Teerameth Rassameecharoenchai

61340500032

Object Recognition

The objective of this lab is very simple, to recognize objects in images. You will be working with a well-known dataset called CIFAR-10.

You can learn more about this dataset and download it here:

https://www.cs.toronto.edu/~kriz/cifar.html

In the webpage above, they also included a few publications based on CIFAR-10 data, which showed some amazing accuracies. The worst network on the page (a shallow convolutional neural network) can classify images with rouhgly 75% accuracy.

1. Write a function to load data

The dataset webpage in the previous section also provide a simple way to load data from your harddrive using pickle. You may use their function for this exercise.

Construct two numpy arrays for train images and train labels from data_batch_1 to data_batch_5. Then, construct two numpy arrays for test images, and test labels from test batch file. The original image size is 32 x 32 x 3. You may flatten the arrays so the final arrays are of size 1 x 3072.

1

```
# Using dataset & loding function from https://www.cs.toronto.edu/~kriz/cifar.html

import numpy as np
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
# Extract downloaded file in same folder
file_path = 'cifar-10-python/cifar-10-batches-py/'
file_names_train = ['data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5']
file_name_test = 'test_batch'
```

2

```
dataset_test = unpickle(file_path+file_name_test)
```

```
classes_name = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
X_test = dataset_test[b'data']
y_test = dataset_test[b'labels']
print("Testing set contain %d images."%X_test.shape[0]) # got (10000, 3072)
X_train = []
y_train = []
for file_name_train in file_names_train:
    dataset_train = unpickle(file_path+file_name_train)
    # concatenate training set to single list
    X_train.append(dataset_train[b'data'])
    y_train = y_train+dataset_train[b'labels']
X_train = np.concatenate(tuple(X_train), axis=0)
print("Training set contain %d images."%X_train.shape[0]) # got (50000, 3072)
```

```
Testing set contain 10000 images.
Training set contain 50000 images.
```

2. Classify Dogs v.s. Cats

Let’s start simple by creating logistic regression model to classify images. We will select only two classes of images for this exercise.

1. From 50,000 train images and 10,000 test images, we want to reduce the data size. Write code to filter only dog images (label = 3) and cat images (label = 5).
2. Create a logistic regression model to classify cats and dogs. Report your accuracy.

วง loop เพื่อกรองออกมาเฉพาะ class 3 และ 5

3
```
# CAT -> label 3
# DOG -> label 5
cat_vs_dog = {'X_train':[], 'y_train':[], 'X_test':[], 'y_test':[]}
for i in range(X_train.shape[0]):
    if y_train[i] in [3, 5]:
        cat_vs_dog['X_train'].append(X_train[i])
        cat_vs_dog['y_train'].append(y_train[i])
print("Filtered training set: %d images."%len(cat_vs_dog['X_train']))
for i in range(X_test.shape[0]):
    if y_test[i] in [3, 5]:
        cat_vs_dog['X_test'].append(X_test[i])
        cat_vs_dog['y_test'].append(y_test[i])
print("Filtered testing set: %d images."%len(cat_vs_dog['X_test']))
```

```
Filtered training set: 10000 images.
Filtered testing set: 2000 images.
```

4
```
# Create Logistic Regression Model

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
LR = LogisticRegression(max_iter=1000, random_state=136)
LR.fit(cat_vs_dog['X_train'], cat_vs_dog['y_train'])
y_pred = LR.predict(cat_vs_dog['X_test']) # Predict test set
print("Confusion Matrix")
print(confusion_matrix(cat_vs_dog['y_test'], y_pred))
print("Classification report")
```

```
print(classification_report(cat_vs_dog['y_test'], y_pred))
print("Accuracy")
print(accuracy_score(cat_vs_dog['y_test'], y_pred))
```

```
Confusion Matrix
[[538 462]
 [428 572]]
Classification report
              precision    recall  f1-score   support

           3       0.56      0.54      0.55      1000
           5       0.55      0.57      0.56      1000

    accuracy                           0.56      2000
   macro avg       0.56      0.55      0.55      2000
weighted avg       0.56      0.56      0.55      2000

Accuracy
0.555
```

```
/home/teera/anaconda3/envs/tf2/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:765: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

5 ได้ Accuracy ต่ำมาก ดีกว่าสุ่มแค่นิดเดียวเอง

3. The Real Challenge

The majority of your score for this lab will come from this real challenge. You are going to construct a neural network model to classify 10 classes of images from CIFAR-10 dataset. You will get half the credits for this one if you complete the assignment, and will get another half if you can exceed the target accuracy of 75%. (You may use any combination of sklearn, opencv, or tensorflow to do this exercise).

Design at least 3 variants of neural network models. Each model should have different architectures. (Do not vary just a few parameters, the architecture of the network must change in each model). In your notebook, explain your experiments in details and display the accuracy score for each experiment.

ใช้ CNN เนื่องจากเหมาะสมกับการ classify ข้อมูลที่เป็นภาพ

และต้อง Preprocess ภาพก่อนเนื่องจากข้อมูลภาพที่ให้มาถูก flatten ไปแล้วแต่เราต้องการทำ convolution ก่อนเลยต้องแปลงกลับเป็น (32,32,3)

ใน web ของ cifar-10 บอกว่า

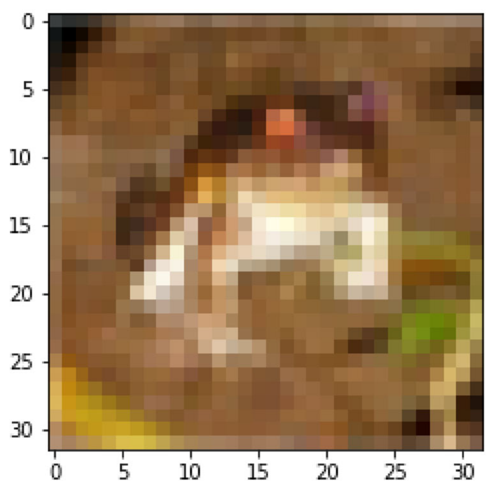
"data -- a 10000x3072 numpy array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image."

ก็เลยใช้ NumPy ในการแปลงภาพตามคำอธิบายดังกล่าวได้แบบนี้เพื่อให้ได้เป็นภาพในแบบ BGR ตามมาตรฐาน OpenCV

5
```
X_train = [np.transpose(np.reshape(X_train[i],(3,32,32)), (1,2,0)) for i in range(len(X_train))]
X_test = [np.transpose(np.reshape(X_test[i], (3,32,32)), (1,2,0)) for i in range(len(X_test))]
```

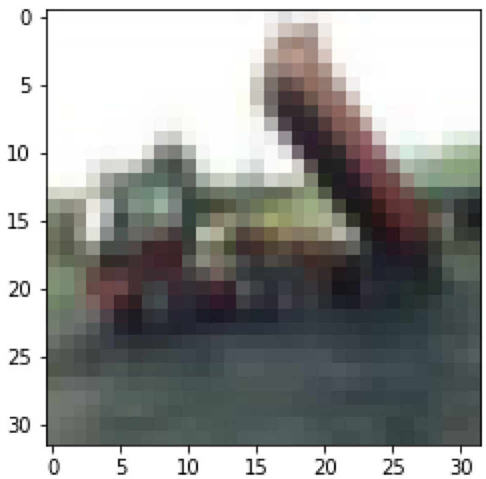
6
```
# ลองตรวจดูว่าแปลง array มาถูกมั้ย

import matplotlib.pyplot as plt
# Training set
for i in range(5):
    print(classes_name[y_train[i]])
    plt.imshow(X_train[i])
    plt.show()
# Test set
for i in range(5):
    print(classes_name[y_test[i]])
    plt.imshow(X_test[i])
    plt.show()
```

```
frog
```

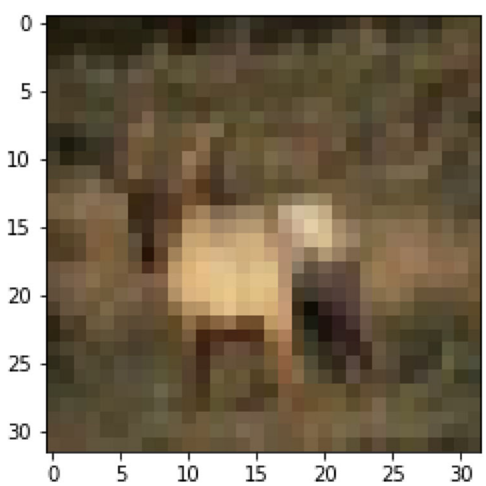



```
truck
```

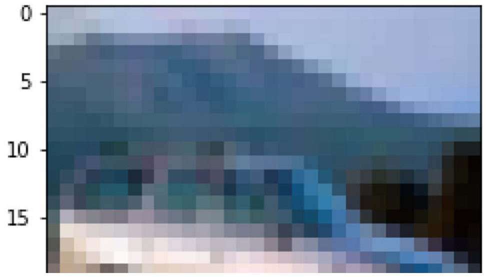
truck



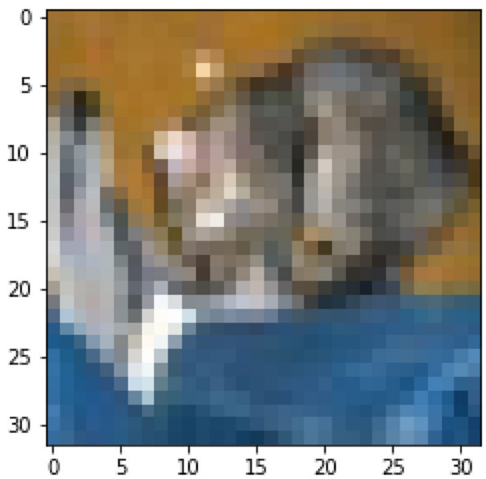
deer



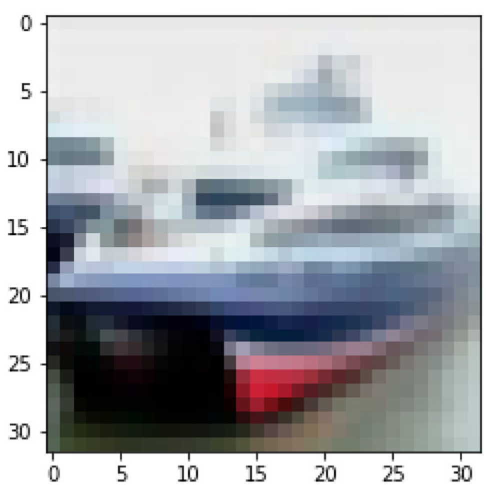
automobile



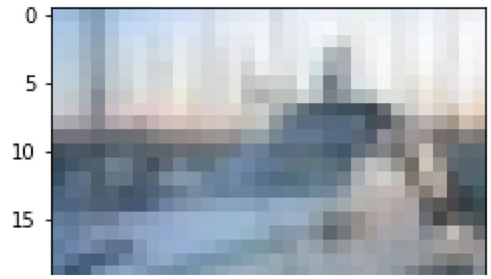
cat



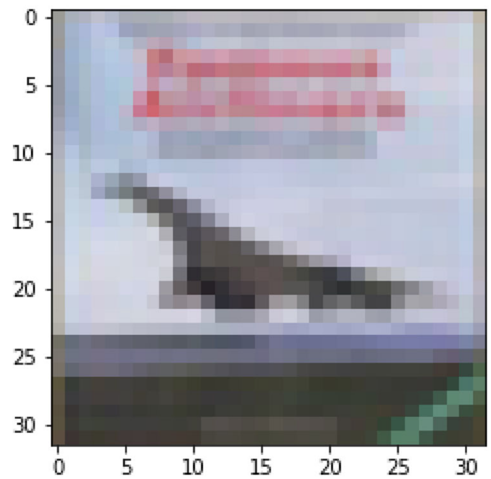
ship



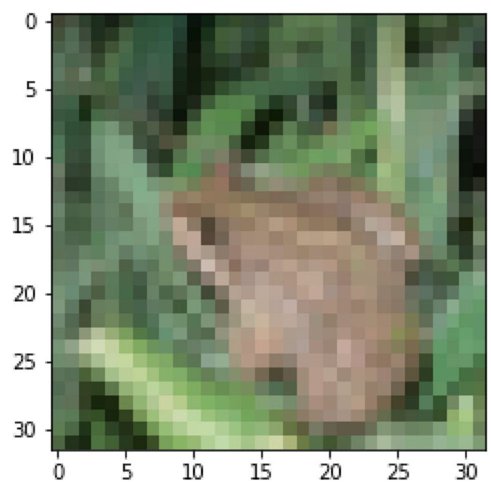
ship



airplane



frog



ใช้ model เดียวกับที่ใช้ใน AI ของ Module8-9 ใช้ optimizer เป็น SGD

7

```
from sklearn.preprocessing import LabelBinarizer
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.core import Activation, Flatten, Dense, Dropout
```

```
# Scale dataset in to range [0, 1]
trainX = np.asarray(X_train).astype("float") / 255.0
testX = np.asarray(X_test).astype("float") / 255.0
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(y_train)
testY = lb.transform(y_test)
```

Model 1: ShallowNet (ไม่ได้ทำ custom convolution layer เอง แบบใน module)

INPUT => CONV => RELU => FC ใช้แค่ 1 convolution layer แล้ว flatten เข้า fully connected layer เลย

8
```
# Build model

class ShallowNet:
        @staticmethod
        def build():
                model = Sequential()
                model.add(Conv2D(32, (3, 3), padding="same", input_shape=(32,32,3))) # input shape=(32,32
                model.add(Activation("relu"))
                model.add(Flatten())
                model.add(Dense(10)) # 10 classes
                model.add(Activation("softmax"))
                return model
```

9
```
opt = SGD(lr=0.01, decay=0.01 / 40, momentum=0.9, nesterov=True)
model = ShallowNet.build()
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), batch_size=64, epochs=40, verbose=1)
predictions = model.predict(testX, batch_size=64)
print(classification_report(testY.argmax(axis=1), predictions.argmax(axis=1), target_names=classes_name))
```

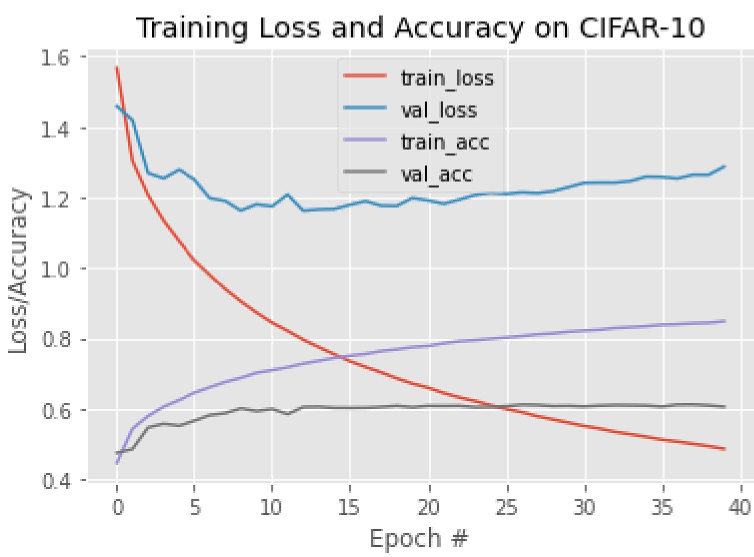
```
Epoch 1/40
782/782 [==============================] - 6s 5ms/step - loss: 1.7642 - accuracy: 0.3736 - val_loss: 1.45
Epoch 2/40
782/782 [==============================] - 2s 3ms/step - loss: 1.3238 - accuracy: 0.5353 - val_loss: 1.41
Epoch 3/40
782/782 [==============================] - 3s 3ms/step - loss: 1.2106 - accuracy: 0.5786 - val_loss: 1.26
Epoch 4/40
782/782 [==============================] - 2s 3ms/step - loss: 1.1362 - accuracy: 0.6057 - val_loss: 1.25
Epoch 5/40
782/782 [==============================] - 2s 3ms/step - loss: 1.0749 - accuracy: 0.6267 - val_loss: 1.27
Epoch 6/40
782/782 [==============================] - 2s 3ms/step - loss: 1.0113 - accuracy: 0.6469 - val_loss: 1.25
Epoch 7/40
782/782 [==============================] - 2s 3ms/step - loss: 0.9758 - accuracy: 0.6621 - val_loss: 1.19
Epoch 8/40
782/782 [==============================] - 2s 3ms/step - loss: 0.9278 - accuracy: 0.6807 - val_loss: 1.19
Epoch 9/40
782/782 [==============================] - 2s 3ms/step - loss: 0.8883 - accuracy: 0.6935 - val_loss: 1.16
Epoch 10/40
782/782 [==============================] - 2s 3ms/step - loss: 0.8641 - accuracy: 0.7053 - val_loss: 1.18
Epoch 11/40
```

```
782/782 [==============================] - 2s 3ms/step - loss: 0.8351 - accuracy: 0.7147 - val_loss: 1.17
Epoch 12/40
782/782 [==============================] - 2s 3ms/step - loss: 0.8181 - accuracy: 0.7194 - val_loss: 1.20
Epoch 13/40
782/782 [==============================] - 2s 3ms/step - loss: 0.7878 - accuracy: 0.7323 - val_loss: 1.16
Epoch 14/40
782/782 [==============================] - 2s 3ms/step - loss: 0.7644 - accuracy: 0.7395 - val_loss: 1.16
Epoch 15/40
782/782 [==============================] - 2s 3ms/step - loss: 0.7485 - accuracy: 0.7452 - val_loss: 1.16
Epoch 16/40
782/782 [==============================] - 2s 3ms/step - loss: 0.7216 - accuracy: 0.7565 - val_loss: 1.17
Epoch 17/40
782/782 [==============================] - 2s 3ms/step - loss: 0.7075 - accuracy: 0.7600 - val_loss: 1.18
Epoch 18/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6962 - accuracy: 0.7679 - val_loss: 1.17
Epoch 19/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6754 - accuracy: 0.7742 - val_loss: 1.17
Epoch 20/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6590 - accuracy: 0.7803 - val_loss: 1.19
Epoch 21/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6515 - accuracy: 0.7813 - val_loss: 1.19
Epoch 22/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6355 - accuracy: 0.7919 - val_loss: 1.18
Epoch 23/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6233 - accuracy: 0.7960 - val_loss: 1.19
Epoch 24/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6185 - accuracy: 0.7981 - val_loss: 1.20
Epoch 25/40
782/782 [==============================] - 2s 3ms/step - loss: 0.6004 - accuracy: 0.8023 - val_loss: 1.21
Epoch 26/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5921 - accuracy: 0.8060 - val_loss: 1.21
Epoch 27/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5760 - accuracy: 0.8145 - val_loss: 1.21
Epoch 28/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5624 - accuracy: 0.8199 - val_loss: 1.21
Epoch 29/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5606 - accuracy: 0.8190 - val_loss: 1.21
Epoch 30/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5523 - accuracy: 0.8218 - val_loss: 1.22
Epoch 31/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5410 - accuracy: 0.8276 - val_loss: 1.24
Epoch 32/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5330 - accuracy: 0.8284 - val_loss: 1.24
Epoch 33/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5228 - accuracy: 0.8344 - val_loss: 1.24
Epoch 34/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5161 - accuracy: 0.8385 - val_loss: 1.24
Epoch 35/40
782/782 [==============================] - 2s 3ms/step - loss: 0.5151 - accuracy: 0.8378 - val_loss: 1.25
Epoch 36/40
782/782 [==============================] - 2s 3ms/step - loss: 0.4982 - accuracy: 0.8441 - val_loss: 1.25
Epoch 37/40
782/782 [==============================] - 2s 3ms/step - loss: 0.4994 - accuracy: 0.8429 - val_loss: 1.25
Epoch 38/40
782/782 [==============================] - 2s 3ms/step - loss: 0.4936 - accuracy: 0.8451 - val_loss: 1.26
Epoch 39/40
782/782 [==============================] - 2s 3ms/step - loss: 0.4876 - accuracy: 0.8474 - val_loss: 1.26
Epoch 40/40
782/782 [==============================] - 2s 3ms/step - loss: 0.4783 - accuracy: 0.8518 - val_loss: 1.28
              precision    recall  f1-score   support

    airplane       0.60      0.68      0.64      1000
  automobile       0.75      0.73      0.74      1000
        bird       0.47      0.43      0.45      1000
         cat       0.45      0.36      0.40      1000
```

```
        deer       0.53      0.55      0.54      1000
         dog       0.56      0.45      0.50      1000
        frog       0.62      0.78      0.69      1000
       horse       0.62      0.70      0.66      1000
        ship       0.72      0.72      0.72      1000
       truck       0.69      0.65      0.67      1000

    accuracy                           0.61     10000
   macro avg       0.60      0.61      0.60     10000
weighted avg       0.60      0.61      0.60     10000
```

10
```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 40), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 40), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 40), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 40), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on CIFAR-10")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

10
```
<matplotlib.legend.Legend at 0x7f1c101a5510>
```



Validation loss ลู่เข้าตั้งแต่ epoch ที่ 10 แล้ว หลังจากนั้นเริ่ม overfit

Model 2: LeNet

(modified version of Yann Lecun's LeNet)

INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC => RELU => FC ลอกมาจาก architecture ของ LeNet ต้นฉบับแต่เปลี่ยน activation function มีการทำ max pooling ขนาด 2x2 จำนวน 2 รอบเพิ่มเข้ามาจาก ShallowNet

11
```
class LeNet:
        @staticmethod
```

```
    def build(activation="relu", n_conv1=20, n_conv2=50, conv_size=(5,5), n_fc=500):
        model = Sequential()
        model.add(Conv2D(n_conv1, conv_size, padding="same", input_shape=(32,32,3)))
        model.add(Activation(activation))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(n_conv2, conv_size, padding="same"))
        model.add(Activation(activation))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Flatten())
        model.add(Dense(n_fc))
        model.add(Activation(activation))
        model.add(Dense(10)) # 10 classes
        model.add(Activation("softmax"))
        return model
```

12
```
opt = SGD(lr=0.01, decay=0.01 / 40, momentum=0.9, nesterov=True)
model = LeNet.build()
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), batch_size=64, epochs=40, verbose=1)
predictions = model.predict(testX, batch_size=64)
print(classification_report(testY.argmax(axis=1), predictions.argmax(axis=1), target_names=classes_name))
```

```
Epoch 1/40
782/782 [==============================] - 9s 8ms/step - loss: 1.8002 - accuracy: 0.3470 - val_loss: 1.27
Epoch 2/40
782/782 [==============================] - 3s 4ms/step - loss: 1.1754 - accuracy: 0.5828 - val_loss: 1.04
Epoch 3/40
782/782 [==============================] - 3s 4ms/step - loss: 0.9539 - accuracy: 0.6660 - val_loss: 0.98
Epoch 4/40
782/782 [==============================] - 3s 4ms/step - loss: 0.8007 - accuracy: 0.7197 - val_loss: 0.93
Epoch 5/40
782/782 [==============================] - 3s 4ms/step - loss: 0.6594 - accuracy: 0.7714 - val_loss: 0.97
Epoch 6/40
782/782 [==============================] - 3s 4ms/step - loss: 0.5379 - accuracy: 0.8146 - val_loss: 0.94
Epoch 7/40
782/782 [==============================] - 3s 4ms/step - loss: 0.4210 - accuracy: 0.8585 - val_loss: 0.93
Epoch 8/40
782/782 [==============================] - 3s 4ms/step - loss: 0.3097 - accuracy: 0.8975 - val_loss: 0.98
Epoch 9/40
782/782 [==============================] - 3s 4ms/step - loss: 0.2201 - accuracy: 0.9313 - val_loss: 1.09
Epoch 10/40
782/782 [==============================] - 3s 4ms/step - loss: 0.1440 - accuracy: 0.9600 - val_loss: 1.15
Epoch 11/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0900 - accuracy: 0.9771 - val_loss: 1.26
Epoch 12/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0535 - accuracy: 0.9899 - val_loss: 1.32
Epoch 13/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0290 - accuracy: 0.9971 - val_loss: 1.40
Epoch 14/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0167 - accuracy: 0.9992 - val_loss: 1.50
Epoch 15/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0103 - accuracy: 0.9997 - val_loss: 1.55
Epoch 16/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0077 - accuracy: 0.9999 - val_loss: 1.57
Epoch 17/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0064 - accuracy: 0.9998 - val_loss: 1.60
Epoch 18/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0053 - accuracy: 0.9999 - val_loss: 1.63
Epoch 19/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0044 - accuracy: 0.9999 - val_loss: 1.66
Epoch 20/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 1.68
Epoch 21/40
```

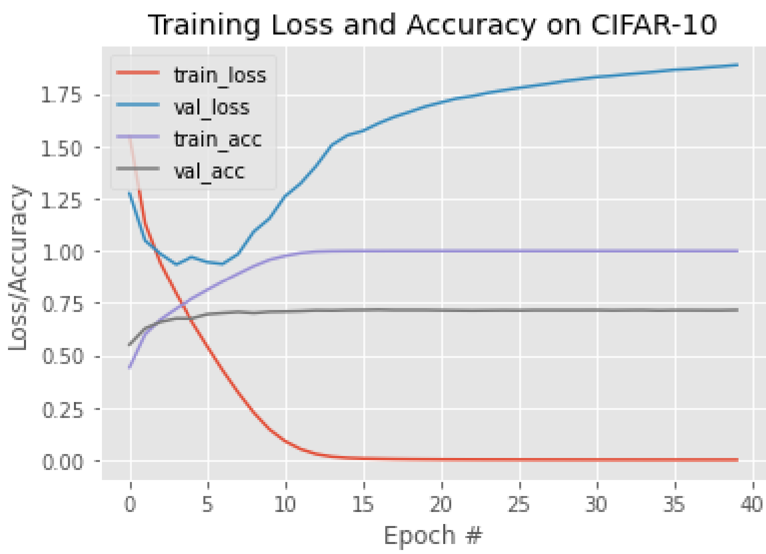
```
782/782 [==============================] - 3s 4ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 1.70
Epoch 22/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 1.72
Epoch 23/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 1.73
Epoch 24/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 1.75
Epoch 25/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 1.76
Epoch 26/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 1.77
Epoch 27/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 1.78
Epoch 28/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 1.79
Epoch 29/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 1.81
Epoch 30/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 1.82
Epoch 31/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 1.82
Epoch 32/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 1.83
Epoch 33/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 1.84
Epoch 34/40
782/782 [==============================] - 4s 5ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 1.84
Epoch 35/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 1.85
Epoch 36/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 1.86
Epoch 37/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.86
Epoch 38/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.87
Epoch 39/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.88
Epoch 40/40
782/782 [==============================] - 3s 4ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.88
              precision    recall  f1-score   support

    airplane       0.73      0.78      0.75      1000
  automobile       0.83      0.81      0.82      1000
        bird       0.63      0.59      0.61      1000
         cat       0.54      0.53      0.53      1000
        deer       0.67      0.67      0.67      1000
         dog       0.62      0.61      0.62      1000
        frog       0.77      0.81      0.79      1000
       horse       0.76      0.77      0.77      1000
        ship       0.84      0.81      0.83      1000
       truck       0.78      0.76      0.77      1000

    accuracy                           0.72     10000
   macro avg       0.72      0.72      0.72     10000
weighted avg       0.72      0.72      0.72     10000
```

13

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 40), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 40), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 40), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 40), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on CIFAR-10")
```

```
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

13 <matplotlib.legend.Legend at 0x7f1bc538f990>



Validation loss ลู่เข้าตั้งแต่ epoch ที่ 5 หลังจากนั้นเกิด overfitting จน val_loss กระโดดแต่ loss ยังคงลดลงเรื่อย ๆ น่าจะลองลดจำนวน neuron ใน Fully Connected Layer ดู

Model 3: MiniVGGNet (ลอกมาจาก https://github.com/matvi/miniVGGNet/blob/master/cnn/MiniVGGNet.py)

INPUT => CONV => RELU => BN => CONV => RELU => BN => POOL => DROPOUT => CONV => RELU => BN => CONV => RELU => BN => POOL => DROPOUT => FC => RELU => BN => DROPOUT => FC => SOFTMAX

เพิ่ม Batch Normalization Layer เข้าไปเพื่อให้การ train มีสเถียรภาพมากขึ้นและเพิ่ม Dropout Layer เพื่อลดการเกิด overfitting

14
```
class MiniVGGNet:
        @staticmethod
        def build(conv_size=(3,3), pool_size=(2,2), dropout1=0.25, n_conv1=32, n_conv2=32, n_conv3=64, n_
                model = Sequential()
                model.add(Conv2D(n_conv1, conv_size, padding="same", input_shape=(32,32,3)))
                model.add(Activation('relu'))
                model.add(BatchNormalization(axis=-1))
                model.add(Conv2D(n_conv2, conv_size, padding="same"))
                model.add(Activation('relu'))
                model.add(BatchNormalization(axis=-1))
                model.add(MaxPooling2D(pool_size=pool_size))
                model.add(Dropout(dropout1))
                model.add(Conv2D(n_conv3, conv_size, padding="same"))
                model.add(Activation('relu'))
                model.add(BatchNormalization(axis=-1))
```

```
                model.add(Conv2D(n_conv4, conv_size, padding="same"))
                model.add(Activation('relu'))
                model.add(BatchNormalization(axis=-1))
                model.add(MaxPooling2D(pool_size=pool_size))
                model.add(Dropout(dropout2))
                model.add(Flatten())
                model.add(Dense(n_fc))
                model.add(Activation("relu"))
                model.add(BatchNormalization())
                model.add(Dropout(dropout3))
                model.add(Dense(10)) # 10 classes
                model.add(Activation("softmax"))
                return model
```

15

```
opt = SGD(lr=0.01, decay=0.01 / 40, momentum=0.9, nesterov=True)
model = MiniVGGNet.build()
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), batch_size=64, epochs=40, verbose=1)
predictions = model.predict(testX, batch_size=64)
print(classification_report(testY.argmax(axis=1), predictions.argmax(axis=1), target_names=classes_name))
```

```
Epoch 1/40
782/782 [==============================] - 19s 16ms/step - loss: 2.0254 - accuracy: 0.3684 - val_loss: 1.
Epoch 2/40
782/782 [==============================] - 6s 8ms/step - loss: 1.2514 - accuracy: 0.5614 - val_loss: 1.05
Epoch 3/40
782/782 [==============================] - 7s 9ms/step - loss: 1.0119 - accuracy: 0.6459 - val_loss: 0.93
Epoch 4/40
782/782 [==============================] - 6s 8ms/step - loss: 0.8860 - accuracy: 0.6893 - val_loss: 0.86
Epoch 5/40
782/782 [==============================] - 6s 8ms/step - loss: 0.8107 - accuracy: 0.7172 - val_loss: 0.71
Epoch 6/40
782/782 [==============================] - 6s 8ms/step - loss: 0.7475 - accuracy: 0.7390 - val_loss: 0.77
Epoch 7/40
782/782 [==============================] - 6s 8ms/step - loss: 0.7080 - accuracy: 0.7545 - val_loss: 0.65
Epoch 8/40
782/782 [==============================] - 6s 8ms/step - loss: 0.6577 - accuracy: 0.7674 - val_loss: 0.70
Epoch 9/40
782/782 [==============================] - 6s 8ms/step - loss: 0.6264 - accuracy: 0.7809 - val_loss: 0.62
Epoch 10/40
782/782 [==============================] - 6s 8ms/step - loss: 0.5816 - accuracy: 0.7946 - val_loss: 0.64
Epoch 11/40
782/782 [==============================] - 6s 8ms/step - loss: 0.5581 - accuracy: 0.8025 - val_loss: 0.61
Epoch 12/40
782/782 [==============================] - 7s 8ms/step - loss: 0.5398 - accuracy: 0.8094 - val_loss: 0.59
Epoch 13/40
782/782 [==============================] - 6s 8ms/step - loss: 0.5163 - accuracy: 0.8180 - val_loss: 0.57
Epoch 14/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4937 - accuracy: 0.8241 - val_loss: 0.58
Epoch 15/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4819 - accuracy: 0.8294 - val_loss: 0.57
Epoch 16/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4600 - accuracy: 0.8351 - val_loss: 0.60
Epoch 17/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4509 - accuracy: 0.8377 - val_loss: 0.58
Epoch 18/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4358 - accuracy: 0.8457 - val_loss: 0.57
Epoch 19/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4200 - accuracy: 0.8499 - val_loss: 0.57
Epoch 20/40
782/782 [==============================] - 6s 8ms/step - loss: 0.4003 - accuracy: 0.8581 - val_loss: 0.56
Epoch 21/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3966 - accuracy: 0.8588 - val_loss: 0.56
```

```
Epoch 22/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3760 - accuracy: 0.8661 - val_loss: 0.55
Epoch 23/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3739 - accuracy: 0.8649 - val_loss: 0.60
Epoch 24/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3601 - accuracy: 0.8707 - val_loss: 0.59
Epoch 25/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3554 - accuracy: 0.8716 - val_loss: 0.57
Epoch 26/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3461 - accuracy: 0.8760 - val_loss: 0.54
Epoch 27/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3340 - accuracy: 0.8821 - val_loss: 0.58
Epoch 28/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3231 - accuracy: 0.8848 - val_loss: 0.55
Epoch 29/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3232 - accuracy: 0.8837 - val_loss: 0.55
Epoch 30/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3091 - accuracy: 0.8881 - val_loss: 0.55
Epoch 31/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2948 - accuracy: 0.8948 - val_loss: 0.57
Epoch 32/40
782/782 [==============================] - 6s 8ms/step - loss: 0.3016 - accuracy: 0.8900 - val_loss: 0.55
Epoch 33/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2932 - accuracy: 0.8944 - val_loss: 0.55
Epoch 34/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2918 - accuracy: 0.8961 - val_loss: 0.56
Epoch 35/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2792 - accuracy: 0.9003 - val_loss: 0.55
Epoch 36/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2806 - accuracy: 0.8995 - val_loss: 0.56
Epoch 37/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2769 - accuracy: 0.8977 - val_loss: 0.55
Epoch 38/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2633 - accuracy: 0.9055 - val_loss: 0.55
Epoch 39/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2580 - accuracy: 0.9075 - val_loss: 0.58
Epoch 40/40
782/782 [==============================] - 6s 8ms/step - loss: 0.2603 - accuracy: 0.9073 - val_loss: 0.56
              precision    recall  f1-score   support

    airplane       0.86      0.81      0.84      1000
  automobile       0.93      0.89      0.90      1000
        bird       0.79      0.70      0.74      1000
         cat       0.67      0.69      0.68      1000
        deer       0.77      0.82      0.79      1000
         dog       0.73      0.77      0.75      1000
        frog       0.84      0.89      0.86      1000
       horse       0.90      0.85      0.87      1000
        ship       0.90      0.91      0.91      1000
       truck       0.87      0.91      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000
```

16
```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 40), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 40), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 40), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 40), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on CIFAR-10")
plt.xlabel("Epoch #")
```

```
plt.ylabel("Loss/Accuracy")
plt.legend()
```

16 `<matplotlib.legend.Legend at 0x7f1b6c0bc150>`