정 (0로 설정하면 보여주지않음)

```
y_pred = model.predict(train_generator, steps=train_steps, verbose=2)
y_label = np.argmax(y_pred, axis = 1)
C = tf.math.confusion_matrix(train_generator.labels, y_label)
print("confusion_matrix(C):", C)
```

model.predict로 train_generator(확장 제네레이터된 학습 데이터)를 예측하고, 가장 큰 값의 위치를 y_label에 저장하고

컨퓨전행렬 C를 계산하고 출력한다

```
y_pred = model.predict(test_generator, steps=test_steps, verbose=2)
y_label = np.argmax(y_pred, axis = 1)
C = tf.math.confusion_matrix(test_generator.labels, y_label)
print("confusion_matrix(C):", C)
```

model.predict로 test_generator(확장 제네레이터된 테스트 데이터)를 예측하고, 가장 큰 값의 위치를 y_label에 저장하고

컨퓨전행렬 C를 계산하고 출력한다

```
train_loss, train_acc = model.evaluate(train_generator,
                                       steps = train_steps,
                                       verbose=2)
test_loss, test_acc = model.evaluate(test_generator,
                                     steps = test_steps,
                                     verbose=2)
```

학습데이터와 테스트데이터를 평가한다.

```
fig, ax = plt.subplots(1, 2, figsize=(10, 6))
ax[0].plot(ret.history['loss'], "g-")
ax[0].set_title("train loss")
ax[0].set_xlabel('epochs')
ax[0].set_ylabel('loss')

ax[1].plot(ret.history['accuracy'], "b-", label="train accuracy")
ax[1].plot(ret.history['val_accuracy'], "r-", label="val_accuracy")
ax[1].set_title("accuracy")
ax[1].set_xlabel('epochs')
ax[1].set_ylabel('accuracy')
plt.legend(loc="best")
fig.tight_layout()
plt.show()
```

plt.show() 그래프로 학습데이터의 손실과 정확도를 보여준다.

결과

```
Found 240 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
Found 32 images belonging to 3 classes.
train_generator.class_indices= {'black': 0, 'blue': 1, 'red': 2}
test_generator.class_indices= {'black': 0, 'blue': 1, 'red': 2}
Epoch 1/100
```

300개중 240개의 학습데이터와 60개의 검증데이터로 나뉘고, 테스트데이터는 총 32개를 읽었다.

분류는 검정 0, 파랑 1, 빨강 2로 나눈다.

```
Epoch 1/100
8/8 - 13s - loss: 30.7439 - accuracy: 0.5833 - val_loss: 1.3805 - val_accuracy: 0.6667
Epoch 2/100
8/8 - 14s - loss: 0.6655 - accuracy: 0.9500 - val_loss: 1.5836 - val_accuracy: 0.6667
Epoch 3/100
8/8 - 11s - loss: 0.1662 - accuracy: 0.9750 - val_loss: 1.2014 - val_accuracy: 0.6667
...
0/0 - 10s - loss: 1.1321e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 94/100
8/8 - 10s - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 95/100
8/8 - 10s - loss: 4.9671e-10 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 96/100
8/8 - 10s - loss: 1.0831e-04 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 97/100
8/8 - 10s - loss: 0.5010 - accuracy: 0.9833 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 98/100
8/8 - 10s - loss: 0.0599 - accuracy: 0.9958 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 99/100
8/8 - 10s - loss: 0.0302 - accuracy: 0.9958 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 100/100
8/8 - 10s - loss: 1.2914e-08 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

100번 학습 모양

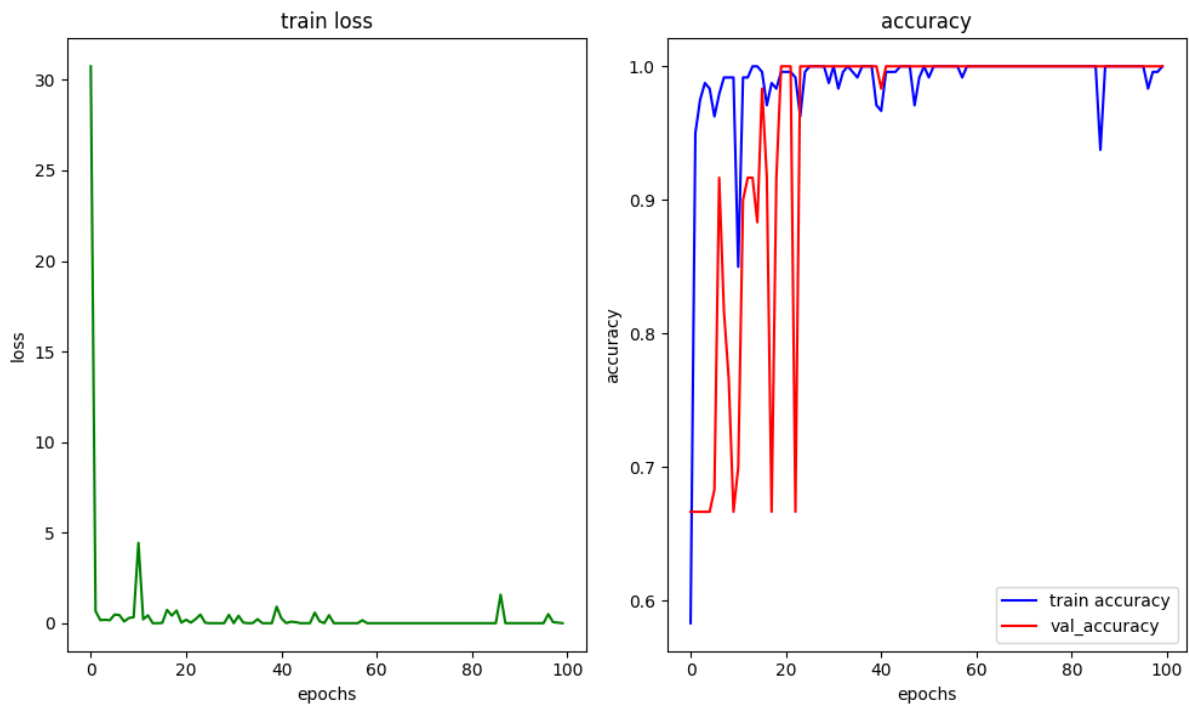
```
0/0 - 10s - loss: 1.1321e+00 - accuracy: 1.0000
8/8 - 4s
confusion_matrix(C): tf.Tensor(
[[ 29 23 28]
 [ 25 24 31]
 [ 26 33 21]], shape=(3, 3), dtype=int32)
```

훈련 데이터에서 컨퓨전 행렬C에서 보면 240개중 166개를 잘못 예측했다.

```
1/1 - 0s
confusion_matrix(C): tf.Tensor(
[[ 3 4 2]
 [ 3 5 5]
 [ 3 4 3]], shape=(3, 3), dtype=int32)
```

테스트 데이터에서 컨퓨전 행렬 C에선 32개중 21개를 잘못 예측했다.

Figure 1



훈련과정에서 손실과 정확도 그래프이다.

3. 결론

결론, 과제 수행 느낌

텐서플로로 틱택토 csv를 학습시켜보고, 옷의 색깔도 학습시켜보았다. 틱택토는 학습데이터의 정확도는 93.21%, 테스트데이터의 정확도는 86.46%, 옷의 색깔은 100%로 나오긴했으나, 학습데이터와 테스트데이터가 터무니 적어서 나온 결과인 것 같다.

이번에 실험한 내용을 확장하면 컴퓨터와 하는 틱택토 게임을 만들 수 있을 것이고, 옷의 색깔은 쇼핑몰이라던가 중고마켓에서 옷의 사진을 업로드하면 자동으로 색이 분류되어 검색에 용이하게 할 수 있을 것이다.

알파고 개발진은 옷 색깔 분류에 사용한 CNN를 이용하고 학습 데이터가 16만개가 된다고 한다는 뉴스를 봤을 때는 저 데이터를 언제 컴퓨터에 입력하고 있을까 라는 생각을 했으나 이번 리포트로 비록 300개의 학습데이터지만 세계를 놀라게 한 ai를 만든 개발진들과 비슷한 일을 해서 흥미롭게 실험을 했다.

참고문헌

-김동근. 『텐서플로 딥러닝 프로그래밍』. 가메출판사, 2020. p.167~169 / p.436~439

코드

1. 틱택토

```
import tensorflow as tf
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
def load_tictactoe(shuffle=False):
```

```
    label={'o':0, 'x':1, 'b':2}
```

```
    result = {'true':1, 'false':0}
```

```
    data = np.loadtxt("./Data/tic-tac-toe.csv", skiprows=1, delimiter=',',
```

```
                    converters={0: lambda name: label[name.decode()],
```

```
                                1: lambda name: label[name.decode()],
```

```
                                2: lambda name: label[name.decode()],
```

```
                                3: lambda name: label[name.decode()],
```

```
                                4: lambda name: label[name.decode()],
```

```
                                5: lambda name: label[name.decode()],
```

```
                                6: lambda name: label[name.decode()],
```

```
                                7: lambda name: label[name.decode()],
```

```
                                8: lambda name: label[name.decode()],
```

```
                                9: lambda resultname: result[resultname.decode()]})
```

```
    if shuffle:
```

```
        np.random.shuffle(data)
```

```
    return data
```

```

def train_test_data_set(tictactoe_data, test_rate=0.2):

    n = int(tictactoe_data.shape[0]*(1-test_rate))

    x_train = tictactoe_data[:n,:-1]
    y_train = tictactoe_data[:n, -1]

    x_test = tictactoe_data[n:,-1]
    y_test = tictactoe_data[n:,-1]

    return (x_train, y_train), (x_test, y_test)

tictactoe_data = load_tictactoe(shuffle=True)

(x_train, y_train), (x_test, y_test) = train_test_data_set(tictactoe_data, test_rate=0.2)

print("x_train.shape:", x_train.shape)
print("y_train.shape:", y_train.shape)
print("x_test.shape:", x_test.shape)
print("y_test.shape:", y_test.shape)

y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

n = 10

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(units=n, input_dim=9, activation='sigmoid'))

model.add(tf.keras.layers.Dense(units=2, activation='softmax'))

```

```
model.summary()
```

```
opt = tf.keras.optimizers.RMSprop(learning_rate=0.01)
```

```
model.compile(optimizer=opt, loss= 'MSE', metrics=['accuracy'])
```

```
model.compile(optimizer=opt, loss='categorical_crossentropy',metrics=['accuracy'])
```

```
ret = model.fit(x_train, y_train, epochs=100, verbose=0)
```

```
print("len(model.layers):", len(model.layers))
```

```
loss = ret.history['loss']
```

```
plt.plot(loss)
```

```
plt.xlabel('epochs')
```

```
plt.ylabel('loss')
```

```
plt.show()
```

```
train_loss, train_acc = model.evaluate(x_train, y_train, verbose=2)
```

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

```
y_pred = model.predict(x_train)
```

```
y_label = np.argmax(y_pred, axis = 1)
```

```
C = tf.math.confusion_matrix(np.argmax(y_train, axis = 1), y_label)
```

```
print("confusion_matrix(C):", C)
```

2. 컬러 분류

```
import tensorflow as tf

from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Dense

from tensorflow.keras.layers import BatchNormalization, Dropout, Flatten

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import numpy as np

import matplotlib.pyplot as plt
```

```
def create_cnn2d(input_shape=(224, 224, 3), num_class = 3):

    inputs = Input(shape=input_shape)

    x=Conv2D(filters=16, kernel_size = (3,3), activation='relu')(inputs)

    x=BatchNormalization()(x)

    x=MaxPool2D()(x)


    x=Conv2D(filters=32, kernel_size=(3,3), activation='relu')(x)

    x=MaxPool2D()(x)

    x=Dropout(rate=0.5)(x)


    x=Flatten()(x)

    x= Dense(units=256, activation='relu')(x)

    x= Dropout(rate=0.5)(x)

    outputs= Dense(units=num_class, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs)
```

```

    opt = RMSprop(learning_rate=0.001)

    model.compile(optimizer=opt, loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = create_cnn2d()

train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2,
    validation_split=0.2)

test_datagen = ImageDataGenerator(rescale= 1./255)

img_width, img_height = 224, 224

train_dir= "C:/Users/82106/.keras/datasets/clothes/train"
test_dir = "C:/Users/82106/.keras/datasets/clothes/test"

train_generator= train_datagen.flow_from_directory(
    train_dir, target_size=(img_width, img_height), batch_size=32,
    class_mode="categorical", subset='training')

valid_generator= train_datagen.flow_from_directory(
    train_dir, target_size=(img_width, img_height), batch_size=32,
    class_mode="categorical", subset='validation')

```

```

test_generator= test_datagen.flow_from_directory(
    test_dir, target_size=(img_width, img_height), batch_size=32,
    class_mode="categorical")

print("train_generator.class_indices=", train_generator.class_indices)
print("test_generator.class_indices=", test_generator.class_indices)


train_steps= int(np.ceil(train_generator.classes.shape[0]/train_generator.batch_size))
valid_steps= int(np.ceil(valid_generator.classes.shape[0]/valid_generator.batch_size))
test_steps= int(np.ceil(test_generator.classes.shape[0]/test_generator.batch_size))


ret = model.fit(train_generator, epochs=3,
                validation_data=valid_generator,
                steps_per_epoch= train_steps,
                validation_steps=valid_steps,
                verbose=2)


y_pred = model.predict(train_generator, steps=train_steps, verbose=2)
y_label = np.argmax(y_pred, axis = 1)
C = tf.math.confusion_matrix(train_generator.labels, y_label)
print("confusion_matrix(C):", C)

```

```
y_pred = model.predict(test_generator, steps=test_steps, verbose=2)

y_label = np.argmax(y_pred, axis = 1)

C = tf.math.confusion_matrix(test_generator.labels, y_label)

print("confusion_matrix(C):", C)
```

```
train_loss, train_acc = model.evaluate(train_generator,

                                       steps = train_steps,

                                       verbose=2)

test_loss, test_acc = model.evaluate(test_generator,

                                       steps = test_steps,

                                       verbose=2)
```

```
fig, ax = plt.subplots(1, 2, figsize=(10, 6))

ax[0].plot(ret.history['loss'], "g-")

ax[0].set_title("train loss")

ax[0].set_xlabel('epochs')

ax[0].set_ylabel('loss')


ax[1].plot(ret.history['accuracy'], "b-", label="train accuracy")

ax[1].plot(ret.history['val_accuracy'], "r-", label="val_accuracy")

ax[1].set_title("accuracy")

ax[1].set_xlabel('epochs')

ax[1].set_ylabel('accuracy')

plt.legend(loc="best")
```

```
fig.tight_layout()
```

```
plt.show()
```