**TERM PROJECT**

**REPORT**

**과 목 : 인공지능설계실습1**

**과 제 명 : 텀프로젝트**

**결과보고서**

**담당교수 : 채승호**

**학 과 : 전자공학과**

**학 번 : 2018144023**

**이 름 : 이상호**

**제 출 일 : 2023. 06.05**

**목차**

1. **주제 배경이론** 
   1. **CNN 개념**
   2. **모델 구조 및 학습 방법**
2. **소스코드 설명** 
   1. **Data labeling**
   2. **Data preprocessing**
   3. **Model**
3. **구현 결과 및 분석**
4. **결론**
   1. **결과 요약**
   2. **한계점 및 개선방안**

**1. 주제 배경 이론**

**A. CNN 개념**

CNN : image를 분류하기 위한 개발된 Network

CNN 구조 예시. NN은 크게 2가지로 나눠진다. 하나는 특징을 추출하는 feature extraction로 이루어져 있고, 다른 하나는 feature extraction를 통과한 이후에 결과값을 도출해 주는 Classification으로 이루어져 있다. 이때 Convolution layer와 Pooling layer가 섞여 있는 것이 feature extraction이고, fully-connected layer로 이루어진 것이 Classification이다.

도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

컨볼루션 레이어에선 입력된 이미지에 필터(커널)을 슬라이딩 윈도우 기법을 적용하여 특징을 추출한다. 오른쪽 그림은 필터 적용 후 결과물인 feature map.

**텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명**

Pooling layer의 목적은 특징을 강화한다. Pooling layer의 종류에는 max pooling, average pooling, overlapping Pooling이 있다

Max pooling

- 계산양이 감소하기 때문에 연산부하가 줄어든다.

- size를 줄이는 것이기 때문에 필연적으로 오차가 발생하므로 오버피팅을 약간 줄여준다.

**B. 모델 구조 및 학습방법**

아래 그림과 같은 2계층 CNN을 사용한다.

도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

학습방법 : Adam optimizer를 사용하여 경사하강법을 진행한다.

Adam (Adaptive Moment Esimation) : Momentum과 RMSProp 두가지를 섞어 쓴 알고리즘이다. 진행하던 속도에 **관성**을 주고, 최근 경로의 곡면의 변화량에 따른 **적응적 학습률**을 갖는 알고리즘이다. 매우 넓은 범위의 아키텍처를 가진 서로 다른 신경망에서 잘 작동한다는 것이 증명되어, 일반적 알고리즘에 현재 가장 많이 사용되고 있다.

폰트, 타이포그래피, 서예, 화이트이(가) 표시된 사진

자동 생성된 설명하이퍼 파라미터로 이 3가지를 주로 사용한다.

가중치 초기화 방법

Xavier Initialization : sigmoid 계열의 activation function을 사용할 때, 가중치를 초기화하는 방법. 입력 데이터의 분산이 출력 데이터에서 유지되도록 가중치를 초기화

He Initialization : 활성 함수가 ReLU일 때, Xavier 초기화를 적용해보면, 입력 데이터가 계층을 통과하면서 분산이 점점 줄어들어 출력이 0이되는 문제를 해결하기 위해 나온 가중치 초기화 방법.

활성화 함수 : Leaky ReLU

* 기존 ReLU함수는 뉴런의 가중치가 바뀌어서 훈련 세트에 있는 모든 샘플에 대해 입력의 가중치 합이 음수가 되면 그레이디언트가 0이라 경사 하강법이 더 일어나지 않음.
* 위 문제를 해결하기 위해 변종 ReLU함수를 사용

폰트, 텍스트, 타이포그래피, 서예이(가) 표시된 사진

자동 생성된 설명

라인, 스크린샷, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

배치 정규화(batch normalization) :

* 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 각 배치별로 평균과 분산을 이용해 정규화하는 것
* batch 단위나 layer에 따라서 입력 값의 분포가 모두 다르지만 정규화를 통하여 분포를 zero mean gaussian 형태로 만듦.
* 평균은 0, 표준 편차는 1로 데이터의 분포를 조정할 수 있다.

**2. 소스코드 설명**

**A. 데이터 라벨링**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

분류를 마친 사진 폴더를 구글드라이브에 업로드 후 코랩에서 읽어와 라벨링을 진행한다.

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* cv2

*import* os

*import* random

*# Average 필터 사용시 속도 개선을 위해 just in compiler 사용.*

*from* numba *import* njit

class\_names = ['buildings', 'forest', 'glacier', 'mountain', 'sea']

def load\_data(*categories*):

*# gpu 사용을 위해 코랩에서 진행함.*

    path = os.getcwd()+"/drive/MyDrive/colab"

    images = []

    labels = []

*for* idx, category *in* enumerate(*categories*):

        category\_path = os.path.join(path, category)

*# 입력 받은 클래스 이름의 폴더가 있으면 내부의 사진들을 읽어온다.*

*if* os.path.isdir(category\_path):

*for* image\_name *in* os.listdir(category\_path):

                image\_path = os.path.join(category\_path, image\_name)

*# 사진 읽기 (외부 라이브러리 사용)*

                image = cv2.imread(image\_path)

*# opencv는 bgr컬러를 사용하고 pyplot은 rgb를 사용하기 때문에*

*# 색공간을 바꿔준다.*

                image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

*# 이미지 크기 조정*

                image = cv2.resize(image, (150, 150))

*# 이미지 타입 변경 (floa32 사용)*

                image = image.astype('float32')

                images.append(image)

                labels.append(idx)

*# 이미지와 라벨을 numpy 배열로 변환*

    images = np.asarray(images)

    labels = np.asarray(labels, *dtype*='int32')

*return* (images, labels)

*# 원본데이터*

raw\_data = load\_data(class\_names)

**B. Data preprocessing**

1. 데이터 정규화

*# 정규화 함수*

def normalize\_data(*data*):

    images, labels = *data*

*# 이미지 BGR 최댓값이 255이므로 255로 나누어 정규화 시킨다.*

    images\_normalized = images / 255.0

*return* (images\_normalized, labels)

normalized\_data = normalize\_data(raw\_data)

2. Average Filter

@njit

def apply\_padding(*image*, *pad\_size*):

*# 필터와 이미지 컨볼루션 시 이미지 사이즈가 줄어드는 것을 막기 위해 원본이미지 크기를 늘린다.*

    padded\_image = np.zeros((*image*.shape[0] + *pad\_size* \* 2, *image*.shape[1] + *pad\_size* \* 2, *image*.shape[2]))

    padded\_image[*pad\_size*:*pad\_size*+*image*.shape[0], *pad\_size*:*pad\_size*+*image*.shape[1], :] = *image*

*return* padded\_image

*# 3중 for문을 쓰기 때문에 매우 느리다. 동적 컴파일러로 속도를 개선한다.*

@njit

def apply\_average\_filter\_1time(*image*, *filter\_size*=3):

*# 패딩 후 컨볼루션*

    pad\_size = *filter\_size* // 2

    padded\_image = apply\_padding(*image*, pad\_size)

    filtered\_image = np.zeros\_like(*image*)

*for* i *in* range(*image*.shape[0]):

*for* j *in* range(*image*.shape[1]):

*for* k *in* range(*image*.shape[2]):

*# 평균 필터이기 때문에 컨볼루션 대신 np.mean으로 평균 취한다.*

                filtered\_image[i, j, k] = np.mean(padded\_image[i:i+*filter\_size*, j:j+*filter\_size*, k])

*return* filtered\_image

def apply\_average\_filter(*images*, *filter\_size*=3):

    filtered\_images = []

*for* image *in* *images*:

        filtered\_image = apply\_average\_filter\_1time(image, *filter\_size*)

        filtered\_images.append(filtered\_image)

*return* np.array(filtered\_images)

normalized\_images, labels = normalized\_data

*# 이미지만 분리하여 평균필터를 통과시킨다.*

filtered\_images = apply\_average\_filter(normalized\_images)

filtered\_data = (filtered\_images, labels)

3. Image Augmentation

상하반전, 좌우반전을 사용하여 3배로 증강한다.

def flip\_horizontal(*image*):

*# 좌우 반전*

*return* np.flip(*image*, *axis*=1)

def flip\_vertical(*image*):

*# 상하반전*

*return* np.flip(*image*, *axis*=0)

def augment\_images(*images*, *labels*):

    augmented\_images = []

    augmented\_labels = []

*for* i, image *in* enumerate(*images*):

        original\_image = image

        horizontal\_flipped = flip\_horizontal(image)

        vertical\_flipped = flip\_vertical(image)

        augmented\_images.extend([original\_image, horizontal\_flipped, vertical\_flipped])

        augmented\_labels.extend([*labels*[i], *labels*[i], *labels*[i]])

*return* np.array(augmented\_images), np.array(augmented\_labels)

X\_train, Y\_train = augment\_images(filtered\_images,labels)

4. Data Shuffle

def shuffle\_data(*images*, *labels*, *seed*=None):

*# 시드값을 주지 않은경우 랜덤으로 섞는다.*

*if* *seed* is not None:

        np.random.seed(*seed*)

*# arange로 생성한 배열을 인덱스로 사용한다.*

    indices = np.arange(*images*.shape[0])

    np.random.shuffle(indices)

    shuffled\_images = *images*[indices]

    shuffled\_labels = *labels*[indices]

*return* shuffled\_images, shuffled\_labels

X\_train, Y\_train = shuffle\_data(X\_train, Y\_train)

shuffled\_train\_data = (X\_train, Y\_train)

5. 전저리 결과 확인

def display\_examples(*class\_names*, *images*, *labels*):

*# 특정 사진을 골라 출력한다.*

    fig = plt.figure(*figsize*=(10,10))

    fig.suptitle("some examples of images of the dataset", *fontsize* = 16)

*for* i *in* range(25):

        plt.subplot(5,5,i+1)

        plt.xticks([])

        plt.yticks([])

        plt.grid(False)

        plt.imshow(*images*[i],*cmap*=plt.cm.binary)

        plt.xlabel(*class\_names*[*labels*[i]])

    plt.show()

display\_examples(class\_names,X\_train,Y\_train)

결과 :

텍스트, 콜라주, 스크린샷, 야외이(가) 표시된 사진

자동 생성된 설명

**C. Model**

첫 번째 Convoluiton Layer

* 필터 수 : 256
* 필터 크기 9 \* 9
* 활성화 함수 : LeakyReLU (alpha = 0.2)

첫 번째 Max Pooling Layer

* Pool size : 5 \* 5

두 번째 Convoluiton Layer

* 필터 수 : 512
* 필터 크기 7 \* 7
* 활성화 함수 : LeakyReLU (alpha = 0.2)

두 번째 Max Pooling Layer

* Pool size : 3 \* 3

Fully Connected Layer

* 노드수 : 512
* 활성화 함수 : LeakyReLU (alpha = 0.2)

Output Layer

* 노드수 : 5
* 활성화 함수 : softmax

학습률 : 1e-4

콜백 메서드를 정의해서 val\_accuracy가 최고일 때 model을 저장한다.

*from* keras.models *import* Sequential,load\_model

*from* keras.layers *import* Dense,Dropout,Flatten,Conv2D,MaxPool2D,BatchNormalization,Activation,LeakyReLU

*from* tensorflow.keras.optimizers *import* Adam

*import* tensorflow *as* tf

*from* tensorflow.keras.callbacks *import* ModelCheckpoint

*with* tf.device('/gpu:0'):

*#GPU 사용한다.*

  model = Sequential()

  model.\_name = "Sangho\_CNN"

  model.add(Conv2D(256, *kernel\_size*=(9, 9), *input\_shape*=(150, 150, 3)))

  model.add(LeakyReLU(*alpha*=0.2))

  model.add(MaxPool2D(*pool\_size*=(5,5)))

  model.add(Conv2D(512, (7, 7)))

  model.add(LeakyReLU(*alpha*=0.2))

  model.add(MaxPool2D(*pool\_size*=(3,3)))

  model.add(Flatten())

  model.add(Dense(512))

  model.add(LeakyReLU(*alpha*=0.2))

  model.add(Dense(5, *activation*='softmax'))

*# Adam 옵티마이저 객체 생성 및 학습률 설정*

  adam\_optimizer = Adam(*learning\_rate*=1e-4)

*# 모델 컴파일에 옵티마이저 객체 전달*

  model.compile(*loss*='sparse\_categorical\_crossentropy', *optimizer*=adam\_optimizer, *metrics*=['accuracy'])

  model.summary()

  checkpoint\_filepath = 'best\_model.h5'

*# val\_accuracy가 최대일 때 모델을 저장하기 위해 콜백 메서드를 정의한다.*

  model\_checkpoint\_callback = ModelCheckpoint(

*filepath*=checkpoint\_filepath,

*save\_weights\_only*=False,

*monitor*='val\_accuracy',

*mode*='max',

*save\_best\_only*=True)

모델 요약

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

학습

배치 크기 : 128

history = model.fit(X\_train, Y\_train, *epochs*=50, *batch\_size*=128,*callbacks*=model\_checkpoint\_callback,*validation\_split* = 0.2)

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

시각화 코드

*# epoch마다 val\_loss와 val\_acc의 변화량 그래프*

fig, (ax1, ax2) = plt.subplots(1, 2, *figsize*=(12, 5))

*# 옵션 값을 설정하셨으니 그래프 이외 주석은 빼도됩니다.*

ax1.plot(history.history['loss'], *label*='Train Loss',*marker*='^')

ax1.plot(history.history['val\_loss'], *label*='Validation Loss',*marker*='^')

ax1.set\_title('Validation Loss per Epoch')

ax1.set\_xlabel('Epoch')

ax1.set\_ylabel('Loss')

ax1.legend()

ax2.plot(history.history['accuracy'], *label*='Train Accuracy',*marker*='^')

ax2.plot(history.history['val\_accuracy'], *label*='Validation Accuracy',*marker*='^')

ax2.set\_title('Validation Accuracy per Epoch')

ax2.set\_xlabel('Epoch')

ax2.set\_ylabel('Accuracy')

ax2.legend()

plt.show()

**3. 구현 결과 및 분석**

학습과정 그래프

텍스트, 그래프, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

Validation loss는 점차 감소하다가 약 35 epoch부근부터 증가한다.

Validation accuracy는 약 40 epoch 부근에서 77.5%로 최고 정확도를 보였으며 이후 점차 감소했다.

**4. 결론**

**A. 결과 요약**

Building, forest, mountain, sea, 그리고 glacier 클래스를 포함하는 200장의 사진을 증강하여 총 600장의 이미지를 만들었다. 이 중 480장은 훈련세트로, 그리고 120장은 검증세트로 사용해 77.5%의 정확도로 이미지 분류에 성공했다.

**B. 한계점 및 개선 방안**

**한계**

* 기본 제공되는 데이터 셋의 크기가 너무 적음.
* 본래 검증세트에 augmentation 적용한 사진은 쓰지 않으나 본 프로젝트는 훈련 조건에 validation\_split 인자를 주게 되어 있어 augmentation 적용한 사진이 검증 세트로 사용됨. 일반화 오차가 발생할 수 있음.
* 가중치 초기화 방법이나 배치 정규화 등 여러 기법들을 적절하게 적용해보지 못했음.
* CNN 모델구조가 2계층으로 고정되어 성능의 한계가 있음.

**개선 방안**

* 데이터 셋의 크기를 늘린다.
* 다양한 기법을 적용해본다 .
* 모델 구조를 바꾼다.