```
In [1]: # Import required packages
import warnings
warnings.filterwarnings("ignore", message="numpy.dtype size changed")
warnings.filterwarnings("ignore", message="numpy.ufunc size changed")

import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import keras
import tensorflow as tf
import utils
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
import timeit
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from tensorflow.keras.utils import to_categorical
```

1. Load the datasets

For the project, we provide a training set with 50000 images in the directory `../data/images/` with:

- noisy labels for all images provided in `../data/noisy_label.csv`;
- clean labels for the first 10000 images provided in `../data/clean_labels.csv`.

```
In [2]: %%time
# [DO NOT MODIFY THIS CELL]

# load the images
n_img = 50000
n_noisy = 40000
n_clean_noisy = n_img - n_noisy
imgs = np.empty((n_img,32,32,3))
for i in range(n_img):
    img_fn = f'../data/images/{i+1:05d}.png'
    imgs[i,:,:,:]=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB)

# load the labels
clean_labels = np.genfromtxt('../data/clean_labels.csv', delimiter=',',
dtype="int8")
noisy_labels = np.genfromtxt('../data/noisy_labels.csv', delimiter=',',
dtype="int8")
```

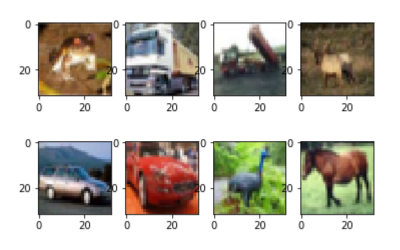
```
CPU times: user 8.55 s, sys: 10.4 s, total: 19 s
Wall time: 41.3 s
```

```
In [3]: predicted_clean_labels = np.genfromtxt('../output/predicted_clean_label.
csv', delimiter=',', dtype="int8")
```

For illustration, we present a small subset (of size 8) of the images with their clean and noisy labels in `clean_noisy_trainset`. You are encouraged to explore more characteristics of the label noises on the whole dataset.

```
In [4]: %%time
# [DO NOT MODIFY THIS CELL]

fig = plt.figure()

ax1 = fig.add_subplot(2,4,1)
ax1.imshow(imgs[0]/255)
ax2 = fig.add_subplot(2,4,2)
ax2.imshow(imgs[1]/255)
ax3 = fig.add_subplot(2,4,3)
ax3.imshow(imgs[2]/255)
ax4 = fig.add_subplot(2,4,4)
ax4.imshow(imgs[3]/255)
ax1 = fig.add_subplot(2,4,5)
ax1.imshow(imgs[4]/255)
ax2 = fig.add_subplot(2,4,6)
ax2.imshow(imgs[5]/255)
ax3 = fig.add_subplot(2,4,7)
ax3.imshow(imgs[6]/255)
ax4 = fig.add_subplot(2,4,8)
ax4.imshow(imgs[7]/255)

# The class-label correspondence
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# print clean labels
print('Clean labels:')
print(' '.join('%5s' % classes[clean_labels[j]] for j in range(8)))
# print noisy labels
print('Noisy labels:')
print(' '.join('%5s' % classes[noisy_labels[j]] for j in range(8)))
```

```
Clean labels:
 frog truck truck  deer   car   car  bird horse
Noisy labels:
  cat   dog truck  frog   dog  ship  bird  deer
CPU times: user 245 ms, sys: 151 ms, total: 396 ms
Wall time: 465 ms
```



2. The predictive model

We consider a baseline model directly on the noisy dataset without any label corrections. RGB histogram features are extracted to fit a logistic regression model.

2.1. Baseline Model

```
In [5]: # [DO NOT MODIFY THIS CELL]
# RGB histogram dataset construction
no_bins = 6
bins = np.linspace(0,255,no_bins) # the range of the rgb histogram
target_vec = np.empty(n_img)
feature_mtx = np.empty((n_img,3*(len(bins)-1)))
i = 0
for i in range(n_img):
    # The target vector consists of noisy labels
    target_vec[i] = noisy_labels[i]

    # Use the numbers of pixels in each bin for all three channels as th
e features
    feature1 = np.histogram(imgs[i][:,:,0],bins=bins)[0]
    feature2 = np.histogram(imgs[i][:,:,1],bins=bins)[0]
    feature3 = np.histogram(imgs[i][:,:,2],bins=bins)[0]

    # Concatenate three features
    feature_mtx[i,] = np.concatenate((feature1, feature2, feature3), axi
s=None)
    i += 1
```

```
In [6]: # [DO NOT MODIFY THIS CELL]
# Train a logistic regression model
clf = LogisticRegression(random_state=0).fit(feature_mtx, target_vec)
```

For the convenience of evaluation, we write the following function `predictive_model` that does the label prediction. **For your predictive model, feel free to modify the function, but make sure the function takes an RGB image of numpy.array format with dimension $32\times32\times3$ as input, and returns one single label as output.**

```
In [7]: # [DO NOT MODIFY THIS CELL]
def baseline_model(image):
    '''
    This is the baseline predictive model that takes in the image and re
turns a label prediction
    '''
    # The class-label correspondence
    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    feature1 = np.histogram(image[:,:,0],bins=bins)[0]
    feature2 = np.histogram(image[:,:,1],bins=bins)[0]
    feature3 = np.histogram(image[:,:,2],bins=bins)[0]
    feature = np.concatenate((feature1, feature2, feature3), axis=None).
reshape(1,-1)
    label_num = int(clf.predict(feature)[0])

    return classes[label_num]
```

```
In [8]: baseline_model(imgs[49998])
Out[8]: 'plane'
```

2.2. Model I

2.2 Model I

```
In [9]: # Splitting data into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(imgs, noisy_labels,
test_size = 0.2, random_state=42)

#validation set

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train,
test_size=0.25, random_state=42)

#normalizing x
x_train = np.array(x_train) / 255
x_test = np.array(x_test) / 255
x_valid = np.array(x_valid) / 255
```

```
In [10]: #create model
model = Sequential()

#add model layers
model.add(Conv2D(32, (3,3), padding="same", activation="relu", input_sha
pe=(32, 32, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dense(10, activation="softmax"))

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

#train the model
history = model.fit(
    x_train,
    to_categorical(y_train),
    epochs=6,
    validation_data=(x_valid, to_categorical(y_valid)))

# Epoch = 6 gives optimal model, Epoch > 6 tends to be overfitting
basecnn = model
```

```
Epoch 1/6
938/938 [==============================] - 59s 62ms/step - loss: 2.2907
- accuracy: 0.1269 - val_loss: 2.2694 - val_accuracy: 0.1520
Epoch 2/6
938/938 [==============================] - 50s 53ms/step - loss: 2.2551
- accuracy: 0.1710 - val_loss: 2.2476 - val_accuracy: 0.1812
Epoch 3/6
938/938 [==============================] - 48s 52ms/step - loss: 2.2360
- accuracy: 0.1911 - val_loss: 2.2399 - val_accuracy: 0.1993
Epoch 4/6
938/938 [==============================] - 49s 52ms/step - loss: 2.2192
- accuracy: 0.2067 - val_loss: 2.2336 - val_accuracy: 0.1991
Epoch 5/6
938/938 [==============================] - 51s 54ms/step - loss: 2.1970
- accuracy: 0.2232 - val_loss: 2.2357 - val_accuracy: 0.2055
Epoch 6/6
938/938 [==============================] - 51s 55ms/step - loss: 2.1690
- accuracy: 0.2377 - val_loss: 2.2333 - val_accuracy: 0.2055
```

```
In [11]: # CNN
def model_I(image):
    '''
    This function should takes in the image of dimension 32*32*3 as inpu
t and returns a label prediction
    '''
    #predict
    #x_test = np.array(image)/255
    predictions = basecnn.predict(image)
    np.argmax(predictions, axis=1)
    return np.argmax(predictions, axis=1)
```

```
In [12]: # test
start = timeit.default_timer()
history = model_I(x_test)
stop = timeit.default_timer()
total_time = (stop - start) / 60
print('Model I took', total_time,'minutes to predict the labels for x_te
st')
```

```
Model I took 0.08315052518333346 minutes to predict the labels for x_tes
t
```

```
In [13]: basecnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          36928

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 64)                262208

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 319,178
Trainable params: 319,178
Non-trainable params: 0
_________________________________________________________________
```

```
In [14]: #model_I(imgs[49999])
#np.argmax(basecnn.predict(np.array(imgs[49999:])/255))
#predictions = basecnn.predict(x_test[:25])
print(model_I(x_test[:25]))
print(y_test[:25])
```

```
[7 8 0 4 1 4 8 9 6 1 8 3 8 7 1 7 3 6 4 5 0 4 6 4 4]
[0 8 1 6 1 2 2 7 6 3 5 5 0 8 1 6 9 5 4 7 0 5 7 0 9]
```

2.3. Model II: cnn with predicted clean labels

```
In [15]: # load the predicted clean labels
predicted_clean_labels = np.genfromtxt('../output/predicted_clean_label.
csv', delimiter=',', dtype="int8")

# Splitting data into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(imgs, predicted_clea
n_labels, test_size = 0.2, random_state=42)

#validation set

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train,
test_size=0.25, random_state=42)

#normalizing x
x_train = np.array(x_train) / 255
x_test = np.array(x_test) / 255
x_valid = np.array(x_valid) / 255
```

```
In [16]: #create model
model = Sequential()

#add model layers
model.add(Conv2D(32, (3,3), padding="same", activation="relu", input_sha
pe=(32, 32, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dense(10, activation="softmax"))

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

#train the model
history = model.fit(
    x_train,
    to_categorical(y_train),
    epochs=8,
    validation_data=(x_valid, to_categorical(y_valid)))

# Epoch = 8 gives optimal model, Epoch > 8 tends to be overfitting
advancecnn = model
```

```
Epoch 1/8
938/938 [==============================] - 64s 67ms/step - loss: 1.6284
- accuracy: 0.4090 - val_loss: 1.3667 - val_accuracy: 0.5242
Epoch 2/8
938/938 [==============================] - 60s 64ms/step - loss: 1.2449
- accuracy: 0.5635 - val_loss: 1.2601 - val_accuracy: 0.5703
Epoch 3/8
938/938 [==============================] - 58s 62ms/step - loss: 1.0466
- accuracy: 0.6357 - val_loss: 1.0725 - val_accuracy: 0.6324
Epoch 4/8
938/938 [==============================] - 58s 62ms/step - loss: 0.9205
- accuracy: 0.6814 - val_loss: 0.9741 - val_accuracy: 0.6727
Epoch 5/8
938/938 [==============================] - 61s 66ms/step - loss: 0.8177
- accuracy: 0.7181 - val_loss: 0.9216 - val_accuracy: 0.6927
Epoch 6/8
938/938 [==============================] - 60s 64ms/step - loss: 0.7347
- accuracy: 0.7486 - val_loss: 0.9363 - val_accuracy: 0.6931
Epoch 7/8
938/938 [==============================] - 83s 88ms/step - loss: 0.6569
- accuracy: 0.7752 - val_loss: 0.9722 - val_accuracy: 0.6801
Epoch 8/8
938/938 [==============================] - 69s 73ms/step - loss: 0.5763
- accuracy: 0.8031 - val_loss: 0.9841 - val_accuracy: 0.6858
```

```
In [19]: # CNN
def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as inpu
t and returns a label prediction
    '''
    #predict
    #x_test = np.array(image)/255
    predictions = advancecnn.predict(image)
    np.argmax(predictions, axis=1)
    return np.argmax(predictions, axis=1)
```

```
In [20]: # test
start = timeit.default_timer()
history = model_II(x_test)
stop = timeit.default_timer()
total_time = (stop - start) / 60
print('Model II took', total_time,'minutes to predict the labels for x_t
est')
```

```
Model II took 0.07237173000000136 minutes to predict the labels for x_te
st
```

```
In [21]: advancecnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 8, 8, 64)          36928

 flatten_1 (Flatten)         (None, 4096)              0

 dense_2 (Dense)             (None, 64)                262208

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 319,178
Trainable params: 319,178
Non-trainable params: 0
_________________________________________________________________
```

```
In [22]: print(model_II(x_test[:25]))
print(y_test[:25])
```

```
[3 8 0 6 1 6 8 9 6 8 0 5 0 7 1 3 3 4 4 5 0 7 5 2 4]
[7 8 0 6 1 6 8 0 6 5 2 7 0 7 1 6 5 6 6 5 0 5 7 2 4]
```

3. Evaluation

For assessment, we will evaluate your final model on a hidden test dataset with clean labels by the `evaluation` function defined as follows. Although you will not have the access to the test set, the function would be useful for the model developments. For example, you can split the small training set, using one portion for weakly supervised learning and the other for validation purpose.

```
In [ ]: # [DO NOT MODIFY THIS CELL]
def evaluation(model, test_labels, test_imgs):
    y_true = test_labels
    y_pred = []
    for image in test_imgs:
        y_pred.append(model(image))
    print(classification_report(y_true, y_pred))
```

```
In [ ]: # [DO NOT MODIFY THIS CELL]
# This is the code for evaluating the prediction performance on a testse
t
# You will get an error if running this cell, as you do not have the tes
tset
# Nonetheless, you can create your own validation set to run the evlauat
ion
n_test = 10000
test_labels = np.genfromtxt('../data/test_labels.csv', delimiter=',', dt
ype="int8")
test_imgs = np.empty((n_test,32,32,3))
for i in range(n_test):
    img_fn = f'../data/test_images/test{i+1:05d}.png'
    test_imgs[i,:,:,:]=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB
)
evaluation(baseline_model, test_labels, test_imgs)
```

The overall accuracy is 0.24, which is better than random guess (which should have a accuracy around 0.10). For the project, you should try to improve the performance by the following strategies:

- Consider a better choice of model architectures, hyperparameters, or training scheme for the predictive model;
- Use both `clean_noisy_trainset` and `noisy_trainset` for model training via **weakly supervised learning** methods. One possible solution is to train a "label-correction" model using the former, correct the labels in the latter, and train the final predictive model using the corrected dataset.
- Apply techniques such as k-fold cross validation to avoid overfitting;
- Any other reasonable strategies.

```
In [23]: %whos
#labels = target_vec[25000:]
#evaluation(baseline_model, labels, imgs[25000:])
#imgs[:2]
```

```
Variable                 Type                  Data/Info
--------------------------------------------------------
Conv2D                   type                  <class 'keras.layers.conv
olutional.Conv2D'>
DataLoader               type                  <class 'torch.utils.data.
dataloader.DataLoader'>
Dense                    type                  <class 'keras.layers.cor
e.dense.Dense'>
Flatten                  type                  <class 'keras.layers.cor
e.flatten.Flatten'>
LogisticRegression       type                  <class 'sklearn.linear_mo
<...>stic.LogisticRegression'>
MaxPooling2D             type                  <class 'keras.layers.pool
ing.MaxPooling2D'>
Sequential               type                  <class 'keras.engine.sequ
ential.Sequential'>
advancecnn               Sequential            <keras.engine.sequential.
<...>object at 0x7fe9d3f36650>
ax1                      AxesSubplot           AxesSubplot(0.125,0.17023
2;0.168478x0.252717)
ax2                      AxesSubplot           AxesSubplot(0.327174,0.17
0232;0.168478x0.252717)
ax3                      AxesSubplot           AxesSubplot(0.529348,0.17
0232;0.168478x0.252717)
ax4                      AxesSubplot           AxesSubplot(0.731522,0.17
0232;0.168478x0.252717)
basecnn                  Sequential            <keras.engine.sequential.
<...>object at 0x7feaad27d950>
baseline_model           function              <function baseline_model
at 0x7feaad8513b0>
bins                     ndarray               6: 6 elems, type `float64
`, 48 bytes
classes                  tuple                 n=10
classification_report    function              <function classification_
<...>report at 0x7feac9eebdd0>
clean_labels             ndarray               10000: 10000 elems, type
`int8`, 10000 bytes
clf                      LogisticRegression    LogisticRegression(random
_state=0)
cv2                      module                <module 'cv2' from '/User
<...>ackages/cv2/__init__.py'>
feature1                 ndarray               5: 5 elems, type `int64`,
40 bytes
feature2                 ndarray               5: 5 elems, type `int64`,
40 bytes
feature3                 ndarray               5: 5 elems, type `int64`,
40 bytes
feature_mtx              ndarray               50000x15: 750000 elems, t
ype `float64`, 6000000 bytes (5.7220458984375 Mb)
fig                      Figure                Figure(432x288)
history                  ndarray               10000: 10000 elems, type
`int64`, 80000 bytes
i                        int                   50000
img_fn                   str                   ../data/images/50000.png
imgs                     ndarray               50000x32x32x3: 153600000
elems, type `float64`, 1228800000 bytes (1171.875 Mb)
keras                    module                <module 'keras' from '/Us
<...>kages/keras/__init__.py'>
model                    Sequential            <keras.engine.sequential.
<...>object at 0x7fe9d3f36650>
model_I                  function              <function model_I at 0x7f
e9d4f0ad40>
model_II                 function              <function model_II at 0x7
fe9ceb6d0e0>
n_clean_noisy            int                   10000
n_img                    int                   50000
n_noisy                  int                   40000
nn                       module                <module 'torch.nn' from
'<...>es/torch/nn/__init__.py'>
no_bins                  int                   6
noisy_labels             ndarray               50000: 50000 elems, type
`int8`, 50000 bytes
np                       module                <module 'numpy' from '/Us
<...>kages/numpy/__init__.py'>
optim                    module                <module 'torch.optim' fro
<...>torch/optim/__init__.py'>
plt                      module                <module 'matplotlib.pyplo
<...>es/matplotlib/pyplot.py'>
```

```
predicted_clean_labels   ndarray                 50000: 50000 elems, type
`int8`, 50000 bytes
shuffle                  function                <function shuffle at 0x7f
eac9e674d0>
start                    float                   1133.500139391
stop                     float                   1137.842443191
target_vec               ndarray                 50000: 50000 elems, type
`float64`, 400000 bytes (390.625 kb)
tf                       module                  <module 'tensorflow' from
<...>/tensorflow/__init__.py'>
timeit                   module                  <module 'timeit' from '/U
<...>lib/python3.7/timeit.py'>
to_categorical           function                <function to_categorical
at 0x7fead1a7bb90>
torch                    module                  <module 'torch' from '/Us
<...>kages/torch/__init__.py'>
torchvision              module                  <module 'torchvision' fro
<...>torchvision/__init__.py'>
total_time               float                   0.07237173000000136
train_test_split         function                <function train_test_spli
t at 0x7feaca099560>
transforms               module                  <module 'torchvision.tran
<...>/transforms/__init__.py'>
utils                    module                  <module 'utils' from '/Us
<...>rj3-group8/doc/utils.py'>
warnings                 module                  <module 'warnings' from
'<...>b/python3.7/warnings.py'>
x_test                   ndarray                 10000x32x32x3: 30720000 e
lems, type `float64`, 245760000 bytes (234.375 Mb)
x_train                  ndarray                 30000x32x32x3: 92160000 e
lems, type `float64`, 737280000 bytes (703.125 Mb)
x_valid                  ndarray                 10000x32x32x3: 30720000 e
lems, type `float64`, 245760000 bytes (234.375 Mb)
y_test                   ndarray                 10000: 10000 elems, type
`int8`, 10000 bytes
y_train                  ndarray                 30000: 30000 elems, type
`int8`, 30000 bytes
y_valid                  ndarray                 10000: 10000 elems, type
`int8`, 10000 bytes
```