

OpenCV 를 사용한 한글 완전체 글자 분류 모델 연구

지능형비전 COM3034

박천수 교수님

2019312552 김서진

2020310100 정소연

2020313241 하유진

I. Introduction

본 연구는 종이 손상 복원을 위한 인공지능 모델의 개발에 초점을 맞추고 있다. 종이는 구겨지거나 찢어지는 등의 손상을 입을 수 있어 종이에 적힌 글자 정보 또한 훼손되는 경우가 종종 발생한다. 이에 딥러닝과 컴퓨터 비전을 활용해 손상된 종이에서 글자를 추출하여 의미를 복원하는 모델을 개발하고자 하였다. 전체적인 모델의 구조는 (1) 찢어진 종이 조각을 하나로 연결 (2) 글자 인식 (OCR) (3) 손실된 문장의 의미 유추, 총 세 단계로 이루어져 있다. 본 연구는 (2) 글자 인식 모델에 초점을 맞추었으며, 본 연구의 모델은 한글 완성형 글자 2,353 종과 자음, 모음 40 자, 영어 대/소문자 52 자, 숫자 10 자의 이미지를 입력으로 받아 글자를 정확하게 예측한다. 이를 통해 손상된 종이 문서의 중요한 정보를 복구할 수 있는 기술적인 해결책을 제시하고자 한다.

II. METHOD

II-A. 데이터 수집 및 전처리 과정

본 연구에서는 2,353 개의 한글 완성형 글자, 40 개의 자/모음, 52 개의 영어 대/소문자, 10 개의 숫자로 총 2,455 종류의 글자 데이터를 활용하여 손글씨 폰트 종류 100 개⁴로 학습을 진행하였다. 데이터는 무료 손글씨 폰트 사이트인 "온글잎"과 "네이버 클로바 나눔손글씨"에서 다운로드하여, 아래의 서식 템플릿에 해당 폰트를 적용하고 이미지 형태로 저장한 후 한 글자 당 하나의 이미지 파일로 분리하여 가공하여 사용하였다. 글자를 분리하는 코드는 **V-A**에 정리되어 있다.

총 245,500 장의 이미지를 불러오는 것에 대하여 로컬 환경에서는 큰 부하가 없었으나, 모델링을 위해 Colab 에 데이터를 업로드할 때 업로드가 중단되거나 취소되는 오류가 발생했다. 이는 파일의 크기보다는 데이터 개수와 관련한 문제였으므로, Google Colab 환경에서 학습을 진행할 때에는 개별 폰트 파일 245,500 장 대신 통합 폰트 파일 100장을 불러오고 이미지를 자르는 과정은 코드 환경에서 하였다. [**V-B**]

위 과정 중에 데이터의 화질이 원본보다 저하되거나 잡음이 추가되는 경우가 있었고, 잡음을 제거하기 위하여 OpenCV 의 Median Blur 을 적용하였고, 중간값 필터 적용에 따른 데이터의 학습 및 예측 성능을 비교하였다.

원본	V-A 과정에서 저장된 데이터	Median Blur (k=3)

II-B. 모델방법론

본 연구에서는 한글 완전체 글자 이미지 분류 모델을 평가하고 최적의 모델을 탐색하기 위한 연구를 수행하였다. 비교 대상으로는 기계학습 모델로는 K-Nearest Neighbors (KNN)와 Support Vector Machines (SVM)을 고려하였으며, 딥러닝 모델로는 직접 학습시킨 Convolutional Neural Network (CNN)과, 전이 학습을 통해 출력 층의 값을 조정한 Inception 과 VGG 를 비교하였다. 기계학습 모델인 KNN 과 SVM 은 OpenCV 에 구현된 KNearrest, HOG 클래스를 이용하여 학습시켰고, test 데이터에 대한 정확도 평가의 경우 모든 모델에 대해 C++ 환경에서 이루어졌다. 각 모델에 대해 정확도, 학습 소요 시간, test 소요 시간을 측정하여 성능을 비교하였다. 각 모델의 학습 및 평가 관련 코드는 [V-C]에 첨부되어 있다.

III. RESULT

III-A. KNN

Fonts per char	30		59		100
n-neighbors	3	4	3	4	3
Accuracy	0.0978955	0.0906993	0.129613	0.128635	0.219837
Train Time (ms)	271	271	525	525	1009
Test Time (ms)	480547	431331	1652370	1279442	4962502

표 1. KNN 모델의 학습 결과 (1) – n-neighbors 비교

가장 기본적인 KNN 모델을 사용한 학습 결과는 위 표와 같다. 데이터 수가 많아질 수록 정확도가 증가하며, K 값인 n-neighbors 로는 4 보다는 3 을 사용하는 것이 효과가 좋다는 것을 볼 수 있다.

Fonts per char	30		59		100	
Use HOG	X	O	X	O	X	O
Accuracy	0.0978955	0.791582	0.129613	0.0857841	0.219837	0.132627
Train Time (ms)	271	162	525	396	1009	56730
Test Time (ms)	480547	321014	1652370	1244373	4962502	3702358

표 2. KNN 모델의 학습 결과 (2) – HOG 사용 여부 비교

HOG Descriptor 을 사용하여 특징을 추출해 학습시켜 보았는데, 학습 시간과 테스트 시간은 줄었으나 정확도 마찬가지로 줄어들었다. HOG 를 구성할 때 winSize(38,42), blockSize(19,21), blockStride(1,3), cellSize(19,21)로 구성하였는데, 38x42 사이즈의 이미지를 리사이징 없이 사용했기 때문에 약수 정수의 개수에 한계가 있어 크기를 적절하게 정하지 못하여 성능이 오히려 저하된 것 같다.

Fonts per char	30		59		100	
Use Blur	O	X	O	X	O	X
Accuracy	0.0978955	0.0839104	0.129613	0.116578	0.219837	0.206965
Train Time (ms)	271	395	525	739	1009	41966
Test Time (ms)	480547	533576	1652370	1334153	4962502	4144353

표 3. KNN 모델의 학습 결과 (3) – Blur 사용 여부 비교

마지막으로, Median Blur 를 사용한 경우와 사용하지 않은 경우의 성능을 비교하였고, 필터 처리를 했을 때 전반적으로 학습 성능이 더 좋은 것을 확인하였다.

III-B. SVM

Fonts per char	30	59	100
Accuracy	0.179498	0.189328	0.243014
Train Time (ms)	176345	548513	1009287
Test Time (ms)	2565329	520489	1222431

표 4. SVM 모델의 학습 결과

SVM 타입은 C-서포트 벡터 분류를 사용하였으며, 커널은 방사 기저 함수 커널을 사용했다. 또한 파라미터를 구하기 위해 30 개 데이터셋에서 trainAuto()를 실행하여, C 는 312.5, Gamma 는 0.50625 라는 값을 얻어 다른 데이터셋을 학습시킬 때에도 사용했다. SVM 의 HOG 는, 사용한 이미지 데이터 크기가 38x42 였기 때문에 셀크기를 그 약수인 19x21 으로 정하였고, 블록 크기는 38x42 로 설정하였다. 30 개, 59 개, 100 개 데이터에 대해서 훈련한 결과 0.17 에서 0.18, 그리고 0.24 정도로 정확도가 증가하였다. 그러나 머신러닝 모델인 만큼, 좋은 성능을 보이지는 못한 것 같다.

III-C. VGG16, InceptionV3 전이 학습 모델

Pre-trained Model	VGG16	InceptionV3
Accuracy	0.0004	0.000366599
Train Time (s)	9683	23647
Test Time (ms)	73514775	1283555

표 5. VGG_15, InceptionV3 모델 전이 학습 결과

먼저 VGG 의 경우에는 VGG16 모델을 베이스 모델로 채택하고, Flatten 레이어와 Dense 레이어를 출력층에 추가하였다. 초기에는 batch_size 를 32, epochs 를 10 으로 설정하여 훈련을 진행하였으나, VGG16 모델의 크기로 인해 실행 시간이 상당히 오래 소요되고, Google Colab 에서 RAM 이 다운되어 연결이 끊어지는 문제가 발생하였다. 이에 훈련 데이터의 수를 줄여 글씨체 15 개를 사용하고, batch_size 와 epochs 를 각각 1 로 조정하였다. 그 결과 정확도는 약 0.04%로 매우 낮게 측정되었다. 실제로 우리의 데이터는 38x42 크기의 1 채널 이미지이지만, VGG 는 224x224 크기의 3 채널 이미지를 사용하는 구조였다. 이로 인해 우리 데이터에 대해 VGG 는 과도하게 복잡한 모델 구조를 가지고 있어 Gradient Vanishing 등의 문제가 발생할 수 있다고 판단하였다.

InceptionV3 모델의 경우에도 마찬가지로 InceptionV3 모델의 출력층에 뉴런 4096 개의 Dense 층과 출력층을 추가하여 학습시켰는데, 이 경우에도 마찬가지로 정답률 0.0003 의 매우 저조한 성능을 보였다. Image Data Loader 를 통해 데이터 양을 증대시켜 학습해 보려 했으나, 크게 성능이 개선되지 않았고, 서비스에 사용될 수 있는 수준의 정확도가 나오지 않았다.

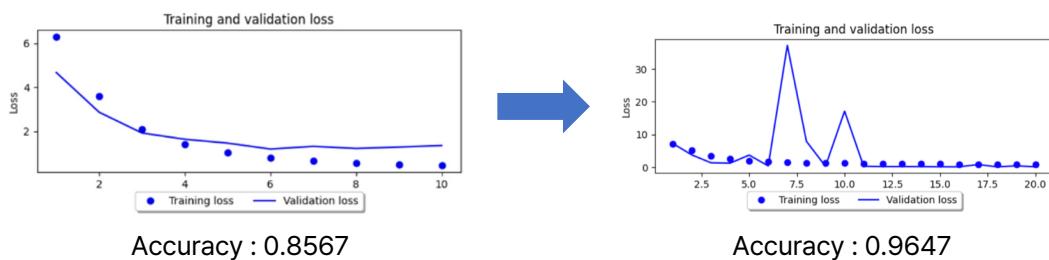
VGG 모델과 Inception 모델의 경우 모두 정확도 면에서 성능이 매우 저조했는데, 이는 VGG16 과 InceptionV3 모델이 학습한 거대 이미지 데이터베이스인 ImageNet 에는 글자 이미지보다 실제 세계의 사진 이미지가 훨씬 많이 담겨 있기 때문에, 해당 이미지를 분류하기 위해 학습된 컨볼루션 층의 가중치는 글씨체 이미지를 분류하는 것에도 적합하게 동작하지 않기 때문이라고 추측하고 있다. 따라서 다양한 데이터를 학습한 일반적으로 좋은 모델을 전이학습시키는 것보다, 손글씨 데이터 인식에 알맞은 새로운 컨볼루션 신경망을 직접 구축하는 것이 더 성능 향상에 도움이 될 것이라고 생각했다.

III-D. CNN

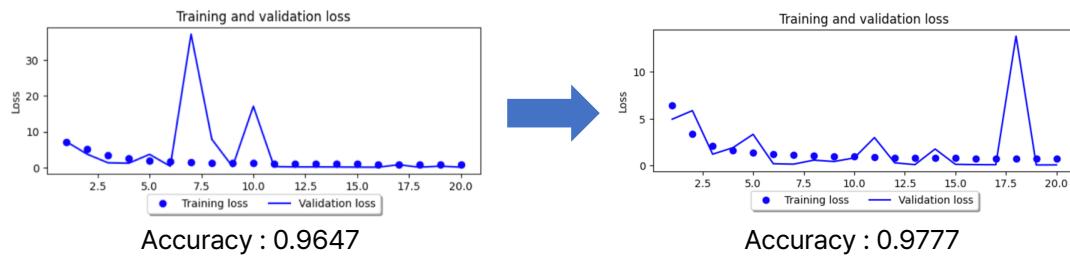
Model	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Accuracy(1)	0.8567	0.9269	0.8967	0.95002	0.9473	0.0430	1
Accuracy(2)	0.7088	0.9647	0.9402	0.9778	0.9777	0.0690	0.9936
Train (s)	3648	1227	17393	2026	2055	13911	264
Test (ms)	25653	14288	22289	15966	15633	17277	6815

표 6. CNN 모델의 학습 결과¹

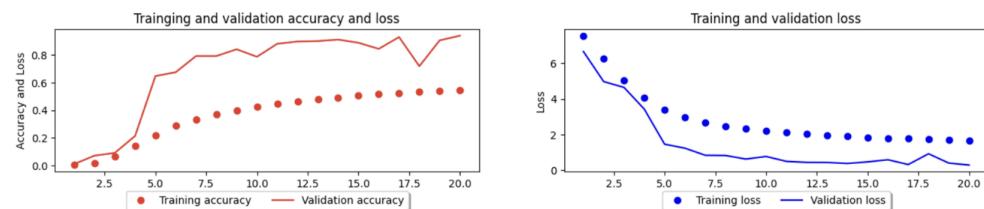
CNN 모델의 학습 결과는 위 표와 같으며, Accuracy(1)은 C++ opencv 환경의 결과이고, Accuracy(2)는 Colab에서의 결과이다. Onnx 파일로 변환하는 과정에서 모델이 조금 변형되어 두 결과가 다르다고 추측된다. CNN 모델의 성능을 증가시킨 방법은 아래와 같다.



첫 모델에서는 컨볼루션 층을 2 개로 쌓는 과정을 반복하여 과적합이 발생했고, 이를 해결하기 위해 각 층에 드롭아웃, 배치 정규화 층을 추가하고 학습 데이터 양을 늘렸으며, epoch를 10에서 20으로 늘렸다. 그 결과 오른쪽 그래프와 같이 Validation Loss 값을 줄였음을 볼 수 있다.

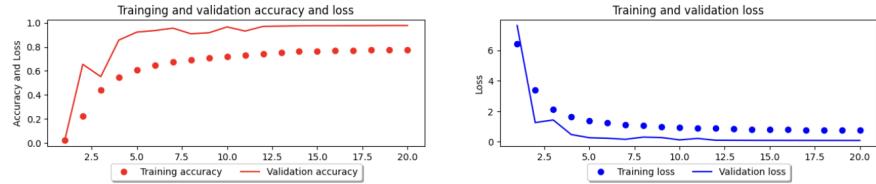


추가적인 성능 개선을 위하여 Early Stopping 콜백을 도입해 validation accuracy 가 1% 증가되지 않는 경우 학습을 중지시키고 과적합 이전의 최상의 모델을 저장하는 방식으로 변경하였으며 이전보다 2 배의 데이터를 넣어 학습시켰다. 데이터 개수가 많아짐에 따라 Test Elapsed Time이 크게 증가하였다.



이후 dropout 비율을 수정해 0.5로 증가시켜 보았으나, 신경망의 복잡도가 감소하여 정확도가 하락하였다.

¹ Train 시간의 경우 batch size나 train dataset size, 신경망의 깊이 등의 영향도 있으나 Colaboratory 런타임 환경의 영향을 가장 크게 받음.



따라서 dropout 비율은 0.25로 유지하면서 학습률 스케줄링을 적용하였고, 이를 통해 학습이 진행되면서 학습률이 조정되어 더 세밀한 학습이 이루어져 결과적으로 모델의 성능이 개선되었다.

위 실험에서는 필터 적용 없이 이미지 데이터를 사용했는데, 이미지의 잡음을 포함한 세부 정보보다는 전반적인 형태를 학습하도록 하면 성능을 향상시킬 수 있을 것으로 예상하고 학습 데이터 셋에 가우시안 필터를 적용하여 학습시켰으나, 가우시안 필터가 적용된 이미지로 학습시킨 모델의 정확도는 0.04로 저조한 모습을 보였다. 따라서 CNN에서 이미지 분류 학습 시 세부 정보가 더 중요하다는 사실을 알게 되었다.

위의 실험을 통해 확인한 결과, TensorFlow 프레임워크에서 손실 함수로는 Cross Entropy Loss를, optimizer로는 adam을 사용했을 때 가장 우수한 성능을 보였다. 또한 ImageDataGenerator 클래스를 활용하여 훈련 데이터셋의 이미지에 회전, 확대/축소, 수평 뒤집기 등의 변형을 적용하여 데이터 양을 늘렸을 때에도 마찬가지로 모델의 성능이 향상되었다.

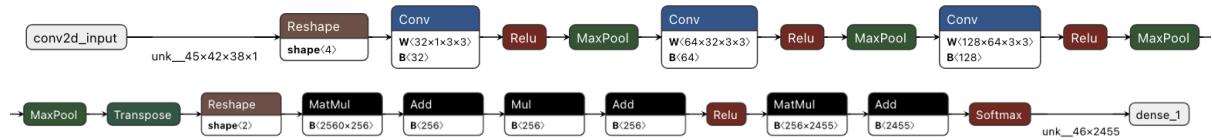
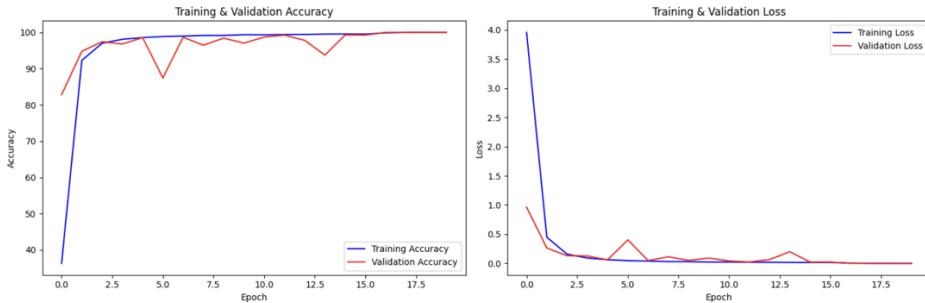


그림 1. CNN 모델의 네트워크 구조 (accuracy 0.9496)



MNIST 손글씨 숫자 데이터 분류 정확도 99.5% 성능의 CNN 모델ⁱⁱ을 본 연구의 한국어 데이터셋에 적용시켜 보았다. 손실 함수로는 Cross Entropy Loss를 사용하였고, optimizer로 Adam을 사용하였다. Learning Rate Scheduler를 사용하여 학습이 진행되지 않을 때 학습률을 조절하도록 하였다. PyTorch 모델이 정확도 1을 달성할 수 있었던 이유는, 모델 아키텍처와 학습과정이 데이터셋의 복잡성과 특징을 잘 반영했기 때문으로 보인다. 물론 실제 세계의 복잡한 데이터셋에 해당 모델을 적용하면 정확도 1이 나올 수 없지만, 새로운 데이터에 대한 모델의 일반화 능력을 검증할 기반을 마련하였다는 점에서 의의가 있다.

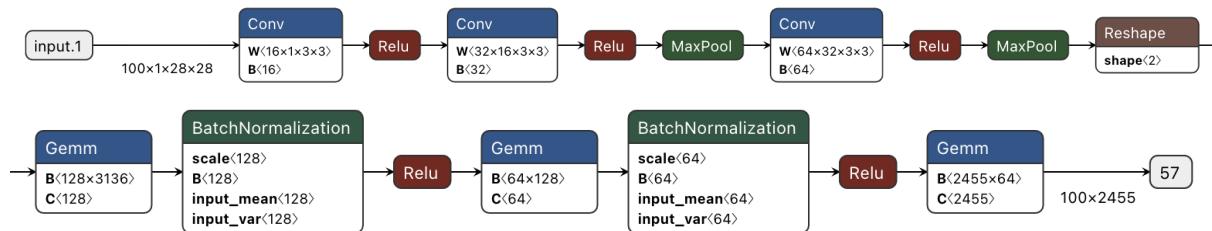


그림 2. CNN 모델의 네트워크 구조 (accuracy 1)

IV. CONCLUSION

본 연구의 한국어 손글씨 이미지 분류에서 가장 높은 정확도를 달성한 모델은 직접 구축한 CNN 모델로 0.9~1의 정확도를 달성하였다. 반면 기계학습 모델의 성능은 0.1~0.2 수준으로 저조한 편이었는데, 이는 24550 개 이상의 레이블로 분류해야 하는 한글 손글씨 분류에 있어서 기계학습 알고리즘에 한계가 있음을 시사한다. 또한, VGG16 과 InceptionV3 의 전이학습을 적용한 모델의 성능이 0.003~0.004 로 가장 저조했는데, 이는 거대 이미지 데이터베이스 ImageNet 을 학습한 VGG, Inception 모델이 한글 손글씨 이미지를 분류하기에는 부적절한 신경망 구조를 갖고 있음을 보여준다.

추후 성능 개선을 위해서 시도해 볼 수 있는 방법은 아래와 같다. 먼저 레이블링 개수를 줄이기 위해 자음과 모음을 따로 추출하여 조합해 글자를 구성하는 방법을 사용해 볼 수 있다. 한글 기본 자음은 14 자, 모음은 21자로, 이미지에서 ROI 영역을 추출해 자음과 모음만으로 분류한다면 기존에 2350 개로 분류하던 신경망을 35 개로 분류하는 것으로 간소화할 수 있기 때문에 예측률 또한 높아질 것이라고 예상한다. 또한, 더 다양한 폰트의 손글씨를 데이터셋에 추가하고, 잡음 없이 데이터를 모으거나 데이터에 있는 잡음을 제거하는 더 다양한 방법도 시도해 볼 수 있을 것이다.

본 연구를 통하여 직접 데이터셋을 모으고, 모델을 구축하여 학습시키고, 결과를 측정하고 시각화한 결과물로 다시 개선책을 생각해내며 딥러닝 분야에 흥미를 느낄 수 있었다. 모델링을 할 때 매개변수 값에 따라 결과물이 크게 변화하는 것을 보면 하이퍼 파라미터 최적화의 중요성을 체감하였으며, 대규모 이미지 데이터를 학습한 VGG16 이나 InceptionV3 의 성능이 딥러닝 모델인데도 기계학습 모델보다 저조한 것을 보고, 주어진 데이터 분류에 알맞은 모델을 선택하여 학습하는 것의 중요성 또한 체감할 수 있었다.

V. CODE

V-A. 데이터 가공 스크립트



아래 스크립트를 통하여 각 폰트 파일에 담긴 폰트 이미지를 분리해 글자 종류별로 파일에 담아 데이터를 로드할 때 사용하였다.

V-A-1. pdf 형식으로 저장한 후 이미지 파일 형식(jpeg)으로 변환

```
for file in ./0_original_pdf/*.pdf; do pdftoppm -jpeg -r 300 "$file"
"1_original_jpeg/${basename "$file"}.pdf"; done
```

V-A-2. 이미지의 Padding 제거, Resize

```
for file in ./1_original_jpeg/*.jpg; do ffmpeg -i "$file" -vf
"crop=1872:2086:208:224" "2_cropped_jpeg/${basename "$file"}.jpg"; done
```

```
for file in 2_cropped_jpeg/*.jpg; do ffmpeg -i "$file" -vf "scale=w=1900:h=2100"
"./3_resized_jpg/${basename "$file"}.jpg"; done
```

V-A-3. 이미지 분리

```
#!/bin/bash
fontnames=( "미깡" "윤스" "재훈" "찬웅" ... ) // 모든 폰트 이름
characters="가각간간갈갈갈감감값갓갓강갓갓같...."; // 모든 글자
for fontname in "${fontnames[@]}"; do
    for ((i=0; i<#${characters}; i++)); do
        char=${characters:i:1} mkdir -p "$char" (ffmpeg -i
        ".../3_resized_jpg/${fontname}.jpg" -vf "crop=38:42:$(( (i % 50) * 38 )):$((
        (i / 50) * 42 ))" "./$char/${fontname}.jpg") &
    done
done
```

V-B. 파일 환경에서의 데이터 로드 스크립트

```
sub_image_width = 38
sub_image_height = 42
num_rows = 50
num_columns = 50

def getSplittedData(fontName):
    font_file_path = file_path + '/' + fontName + '.jpg'
    image = cv2.imread(font_file_path, cv2.IMREAD_GRAYSCALE)
    image_dict = dict()
    for row in range(num_rows):
        for col in range(num_columns):
            x = col * sub_image_width
            y = row * sub_image_height

            sub_image = image[y:y + sub_image_height, x:x
                              + sub_image_width]
            sub_image = np.expand_dims(sub_image, axis=-1)
            try:
                target_char = ALL_TARGET_CHAR[row * num_rows + col]
                image_dict[target_char] = [sub_image]
            except:
                break
    return image_dict

all_dict = dict()
for fn in all_font_names:
    fid = getSplittedData(fn)
    all_dict = merge_dicts(all_dict, fid)

train_index = 0
test_index = 0
trainImages = np.memmap("train_images.npy", dtype=np.float32, mode="w+",
shape=(num_train_images, image_height, image_width, num_channels))
trainLabels = np.memmap("train_labels.npy", dtype=np.int32, mode="w+",
shape=(num_train_images,))
testImages = np.memmap("test_images.npy", dtype=np.float32, mode="w+",
shape=(num_test_images, image_height, image_width, num_channels))
testLabels = np.memmap("test_labels.npy", dtype=np.int32, mode="w+",
shape=(num_test_images,))

for label, image in all_dict.items(): # 글자 하나
    allImages = len(image) # 글자에 대한 각종 폰트의 개수
    number_label = ALL_TARGET_CHAR.index(label)

    testCount = int(allImages * (1-train_ratio))
    testRangeStart = c%3
    testRangeEnd = testRangeStart + testCount
    if(c%1000 == 0) : print("process :", c)
```

```
for i in range(allImages):
    targetImage = image[i]
    reshapedTargetImage = np.repeat(targetImage, 3, axis=2)
    resizedTargetImage = cv2.resize(reshapedTargetImage, (75,75))

    if i >= testRangeStart and i < testRangeEnd:
        testImages[test_index] = resizedTargetImage
        testLabels[test_index] = number_label
        test_index += 1
    else:
        trainImages[train_index] = resizedTargetImage
        trainLabels[train_index] = number_label
        train_index += 1

c += 1
```

V-C-1. 이미지 로드

```
#include <iostream>
#include <string>
#include <vector>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include "allChars.cpp"

using namespace std;
using namespace cv;

class CharImage
{
public:
    string targetChar;
    int targetLabel;
    Mat src;
    CharImage(int fontIndex, int label, string targetChar, bool useBlur = false)
    {
        this->targetChar = targetChar;
        this->targetLabel = label;
        string filename = "resources/images/webFonts/" + targetChar + "/" +
        to_string(fontIndex) + ".jpg";
        Mat tempSrc = imread(filename, IMREAD_GRAYSCALE);
        if (tempSrc.empty())
        {
            throw runtime_error("failed to read image : " + filename);
        }
        if (useBlur)
            medianBlur(tempSrc, this->src, 3);
        else
            this->src = tempSrc;
    }
    friend std::ostream &operator<<(std::ostream &os, const CharImage &charImage)
    {
        os << "\tCharImage: " << charImage.targetLabel << ", " <<
        charImage.targetChar;
        return os;
    };
};

template <typename T>
class TrainTestData
{
public:
    vector<T> trainData;
    vector<T> testData;
    TrainTestData(vector<T> trainData, vector<T> testData)
    {
        this->trainData = trainData;
        this->testData = testData;
    }
};

vector<CharImage> getAllCharImage(int charSize)
```

```

{
    vector<CharImage> charImages = {};
    int label = 0;
    for (string targetChar : ALL_CHARS)
    {
        for (int fontIdx = 1; fontIdx <= charSize; fontIdx++)
        {
            try
            {
                CharImage newCharImage(fontIdx, label, targetChar);
                charImages.push_back(newCharImage);
            }
            catch (exception &e)
            {
                cout << e.what() << endl;
            }
        }
        label++;
    }
    cout << "\t# of total data : " << charImages.size() << endl;
    return charImages;
}

TrainTestData<CharImage> getCharImageData(float trainRatio = 0.9, int charSize =
100)
{
    vector<CharImage> allCharImages = getAllCharImage(charSize);
    vector<CharImage> trainData, testData;
    int testCount = charSize * (1 - trainRatio);

    for (int i = 0; i < allCharImages.size(); i++)
    {
        int testRangeStart = (i / charSize) % 3;
        int testRangeEnd = testRangeStart + testCount;
        if (i % charSize >= testRangeStart && i % charSize < testRangeEnd)
            testData.push_back(allCharImages[i]);
        else
            trainData.push_back(allCharImages[i]);
    }
    cout << "data split completed, " << endl;
    cout << "\t# of train data : " << trainData.size() << endl;
    cout << "\t# of test data : " << testData.size() << endl;

    return TrainTestData<CharImage>(trainData, testData);
}

```

V-C-2-a. Train KNN

```
#include <iostream>
#include <locale>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/ml.hpp>
#include <chrono>
#include "imageLoader.cpp"

using namespace std;
using namespace cv;

int main()
{
    Ptr<ml::KNearest> knn = ml::KNearest::create();
    auto start_time1 = chrono::high_resolution_clock::now();
    TrainTestData<CharImage> data = getCharImageData();
    auto end_time1 = chrono::high_resolution_clock::now();
    auto duration1 = chrono::duration_cast<chrono::milliseconds>(end_time1 -
start_time1);
    cout << "Elapsed time(Data Load): " << duration1.count() << " milliseconds" <<
endl;

    Mat trainImages, trainLabels;

    int cnt = 0;
    for (const auto &trainCharImage : data.trainData)
    {
        Mat flattenRoi;
        trainCharImage.src.convertTo(flattenRoi, CV_32F);
        flattenRoi = flattenRoi.reshape(1, 1);

        trainImages.push_back(flattenRoi);
        trainLabels.push_back(trainCharImage.targetLabel);
    }
    auto start_time2 = chrono::high_resolution_clock::now();
    knn->train(trainImages, ml::ROW_SAMPLE, trainLabels);
    auto end_time2 = chrono::high_resolution_clock::now();
    auto duration2 = chrono::duration_cast<chrono::milliseconds>(end_time2 -
start_time2);
    cout << "Elapsed time(Train): " << duration2.count() << " milliseconds" <<
endl;

    int answerCount = 0;
    int totalCount = 0;
    auto start_time3 = chrono::high_resolution_clock::now();
    for (const auto &testCharImage : data.testData)
    {
        totalCount++;
        Mat flattenRoi;
        testCharImage.src.convertTo(flattenRoi, CV_32F);
        flattenRoi = flattenRoi.reshape(1, 1);

        answerCount += knn->nearest(&flattenRoi, 1);
    }
    auto end_time3 = chrono::high_resolution_clock::now();
    auto duration3 = chrono::duration_cast<chrono::milliseconds>(end_time3 -
start_time3);
    cout << "Elapsed time(Test): " << duration3.count() << " milliseconds" <<
endl;
```

```

    Mat flattenRoi;
    testCharImage.src.convertTo(flattenRoi, CV_32F);
    flattenRoi = flattenRoi.reshape(1, 1);

    Mat res;
    knn->findNearest(flattenRoi, 4, res);
    int predictLabel = res.at<float>(0, 0);
    if (predictLabel == testCharImage.targetLabel)
        answerCount++;
    if (totalCount % 100 == 0)
    {
        cout << "\t" << totalCount << " accuracy : " <<
static_cast<double>(answerCount) / totalCount << endl;
    }
    auto end_time3 = chrono::high_resolution_clock::now();
    auto duration3 = chrono::duration_cast<chrono::milliseconds>(end_time3 -
start_time3);
    cout << "Elapsed time(Test): " << duration3.count() << " milliseconds" << endl;

    cout << "test completed" << endl;
    cout << "\t# of correct answer : " << answerCount << endl;
    cout << "\t# of total test set : " << totalCount << endl;

    double accuracy = static_cast<double>(answerCount) / totalCount;
    cout << "\taccuracy : " << accuracy << endl;

    return 0;
}

```

V-C-2-b. Train KNN with HOG

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <locale>
#include <chrono>
#include "imageLoader.cpp"

using namespace cv;
using namespace std;

int main()
{
    auto start_time1 = chrono::high_resolution_clock::now();
    TrainTestData<CharImage> data = getCharImageData();
    auto end_time1 = chrono::high_resolution_clock::now();
    auto duration1 = chrono::duration_cast<chrono::milliseconds>(end_time1 -
start_time1);
    cout << "Elapsed time(Data Load): " << duration1.count() << " milliseconds" <<
endl;

    Size winSize(38, 42), blockSize(19, 21), blockStride(1, 3), cellSize(19, 21);
    int nbins = 9;

    vector<Mat> trainImages;
    vector<int> trainLabels;
    for (const auto &trainCharImage : data.trainData)
    {
        trainImages.push_back(trainCharImage.src);
        trainLabels.push_back(trainCharImage.targetLabel);
    }

    vector<vector<float>> trainDescriptors;
    HogDescriptor hog(winSize, blockSize, blockStride, cellSize, nbins);
    for (const auto &img : trainImages)
    {
        vector<float> descriptors;
        hog.compute(img, descriptors);
        trainDescriptors.push_back(descriptors);
    }

    Mat trainData(trainDescriptors.size(), trainDescriptors[0].size(), CV_32FC1);
    Mat trainLabelsMat(trainLabels.size(), 1, CV_32SC1);
    for (int i = 0; i < trainDescriptors.size(); ++i)
    {
        for (int j = 0; j < trainDescriptors[i].size(); ++j)
        {
            trainData.at<float>(i, j) = trainDescriptors[i][j];
        }
        trainLabelsMat.at<int>(i, 0) = trainLabels[i];
    }
}
```

```

Ptr<ml::KNearest> knn = ml::KNearest::create();
auto start_time2 = chrono::high_resolution_clock::now();
knn->train(trainData, ml::ROW_SAMPLE, trainLabelsMat);
auto end_time2 = chrono::high_resolution_clock::now();
auto duration2 = chrono::duration_cast<chrono::milliseconds>(end_time2 -
start_time2);
cout << "Elapsed time(Train): " << duration2.count() << " milliseconds" <<
endl;
int answerCount = 0;
int totalCount = 0;
auto start_time3 = chrono::high_resolution_clock::now();
for (const auto &testCharImage : data.testData)
{
    totalCount++;
    vector<float> testDescriptors;
    hog.compute(testCharImage.src, testDescriptors);

    Mat testData(1, testDescriptors.size(), CV_32FC1);
    for (int i = 0; i < testDescriptors.size(); ++i)
    {
        testData.at<float>(0, i) = testDescriptors[i];
    }
    Mat res;
    knn->findNearest(testData, 3, res);
    int predictLabel = res.at<float>(0, 0);
    if (predictLabel == testCharImage.targetLabel)
        answerCount++;
    if (totalCount % 100 == 0)
    {
        cout << "\t" << totalCount << " accuracy : " <<
static_cast<double>(answerCount) / totalCount << endl;
    }
}
auto end_time3 = chrono::high_resolution_clock::now();
auto duration3 = chrono::duration_cast<chrono::milliseconds>(end_time3 -
start_time3);
cout << "Elapsed time(Test): " << duration3.count() << " milliseconds" << endl;

cout << "test completed" << endl;
cout << "\t# of correct answer : " << answerCount << endl;
cout << "\t# of total test set : " << totalCount << endl;

double accuracy = static_cast<double>(answerCount) / totalCount;
cout << "\taccuracy : " << accuracy << endl;

return 0;
}

```

V-C-3. Train SVM

```
#include <iostream>
#include <locale>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include "imageLoader.cpp"
#include <opencv2/ml.hpp>
#include "opencv2/opencv.hpp"
#include <chrono>

using namespace std;
using namespace cv;
using namespace cv::ml;

int main()
{
    //HOGDescriptor(글씨영상하나크기, 블록크기, 블록 이동 크기, 셀 크기)    // 블록 = 2x2 셀
    HOGDescriptor hog(Size(38, 42), Size(38, 42), Size(19, 21), Size(19, 21), 9);

    Ptr<SVM> svm = SVM::create();
    auto start_time1 = std::chrono::high_resolution_clock::now();
    TrainTestData<CharImage> data = getCharImageData();
    auto end_time1 = std::chrono::high_resolution_clock::now();
    auto duration1 =
        std::chrono::duration_cast<std::chrono::milliseconds>(end_time1 - start_time1);
    std::cout << "Elapsed time(Load Data): " << duration1.count() << " milliseconds" << endl << endl;

    Mat trainHOG, trainLabels;
    for (const auto &trainCharImage : data.trainData)
    {
        vector<float> desc;
        hog.compute(trainCharImage.src, desc);

        Mat desc_mat(desc);
        desc_mat.convertTo(desc_mat, CV_32F);

        Mat transpose_desc = desc_mat.t();
        trainHOG.push_back(transpose_desc.reshape(1, 1));
        trainLabels.push_back(trainCharImage.targetLabel);
    }
    svm->setType(SVM::Types::C_SVC);
    svm->setKernel(SVM::KernelTypes::RBF);
    svm->setC(312.5);
    svm->setGamma(0.50625);

    auto start_time2 = std::chrono::high_resolution_clock::now();
    svm->train(trainHOG, ml::ROW_SAMPLE, trainLabels);
    auto end_time2 = std::chrono::high_resolution_clock::now();
```

```

    auto duration2 =
std::chrono::duration_cast<std::chrono::milliseconds>(end_time2 - start_time2);
    std::cout << "Elapsed time(Train): " << duration2.count() << " milliseconds" <<
endl;

    cout << "train completed" << endl;
    cout << "getC: " << svm->getC() << endl;
    cout << "getGamma: " << svm->getGamma() << endl;

    auto start_time3 = std::chrono::high_resolution_clock::now();
    int answerCount = 0;
    int totalCount = 0;
    int t = 0;
    for (const auto &testCharImage : data.testData)
{
    totalCount++;

    vector<float> desc;
    hog.compute(testCharImage.src, desc);

    Mat desc_mat(desc);
    desc_mat.convertTo(desc_mat, CV_32F);

    Mat transpose_desc = desc_mat.reshape(1,1);

    int predictLabel = svm->predict(transpose_desc);
    if (predictLabel == testCharImage.targetLabel)
        answerCount++;
    else if (t++ < 5)
        cout << t << " : " << testCharImage.targetLabel << " " << predictLabel
<< endl;
}
    auto end_time3 = std::chrono::high_resolution_clock::now();
    auto duration3 =
std::chrono::duration_cast<std::chrono::milliseconds>(end_time3 - start_time3);
    std::cout << "Elapsed time(Test): " << duration3.count() << " milliseconds" <<
std::endl;

    cout << "test completed" << endl;
    cout << "\t# of correct answer : " << answerCount << endl;
    cout << "\t# of total test set : " << totalCount << endl;

    double accuracy = static_cast<double>(answerCount) / totalCount;
    cout << "\taccuracy : " << accuracy << endl;

    return 0;
}

```

V-C-4. Evaluate ONNX models

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <locale>
#include <chrono>
#include "imageLoader.cpp"

using namespace cv;
using namespace std;

int main()
{
    auto start_time1 = chrono::high_resolution_clock::now();
    TrainTestData<CharImage> data = getCharImageData();
    auto end_time1 = chrono::high_resolution_clock::now();
    auto duration1 = chrono::duration_cast<chrono::milliseconds>(end_time1 -
start_time1);
    cout << "Elapsed time(Data Load): " << duration1.count() << " milliseconds" <<
endl;

    vector<string> filePaths = {/*onnx 파일의 경로*/};
    for (const auto &filePath : filePaths)
    {
        string fullFilePath = "resources/networks/" + filePath;
        dnn::Net net = dnn::readNetFromONNX(fullFilePath);
        cout << filePath << " start" << endl;

        int answerCount = 0;
        int totalCount = 0;
        auto start_time3 = chrono::high_resolution_clock::now();
        for (const auto &testCharImage : data.testData)
        {
            totalCount++;
            Mat inputBlob = dnn::blobFromImage(testCharImage.src, 1.0, Size(),
Scalar(), true, false);
            net.setInput(inputBlob);
            Mat output = net.forward();

            Mat scores = output.reshape(1, 1);

            Point classIdPoint;
            double confidence;
            minMaxLoc(scores, nullptr, &confidence, nullptr, &classIdPoint);

            int predictLabel = classIdPoint.x;
            if (predictLabel == testCharImage.targetLabel) answerCount++;
        }
        auto end_time3 = chrono::high_resolution_clock::now();
        auto duration3 = chrono::duration_cast<chrono::milliseconds>(end_time3 -
start_time3);
```

```
    cout << "Elapsed time(Test): " << duration3.count() << " milliseconds" <<
endl;

    cout << "test completed" << endl;
    cout << "\t# of correct answer : " << answerCount << endl;
    cout << "\t# of total test set : " << totalCount << endl;

    double accuracy = static_cast<double>(answerCount) / totalCount;
    cout << "\taccuracy : " << accuracy << endl;
}

return 0;
}
```

V-D-1. Train VGG in Colaboratory

```
import math
import time
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(len(ALL_TARGET_CHAR), activation='softmax'))

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

batch_size = 1
image_size = (224, 224)

epochs = 1
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
start1 = time.time()
model.fit(trainImages, train_labels_encoded,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
end1 = time.time()
print(f"Elapsed time. train{end1-start1:.5f} sec")

start2 = time.time()
loss, accuracy = model.evaluate(testImages, testLabels)
print(f'Test loss: {loss:.4f}')
print(f'Test accuracy: {accuracy:.4f}')
end2 = time.time()
print(f"Elapsed time. test{end2-start2:.5f} sec")
```

V-D-2. Train Inception in Colaboratory

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(75,
75, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(len(ALL_TARGET_CHAR), activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

train_labels_encoded = to_categorical(trainLabels,
num_classes=len(ALL_TARGET_CHAR))
model.fit(trainImages, train_labels_encoded, epochs=5, batch_size=32)

final_model_path = '/content/drive/My Drive/COM3034/final_model.h5'
model.save(final_model_path)
```

V-E. Train CNN in Colaboratory

- (1) <https://colab.research.google.com/drive/1vUdfuACjuQNSIAgdwUMyxuo6WlrHtQnh?usp=sharing>
- (2) <https://colab.research.google.com/drive/1zKHpnST-jqLXzc4WGpHONVO95JRgODfS?usp=sharing>
- (3) https://colab.research.google.com/drive/1k7CObZOHdeKTMksbONbROLs5UIZcqh_2?usp=sharing
- (4) https://colab.research.google.com/drive/1TejVc229t019FIotHhdn_JZn6v1eOcqO?usp=sharing
- (5) <https://colab.research.google.com/drive/1-tpEf1gZ6pM01ra45tnF04CwboKvBWaO?usp=sharing>
- (6) <https://colab.research.google.com/drive/1NSI5gXN3ikGY2BHyOsKW3y0rLqfEr5wJ?usp=sharing>
- (7) <https://colab.research.google.com/drive/1ljsHxu0CY9cHRLcher6wl3GayRfoRVxl?usp=sharing>

VI. REFERENCE

ⁱ 사용된 폰트 100종은 아래와 같다. 희야체, 황유체, 호치코리, 호두할아버지, 하우동, 포이리, 포말랑이, 톳, 태선, 큐치폰트, 콩당근밀, 추정, 찬웅, 찌니찐, 지현남자친구체, 지은체, 주안파크, 제비꽃, 재훈, 작은덕골체, 이정우, 윤종우, 윤스, 우희공원체, 우밍글씨, 온글잎, 예리흘림체, 앵두몬, 승월혜광, 소진샘, 뺑글이, 블록이, 부홍채건, 부홍시환, 부홍성수, 부홍도윤엘, 병수필기, 박운슬체, 미강, 문순체, 몽글이폰트, 모학외할머니, 모학사이니즈백, 모모는달행피라이브, 망갈판디, 동구승훈, 더그레잇하영, 대강대강, 나영70체, 나는나대로, 꿈빛, 그농이체, 구슬, 공지, 곰재민날림, 경임, 강변대나무, 가윤체, 24용빈, 가람연꽃, 고딕아니고고딩, 곱신, 규리의일기, 규폰트, 금은보화, 기쁨밝음, 김유이, 꽃내음, 나는이겨낸다, 나무정원, 나의봉글이체, 나의아내손글씨, 남고생폰트, 노력하는동희, 느릿느릿체, 다진, 다채사랑, 다행, 등근인연, 디호봉, 딘두덩, 말랑몽글, 무궁화, 박도현사랑해, 사이사이, 서아글씨체, 성실, 세아, 수연, 수영, 쏘쏘체, 외할머니글씨, 올울체, 정덩이, 정정당당, 지혜, 체가은체, 탁1, 하은체, 해인이.

ⁱⁱ MNIST 숫자 분류 정확도 99.5% 모델, 김형준, gisdeveloper.go.kr, 최종 수정일 2019-11-04, <http://www.gisdeveloper.co.kr/?p=8436>