

최종 보고서

과제	이미지 노이즈 제거 클래스 분류기
수업	머신 러닝 (4-6)
지도교수	김인겸 교수님
팀	셀프러닝
연구참여자	팀장 김찬영(20190895) : 프로젝트 총괄 및 보고서 작성
	팀원 우상욱(20190919) : 프로그래밍, 회의록 작성, ppt 제작
	팀원 지성원(20190948) : 프로그래밍, 자료조사
	팀원 허진환(20190954) : 프로그래밍 총괄
제출일자	2023.06.12

목차

1. 설계과제 제목 및 소개	1
2. 설계과제 추진 목적.....	4
3. 설계과제의 필요성.....	4
4. 설계과제의 목표.....	4
5. 설계과정	4
5.1 개요	2
5.2 설계기초이론	2
5.2.1 CNN(합성곱 신경망)	6
5.2.2 오토인코더	6
5.2.3 코드레이어	6
5.2.4 오토인코더의 수학적 원리	6
5.2.5 재구성 오차	6
6. 제작	4
6.1 제작과정.....	2
6.1.1 이미지 노이즈 제거.....	6
6.1.2 Classification 수행	6
6.2 제작 시 문제점 및 개선사항.....	2
6.2.1 이미지 노이즈 제거기의 손실함수, 정확도.....	6
6.2.2 노이즈 제거 성능에 따라 이미지 분류기의 정확도 편차 증가.....	6
7. 시험	4
8. 평가	4
9. 추진체계	4
10. 설계 추진 일정	4
11. 결론	4

1. 설계과제 제목 및 소개



이미지 노이즈 제거 클래스 분류기

머신 러닝을 통하여 노이즈가 포함된 이미지를 classification 하는 모델을 설계한다. 성능의 증가를 위해 오토인코더를 사용하여 이미지의 노이즈 제거 후, '자동차', '비행기', '얼굴'과 같은 101 가지 다른 클래스의 이미지를 분류하기 위한 알고리즘을 개발한다. Sequential 모델을 사용하여 인공적으로 특징을 추출하고, 모델을 훈련하여 새로운 입력 이미지의 클래스를 분류하고 예측하는 것을 목표로 한다. 이미지의 특징을 추출하고 학습하기 위해 합성곱 신경망(CNN)을 사용하고, 모델을 분류하기 위해 훈련시킨다.

2. 설계과제 추진 목적

- ① 합성곱 신경망(CNN) 기본 개념 이해
- ② 인공지능에서의 파이썬 활용
- ③ CNN 기반의 오토인코더 모델 설계
- ④ 오토인코더 모델을 통한 이미지 잡음 제거
- ⑤ CNN 기반의 이미지 클래스 분류기 모델 설계
- ⑥ 이미지 클래스 분류기 모델을 통한 이미지 클래스 분류
- ⑦ 오토인코더와 이미지 클래스 분류기 모델의 합성
- ⑧ CNN 활용 범위와 필요성에 대한 인식 제고

3. 설계과제의 필요성

다양한 영상 처리에 있어 이미지 데이터의 처리는 영상처리의 기본으로, 통신 과정 중 데이터에 끼는 노이즈는 항상 존재한다고 할 수 있다. 노이즈가 낀 데이터를 수신해도 해당 데이터를 제대로 해석하여 수신한다면, 통신 과정에서 비용을 들이지 않고도 BER 성능을 높일 수 있으므로, 데이터를 처리하고 분류하는 데 있어 cnn 기반의 오토인코더 모델을 통해 노이즈를 제거한 모델을 S/W 적으로 구현하는 것은 H/W 적으로의 현저한 비용 절감을 가져올 수 있다. 또한, 이렇게 수신한 데이터를 클래스화하여 분류하는 작업을 통해 데이터 분류에 소모되는 인력 및 시간을 절약할 수 있고, 더욱 다양한 방향으로 데이터가 이용되어질 수 있으며, 다양한 문제 해결 (의료 진단 또는 자율주행차에서의 표지판 인지) 등에 쓰일 수 있으므로 설계 과제를 추진하게 되었다.

4. 설계과제의 목표

- ① 이미지 데이터셋 (cifar-10) 로드 및 잡음 추가
- ② 잡음 제거 기능 오토인코더 모델 설계
- ③ 오토인코더 모델 훈련을 통한 예측 및 2 가지 손실함수(이진교차엔트로피, MSE)에 대한 비교
- ④ 이미지 클래스 분류기 모델 설계
- ⑤ 이미지 클래스 분류기 모델 훈련을 통한 예측 (이미지 클래스 분류 수행)
- ⑥ 오토인코더 모델과 이미지 클래스 분류기 모델의 합성

(table01_ 현실적 제한 요소)

현실적 제한 요소	내용
경제	모델 훈련에 사용되는 H/W 비용에 대한 문제를 인식하여야 한다.
편리	이미지 처리에 있어 기존 사람이 하던 방식만큼 정확하고, 사람이 하던 방식보다 편리해야 한다.
윤리	활용되는 CNN 이미지 처리 기술이 불법적이나 비윤리적으로 이용되어서는 안된다.
사회	인공지능의 발달로 인한 일자리 감소 문제를 인식하여야 한다.

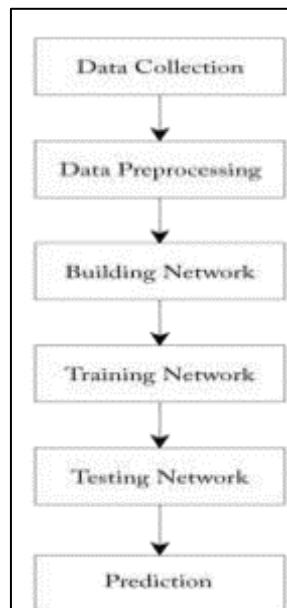
5. 설계과정

5.1 개요

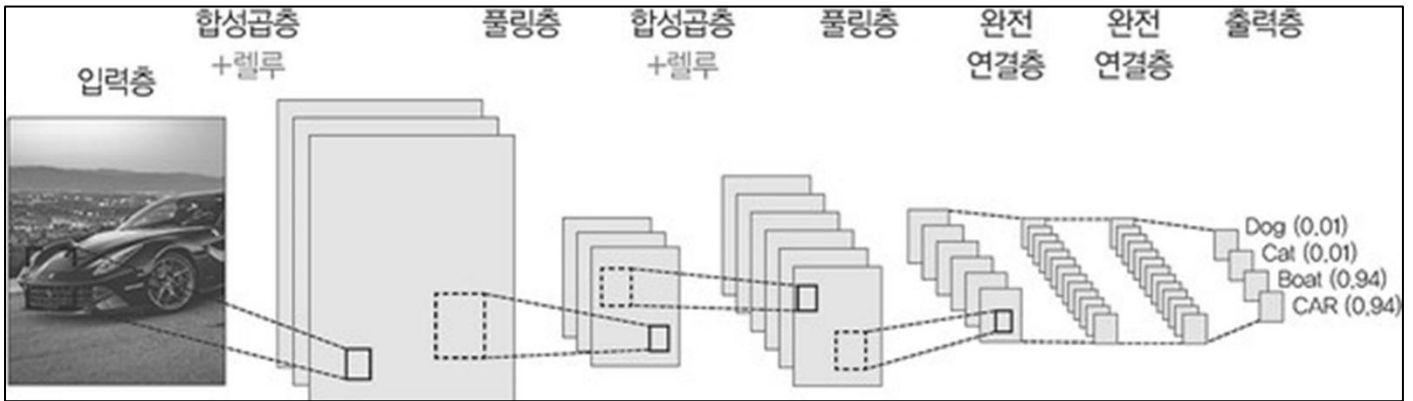
- ① 라이브러리 불러오기 및 이미지 출력
- ② 이미지 정규화 및 가공, 이미지 노이즈 추가
- ③ 오토인코더 모델 생성
- ④ 오토인코더 모델 학습
- ⑤ 오토인코더 모델 성능 파라미터 및 결과 출력

5.2 설계 기초이론

5.2.1 CNN(합성곱 신경망)



신경망: 신경망의 기본 단위는 퍼셉트론이다. 입력 레이어에서 이 퍼셉트론으로 입력이 들어가고, 이 퍼셉트론은 결과를 출력한다. 본 프로젝트에서의 입력은 이미지로 들어가게 되는데, 이 이미지는 훈련 및 테스트 데이터셋으로 나뉘진다. 그런 다음 훈련 데이터셋으로 모델을 학습시킨다. 그 후 테스트 데이터셋으로 모델의 성능을 테스트한다. 프로세스의 마지막 부분은 임의의 입력 데이터의 클래스를 예측하는 것이다.

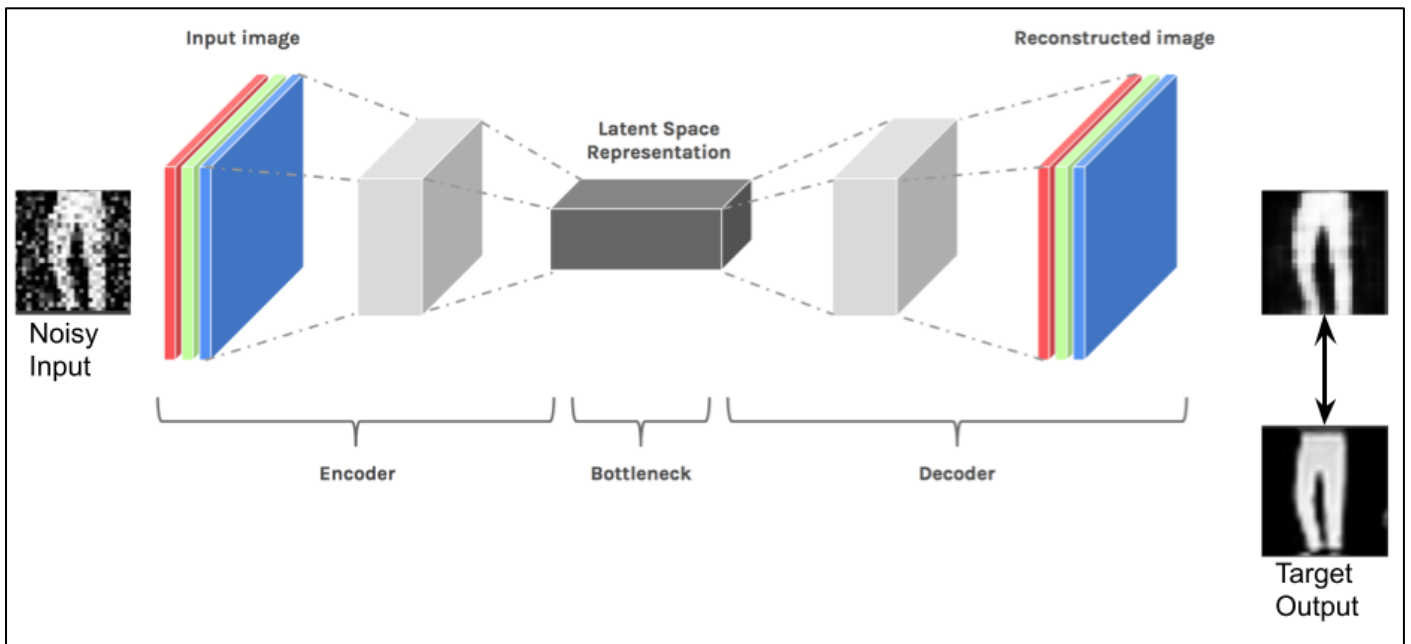
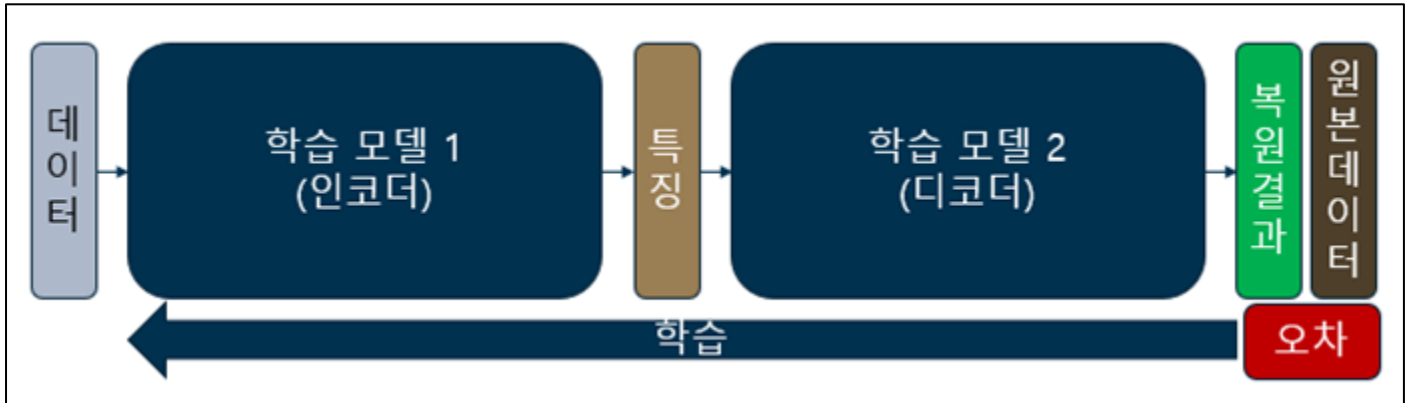


- ① 입력층에서의 이미지 정규화: 첫 번째 단계는 입력 이미지를 정규화하는 것으로, 이미지는 RGB 값의 집합으로 이루어져 있으며, 이 값들은 채널로써 주어진다. 채널의 값이 1 이면 그레이스케일, 3 이면 RGB 이다. 따라서 이러한 값을 공통 범위로 정규화해야 한다. 첫 번째 레이어에서 이미지를 RGB 로 정규화한다 (픽셀 단위).
- ② 활성화 함수: 입력을 임계값으로 처리하기 위해 ReLU 를 사용한다. 합성곱 계층에서 정의된다. ReLU 함수는 음수를 0 으로 만들고, 양수는 그대로 유지하는 함수이다.
- ③ 합성곱 계층 : 입력 데이터로부터 이미지의 특징을 추출하는 역할을 한다. 각 합성곱 층은 여러 개의 필터 또는 커널로 구성되어 있다. 필터는 입력 데이터에 대해 지역적인 패턴을 탐지하는 작은 윈도우로, 입력 데이터를 윈도우마다 합성곱 연산을 수행하여 특징 맵을 생성한다. 합성곱 층의 출력에서 활성화 함수인 ReLU 가 사용된다.
- ④ 풀링 층 : 특징 맵의 크기를 줄이거나 중요 정보를 강조하는 역할을 한다. 일반적으로 최대 풀링이 사용되고, 최대 풀링은 윈도우 내에서 가장 큰 값을 선택하여 특징을 강조하며, 나머지 값은 버리는 방식으로 동작한다.
- ⑤ 완전 연결 층 : 추출된 특징을 기반으로 클래스의 예측을 수행한다. 풀링 층의 출력을 일렬로 펼친 후 완전 연결 층에 전달한다. 완전 연결 층은 일반적인 다층 퍼셉트론 구조로 구성되어 있으며, 출력층은 클래스 분류를 위한 뉴런 수와 활성화 함수를 설정하여 최종 예측을 수행하게 된다.

CNN 은 합성곱층과 풀링층을 거치면서 입력 이미지의 주요 특성 벡터를 추출하여, 완전연결층을 거치면서 1 차원 벡터로 변환되고, 마지막으로 출력층에서 활성화 함수인 소프트맥스 함수를 사용하여 최종 결과가 출력된다.

5.2.2 오토인코더

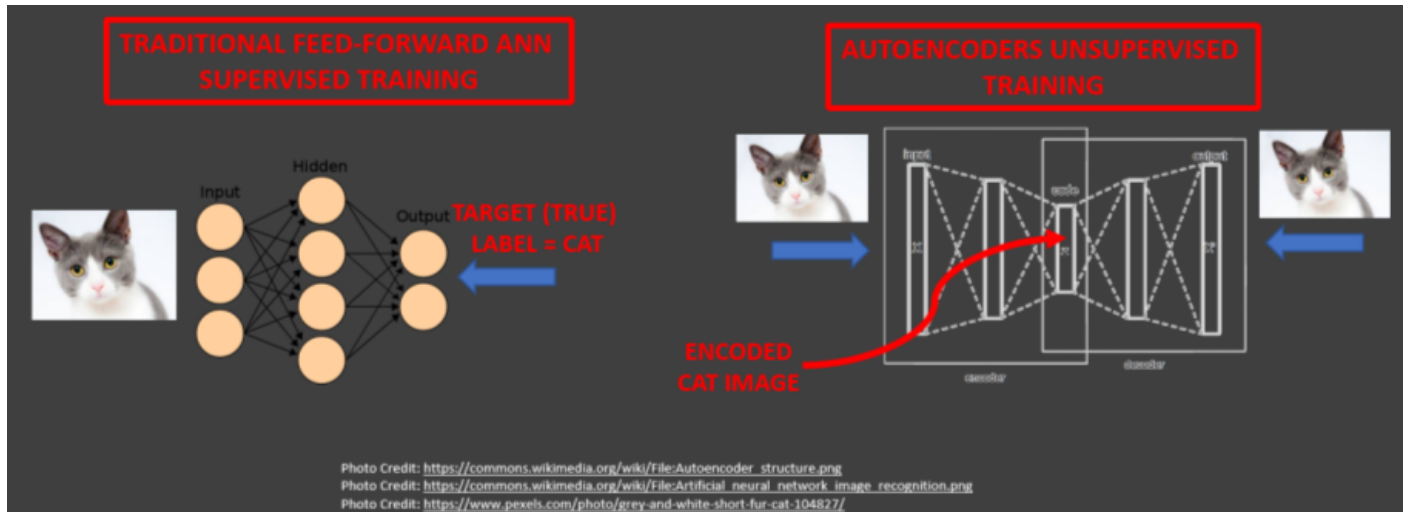
데이터 인코딩(representation learning) 작업을 수행하는 인공 신경망의 한 유형으로, 오토인코더의 특징으로는 입력 데이터를 입력 및 출력으로 사용한다.



본 프로젝트에서는 (3, 3) 윈도우 크기의 32 개 filter 를 사용하여 합성곱 연산 및 출력(ReLU 함수 사용)하여 특성을 추출한 후, 풀링층으로 진입해 특징 맵 크기를 1/2 로 줄이고 특징을 강조한 뒤, 2 차 합성곱층으로 진입하여 인코딩 과정을 거치는 인코더 부분과 입력 데이터의 각 차원을 2 배로 확장하고 이미지 특성을 3 개의 필터로 추출하여 디코딩 과정을 거치는 디코더 부분으로 구성된 오토인코더 모델을 설계하고 모델 훈련 및 예측에 활용하였다.

5.2.3 코드레이어

오토인코더는 네트워크에 병목 현상을 만들어 줌으로서 동작한다. 이 병목 현상은 네트워크가 원래 입력의 압축된(encoded) 버전을 생성하도록 한다. 오토인코더는 입력 데이터와 상관관계가 있는 경우에 잘 작동한다.(입력 데이터가 모두 독립적인 경우에는 잘 작동하지 않을 수 있다)

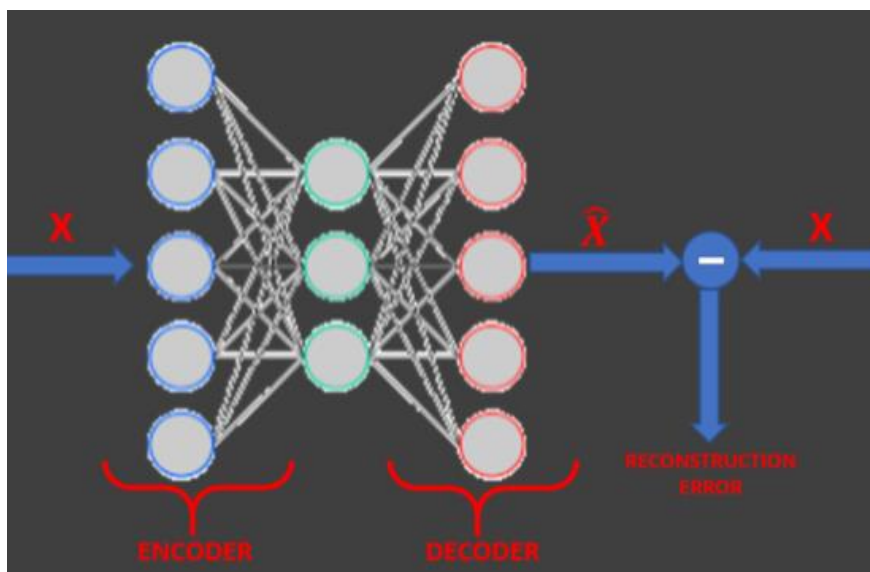


5.2.4 오토인코더 수학적 원리

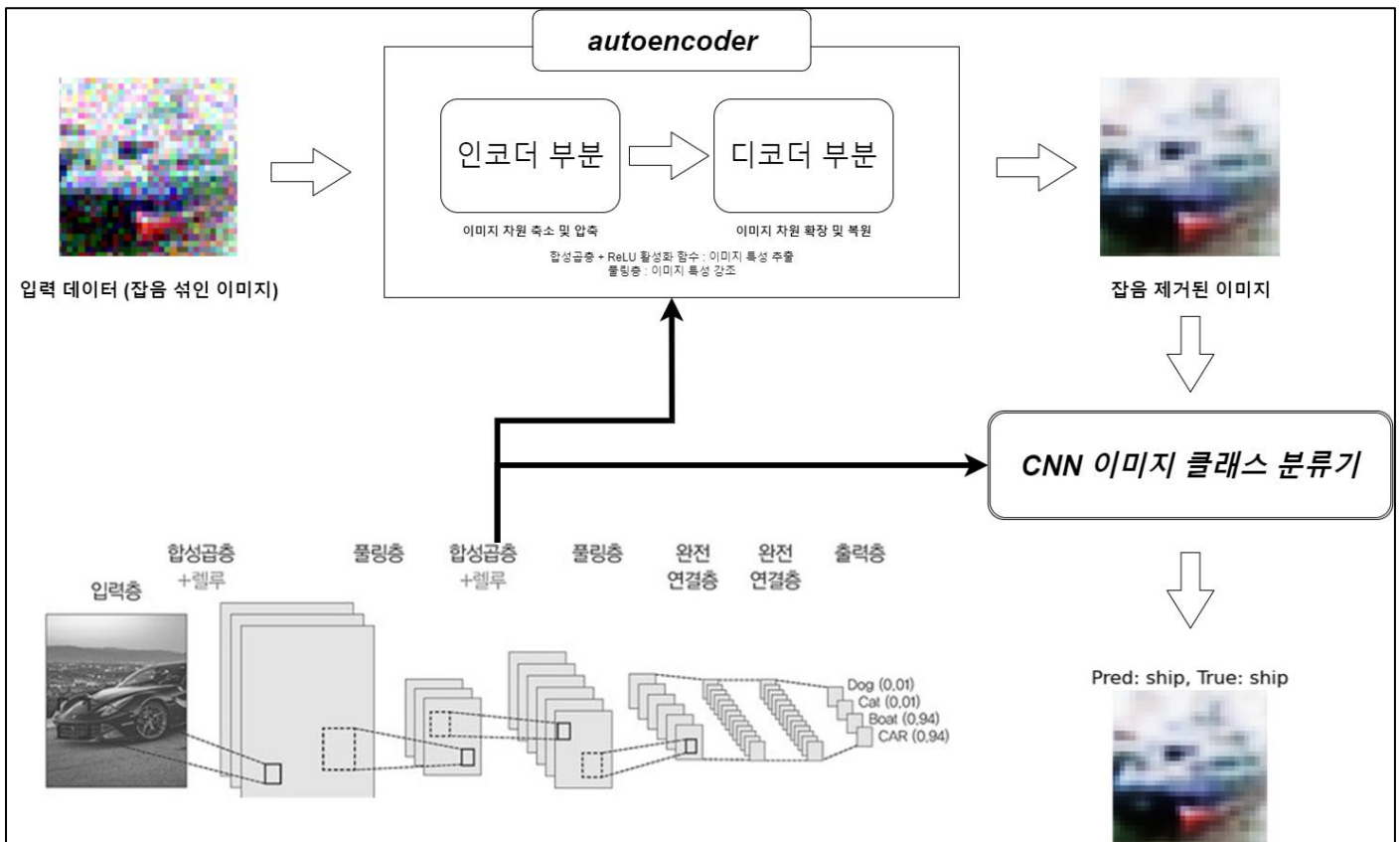
Encoder: $h(x) = \text{sigmoid}(W * x + b)$

Decoder: $\hat{x} = \text{sigmoid}(W^* * h(x) + c)$

5.2.5 재구성 오차: 오토인코더의 목표는 입력 X 와 네트워크 출력 X^{\wedge} 사이의 재구성 오차를 최소화하는 것이다. 오토인코더의 차원 축소(잠재 공간)는 선형 활성화 함수를 사용하는 경우 PCA(주성분 분석)와 매우 유사하다.



5.3.1 기능 블록도



5.3.2 기능적 요구사항 명세서

ID	요구사항	내용	설명	우선순위
No_01	이미지 데이터셋 로드	이미지 노이즈 추가/제거 및 클래스화를 위한 이미지 데이터셋 로드	클래스가 10 개인 이미지 데이터셋인 Cifar-10 데이터셋을 훈련용/검증용 데이터로 나누어 로드한다.	1
No_02	이미지 잡음 추가	로드된 cifar 데이터셋에 이미지 잡음 추가	오토인코더의 입력 데이터로 들어가 잡음 제거가 수행되도록 하기 위해 임의의 노이즈 강도를 설정해 각 이미지에 잡음을 추가한다.	2

No_03	오토인코더 모델 설계	이미지 잡음 제거를 위한 오토인코더 모델 설계	ReLU 활성화 함수를 사용한 합성곱층과 이미지 크기를 1/2 로 줄이는 풀링층 이후 다시 합성곱층으로 이어지는 인코더 부분과 이미지 크기를 원복하고, 합성곱층을 사용하여 이미지 특성 추출하여 압축으로부터 복원하는 디코더 부분을 가진 오토인코더 모델을 설계한다.	3
No_04	오토인코더 모델 훈련	설계된 오토인코더 모델 훈련	잡음이 추가된 이미지 훈련용 데이터에 대해 원본 훈련용 데이터를 비교를 위한 정답 데이터로 사용해 오토인코더 모델을 훈련한다.	4
No_05	잡음 제거 이미지 출력	잡음이 제거된 이미지 출력	잡음이 추가된 이미지가 오토인코더 모델 훈련을 통해 예측 결과로 잡음 제거 이미지를 반납하고, 이를 정답 데이터(원본 훈련용 데이터)와 같이 출력함으로써 잡음 제거 수행 결과를 확인한다.	5
No_06	잡음 제거 이미지 데이터화	잡음이 제거된 이미지 훈련용/검증용 분류	잡음이 추가된 이미지에 대해 이미지 분류를 수행해야 하므로 훈련용/검증용 데이터로 나누어야 한다.	6
No_07	이미지 클래스 분류기 모델 생성	잡음 제거 이미지 데이터셋에 대한 이미지 클래스 분류를 위한 이미지 분류기 모델 생성	ReLU 활성화 함수를 사용한 합성곱층과 풀링층을 3번 통과하고 차원 축소를 거친 후 완전 연결층으로 이어진 후 확률 값을 출력하는 이미지 클래스 분류기 모델을 생성한다.	7
No_08	이미지 클래스 분류기 모델 훈련	설계된 이미지 클래스 분류기 모델 훈련	(잡음 제거 과정을 거친)훈련용 데이터에 대해 정답 데이터인	8
No_09	클래스 분류 결과 출력	잡음 제거 과정을 거친 데이터의 클래스 분류 결과 출력	잡음 제거 과정을 거친 데이터의 클래스 분류 결과를 이미지와 예측/정답 클래스가 함께 나오도록 출력	9

6. 제작

6.1 제작과정

6.1.1 이미지 노이즈 제거기

STEP #1: 라이브러리 및 데이터셋 불러오기

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
import seaborn as sns
import random
import os
```

✓ 0.0s

Python

```
# keras 라이브러리 불러오기
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras import regularizers
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint
```

✓ 0.0s

Python

```
# 이미지 분류기를 위해 다양한 카테고리를 가진 cifar10 데이터셋 사용
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

✓ 0.6s

Python

```
X_train.shape
```

✓ 0.0s

Python

(50000, 32, 32, 3)

```
X_test.shape
```

✓ 0.0s

Python

(10000, 32, 32, 3)

이미지 분류를 위해 다양한 카테고리를 가진 이미지 dataset 인 cifar10 을 사용하고, 훈련용/검증용 데이터로 구분하였다.

STEP #2: 데이터 이미지 출력

```
# 이미지 여러 개 출력
# 15 * 15 형식으로 출력
W_grid = 15
L_grid = 15

# subplots()함수 fig, axes 객체 리턴
fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))

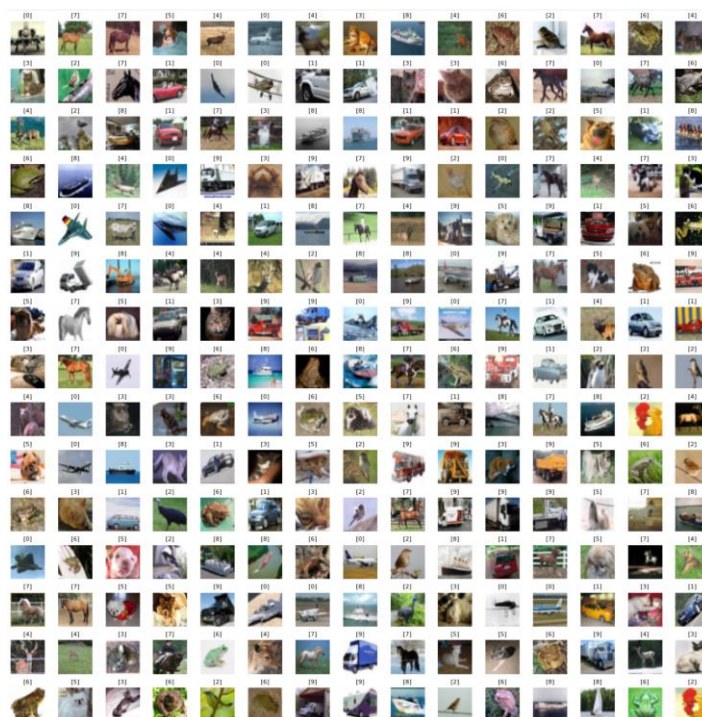
# 15 * 15 행렬을 255크기의 배열로 바꿈
axes = axes.ravel()

# 훈련용 데이터셋 크기 반환
n_training = len(X_train)

# 0 ~ n_training 까지 중 랜덤한 인덱스의 데이터 이미지 출력
for i in np.arange(0, W_grid * L_grid):
    # 랜덤한 인덱스 지정
    index = np.random.randint(0, n_training)
    # 인덱스에 해당하는 데이터 이미지 출력
    axes[i].imshow( X_train[index] )
    axes[i].set_title(y_train[index], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```

✓ 15.3s



이미지 데이터셋이 제대로 로드되어 훈련용 데이터로 들어갔는지 확인하기 위해 데이터 이미지를 랜덤한 인덱스로 접근하여 위와 같이 15*15 형식으로 출력(출력 화면 밑으로 스크롤 시 15 * 15 형식)하였다.

STEP #3: 이미지 노이즈 추가

```

X_train = X_train / 255
X_test = X_test / 255

# 노이즈 강도 0.15
noise_factor = 0.15

# 노이즈 추가된 이미지 저장할 배열
noise_dataset = []

# 훈련용 데이터 노이즈 추가
for img in X_train:
    noisy_image = img + noise_factor * np.random.randn(*img.shape) # 이미지 픽셀에 노이즈 추가
    noisy_image = np.clip(noisy_image, 0., 1.) # 이미지 클리핑
    noise_dataset.append(noisy_image) # 노이즈 추가된 이미지 noise_dataset 배열에 추가

noise_dataset = np.array(noise_dataset)
noise_dataset.shape

```

✓ 13.4s

Python Python

(50000, 32, 32, 3)

```

# 검증용 데이터 노이즈 추가
noise_test_set = []
for img in X_test:
    noisy_image = img + noise_factor * np.random.randn(*img.shape) # 이미지 픽셀에 노이즈 추가
    noisy_image = np.clip(noisy_image, 0., 1.) # 이미지 클리핑
    noise_test_set.append(noisy_image) # 노이즈 추가된 이미지 noise_dataset 배열에 추가

noise_test_set = np.array(noise_test_set)
noise_test_set.shape

```

✓ 2.3s

Python

(10000, 32, 32, 3)

```

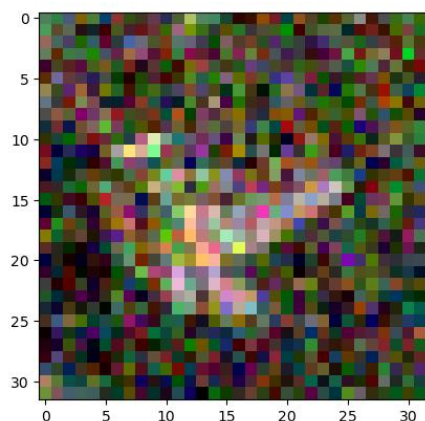
# 노이즈 추가된 이미지 하나 임의 출력
plt.imshow(noise_dataset[22], cmap="gray")

```

✓ 0.2s

Python

<matplotlib.image AxesImage at 0x1362fc7hd00>



이미지 노이즈 추가를 위해 노이즈 강도 설정 후 각 이미지 데이터 픽셀에 노이즈를 추가하고, 노이즈가 잘 추가되었는지 확인하기 위해 노이즈 추가된 이미지 한 장을 임의 출력하였다.

오토인코더 컴파일 과정에서 2 가지 손실 함수 방식을 사용하여 모델 예측의 정확도를 비교하기 위해 과제를 진행하였다. 첫번째로 이진 교차 엔트로피(Binary Cross Entropy) 방식으로 진행하였으며, 두번째로는 평균 제곱 오차(MSE) 방식을 사용하여 2 가지 손실 함수에 대한 정확도를 비교하기로 하였다.

STEP #4.1 : 손실 함수 : 이진 교차 엔트로피

STEP #4: 오토인코더 모델 설계 및 훈련(.fit())

```
# 입력 이미지의 형태 저장
input_shape = X_train.shape[1:]

# 오토인코더 모델 생성
autoencoder = tf.keras.models.Sequential([
    ### 인코더 부분
    # 이미지 특성 추출
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
    # 2x2 최대 풀링을 사용하여 이미지 크기를 1/2배 (차원 축소 및 압축)
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    # 이미지 특성 추출
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),

    ### 디코더 부분
    # 2x2 업샘플링을 사용하여 이미지 크기를 2배 (차원 복원 및 복원)
    tf.keras.layers.UpSampling2D((2, 2)),
    # 이미지 특성 추출
    tf.keras.layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')
])
```

```
# 오토인코더 모델 컴파일
autoencoder.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.001))
# 오토인코더 모델 요약 출력
autoencoder.summary()
```

WARNING:abs1:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimi
Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_7 (Conv2D)	(None, 16, 16, 32)	9248
up_sampling2d (UpSampling2D)	(None, 32, 32, 32)	0
conv2d_8 (Conv2D)	(None, 32, 32, 3)	867

오토 인코더 모델의 인코더 부분을 설계할 때 이미지 데이터 특성을 추출하기 위해 Conv2D 함수를 이용하고, MaxPooling2D()함수를 사용하여 차원 축소 및 압축을 시행하도록 하였다. 디코더 부분을 설계할 때는 그의 역순으로 Upsampling2D() 함수를 이용하여 차원 복원 및 압축으로부터의 복원을 시행하도록 하였다.

```
# 데이터 훈련
autoencoder.fit(noise_dataset, X_train, epochs=10, batch_size=128, validation_data=(noise_test_set, X_test))
```

설계한 오토 인코더 모델을 통해 데이터 훈련을 fit() 함수를 통해 진행하였다. 학습 반복 횟수인 epochs 를 10 으로 두었고, 노이즈 추가 데이터 noise_dataset 입력에 대해 정답 레이블인 원본 데이터 X_train 과 비교하여 학습하도록 하였다.

STEP #5: 모델 성능 측정 및 노이즈 제거 예시 출력

```
# 모델 성능 측정 (예측 정확도 파라미터)
evaluation = autoencoder.evaluate(noise_test_set, X_test)
print('Test Accuracy : {:.3f}'.format(evaluation))
```

```
313/313 [=====] - 4s 14ms/step - loss: 0.5659
Test Accuracy : 0.566
```

```
# 오토인코더 노이즈 제거 출력을 위한 예측 결과 저장
predicted = autoencoder.predict(noise_test_set[:10])
```

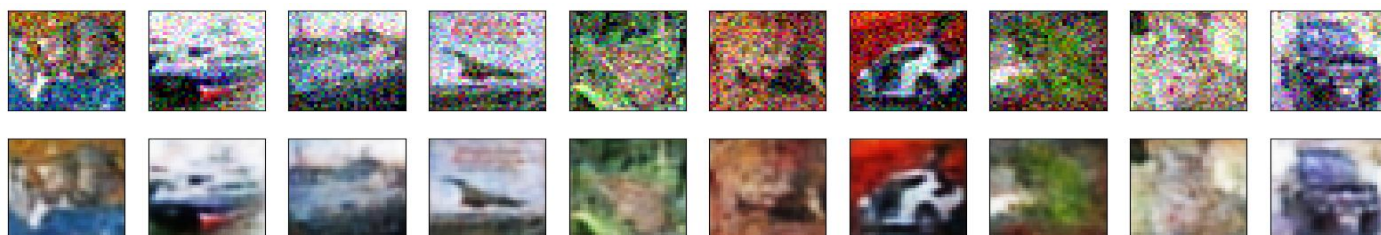
```
1/1 [=====] - 0s 59ms/step
```

```
predicted.shape
```

```
(10, 32, 32, 3)
```

모델 성능 측정을 위해 정확도 파라미터를 사용하여 모델 성능을 측정하였고, 해당 모델에서 0.566 의 정확도를 보였다. 또한, predict()함수를 통해 입력데이터 noise_test_set[:10] (10 개)에 대한 오토인코더 모델 출력의 예측 결과를 predicted 에 담고 아래와 같이 예측 결과를 출력하였다.

```
# 예측 결과 출력
fig, axes = plt.subplots(nrows=2, ncols=10, sharex=True, sharey=True, figsize=(20,4))
for images, row in zip([noise_test_set[:10], predicted], axes):
    for img, ax in zip(images, row):
        ax.imshow(img.reshape((32, 32, 3)), cmap='Greys_r')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
```



STEP #4.2 : 손실 함수 : 평균 제곱 오차

STEP #4: 오토인코더 모델 설계 및 훈련(.fit())

```
# 입력 이미지의 형태 저장
input_shape = X_train.shape[1:]

# 오토인코더 모델 생성
autoencoder = tf.keras.models.Sequential([
    ### 인코더 부분
    # 이미지 특성 추출
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
    # 2x2 최대 풀링을 사용하여 이미지 크기를 1/2배 (차원 축소 및 압축)
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
    # 이미지 특성 추출
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),

    ### 디코더 부분
    # 이미지 특성 복원
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    # 2x2 업샘플링을 사용하여 이미지 크기를 2배 (차원 복원 및 복원)
    tf.keras.layers.UpSampling2D((2, 2)),
    # 이미지 복원
    tf.keras.layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')
])

# 오토인코더 모델 컴파일
autoencoder.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.Adam(lr=0.001))
```

Python

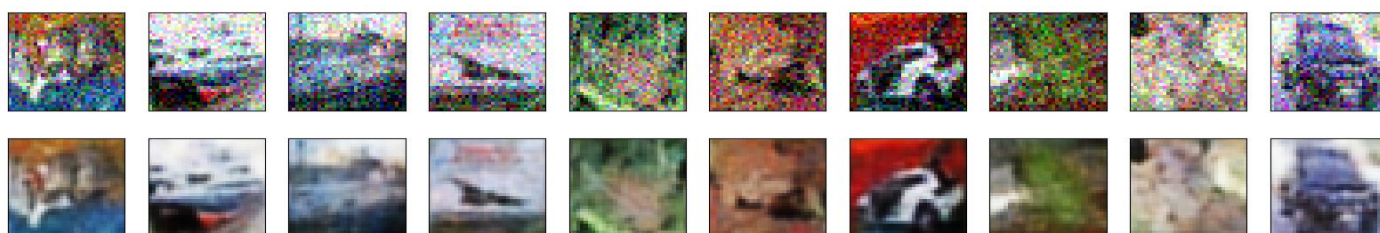
전체적인 과정은 STEP#4.1의 과정과 동일하나, 위와 다르게 오토인코더 모델 컴파일 과정에서 손실 함수로 MSE를 사용하여 진행하였다.

STEP #5: 모델 성능 측정 및 노이즈 제거 예시 출력

```
# 모델 성능 측정 (예측 정확도 파라미터)
evaluation = autoencoder.evaluate(noise_test_set, X_test)
print('Test Accuracy : {:.3f}'.format(evaluation))
```

Python

```
313/313 [=====] - 8s 24ms/step - loss: 0.0033
Test Accuracy : 0.003
```



손실 함수로 MSE를 사용한 결과, Test Accuracy는 0.003, 노이즈 제거 결과는 위와 같았다. 테스트 정확도와 실제 노이즈 제거 예시 출력 결과를 확인하여 손실 함수로 이진 교차 엔트로피 방식으로 진행하기로 결정하였다.

Classification 수행

STEP #1 : 오토인코더를 거친 잡음제거 이미지 데이터 훈련용/검증용 분류

```
from sklearn.model_selection import train_test_split

num_classes = 10

# 이미지 노이즈 제거를 위한 훈련 데이터 준비
X_train_autoencoder = predicted

# 훈련 데이터와 정답 레이블 준비
X_train_classifier = X_train_autoencoder
y_train_classifier = tf.keras.utils.to_categorical(y_test[:10000], num_classes)

# 훈련 및 검증 데이터로 분할
X_train_class, X_val_class, y_train_class, y_val_class = train_test_split(X_train_classifier, y_train_classifier, test_size=0.2, random_state=42)
```

✓ 0.1s

Python

Image Classification 을 수행하기 위해 다음과 같이 훈련 데이터를 준비하였다. 우선 훈련 및 검증 set 로 분류하기 위하여 `train_test_split` 함수를 임포트하였다. 앞서 언급한 바와 같이 `cifar10` dataset 은 10 가지 클래스가 존재하기에 `num_classes` 를 10 으로 하였으며, 노이즈가 제거된 이미지세트인 `predicted` 를 `X_train_autoencoder` 에 저장하였다. 그 후, `classifier` 의 훈련 데이터와 정답 레이블을 각각 `X_train_classifier` 와 `y_train_classifier` 에 준비하였으며, `to_categorical` 함수로 정답 레이블을 원-핫 인코딩으로 변환하였다. 마지막으로 `train_test_split` 함수를 사용하여 훈련 데이터를 훈련 세트와 검증 세트로 분할하였다. `test_size=0.2` 는 검증 세트의 크기를 전체 데이터의 20%로 설정하고, `random_state=42` 는 재현성을 위해 랜덤 시드를 설정하였다. 결과적으로 분할된 훈련 및 검증 데이터를 각각 `X_train_class`, `X_val_class`, `y_train_class`, `y_val_class` 변수에 할당하였다.

STEP #2 : 이미지 분류기 모델 생성

```

classifier = tf.keras.models.Sequential([
    # Step 1 - Convolution
    tf.keras.layers.Conv2D(128, (3, 3), input_shape=(32, 32, 3), activation='relu'), # 128개의 3x3 필터를 사용하여 합성곱을 수행하는 층
    # 입력 이미지의 크기: 32x32, 입력 채널 수: 3 (RGB 이미지)
    # 활성화 함수로 ReLU를 사용하여 비선형성을 도입

    # Adding another layer
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'), # 64개의 3x3 필터를 사용하여 합성곱을 수행하는 층
    # 이전 층에서 추출한 이미지 특성을 더욱 추상화하여 표현

    # Pooling
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)), # 2x2 최대 풀링을 사용하여 이미지 크기를 절반으로 줄임
    # 이미지의 중요한 특징을 보존하면서 공간 크기를 줄이고 계산 부담을 줄임

    # Adding another layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'), # 32개의 3x3 필터를 사용하여 합성곱을 수행하는 층
    # 더욱 추상화된 이미지 특성을 추출

    # Pooling
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)), # 2x2 최대 풀링을 사용하여 이미지 크기를 절반으로 줄임

    # Adding another layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'), # 32개의 3x3 필터를 사용하여 합성곱을 수행하는 층
    # 더욱 추상화된 이미지 특성을 추출

    # Pooling
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)), # 2x2 최대 풀링을 사용하여 이미지 크기를 절반으로 줄임

    # Flatten the layer
    tf.keras.layers.Flatten(), # 다차원의 특성 맵을 1차원으로 평탄화

    # Fully connected layer
    tf.keras.layers.Dense(128, activation='relu'), # 128개의 뉴런을 가지는 완전 연결층
    # 추출된 이미지 특성을 입력으로 받아 뉴런의 가중치와 편향을 사용하여 계산을 수행

    # Output layer
    tf.keras.layers.Dense(num_classes, activation='softmax') # 클래스 개수만큼의 뉴런을 가지는 출력층
    # 소프트맥스 활성화 함수를 사용하여 각 클래스에 대한 확률 값을 출력
])

```

이미지 분류기 모델인 classifier 을 tf.keras.models.Sequential 모델 클래스로 간단한 신경망 구조를 정의하였다. 합성곱층에서의 이미지 특성 추출과 풀링층에서의 이미지 특성 강조 단계를 여러 번 거친 후, 레이어를 Flatten 하여 1차원으로 차원 축소한 후, 완전 연결층에서 추출된 이미지 특성을 입력으로 받아 계산을 수행하도록 한 후, softmax 함수를 사용하여 출력 층에서 클래스 개수(10 개 : cifar-10 데이터셋 활용)만큼의 뉴런을 가지도록 하여 각 클래스에 대한 확률 값을 출력하도록 하였다. 컴파일시 손실 함수로는 categorical_crossentropy 함수를 사용하였다.

```

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

```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 30, 30, 128)	3584
conv2d_13 (Conv2D)	(None, 28, 28, 64)	73792
max_pooling2d_5 (MaxPooling 2D)	(None, 14, 14, 64)	0
conv2d_14 (Conv2D)	(None, 12, 12, 32)	18464
max_pooling2d_6 (MaxPooling 2D)	(None, 6, 6, 32)	0
conv2d_15 (Conv2D)	(None, 4, 4, 32)	9248
max_pooling2d_7 (MaxPooling 2D)	(None, 2, 2, 32)	0
flatten_1 (Flatten)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512

...
Total params: 122,896
Trainable params: 122,896
Non-trainable params: 0

STEP #3 : 이미지 분류기 훈련 및 정확도 평가

```
#fit()과정 batch=32, epochs=10
classifier.fit(X_train_class, y_train_class, batch_size=32, epochs=10, validation_data=(X_val_class, y_val_class))
```

✓ 8m 1.3s Python

Epoch 1/10
250/250 [=====] - 45s 174ms/step - loss: 2.0769 - accuracy: 0.2307 - val_loss: 1.7824 - val_accuracy: 0.3560
Epoch 2/10
250/250 [=====] - 47s 187ms/step - loss: 1.6999 - accuracy: 0.3739 - val_loss: 1.6809 - val_accuracy: 0.3835
Epoch 3/10
250/250 [=====] - 49s 197ms/step - loss: 1.5636 - accuracy: 0.4329 - val_loss: 1.5548 - val_accuracy: 0.4280
Epoch 4/10
250/250 [=====] - 48s 194ms/step - loss: 1.4686 - accuracy: 0.4690 - val_loss: 1.5355 - val_accuracy: 0.4520
Epoch 5/10
250/250 [=====] - 46s 185ms/step - loss: 1.3999 - accuracy: 0.4873 - val_loss: 1.5503 - val_accuracy: 0.4450
Epoch 6/10
250/250 [=====] - 50s 202ms/step - loss: 1.3351 - accuracy: 0.5185 - val_loss: 1.4977 - val_accuracy: 0.4535
Epoch 7/10
250/250 [=====] - 49s 196ms/step - loss: 1.2836 - accuracy: 0.5345 - val_loss: 1.4466 - val_accuracy: 0.4830
Epoch 8/10
250/250 [=====] - 50s 201ms/step - loss: 1.2337 - accuracy: 0.5570 - val_loss: 1.4970 - val_accuracy: 0.4630
Epoch 9/10
250/250 [=====] - 50s 199ms/step - loss: 1.2008 - accuracy: 0.5696 - val_loss: 1.4493 - val_accuracy: 0.4895
Epoch 10/10
250/250 [=====] - 46s 186ms/step - loss: 1.1450 - accuracy: 0.5840 - val_loss: 1.4581 - val_accuracy: 0.4850

<keras.callbacks.History at 0x1363176f340>

```
class_predicted = classifier.predict(X_val_class)
```

✓ 3.5s Python

63/63 [=====] - 3s 52ms/step

이미지 클래스 분류기를 X_train_class(이미지 노이즈 제거가 된 결과)와 y_train_class 를 사용하여 훈련시켰다. (훈련 시 batch_size=32, epochs=10 으로 설정)

그 후 모델의 성능 정도를 검증하기 위하여 X_val_class(검증 데이터)로 예측을 진행하였다.

```
# 모델 성능 측정 (예측 정확도 파라미터)
class_evaluation = classifier.evaluate(X_val_class, y_val_class)
print('Test Accuracy: {:.3f}'.format(class_evaluation[1]))
```

✓ 3.1s Python

63/63 [=====] - 3s 48ms/step - loss: 1.4581 - accuracy: 0.4850
Test Accuracy: 0.485

evaluate() 함수를 사용하여 X_val_class 와 y_val_class (검증 set)를 사용해 모델의 정확도를 평가하였다. 평가 결과, 0.485 라는 결과를 얻을 수 있었다.

```
class_names = ['airpl', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# 예측 결과 출력
predicted_labels = np.argmax(class_predicted, axis=1)
true_labels = np.argmax(y_val_class, axis=1)

# 예측 결과 시각화
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(10, 10))
fig.tight_layout()

for i, ax in enumerate(axes.flatten()):
    # 이미지 출력
    ax.imshow(X_val_class[i])
    ax.axis('off')

    # 예측된 레이블과 실제 레이블 출력
    predicted_label = class_names[predicted_labels[i]]
    true_label = class_names[true_labels[i]]
    ax.set_title(f'Pred: {predicted_label}, True: {true_label}')

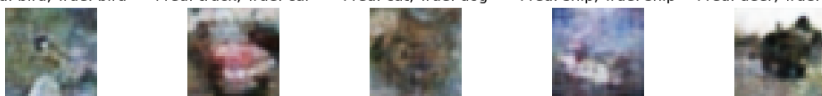
# 이미지 출력 간격 조절
plt.subplots_adjust(wspace=1.0, hspace=1.0)

plt.show()
```

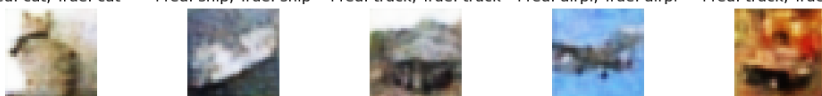
✓ 1.9s

Python

Pred: bird, True: bird Pred: truck, True: car Pred: cat, True: dog Pred: ship, True: ship Pred: deer, True: truck



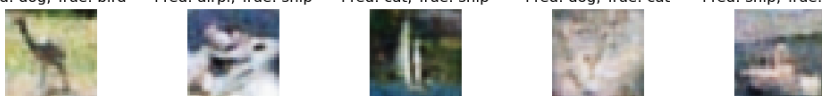
Pred: cat, True: cat Pred: ship, True: ship Pred: truck, True: truck Pred: airpl, True: airpl Pred: truck, True: car



Pred: dog, True: dog Pred: car, True: car Pred: dog, True: cat Pred: dog, True: dog Pred: ship, True: truck



Pred: dog, True: bird Pred: airpl, True: ship Pred: cat, True: ship Pred: dog, True: cat Pred: ship, True: ship



Pred: frog, True: deer Pred: airpl, True: airpl Pred: dog, True: bird Pred: dog, True: horse Pred: cat, True: dog



Classifier 의 성능을 직관적으로 확인하기 위해 노이즈가 제거된 이미지의 예측 결과를 출력하였다. 출력 결과, 모델의 정확도인 0.48 정도에 부합하는 결과를 눈으로 확인할 수 있었다. 또한, 모델이 분류를 하는 과정에서 트럭과 자동차나 고양이와 강아지 같은 비슷한 형태의 물체에 대한 이미지 분류의 정확도가 떨어지는 것을 확인할 수 있었다.

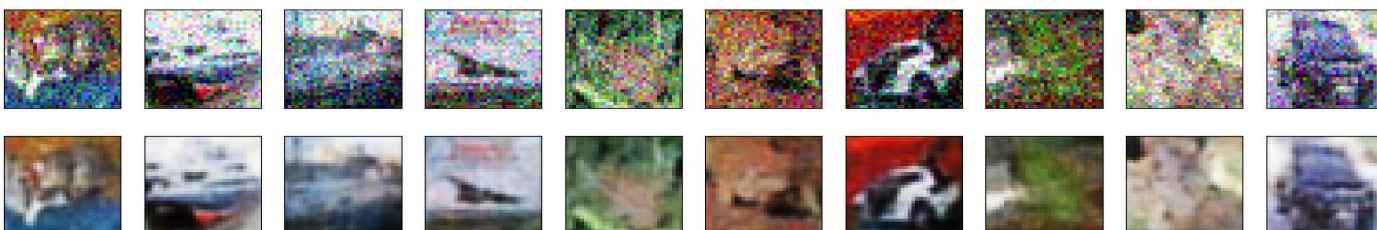
6.2 제작 시 문제점 및 개선사항

6.2.1 이미지 노이즈 제거기의 손실함수, 정확도

STEP #5: 모델 성능 측정 및 노이즈 제거 예시 출력

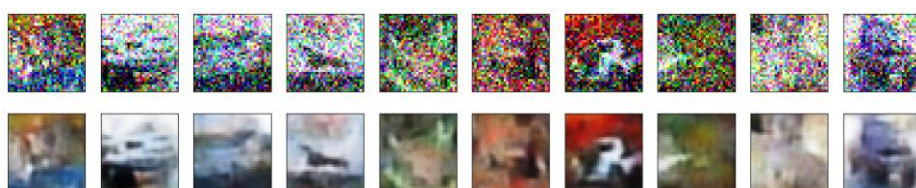
```
# 모델 성능 측정 (예측 정확도 파라미터)
evaluation = autoencoder.evaluate(noise_test_set, X_test)
print('Test Accuracy : {:.3f}'.format(evaluation))
```

313/313 [=====] - 8s 24ms/step - loss: 0.0033
Test Accuracy : 0.003



앞의 제작과정에서 STEP#4.2 처럼 손실함수로 MSE 를 사용할 경우 정확도가 0.003 으로 매우 낮은 수치를 얻을 수 있었다. 실제 MSE 방식으로 진행한 결과 이미지는 눈으로 보기에 나쁘지 않은 결과를 얻을 수 있었지만, 수치적으로는 매우 낮은 정확도가 나왔다. 따라서 STEP#4.1 의 Binary CrossEntropy 방식과 수치적인 비교에서 문제점이 있었다.

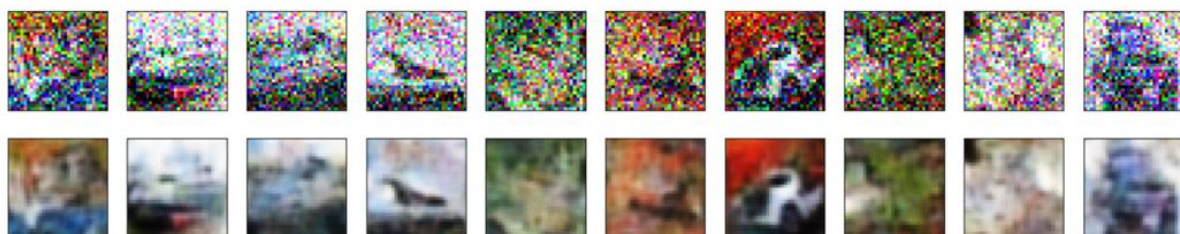
6.2.2 노이즈 제거 성능에 따라 이미지 분류기의 정확도 편차 증가



노이즈 강도를 0.5 이상으로 설정한 뒤 이미지에 노이즈를 가하면 매우 강한 노이즈가 섞인 이미지가 생성되는데, 이를 노이즈 제거 모델에 입력한 뒤, 그것들로 이미지 분류기를 실행할 경우 정상적으로 학습이 되지 않는 문제점이 발생하였다. 원인을 분석한 결과 이를 해결하기 위해 노이즈 강도 0.5 이상의 이미지에 대하여 노이즈 제거가 제대로 되지 않았고, 그것들로 이미지 분류기 또한 학습하기 때문에 분류 학습 과정이 정상적이지 않다는 것을 인지하였다. 따라서 이를 해결하기 위해 이미지 노이즈 제거기의 성능을 높이려고 여러 시도(손실함수, 필터수, 학습률, batch size, ephocs)를 시도한 결과, 위의 6.1 제작과정과 같은 모델을 설계하였으며 노이즈 강도 0.5 이상의 이미지에 대해서도 분류기 성능이 어느 정도 안정된 수치를 출력하는 것을 확인하였다.

7. 시험

Autoencoder Output - 1 열은 노이즈가 추가된 이미지, 2 열은 오토인코더를 통해 노이즈 제거된 이미지이다. 성능 측정 시 0.5576 수치를 획득하였다.

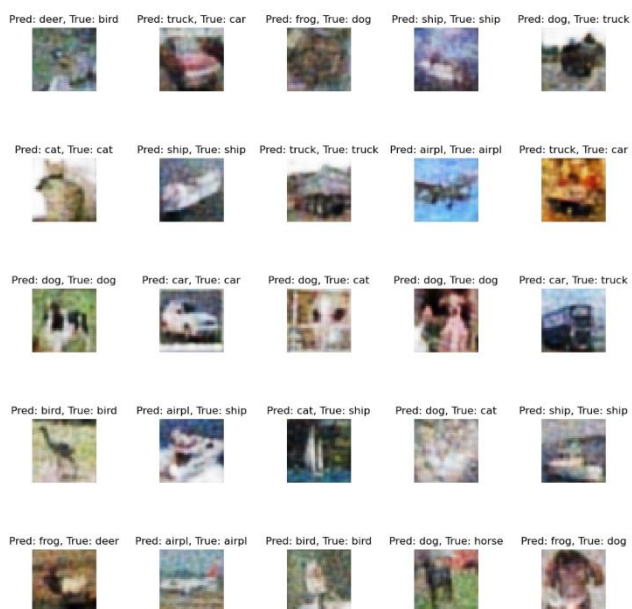


```
# 모델 성능 측정 (예측 정확도 파라미터)
evaluation = autoencoder.evaluate(noise_test_set, X_test)
print('Test Accuracy : {:.3f}'.format(evaluation))
```

✓ 7.0s

313/313 [=====] - 7s 21ms/step - loss: 0.5576
Test Accuracy : 0.558

Classifier Output - Pred 는 예측 결과, True 는 정답 레이블이다. 성능 측정 시 0.482 수치를 획득하였다.



```
# 모델 성능 측정 (예측 정확도 파라미터)
class_evaluation = classifier.evaluate(X_val_class, y_val_class)
print('Test Accuracy: {:.3f}'.format(class_evaluation[1]))
```

✓ 1.7s

63/63 [=====] - 2s 26ms/step - loss: 1.4851 - accuracy: 0.4820
Test Accuracy: 0.482

8. 평가

8.1 정량적/정성적 목표달성도 평가

항목	목표	달성률	비고
1	이미지 데이터셋 (cifar-10) 로드 및 잡음 추가	100%	.
2	잡음 제거 기능 오토인코더 모델 설계	100%	.
3	오토인코더 모델 훈련을 통한 예측 및 2 가지 손실함수(이진교차엔트로피, MSE)에 대한 비교	100%	.
4	이미지 클래스 분류기 모델 설계	100%	.
5	이미지 클래스 분류기 모델 훈련을 통한 예측 (이미지 클래스 분류 수행)	100%	.
6	오토인코더 모델과 이미지 클래스 분류기 모델의 합성	100%	.

8.2 현실적 제한요소 달성도 평가

현실적 제한 요소	내용	달성 결과
경제	모델 훈련에 사용되는 H/W 비용에 대한 문제를 인식하여야 한다.	달성 완료
편리	이미지 처리에 있어 기존 사람이 하던 방식만큼 정확하고, 사람이 하던 방식보다 편리해야 한다.	달성 완료
윤리	활용되는 CNN 이미지 처리 기술이 불법적이나 비윤리적으로 이용되어서는 안된다.	달성 완료
사회	인공지능의 발달로 인한 일자리 감소 문제를 인식하여야 한다.	달성 완료

8.3 기능적 요구사항 달성도 평가

ID	요구사항	내용	설명	달성률
No_01	이미지 데이터셋 로드	이미지 노이즈 추가/제거 및 클래스화를 위한 이미지 데이터셋 로드	클래스가 10 개인 이미지 데이터셋인 Cifar-10 데이터셋을 훈련용/검증용 데이터로 나누어 로드한다.	100%
No_02	이미지 잡음 추가	로드된 cifar 데이터셋에 이미지 잡음 추가	오토인코더의 입력 데이터로 들어가 잡음 제거가 수행되도록 하기 위해 임의의 노이즈 강도를 설정해 각 이미지에 잡음을 추가한다.	100%

No_03	오토인코더 모델 설계	이미지 잡음 제거를 위한 오토인코더 모델 설계	ReLU 활성화 함수를 사용한 합성곱층과 이미지 크기를 1/2 로 줄이는 풀링층 이후 다시 합성곱층으로 이어지는 인코더 부분과 이미지 크기를 원복하고, 합성곱층을 사용하여 이미지 특성 추출하여 압축으로부터 복원하는 디코더 부분을 가진 오토인코더 모델을 설계한다.	100%
No_04	오토인코더 모델 훈련	설계된 오토인코더 모델 훈련	잡음이 추가된 이미지 훈련용 데이터에 대해 원본 훈련용 데이터를 비교를 위한 정답 데이터로 사용해 오토인코더 모델을 훈련한다.	100%
No_05	잡음 제거 이미지 출력	잡음이 제거된 이미지 출력	잡음이 추가된 이미지가 오토인코더 모델 훈련을 통해 예측 결과로 잡음 제거 이미지를 반납하고, 이를 정답 데이터(원본 훈련용 데이터)와 같이 출력함으로써 잡음 제거 수행 결과를 확인한다.	100%
No_06	잡음 제거 이미지 데이터화	잡음이 제거된 이미지 훈련용/검증용 분류	잡음이 추가된 이미지에 대해 이미지 분류를 수행해야 하므로 훈련용/검증용 데이터로 나누어야 한다.	100%
No_07	이미지 클래스 분류기 모델 생성	잡음 제거 이미지 데이터셋에 대한 이미지 클래스 분류를 위한 이미지 분류기 모델 생성	ReLU 활성화 함수를 사용한 합성곱층과 풀링층을 3번 통과하고 차원 축소를 거친 후 완전 연결층으로 이어진 후 확률 값을 출력하는 이미지 클래스 분류기 모델을 생성한다.	100%
No_08	이미지 클래스 분류기 모델 훈련	설계된 이미지 클래스 분류기 모델 훈련	(잡음 제거 과정을 거친)훈련용 데이터에 대해 정답 데이터인	100%
No_09	클래스 분류 결과 출력	잡음 제거 과정을 거친 데이터의 클래스 분류 결과 출력	잡음 제거 과정을 거친 데이터의 클래스 분류 결과를 이미지와 예측/정답 클래스가 함께 나오도록 출력	100%

11. 결론



이 프로젝트를 통해 합성곱 신경망, 오토인코더 등 여러 기계학습 분야에서 사용되는 이론에 대한 이해 및 응용 능력 부분에서 팀원 모두가 발전하였다고 생각한다. 또한, 이미지/영상 통신 과정에서 발생할 수 있는 노이즈를 제거하는 것과, 그것들을 분류기를 통해 라벨링을 하는 부분에서 우리가 목표로 달성하고자 한 정도를 달성하였다고 생각한다.

이미지 노이즈 제거와 분류를 위한 모델을 개발하는 과정은 처음에는 어려움을 겪었지만, 노력과 학습을 통해 점점 이해도를 높일 수 있었다. 초기에는 신경망 모델의 구조와 레이어의 역할에 대해 이해하는 데 어려움을 겪었지만, 점차 적용해보면서 모델이 어떻게 동작하는지 이해할 수 있었다.

모델 개발 과정에서는 데이터 전처리, 모델 아키텍처 설계, 하이퍼파라미터 조정 등 다양한 단계를 거쳤다. 가장 힘들었던 부분은 모델의 성능을 높이기 위해 여러 시도 과정에서, 신경망을 통과시키고 학습시키는 과정에 시간이 오래 들었던 부분이다. 그것에 더하여 이미지 노이즈 제거를 위한 오토인코더 모델과 분류를 위한 분류기 모델의 조합은 복잡한 작업이었지만, 문제에 대한 이해와 실험을 통해 성취해냈다.

최종적으로 우리가 개발한 모델은 노이즈가 추가된 이미지를 입력으로 받아 원본 이미지를 복원하고, 분류 작업을 수행할 수 있었다. 이를 통해 이미지 노이즈 제거와 분류를 동시에 수행하는 종합적인 기능을 구현할 수 있었다. 이 프로젝트를 통해 머신러닝과 딥러닝 기술에 대한 이해와 적용 능력이 크게 향상되었으며, 문제 해결 과정에서의 문제 분석, 실험, 결과 분석 등 다양한 과정을 경험하며 데이터 과학에 대한 통찰력과 문제 해결 능력을 키워갈 수 있었던 경험이었다고 생각한다.