main

November 2, 2022

[1]:
```
#!pip install opencv-python
```

[45]:
```
# Import required packages
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from tensorflow import keras
from matplotlib.pyplot import step
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.callbacks import ModelCheckpoint
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score, confusion_matrix,precision_score
from keras.layers import concatenate
from keras.layers import BatchNormalization
from sklearn.model_selection import train_test_split
import time
import pandas as pd
```

0.1 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`.

[46]:
```
# [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'train_data/images/{i+1:05d}.png'
    imgs[i,:,:,:]=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB)
```

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

[48]:
```
# [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
```

0.2 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.

0.2.1 2.1. Baseline Model

[49]:
```
# [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 the␣
 ↪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), axis=None)
    i += 1
```

[50]:
```
# [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 $32 \times 32 \times 3$ as input, and returns one single label as output.**

[51]:
```
# [DO NOT MODIFY THIS CELL]
def baseline_model(image):
    '''
    This is the baseline predictive model that takes in the image and returns a␣
 ↪label prediction
    '''
    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)
    return clf.predict(feature)
```

0.2.2 2.2. Model I

[52]:
```
class TimeHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, epoch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, epoch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)
```

[53]:
```
# [BUILD A MORE SOPHISTICATED PREDICTIVE MODEL]

# write your code here...
start_time = time.time()
#CNN model
model1 = tf.keras.Sequential()
model1.add(tf.keras.layers.Conv2D(32,(3,3),activation =␣
 ↪'linear',input_shape=(32,32,3),padding='same'))
model1.add(tf.keras.layers.BatchNormalization())
model1.add(tf.keras.layers.MaxPooling2D((2,2),padding='same'))
model1.add(tf.keras.layers.Dropout(0.2))
model1.add(tf.keras.layers.Conv2D(128,(3,3),activation =␣
 ↪'linear',padding='same'))
model1.add(tf.keras.layers.MaxPooling2D((2,2)))
```

```
model1.add(tf.keras.layers.Dropout(0.2))
model1.add(tf.keras.layers.Dense(200,activation='linear'))
model1.add(tf.keras.layers.MaxPooling2D((2,2)))
model1.add(tf.keras.layers.Dropout(0.2))
model1.add(tf.keras.layers.BatchNormalization())
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(10,activation='softmax'))


#compile model
model1.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.
 ↪optimizers.Adam(0.0005),metrics=['accuracy'])

#model fitting
timer = TimeHistory()
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
model1.fit(imgs,noisy_labels,epochs = 10, validation_split =0.
 ↪2,callbacks=[timer,early_stop])
print('-----Model1 run time: %s seconds-----'%(time.time()-start_time))
```

```
Epoch 1/10
1250/1250 [==============================] - 48s 38ms/step - loss: 2.5923 -
accuracy: 0.1313 - val_loss: 2.3459 - val_accuracy: 0.1572
Epoch 2/10
1250/1250 [==============================] - 49s 39ms/step - loss: 2.4362 -
accuracy: 0.1517 - val_loss: 2.3328 - val_accuracy: 0.1663
Epoch 3/10
1250/1250 [==============================] - 48s 38ms/step - loss: 2.3710 -
accuracy: 0.1691 - val_loss: 2.3884 - val_accuracy: 0.1598
Epoch 4/10
1250/1250 [==============================] - 48s 38ms/step - loss: 2.3254 -
accuracy: 0.1821 - val_loss: 2.3307 - val_accuracy: 0.1873
Epoch 5/10
1250/1250 [==============================] - 51s 41ms/step - loss: 2.2857 -
accuracy: 0.1935 - val_loss: 2.2857 - val_accuracy: 0.1938
Epoch 6/10
1250/1250 [==============================] - 50s 40ms/step - loss: 2.2571 -
accuracy: 0.2071 - val_loss: 2.2822 - val_accuracy: 0.1935
Epoch 7/10
1250/1250 [==============================] - 49s 39ms/step - loss: 2.2356 -
accuracy: 0.2166 - val_loss: 2.2563 - val_accuracy: 0.2108
Epoch 8/10
1250/1250 [==============================] - 47s 38ms/step - loss: 2.2118 -
accuracy: 0.2267 - val_loss: 2.2409 - val_accuracy: 0.2272
Epoch 9/10
1250/1250 [==============================] - 48s 38ms/step - loss: 2.1961 -
accuracy: 0.2367 - val_loss: 2.2331 - val_accuracy: 0.2260
```

```
Epoch 10/10
1250/1250 [==============================] - 47s 38ms/step - loss: 2.1817 -
accuracy: 0.2448 - val_loss: 2.2690 - val_accuracy: 0.2065
-----Model1 run time: 485.97716307640076 seconds-----
```

[54]:
```
#save trained model
model1.save('model1')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 2 of 2). These functions will
not be directly callable after loading.

INFO:tensorflow:Assets written to: model1/assets

INFO:tensorflow:Assets written to: model1/assets
```

[70]:
```
def model_I(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and␣
 ↪returns a label prediction
    '''
    # write your code here...
    result = model1.predict(np.array([image/255,]))
    result = np.argmax(result,axis = 1)
    return result
```

0.2.3 2.3. Model II

[56]:
```
#create training and testing set for model
imgs_c = imgs[:10000].reshape(-1,32,32,3)
noisy_lab1 = (np.array(noisy_labels[0:10000])).reshape(-1,1)
x_tr, x_te, y_tr, y_te = train_test_split(imgs_c,clean_labels,test_size=0.
 ↪2,random_state=42)
x_tr = tf.cast(x_tr,dtype='float32')/255.0
x_te = tf.cast(x_te,dtype='float32')/255.0
```

[60]:
```
#Label Correction model
label_corr = tf.keras.Sequential()
label_corr.add(tf.keras.layers.Conv2D(32,(3,3),activation =␣
 ↪'relu',input_shape=(32,32,3),padding='same'))
label_corr.add(tf.keras.layers.MaxPooling2D((2,2)))
label_corr.add(tf.keras.layers.BatchNormalization())
label_corr.add(tf.keras.layers.Dropout(0.2))
label_corr.add(tf.keras.layers.Conv2D(64,(3,3),activation =␣
 ↪'relu',padding='same'))
label_corr.add(tf.keras.layers.MaxPooling2D((2,2)))
```

```
label_corr.add(tf.keras.layers.Dropout(0.2))
label_corr.add(tf.keras.layers.Conv2D(64,(3,3),activation =␣
 ↪'relu',padding='same'))
label_corr.add(tf.keras.layers.Dropout(0.2))
label_corr.add(tf.keras.layers.Flatten())
label_corr.add(tf.keras.layers.Dense(128,activation='relu'))
label_corr.add(tf.keras.layers.Dropout(0.2))
label_corr.add(tf.keras.layers.Dense(10,activation='softmax'))

#compile model
timer = TimeHistory()
label_corr.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), optimizer=tf.keras.
 ↪optimizers.Adam(1e-4),metrics=['accuracy'])

#model fitting
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
label_corr.fit(x=x_tr, y=y_tr, epochs=40,batch_size=64,validation_split =0.
 ↪2,callbacks=[early_stop,timer])
time_label = sum(timer.times)
```

```
Epoch 1/40

/opt/anaconda3/lib/python3.8/site-packages/keras/backend.py:5582: UserWarning:
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `output`
argument was produced by a Softmax activation and thus does not represent
logits. Was this intended?
  output, from_logits = _get_logits(

100/100 [==============================] - 7s 59ms/step - loss: 2.1697 -
accuracy: 0.2037 - val_loss: 2.2251 - val_accuracy: 0.2256
Epoch 2/40
100/100 [==============================] - 6s 57ms/step - loss: 1.8844 -
accuracy: 0.3097 - val_loss: 2.1025 - val_accuracy: 0.3587
Epoch 3/40
100/100 [==============================] - 6s 60ms/step - loss: 1.7239 -
accuracy: 0.3742 - val_loss: 1.9474 - val_accuracy: 0.4000
Epoch 4/40
100/100 [==============================] - 6s 59ms/step - loss: 1.6369 -
accuracy: 0.4103 - val_loss: 1.7683 - val_accuracy: 0.4187
Epoch 5/40
100/100 [==============================] - 6s 59ms/step - loss: 1.5598 -
accuracy: 0.4347 - val_loss: 1.6246 - val_accuracy: 0.4300
Epoch 6/40
100/100 [==============================] - 6s 61ms/step - loss: 1.4968 -
accuracy: 0.4569 - val_loss: 1.4945 - val_accuracy: 0.4694
Epoch 7/40
100/100 [==============================] - 6s 61ms/step - loss: 1.4357 -
```

```
accuracy: 0.4834 - val_loss: 1.4203 - val_accuracy: 0.4925
Epoch 8/40
100/100 [==============================] - 6s 61ms/step - loss: 1.3904 -
accuracy: 0.5055 - val_loss: 1.3974 - val_accuracy: 0.4881
Epoch 9/40
100/100 [==============================] - 6s 61ms/step - loss: 1.3330 -
accuracy: 0.5167 - val_loss: 1.3502 - val_accuracy: 0.5088
Epoch 10/40
100/100 [==============================] - 6s 61ms/step - loss: 1.2950 -
accuracy: 0.5267 - val_loss: 1.3461 - val_accuracy: 0.5094
Epoch 11/40
100/100 [==============================] - 6s 62ms/step - loss: 1.2790 -
accuracy: 0.5391 - val_loss: 1.3450 - val_accuracy: 0.5175
Epoch 12/40
100/100 [==============================] - 6s 61ms/step - loss: 1.2458 -
accuracy: 0.5556 - val_loss: 1.2959 - val_accuracy: 0.5312
Epoch 13/40
100/100 [==============================] - 6s 62ms/step - loss: 1.2047 -
accuracy: 0.5720 - val_loss: 1.2878 - val_accuracy: 0.5350
Epoch 14/40
100/100 [==============================] - 6s 62ms/step - loss: 1.1785 -
accuracy: 0.5806 - val_loss: 1.2567 - val_accuracy: 0.5425
Epoch 15/40
100/100 [==============================] - 6s 61ms/step - loss: 1.1398 -
accuracy: 0.5970 - val_loss: 1.2580 - val_accuracy: 0.5550
Epoch 16/40
100/100 [==============================] - 6s 62ms/step - loss: 1.1174 -
accuracy: 0.5989 - val_loss: 1.2375 - val_accuracy: 0.5475
Epoch 17/40
100/100 [==============================] - 6s 61ms/step - loss: 1.0794 -
accuracy: 0.6152 - val_loss: 1.2504 - val_accuracy: 0.5525
Epoch 18/40
100/100 [==============================] - 6s 63ms/step - loss: 1.0707 -
accuracy: 0.6189 - val_loss: 1.2537 - val_accuracy: 0.5544
Epoch 19/40
100/100 [==============================] - 7s 66ms/step - loss: 1.0407 -
accuracy: 0.6373 - val_loss: 1.2159 - val_accuracy: 0.5713
Epoch 20/40
100/100 [==============================] - 7s 66ms/step - loss: 1.0189 -
accuracy: 0.6366 - val_loss: 1.2125 - val_accuracy: 0.5663
Epoch 21/40
100/100 [==============================] - 7s 65ms/step - loss: 0.9815 -
accuracy: 0.6470 - val_loss: 1.2023 - val_accuracy: 0.5675
Epoch 22/40
100/100 [==============================] - 7s 66ms/step - loss: 0.9686 -
accuracy: 0.6558 - val_loss: 1.1945 - val_accuracy: 0.5831
Epoch 23/40
100/100 [==============================] - 6s 62ms/step - loss: 0.9540 -
```

```
accuracy: 0.6580 - val_loss: 1.1845 - val_accuracy: 0.5975
Epoch 24/40
100/100 [==============================] - 6s 61ms/step - loss: 0.9183 -
accuracy: 0.6727 - val_loss: 1.1706 - val_accuracy: 0.5919
Epoch 25/40
100/100 [==============================] - 6s 61ms/step - loss: 0.8984 -
accuracy: 0.6787 - val_loss: 1.1614 - val_accuracy: 0.5875
Epoch 26/40
100/100 [==============================] - 6s 62ms/step - loss: 0.8866 -
accuracy: 0.6894 - val_loss: 1.1935 - val_accuracy: 0.5750
Epoch 27/40
100/100 [==============================] - 7s 66ms/step - loss: 0.8462 -
accuracy: 0.6975 - val_loss: 1.1596 - val_accuracy: 0.5956
Epoch 28/40
100/100 [==============================] - 6s 61ms/step - loss: 0.8386 -
accuracy: 0.6966 - val_loss: 1.1571 - val_accuracy: 0.5950
Epoch 29/40
100/100 [==============================] - 6s 61ms/step - loss: 0.8137 -
accuracy: 0.7111 - val_loss: 1.1740 - val_accuracy: 0.6025
Epoch 30/40
100/100 [==============================] - 6s 62ms/step - loss: 0.8000 -
accuracy: 0.7117 - val_loss: 1.1813 - val_accuracy: 0.5981
Epoch 31/40
100/100 [==============================] - 6s 61ms/step - loss: 0.7767 -
accuracy: 0.7219 - val_loss: 1.1764 - val_accuracy: 0.5888
```

[61]:
```
y_pred = label_corr.predict(x_te)
y_pred = np.argmax(y_pred,axis=-1)
accuracy_score(y_pred,y_te)
#got 57% accuracy for label correction model
```

```
63/63 [==============================] - 1s 8ms/step
```

[61]:
```
0.6125
```

[62]:
```
#get cleaned labels
imgs_n = imgs[10000:].reshape(-1,32,32,3)
imgs_n = tf.cast(imgs_n,dtype='float32')/255.0
cleaned_labels = np.argmax(label_corr.predict(imgs_n),axis=1)
labels = np.append(clean_labels,cleaned_labels)
#create training and testing set for modelII
x_tr, x_te, y_tr, y_te = train_test_split(imgs,labels,test_size=0.
 ↪1,shuffle=False)
x_tr = tf.cast(x_tr,dtype='float32')/255.0
x_te = tf.cast(x_te,dtype='float32')/255.0
```

```
1250/1250 [==============================] - 10s 8ms/step
```

[63]:
```
start = time.time()
#the model 2 here is exactly same as model 1
model2 = tf.keras.Sequential()
model2.add(tf.keras.layers.Conv2D(32,(3,3),activation =␣
 ↪'linear',input_shape=(32,32,3),padding='same'))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.MaxPooling2D((2,2),padding='same'))
model2.add(tf.keras.layers.Dropout(0.2))
model2.add(tf.keras.layers.Conv2D(128,(3,3),activation =␣
 ↪'linear',padding='same'))
model2.add(tf.keras.layers.MaxPooling2D((2,2)))
model2.add(tf.keras.layers.Dropout(0.2))
model2.add(tf.keras.layers.Dense(200,activation='linear'))
model2.add(tf.keras.layers.MaxPooling2D((2,2)))
model2.add(tf.keras.layers.Dropout(0.2))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.Dense(10,activation='softmax'))



#compile model
model2.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.
 ↪optimizers.Adam(0.0005),metrics=['accuracy'])

#model fitting
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
model2.fit(x=x_tr, y=y_tr, epochs=10, validation_split =0.
 ↪2,callbacks=[early_stop,timer])

print(f'Model 2 took {sum(timer.times)+time_label} seconds to train, which is␣
 ↪about {(sum(timer.times)+time_label)/60} minutes.')
```

```
Epoch 1/10
1125/1125 [==============================] - 41s 35ms/step - loss: 1.4611 -
accuracy: 0.5014 - val_loss: 1.1202 - val_accuracy: 0.5953
Epoch 2/10
1125/1125 [==============================] - 41s 36ms/step - loss: 1.1185 -
accuracy: 0.6107 - val_loss: 0.9201 - val_accuracy: 0.6626
Epoch 3/10
1125/1125 [==============================] - 41s 37ms/step - loss: 1.0180 -
accuracy: 0.6426 - val_loss: 0.9544 - val_accuracy: 0.6736
Epoch 4/10
1125/1125 [==============================] - 41s 36ms/step - loss: 0.9648 -
accuracy: 0.6597 - val_loss: 0.9282 - val_accuracy: 0.6717
Epoch 5/10
1125/1125 [==============================] - 41s 36ms/step - loss: 0.9295 -
```

```
accuracy: 0.6702 - val_loss: 0.8260 - val_accuracy: 0.7032
Epoch 6/10
1125/1125 [==============================] - 41s 36ms/step - loss: 0.9003 -
accuracy: 0.6815 - val_loss: 0.7825 - val_accuracy: 0.7082
Epoch 7/10
1125/1125 [==============================] - 41s 36ms/step - loss: 0.8754 -
accuracy: 0.6927 - val_loss: 0.7933 - val_accuracy: 0.7138
Epoch 8/10
1125/1125 [==============================] - 41s 37ms/step - loss: 0.8475 -
accuracy: 0.6989 - val_loss: 0.8396 - val_accuracy: 0.6934
Epoch 9/10
1125/1125 [==============================] - 41s 36ms/step - loss: 0.8346 -
accuracy: 0.7036 - val_loss: 0.7417 - val_accuracy: 0.7250
Epoch 10/10
1125/1125 [==============================] - 42s 37ms/step - loss: 0.8214 -
accuracy: 0.7067 - val_loss: 0.8213 - val_accuracy: 0.7054
Model 2 took 603.0154283046722 seconds to train, which is about
10.050257138411204 minutes.
```

[64]:
```
#save model
model2.save('model2')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 2 of 2). These functions will
not be directly callable after loading.

INFO:tensorflow:Assets written to: model2/assets

INFO:tensorflow:Assets written to: model2/assets
```

[65]:
```
m2 = tf.keras.models.load_model('model2')
y_pred = m2.predict(x_te)
y_pred = np.argmax(y_pred,axis=-1)
accuracy_score(y_pred,y_te)
```

```
157/157 [==============================] - 2s 10ms/step
```

[65]: 0.6998

[71]:
```
# [ADD WEAKLY SUPERVISED LEARNING FEATURE TO MODEL I]

# write your code here...

def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and␣
 ↪returns a label prediction
```

```
    '''
    # write your code here...
    image = tf.reshape(image,((1,)+ image.shape))
    pred = model2.predict(image)
    pred = np.argmax(pred,axis=-1)
    return pred
```

0.3 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.

[67]:
```
# [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))
```

[20]:
```
# [DO NOT MODIFY THIS CELL]
# This is the code for evaluating the prediction performance on a testset
# You will get an error if running this cell, as you do not have the testset
# Nonetheless, you can create your own validation set to run the evlauation
n_test = 10000
test_labels = np.genfromtxt('../data/test_labels.csv', delimiter=',',␣
 ↪dtype="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)
```

```
      ␣
↪---------------------------------------------------------------------------

      OSError                                   Traceback (most recent call␣
↪last)

      <ipython-input-20-82821a72951b> in <module>
        4 # Nonetheless, you can create your own validation set to run the␣
↪evlauation
        5 n_test = 10000
```

```
   ----> 6 test_labels = np.genfromtxt('../data/test_labels.csv',␣
↪delimiter=',', dtype="int8")
         7 test_imgs = np.empty((n_test,32,32,3))
         8 for i in range(n_test):


       ~/anaconda3/lib/python3.7/site-packages/numpy/lib/npyio.py in␣
↪genfromtxt(fname, dtype, comments, delimiter, skip_header, skip_footer,␣
↪converters, missing_values, filling_values, usecols, names, excludelist,␣
↪deletechars, replace_space, autostrip, case_sensitive, defaultfmt, unpack,␣
↪usemask, loose, invalid_raise, max_rows, encoding)
      1747             fname = os_fspath(fname)
      1748         if isinstance(fname, str):
   -> 1749             fid = np.lib._datasource.open(fname, 'rt',␣
↪encoding=encoding)
      1750             fid_ctx = contextlib.closing(fid)
      1751         else:


       ~/anaconda3/lib/python3.7/site-packages/numpy/lib/_datasource.py in␣
↪open(path, mode, destpath, encoding, newline)
       193
       194     ds = DataSource(destpath)
   --> 195     return ds.open(path, mode, encoding=encoding, newline=newline)
       196
       197


       ~/anaconda3/lib/python3.7/site-packages/numpy/lib/_datasource.py in␣
↪open(self, path, mode, encoding, newline)
       533                                     encoding=encoding,␣
↪newline=newline)
       534         else:
   --> 535             raise IOError("%s not found." % path)
       536
       537


       OSError: ../data/test_labels.csv not found.
```

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.

[68]:
```
test_imgs = imgs[0:10000]
test_labels = clean_labels
```

[ ]:
```
start=time.time()
evaluation(model_I, test_labels, test_imgs)
print('-----Evaluation for model I run time: %s seconds-----'%(time.
 ↪time()-start))
```

[ ]:
```
start=time.time()
evaluation(model_II, test_labels, test_imgs)
print('-----Evaluation for model II run time: %s seconds-----'%(time.
 ↪time()-start))
```

[77]:
```
#generate classification report
##1. baseline
feature_mtx_pred = np.empty((10000,3*(len(bins)-1)))
for i in range(10000):

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

    # Concatenate three features
    feature_mtx_pred[i,] = np.concatenate((feature1, feature2, feature3),␣
 ↪axis=None)
    i += 1

pred_base = clf.predict(feature_mtx_pred)

report_base = classification_report(test_labels,pred_base)
print('---------baseline_model classification report----------')
print(report_base)

#model 1
model1_pred = tf.keras.models.load_model('model1')
pred_1 = model1_pred.predict(test_imgs)
pred_1 = np.argmax(pred_1,axis=-1)

report_1 = classification_report(test_labels,pred_1)
print('---------model1 classification report----------')
```

```
print(report_1)

#model 2
model2_pred = tf.keras.models.load_model('model2')
pred_2 = model2_pred.predict(test_imgs)
pred_2 = np.argmax(pred_2,axis=-1)
report_2 = classification_report(test_labels,pred_2)
print('---------model 2 classification report----------')
print(report_2)
```

```
---------baseline_model classification report----------
              precision    recall  f1-score   support

           0       0.32      0.43      0.37      1005
           1       0.18      0.29      0.22       974
           2       0.22      0.04      0.07      1032
           3       0.19      0.12      0.14      1016
           4       0.24      0.48      0.32       999
           5       0.22      0.13      0.16       937
           6       0.26      0.35      0.30      1030
           7       0.29      0.04      0.07      1001
           8       0.28      0.43      0.34      1025
           9       0.19      0.11      0.14       981

    accuracy                           0.24     10000
   macro avg       0.24      0.24      0.21     10000
weighted avg       0.24      0.24      0.21     10000

313/313 [==============================] - 4s 11ms/step
---------model1 classification report----------
              precision    recall  f1-score   support

           0       0.59      0.47      0.52      1005
           1       0.75      0.53      0.62       974
           2       0.27      0.61      0.38      1032
           3       0.47      0.18      0.26      1016
           4       0.48      0.40      0.43       999
           5       0.41      0.39      0.40       937
           6       0.33      0.86      0.48      1030
           7       0.84      0.23      0.36      1001
           8       0.75      0.45      0.56      1025
           9       0.72      0.38      0.50       981

    accuracy                           0.45     10000
   macro avg       0.56      0.45      0.45     10000
weighted avg       0.56      0.45      0.45     10000
```

```
313/313 [==============================] - 4s 11ms/step
---------model 2 classification report----------
              precision    recall  f1-score   support

           0       0.45      0.71      0.55      1005
           1       0.47      0.93      0.62       974
           2       0.75      0.21      0.33      1032
           3       0.79      0.13      0.22      1016
           4       0.93      0.04      0.07       999
           5       0.50      0.63      0.56       937
           6       0.87      0.42      0.57      1030
           7       0.51      0.78      0.62      1001
           8       0.56      0.83      0.67      1025
           9       0.55      0.73      0.63       981

    accuracy                           0.54     10000
   macro avg       0.64      0.54      0.48     10000
weighted avg       0.64      0.54      0.48     10000
```

[78]:
```
#generate prediction for models and export csv file
output_df = pd.DataFrame()

#baseline
feature_mtx_pred = np.empty((10000,3*(len(bins)-1)))
for i in range(10000):

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

    # Concatenate three features
    feature_mtx_pred[i,] = np.concatenate((feature1, feature2, feature3),␣
 ↪axis=None)
    i += 1

pred_base = clf.predict(feature_mtx_pred)
pred_base = clf.predict(feature_mtx_pred)
#model1
model1_pred = tf.keras.models.load_model('model1')
pred_1 = model1_pred.predict(test_imgs)
pred_1 = np.argmax(pred_1,axis=-1)
#model 2
model2_pred = tf.keras.models.load_model('model2')
pred_2 = model2_pred.predict(test_imgs)
```

```
pred_2 = np.argmax(pred_2,axis=-1)


output_df['baseline_pred'] = pred_base
output_df['modelI_pred'] = pred_1
output_df['modelII_pred'] = pred_2
output_df.to_csv('label_prediction.csv')
```

```
313/313 [==============================] - 4s 12ms/step
313/313 [==============================] - 4s 12ms/step
```