**R E P O R**

[인공지능 학기말 프로젝트]



|  |  |
| --- | --- |
| **•과 목 명:** | 인공지능 |
| **•담당교수:** | 김동근 교수님 |
| **•제 출 일:** | 2023. 12. 06. |
| **•학 과:** | 컴퓨터 공학과 |
| **• 학 번 :** | 201601848 |
| **•성 명:** | 이성남 |

# 서론

* 삼목 게임 학습  
  https://github.com/datasets/tic-tac-toe 사이트의 “tic-tac-toe.csv“의 삼목 게임을 tensorflow로 학습한다.
* 컬러(color) 분류  
  목적: 카메라로 촬영한 컬러( R, G, B, Yellow, Pink 등) 식별, 색상 종류는 각자 정하기

# 본론

# 삼목게임 학습

1. 서론 (문제 설명)

* 깃허브(<https://github.com/datasets/tic-tac-toe>) 에서 “tic-tac-toe.csv”라는 삼목 게임 데이터를 가져와서 tenserflow로 학습했습니다. 해당 데이터에서 6 대 4 비율로 훈련 데이터와 테스트 데이터로 나누었습니다. 그리고 훈련 데이터를 통해 모델을 학습시키고 학습시킨 모델을 통해 훈련 데이터와 테스트 데이터를 평가해 보았습니다.
* 교재 167p. 2층신경망: iris 데이터 분류 (step2202) 예제를 활용하여 tic-tac-toe 삼목게임을 학습했습니다.

1. 본론 (구현 내용을 코드, 그래프 등을 사용하여 설명)

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 데이터를 불러와서 csv파일을 txt파일로 만들었습니다. 그리고 파일의 첫번째 행에는 데이터 칼럼의 헤더값이 있기 때문에 해당 행을 무시하고 넘어가도록 했습니다. 그리고 칼럼들은 쉼표( , ) 로 구분되어 있기 때문에 delimiter = “,” 을 설정하였습니다. 그리고 라벨로 (true, false, x, b, o)를 딕셔너리로 만들어 converters를 통해서 문자를 정수형으로 바꾸었습니다. 그리고 학습이 잘되게 하기위해 셔플을 할 수 있도록 매개변수로 셔플을 true로 바꿀 수 있도록 설정하였습닌다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 데이터를 첫번째 행부터8대 2 비율로 나누어 훈련데이터와 테스트 데이터로 나누도록 함수를 만들었습니다. 해당 함수는 데이터를 나누는 비율을 매개변수로 받도록 했습니다. 해당 매개변수는 8대 2 비율이 고정값으로 되어있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Train\_test\_data\_set() 함수를 사용하여 6대 4 비율로 훈련데이터와 테스트 데이터를 구분했습니다. 그리고 모델이 나눠진 데이터(y값)를 구분하게 하기 위해서 원핫 인코딩을 하였습니다. 그리고 뉴런 개수 30개로 정하고 모델은 Sequential()모델을 사용하였습니다 그리고 인풋층에 relu활성화 함수를 사용하고, 아웃풋 층은 softmax활성화함수를 사용하였씁니다. Unit2개는 true와 false값을 인식할것이기 때문에 units = 2를 하였습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 손실함수로 원핫인코딩된 값에 적합한CategoricalCrossentropy()를 사용하였습니다. 그리고 최적화 알고리즘으로 RMSprop을 사용하였고 학습율을 0.01로 설정하였습니다. 그리고 정확도 손실율등을 주기적으로 확인하기위해 compile함수를 사용하였고, model.fit을 통해 데이터를 학습했습니다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

* Epoch 당 손실율을 그래프를 통해서 보여줍니다.

텍스트, 폰트, 친필, 화이트이(가) 표시된 사진

자동 생성된 설명

* 마지막으로 훈련데이터와 테스트데이터를 평가한값을 보여주고 원핫으로 나눠진 데이터결과값을 출력합니다.

1. 결론 (화면 덤프 및 결과 분석 설명)

텍스트, 스크린샷, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

* 모델 학습을 진행하면서(epoch이 증가하면서) 손실율이 줄어드는 것을 확인 하였습니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* 훈련데이터의 정확도와 테스트 데이터의 정확도를 확인했습니다. 각각의 정확도가 0.9861 , 0.9479임을 확인하였습니다. 해당 데이터의 오차가 생긴이유는 true값에서 3개 false값에서 5개의 오차가 발생했기 때문임을 알 수 있었습니다.

# 컬러(color) 분류

1. 서론 (문제설명)

* 저는 의류 이미지를 데이터로 활용하기 위하여 구글을 통해 의류 이미지 데이터셋을 찾아 훈련 데이터로 사용하였습니다. 그리고 테스트 데이터로 제가 직접 찍은 사진을 테스트 데이터로 활용하였습니다. 컬러 구분은 흑색, 회색 ,백색을 기준으로 컬러를 구분하였습니다. 해당 컬러는 y값 (y\_test, y\_train)으로 사용하였습니다. 그리고 훈련 데이터는 약 1200장 정도의 의류 이미지를 활용하였고, 테스트 데이터로는 약 35장의 이미지를 사용하였습니다.
* 교제 387p. VGG전이학습(transfer learning): CIFAR-10 분류 (4504) 예제를 활용하여 VGG16모델을 사용하였습니다. 그리고 교제 539p. COLAB 사용법을 참고하여 COLAB을 사용하여 모델을 학습하고 학습한 모델로 데이터를 평가했습니다.
* 추가적으로 최적화 알고리즘으로 RMSprop와 Adam을 비교해보았습니다

1. 본론 (구현내용을 코드, 그래프 등을 사용하여 설명)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 우선적으로 학습을 하기 위한 데이터(이미지)가 저장되어 있는 위치를 각 변수에 저장합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* Y(train, test) 값을 (백색, 회색, 흑색)으로 딕셔너리를 활용하여 문자열과 해당값에 매칭되도록 밸류값으로 정수형을 선언하였습니다. 그리고 훈련 데이터와 테스트 데이터를 불러와 각각의 이미지에 대해 크기 조정 및 정규화를 진행하도록 하였습니다.

텍스트, 폰트, 화이트, 친필이(가) 표시된 사진

자동 생성된 설명

* 그리고 for문을 돌면서 추가된 값들을 numPy 배열로 변환하도록 하였습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 클래스 라벨을 원핫 인코딩을 통해서 백색, 흑색, 회색을 구분하도록 하였습니다. 그리고 이미지를 딥러닝 모델에 입력하여 사용하기 위해 preprocess\_input() 함수를 사용하여 x\_train과 x\_test를 전처리 하도록 하였습니다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

* 모델의 입력층을 설정했습니다. 그리고 VGG16모델에 적합한 이미지 사이즈를 만들기위해 resize\_layer를 만들어서 32, 32 사이즈의 이미지를 224, 224로 변환합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* VGG16모델을 생성합니다. 그리고 trainable 를 false로 둠으로써 모델의 가중치를 동결합니다. 그리고 모델의 중간출력을 담고 있는 변수를 만들고 VGG16의 출력을 x에 할당합니다. 그리고 Flatten() 을 사용하여 VGG16의 출력을 1차원으로 펼칩니다. 그리고 1024개의 Dense레이어를 추가하고 활성화 함수로 Relu를 사용합니다. 추가적으로 출력을 담당하는 Dense레이어를 추가합니다 해당 과제에서는 흑색, 회색, 백색 3개의 클래스에대한 확률을 출력하기 때문에 Dense(3, -)으로 하였고 활성화 함수로 softmax를 사용하였습니다.

그리고 3개의 클래스에 대한 확률을 출력하는 모델을 생성합니다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

* 콜백함수를 사용하여 특정조건에 따라 모델의 체크포인트를 filepath의 파일 경로로 저장합니다. 그리고 콜백함수의 매개변수로 save\_best\_only = true를 설정하여 가장 좋은 체크포인트만 저장하도록 설정하였습니다.

옵티마이저로 RMSprop를 설정하고 학습율로 0.001을 설정하였습니다.

(추가적으로 저는 Adam으로 활성화 알고리즘을 설정하고 학습율 0.001로 설정해서 실험해보았습니다. 결과는 아래 결과덤프를 통하여 보이도록하겠습니다.)

그리고 원핫 인고딩에 특화된categofical\_crossentropy손실 함수를 사용하여 컴파일하였고 정확도를 모니터링하도록 컴파일 하였습니다.

그리고 epoch 150 batchsize 30으로 하여 학습을 진행하였습니다. 과적합을 방지하고 더 좋은 학습 결과를 만들기위해 validation\_spit=0.2 를 통해 학습데이터의 20퍼센트는 검증데이토로 사용하도록 하였습니다.

그리고 에폭을 진행할때마다 콜백이 실행되도록 하였습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 마지막으로 학습된 모델을 통해서 테스트 데이터와 학습 데이터를 평가하였습니다. 그리고 훈련중 발생한 손실율과 정확도를 그래프로 표현하도록 하였습니다.

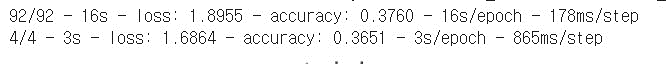
+ 아담알고리즘 코드

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

1. 결론 (화면 덤프 및 결과 분석 설명)

RMSprop 알고리즘 실행결과

-> 실행결과 정확도가 0.37로 적게나왔습니다.

텍스트, 그래프, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

* Adam 알고리즘 실행결과



아담알고리즘은 에폭을 120으로 RMSprop 보다 적게 설정하였습니다 . 하지만 정확도가 더 높게 나온 것을 확인할 수있었습니다.

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명

# 결론(과제 수행 느낌)

* 이번 과제를 수행하면서 딥러닝을 직접해보면서 인공지능을 조금 더 알아가는 계기가 되었습니다. 그리고 RMSprop와 Adam 같은 최적화 알고리즘을 사용해보면서 수업에서 들었던 서로 다른 결과를 도출해낸다는 것을 몸소 느낄 수 있었습니다.

.

# 부록( 코드 )

* 삼목 게임 코드

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

#1 데이터 읽어오는 함수

# csv파일을 txt파일로 불러온다 txt파일로 불러 오면 ", " 로 구분되어 나누어진다.

# 첫번째 행은 헤더값이 온다. 따라서 skiprows = 1 을 통해 1행을 무시하고 넘어간다.

# converters 를 통해서 문자열을 정수형으로 바꾼다.

def load\_Iris(shuffle=False):

label={'true':0, 'false':1, 'x': 2, 'b': 3, 'o': 4}

data = np.loadtxt(r'C:\Users\dltjd\Artificial Intelligence\tic-tac-toe.csv', skiprows=1, delimiter=',',

converters={i: lambda name: label[name.decode()] for i in range(10)})

#셔플이 트루라면 데이터를 섞는다.

if shuffle:

np.random.shuffle(data)

return data

# 데이터에서 훈련세트와 테스트세트를 나눈다.

# test\_rate -> 테스트 세트 40% iris\_data -> 훈련세트 60%

# 80%를 학습하고 20%는 성능을 테스

def train\_test\_data\_set(iris\_data, test\_rate=0.2): # train: 0.8, test: 0.2

n = int(iris\_data.shape[0]\*(1-test\_rate)) # iris\_data -> (958, 10) 중 [0] 즉, 958 \* 0.8 (766)개 훈련데이터로 설

#훈련 데이터 574개

x\_train = iris\_data[:n,:-1] # 958의 행중 1행부터 765행까지 사용 + 마지막 열 제거 (입력)

y\_train = iris\_data[:n, -1] # 958의 행중 1행부터 765행까지 사용 + 마지막 열만 사용 (출력)

# 테스트 데이터 384

x\_test = iris\_data[n:,:-1] # 766행 부터 958행 까지 사용 + 마지막 열 제거 (입력)

y\_test = iris\_data[n:,-1] # 766행 부터 958행 까지 사용 + 마지막 열만 사용 (출력)

return (x\_train, y\_train), (x\_test, y\_test)

# 데이터 읽어오는 함수를 사용하여 데이터를 불러온다

# 데이터를 섞어서 불러온다 (shuffle = true) -> 데이터를 섞어야 학습이 더 잘된다.

iris\_data = load\_Iris(shuffle=True)

(x\_train, y\_train), (x\_test, y\_test) = train\_test\_data\_set(iris\_data, test\_rate=0.4) #test데이터의 비율을 40%로 높인다.

print("x\_train.shape:", x\_train.shape) #(574,4)

print("y\_train.shape:", y\_train.shape) #(574, 1)

print("x\_test.shape:", x\_test.shape) #(384, 4)

print("y\_test.shape:", y\_test.shape) #(384, 1)

# one-hot encoding: 'mse', 'categorical\_crossentropy'

y\_train = tf.keras.utils.to\_categorical(y\_train)

y\_test = tf.keras.utils.to\_categorical(y\_test)

##print("y\_train=", y\_train)

##print("y\_test=", y\_test)

#2

# 모델을 만든다. n은 뉴런의 갯수 , input\_dim은 입력갯수, 첫벗째 input층 마지막줄 output층

# output층은 3가지 종류로 인식할것이기 때문에 unit = 3

n = 30 # number of neurons in a hidden layer

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(units=n, input\_dim=9, activation='relu'))

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

model.summary()

#3

def MSE(y, t):

return tf.reduce\_mean(tf.square(y - t)) # (y - t)\*\*2

CCE = tf.keras.losses.CategoricalCrossentropy()

opt = tf.keras.optimizers.RMSprop(learning\_rate=0.01)

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

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

##model.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=['accuracy'])

# model.compile을 통해서 학습환경을 정한다. metrics -> 정확도를 출력

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

# model.fit을 통해 학습하고 값을 ret에 저장

ret = model.fit(x\_train, y\_train, epochs=1000, batch\_size=7, verbose=2) # batch\_size=32

print("len(model.layers):", len(model.layers)) # 2

loss = ret.history['loss']

plt.plot(loss)

plt.xlabel('epochs')

plt.ylabel('loss')

plt.show()

#4

##print(model.get\_weights())

##for i in range(len(model.layers)):

## print("layer :", i, '-'\*20)

## w = model.layers[i].weights[0].numpy()

## b = model.layers[i].bias.numpy()

## print("weights[{}]: {}".format(i, np.array2string(w)))

## print("bias[{}]: {}".format(i, np.array2string(b)))

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)

* 컬러 분류 코드
* import tensorflow as tf
* from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array
* from tensorflow.keras.layers   import Input, Dense, Flatten
* from tensorflow.keras.applications.vgg16 import preprocess\_input, VGG16
* import numpy as np
* import matplotlib.pyplot as plt
* import os
* #1:
* ##gpus = tf.config.experimental.list\_physical\_devices('GPU')
* ##tf.config.experimental.set\_memory\_growth(gpus[0], True)
* #2
* # 이미지 파일들이 있는 디렉토리 경로(xtrain, xtest)
* x\_test\_images\_dir = "/content/drive/My Drive/ai/clothData/test"
* x\_train\_image\_dir = "/content/drive/My Drive/ai/clothData/train"
* # 클래스 레이블 정의
* class\_labels = {'white': 0, 'gray': 1, 'black': 2}
* # 데이터 저장을 위한 리스트 초기화
* x\_train = []
* y\_train = []
* x\_test = []
* y\_test = []
* # 이미지 파일들의 경로를 리스트로 수집
* for class\_name, class\_label in class\_labels.items():
* # 훈련 데이터
* train\_dir = os.path.join(x\_train\_image\_dir, class\_name)
* for filename in os.listdir(train\_dir):
* if filename.endswith(".jpg"):
* img\_path = os.path.join(train\_dir, filename)
* img = load\_img(img\_path, target\_size=(32, 32))
* img\_array = img\_to\_array(img) / 255.0
* x\_train.append(img\_array)
* y\_train.append(class\_label)
* # 테스트 데이터
* test\_dir = os.path.join(x\_test\_images\_dir, class\_name)
* for filename in os.listdir(test\_dir):
* if filename.endswith(".jpg"):
* img\_path = os.path.join(test\_dir, filename)
* img = load\_img(img\_path, target\_size=(32, 32))
* img\_array = img\_to\_array(img) / 255.0
* x\_test.append(img\_array)
* y\_test.append(class\_label)
* # 리스트를 NumPy 배열로 변환
* x\_train = np.array(x\_train)
* y\_train = np.array(y\_train)
* x\_test = np.array(x\_test)
* y\_test = np.array(y\_test)
* x\_train = x\_train.astype('float32') # (사진갯수, 사진높이, 너비, 채널수)
* x\_test  = x\_test.astype('float32')  # (사진갯수, 사진높이, 너비, 채널수)
* # one-hot encoding
* y\_train = tf.keras.utils.to\_categorical(y\_train)
* y\_test  = tf.keras.utils.to\_categorical(y\_test)
* # preprocessing, 'caffe', x\_train, x\_test: BGR
* x\_train = preprocess\_input(x\_train)
* x\_test = preprocess\_input(x\_test)
* #3: resize\_layer
* inputs = Input(shape=(32, 32, 3))
* resize\_layer = tf.keras.layers.Lambda(
* lambda img: tf.image.resize(img,(224, 224)))(inputs)
* #4:
* ##W = 'C:/Users/user/.keras/models/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'
* ##W = './Data/models/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5'
* vgg\_model = VGG16(weights='imagenet', include\_top= False,
* input\_tensor= resize\_layer) # input\_tensor= inputs
* vgg\_model.trainable=False
* ##for layer in vgg\_model.layers:
* ##    layer.trainable = False
* #4-1: output: classification
* x = vgg\_model.output
* x = Flatten()(x)   # x = GlobalAveragePooling2D()(x)
* x = Dense(1024, activation='relu')(x)
* outs  = Dense(3, activation='softmax')(x)
* model = tf.keras.Model(inputs=inputs, outputs=outs)
* model.summary()
* #5: train and evaluate the model
* filepath = "./RES/finalAssignment.h5"
* cp\_callback = tf.keras.callbacks.ModelCheckpoint(
* filepath, verbose=0, save\_best\_only=True)
* opt = tf.keras.optimizers.Adam(learning\_rate=0.001)
* model.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=['accuracy'])
* ret = model.fit(x\_train, y\_train, epochs=120, batch\_size= 30,
* validation\_split=0.2, verbose=2, callbacks = [cp\_callback])
* train\_loss, train\_acc = model.evaluate(x\_train, y\_train, verbose=2)
* test\_loss, test\_acc = model.evaluate(x\_test,  y\_test, verbose=2)
* #6: plot accuracy and loss
* 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].set\_ylim(0, 1.1)
* 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='lower right')
* fig.tight\_layout()
* plt.show()