자유 Project

컴퓨터공학과

20155137

안원영

목차

Project

목표를 크게 4가지로 설정하였다.

1. CIFAR 10 데이터에 대해서 분석한다.
2. 성능이 좋았던 RandomForest를 이용해 성능을 높인다.
3. 인공신경망으로 다시 구축해보고 성능을 높인다.
4. 오차행렬을 이용해 많이 틀리는 이미지 2가지를 비교해본다.

Chapter 1

데이터 읽기

- 먼저 사용할 데이터를 읽어온다. 사용할 데이터는 Keras에 있는 CIFAR10 데이터셋 이다.
- 총 10개의 클래스로 이루어져 있으며, 한 클래스당 6000개의 컬러 이미지로 이루어져 있다.
- 데이터는 총 6만개, 32*32*3 픽셀의 형태로 저장되어있다.
  - 컬러이미지라 3차원으로 이루어져 있다.
  - 따라서 뒤의 3은 채널(RGB)을 뜻한다.

```
X_train = X_train[:10000].reshape(-1,32*32*3)
y_train = y_train[:10000]

X_test = X_test[:10000].reshape(-1,32*32*3)
y_test = y_test[:10000]

print("X_train shape : ",X_train.shape)
print("y_train shape : ",y_train.shape)
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
class_names = np.array(class_names)
```

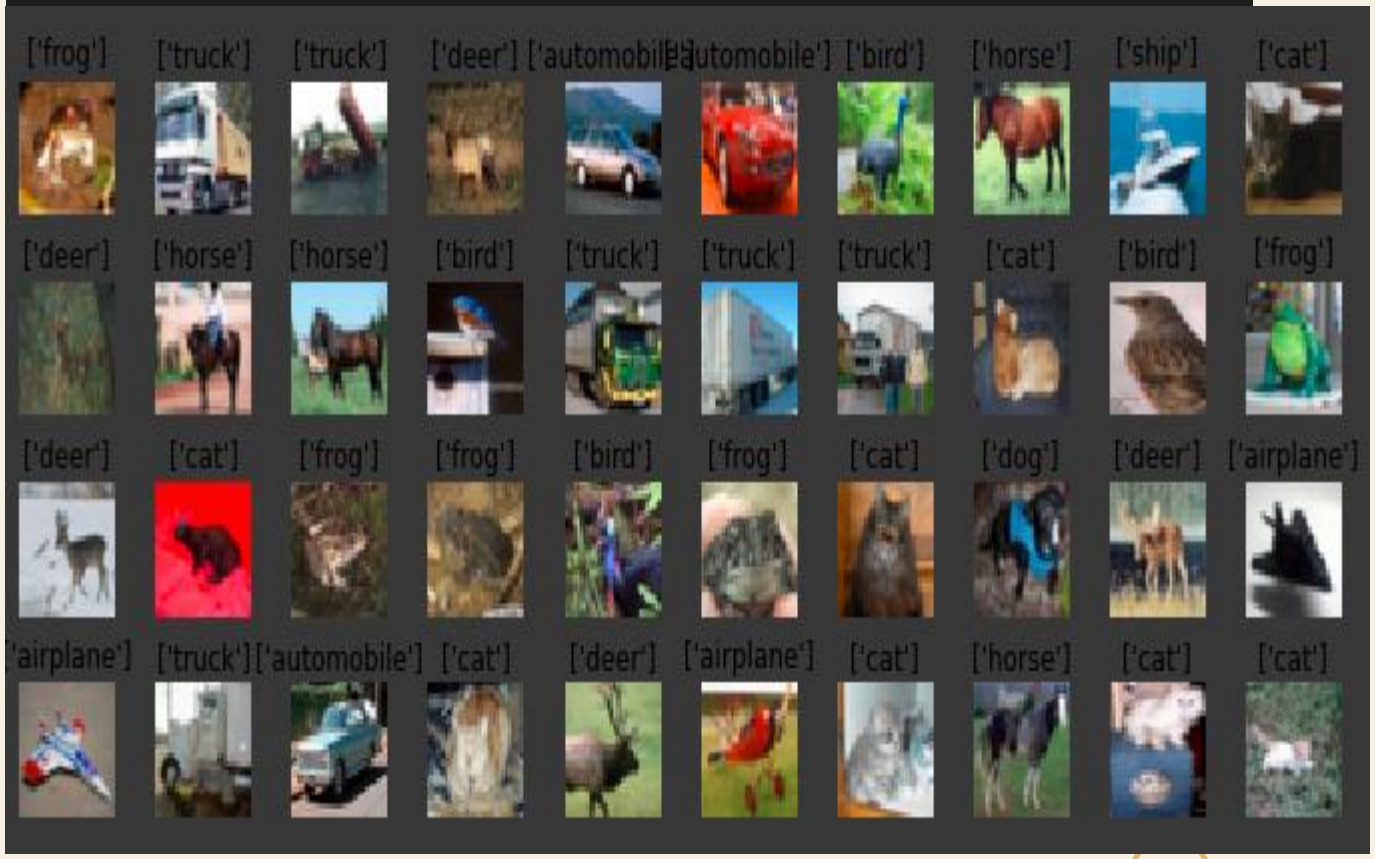
Chapter 2

데이터 분석1

- 쉽게 다루기 위해 1만개의 이미지만 가지고 온다.
- 이미지를 계산하기 쉽도록 1차원 형태인(-1,3072)로 바꿔준다.
- y_train 에는 데이터의 라벨들이 인덱스의 형태로 저장되어 있다.

```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

n_rows = 4
n_cols = 10
```



데이터분석2

- 이미지를 구체적으로 보기 위해 40개의 이미지와 각 라벨을 plot

```
model_RandomF = RandomForestClassifier(n_estimators=200, oob_score=True, random_state=42)
model_RandomF.fit(X_train,y_train)

# test셋으로 값을 예측
y_pred = model_RandomF.predict(X_test)
print('test샘플에 대한 정확도 :', accuracy_score(y_test, y_pred))

# train셋으로 값을 예측 -> 과적합이 된것을 알 수 있다.
y_pred_tra = model_RandomF.predict(X_train)
print('train샘플에 대한 정확도 :', accuracy_score(y_train, y_pred_tra))
```

```
test샘플에 대한 정확도 : 0.4316
train샘플에 대한 정확도 : 1.0
```

Chapter 3

RandomForest로 모델 만들기

- **RandomForest**는 결정트리의 앙상블로, 성능이 아주 뛰어난 분류기이다.
- **Validation set** 별도로 만들지 않고 **oob** 했다.
- 성능 올리는 방법 2가지
  - 규제 - 과대적합 일 경우 사용
  - 학습 데이터 수 증가 - 과소적합 일 경우 사용

RandomForest
성능올리기 1
(규제)

- 규제 편하게 하기위해 함수를 만들어준다.

규제를 통해 **33%** 에서 **44%** 까지 증가시킬수 있었다.

```
def learning(
                # model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=5000,
                #                                        min_samples_split=100)#37%, test:33%
                # model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=2000,
  model_RandomF.fit#                                    min_samples_split=100)#leaf_node 갯수 줄임-> test:39%
  train_accuracy = 0    # model_RandomF = RandomForestClassifier(n_estimators = 1000, random_state=42, max_leaf_nodes=5000,
  test_accuracy = 0     #                                       min_samples_split=100)#n_estimator 증가-> test:39%
                # model_RandomF = RandomForestClassifier(n_estimators = 200, random_state=42, max_leaf_nodes=2000,
  train_accuracy = model_Ran(#                                 min_samples_split=500)#sample_split 증가 -> test:39%
  test_accuracy = model_RandomF.# model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=2000,
  a = (train_accuracy-test_accur#                              min_samples_split=50)#sample_split 감소 -> test:41%
                # model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=1000,
  print("train 데이터정확도=",tra#                             min_samples_split=200)#sample_split, leaf_node 증가 -> test:43%
        ", test 데이터정확도=",test_accura(# model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=2000,
                                #                              min_samples_split=10)#99%, test:44%

learning()
```

RandomForest
성능올리기 2
(학습 데이터 수 늘리기)

- 데이터 2만개 사용
  - 1만개(43%)보다 2만개(45%)가 2%올랐다.

- 데이터 3만개 사용
  - 규제 사용했지만 성능에는 크게 차이 없었다.

```
X_train shape :   (20000, 3072)
y_train shape :   (20000, 1)
```

```
# model_RandomF = RandomForestClassifier(n_estimators = 500, random_state=42, max_leaf_nodes=2000,
#                                        min_samples_split=50)#기준-> test:43%
```

```
# model_RandomF = RandomForestClassifier(n_estimators = 300, random_state=42, max_leaf_nodes=2000,
#                                        min_samples_split=50)#n_estimator 증가 -> test:43%
```

```
# model_RandomF = RandomForestClassifier(n_estimators=300,
#                                        oob_score=True, random_state=42)# 규제 없음(과적합) -> 46%
```

```
train 데이터정확도= 1.0 , test 데이터정확도= 0.458 , 차이= 0.542
```

따라서 모델을 직접 만들어 보기로 하였다.

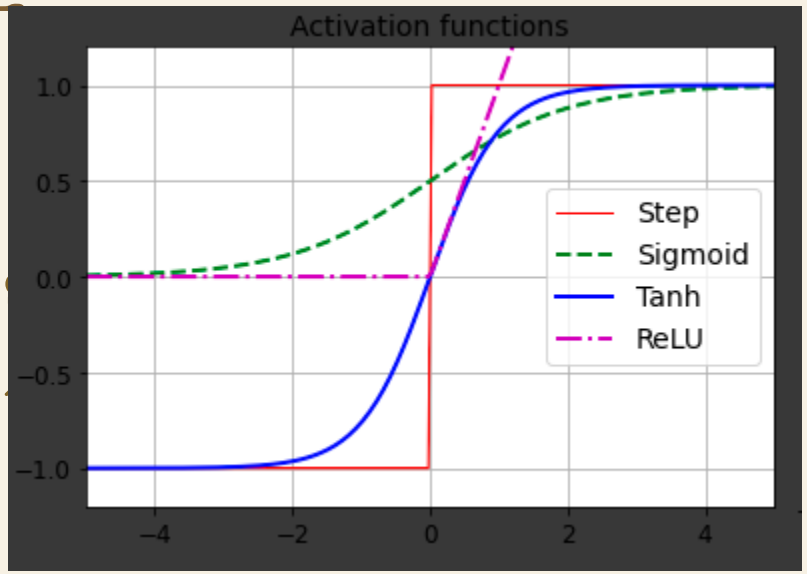
```
X_train shape :   (30000, 3072)
y_train shape :   (30000, 1)
```

```
train 데이터정확도= 1.0 , test 데이터정확도= 0.4655 , 차이= 0.5345
```

Chapter 4

인공신경망 시작

인공신경망을 이용한 모델 만들기



- **Keras**를 통해 모델 만드는 방법
  - **Sequential API (Layer**를 순서적으로 연결, 복잡한것은
  - **Function API (** 복잡한 **Layer** 연결구조 표현 가능)
  - **SubClassing API (** 사용자가 클래스 만들어서 동적으로
- 동적모델로 구현하지 않아도 되기 때문에 간단한 택함
- 활성화 함수 (미분을 가능하도록 하는 함수)
  - **Sigmoid (0**과 **1**사이의 값만 다룬다.**)**
  - **Tanh (Sigmoid**의 단점을 보완**)**
  - **Relu (0**보다 작은 값이 나온 경우 **0**을 만환하고, **0** 보다 큰값일 경우 그대로 값을 반환한다.**)**

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[32, 32,3]),
    keras.layers.Dense(1536, activation="relu"),
    keras.layers.Dense(718, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 1536)              4720128
_________________________________________________________________
dense_1 (Dense)              (None, 718)               1103566
_________________________________________________________________
dense_2 (Dense)              (None, 10)                7190
=================================================================
Total params: 5,830,884
Trainable params: 5,830,884
Non-trainable params: 0
```

인공신경망 구축-1

- **Sequential** 사용
  - **Dense :** 중간 노드**(hidden)**를 몇 개 만들 것인지 결정
  - **Flatten : input**형태를 지정**(1**차원으로 바꿔줌**)**
  - **Softmax (**중요요소**)**

- **Softmax**
  - 다중 분류에 사용
  - 출력값을 **0~1** 사이 값으로 정규화하여 출력값의 총합이 항상 **1**이 되게 하는 것
  - 따라서 가장 큰 출력값을 가지는 클래스가 정답일 확률이 가장 높다**.**

- 마지막 **Dense**는 최종 **output**이기 때문에 데이터의 클래스 개수와 맞춰줘야 함

```
model.compile(loss="sparse_categorical_crossentropy",# 분류문제이기 때문에
              optimizer="sgd",
              metrics=["accuracy"])

early_stopping_cb = keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True)

cifar10_model = model.fit(X_train, y_train, epochs=10, callbacks=[early_stopping_cb])

keras.utils.plot_model(model, show_shapes=True)#모델 구성 plot

model.evaluate(X_test, y_test)#모델성능 test
```

- 이러한 형태로 서로다른 4개의 **Sequential** 을 만들어 비교를 진행한다.
- 성능 비교 후 모델의 구조를 **plot**하고 성능을 측정해본다.

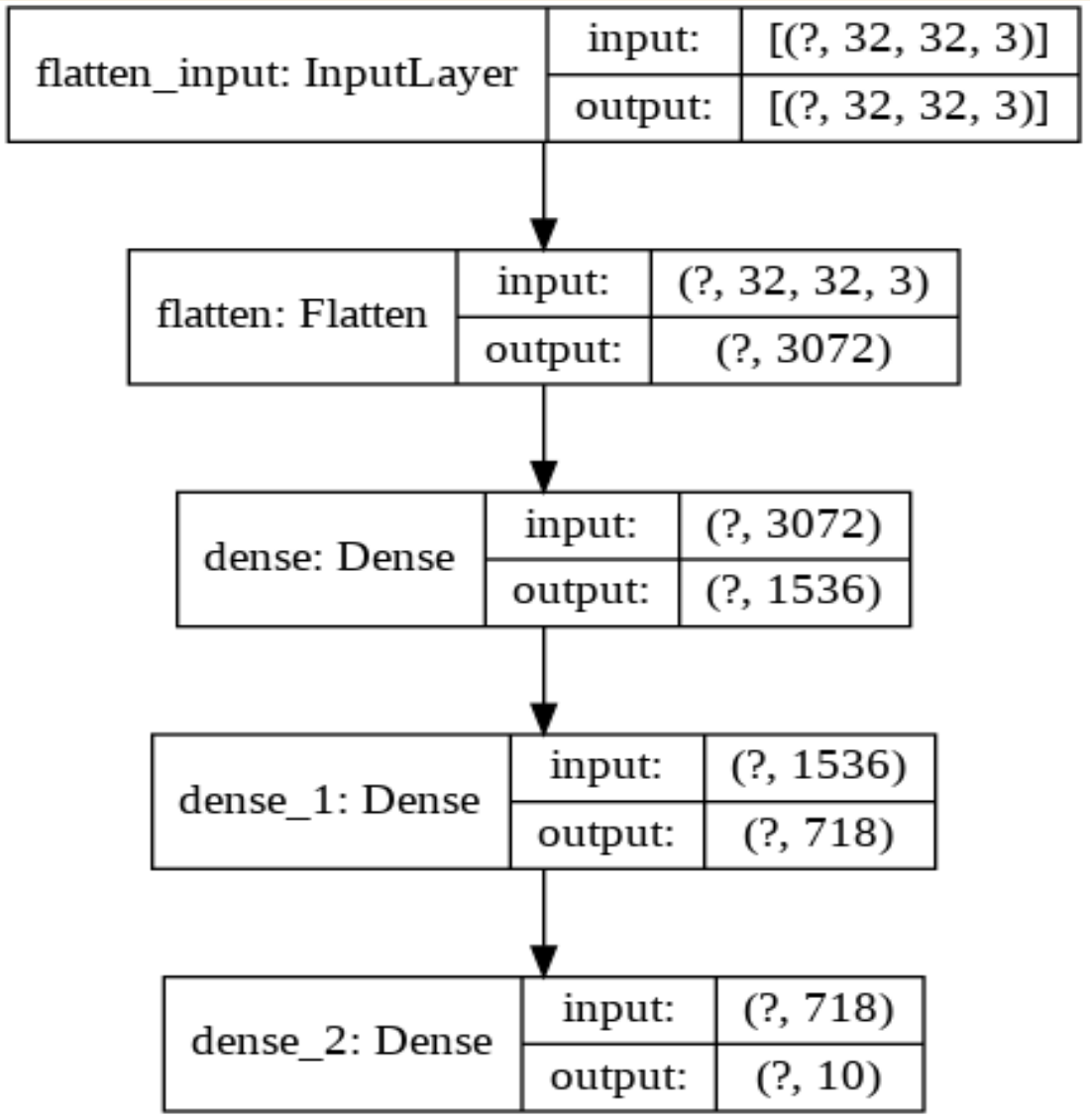
- 콜백
  - 그 상황이 일어났을 때 지정해놓은 함수를 호출한다.

인공신경망 구축-2

- **Compile**
  - 앞서 만든 모델의 파라미터들을 등록하는 과정
  - 어떻게 학습시킬지에 대한 정보가 들어감
- **Optimizer**
  - **SGD(** 확률적 경사 하강법**)** 사용
- **Metrics**
  - 한 **epock** 수행시 어떤 유형으로 평가할 것인가를 결정
- **EarlyStoping**
  - **Patience**번만큼 오차가 전보가 커졌을때 학습을 중단
- **Loss**를 줄여가는 것이 목표이다,

	분류기 1	분류기 2	분류기 3	분류기4
이미지 갯수	3만개		4만개	
Dense Layer 갯수	2개	3개		
Epoch	15번	10번	15번	15번
Train _ Accuracy _ start	40%	40%	40%	42%
Train _ Accuracy _ end	85%	86%	96%	95%
Train _ Accuracy 차이	총 45% 증가	총 46% 증가	총 56% 증가	총 53% 증가
Test_ Accuracy	49.6%	50%	52.7%	53.3%
성능 순위	4	3	2	1

성능 평가 및 결과 정리

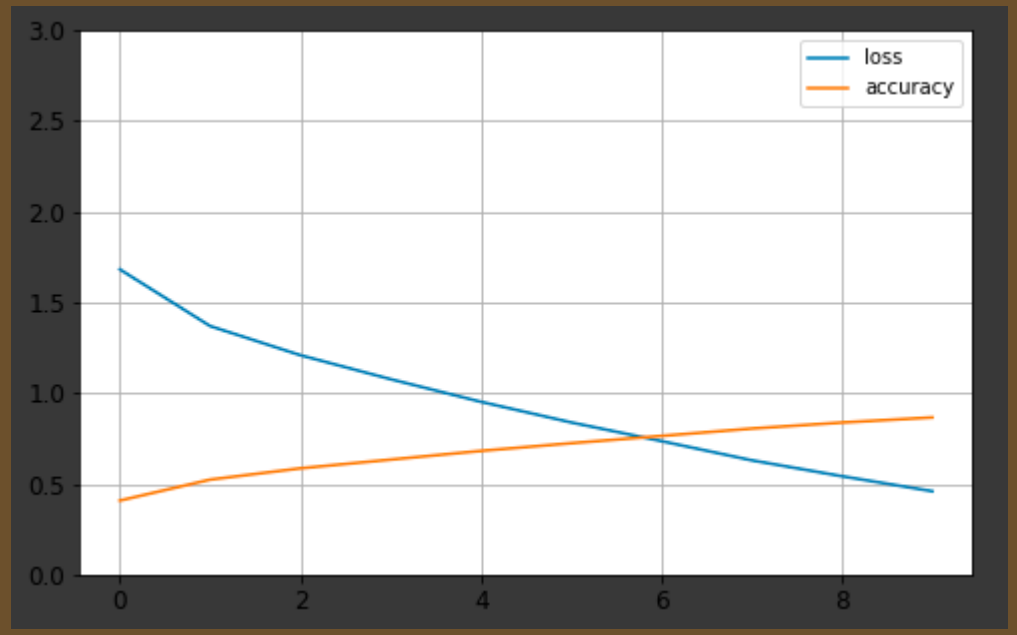


Sequential 모델 plot

- **Sequential** 모델의 구조를 파악할 수 있다.

```
3 import pandas as pd
4
5 df = pd.DataFrame(cifar10_model.history)
6 df.head()
```

	loss	accuracy
0	1.681335	0.407300
1	1.369007	0.522733
2	1.208362	0.584967
3	1.076084	0.633067
4	0.951556	0.680967



모델

loss와 accuracy plot

- Pandas 패키지를 이용해서 DataFrame의 형태로 바꿔준다.
- Loss와 accuracy의 변화를 볼 수 있다.

Chapter 5

오차행렬

오차행렬 구현

<Confusion Matrix error>

- 오차행렬을 이용해 어떤 데이터가 많이 틀리는지 파악할 수 있다.

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

def confu_matrix(X_train,y_train,model_RandomF):

  scaler = StandardScaler()
  model_RandomF_scaler = scaler.fit_transform(X_train.astype(np.float64))
  y_pred = cross_val_predict(model_RandomF, model_RandomF_scaler, y_train, cv=3)
  cm = confusion_matrix(y_train, y_pred)

  return cm

cm = confu_matrix(X_train,y_train,model_RandomF)#오차행렬을 리턴받음
print(cm)#오차행렬 표시
```

```
[[494  57  59  22  30  25  28  32 209  49]
 [ 39 427  17  35  40  34  61  32  69 220]
 [123  47 283  62 179  56 148  67  39  28]
 [ 53  52  76 206 109 183 170  60  28  79]
 [ 67  23 147  44 388  49 148  69  28  36]
 [ 25  36  84 152  85 312 105  70  31  37]
 [ 19  33 106  67 128  57 535  41   6  38]
 [ 52  51  52  54 141  69  66 393  28  95]
 [106  75  26  38  20  43  24  21 588  84]
 [ 55 130  12  33  13  24  41  34  98 541]]
```

1인덱스와 9 인덱스 의 이미지를 가장 많이 틀린것을 알 수 있다.

오차 행렬 분석
<Error Analysis>

- 오차행렬 plot에서 가장 많이 오류가 발생한 인덱스(1,9)이 어떤 데이터인지 분석하고 이유를 찾아본다.
- 인덱스 1 = automobile
- 인덱스 9 = truck

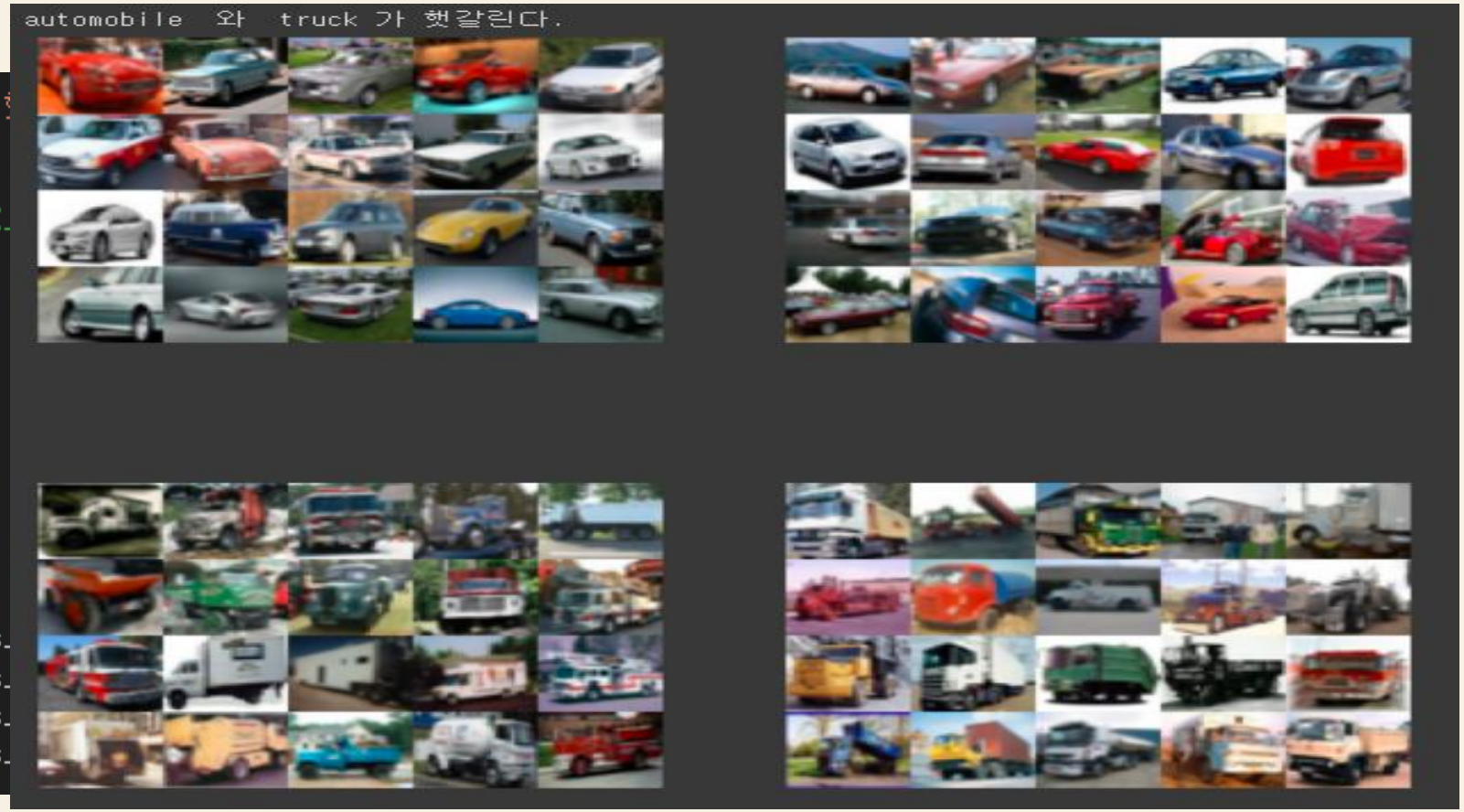
```
print(class_names[1]," 와 ", class_names[9], "가 헷

X_train = X_train.reshape(10000,3072)
y_train = y_train.reshape(-1,)# 차원이 안맞아서 X_

X_aa = X_train[(y_train == 1) & (y_pred == 1)]
X_ab = X_train[(y_train == 1) & (y_pred == 9)]
X_ba = X_train[(y_train == 9) & (y_pred == 1)]
X_bb = X_train[(y_train == 9) & (y_pred == 9)]

plt.figure(figsize=(12,12))
plt.subplot(221); plot_cifar10s(X_aa[:20], images_
plt.subplot(222); plot_cifar10s(X_ab[:20], images_
plt.subplot(223); plot_cifar10s(X_ba[:20], images_
plt.subplot(224); plot_cifar10s(X_bb[:20], images_
```

```
# 왼쪽 위 = automobil 인데 automobil이라고 한것. ->맞춘것
# 오른쪽 위 = automobil 인데 truck 이라고 한것. -> 틀린것
# 왼쪽 밑 = truck 인데 automobil이라고 한것. ->틀린것
# 오른쪽 밑 = truck 인데 truck 이라고 한것. -> 맞춘것
```



Project Result

프로젝트 결과

- CIFAR10 데이터셋은 10가지 클래스로 구성된 총 6만개의 컬러 이미지가 (32*32*3)의 형태로 되어있다.
- RandomForest 모델로는 성능에 한계가 있다.
- 인공신경망을 이용해도 성능에 한계가 존재했다. 부가적인 기능들을 더 추가해서 성능을 높여야 한다.
- 가장 많이 틀리는 데이터는 automobile 과 truck이다.

Thank You!

감사합니다.