In [2]:
```
# 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
from sklearn.model_selection import train_test_split
import tensorflow as tf
import time
from tensorflow import keras
from sklearn.metrics import accuracy_score, confusion_matrix,precision_s
import pandas as pd
```

1. Load the datasets

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

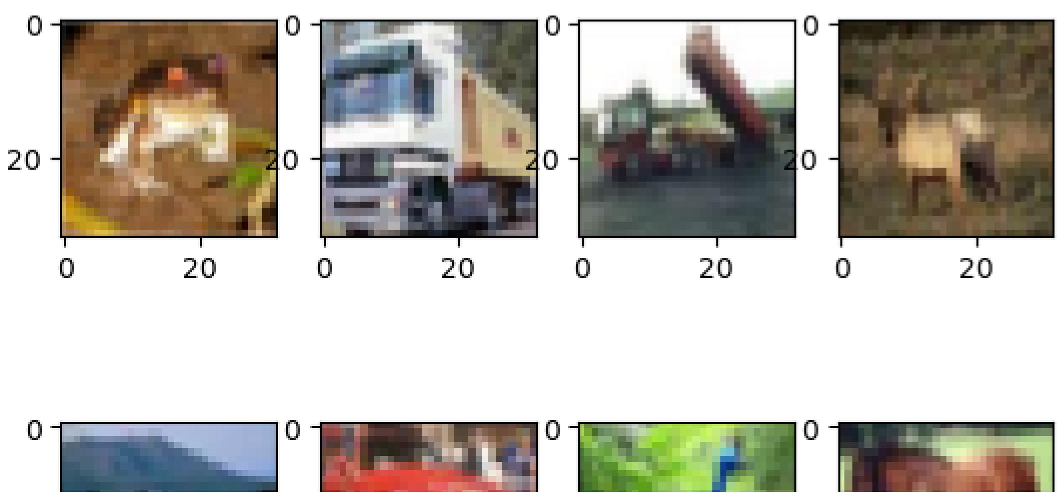
- noisy labels for all images provided in `../data/noisy_label.csv` ;
- clean labels for the first 10000 images provided in `../data/clean_labels.csv` .

In [3]:
```
# [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=',', 
noisy_labels = np.genfromtxt('../data/noisy_labels.csv', delimiter=',', 
```

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]:
```
# [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
```



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
    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
    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 $32 \times 32 \times 3$ 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
    '''
    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).
    return clf.predict(feature)
```

In [10]:
```
# split the data
imgs = imgs.astype('float32') /255.0
# Clean data
x_clean = imgs[0:10000]
y_clean = clean_labels
# Noisy data
x_noisy = imgs[10000:]
y_noisy = noisy_labels[10000:]
# Split the noisy data into training and validation set
x_noisy_train, x_noisy_test, y_noisy_train, y_noisy_test = train_test_sp
# Split the clean data into training and validation set
x_clean_train, x_clean_test, y_clean_train, y_clean_test = train_test_sp
```

In [8]:
```
# Constructed a timer to record the training time for each model in the p
class TimeHistory(keras.callbacks.Callback):
    def begin_train(self, logs={}):
        self.times = []

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

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

2.2. Model I

In [29]:
```
#Record the start time for model 1
start_time = time.time()
#construct a CNN model for model 1
model1 = tf.keras.Sequential()
model1.add(tf.keras.layers.Conv2D(32,(3,3),activation = 'relu',input_sha
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 = 'relu',padding=
model1.add(tf.keras.layers.MaxPooling2D((2,2)))
model1.add(tf.keras.layers.Dropout(0.2))
model1.add(tf.keras.layers.Dense(200,activation='relu'))
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 the CNN model
model1.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras
timer = TimeHistory()
#Set a early stopping callback, if the model does not improve after 3 ep
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Ad
am` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer in
stead, located at `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimize
rs on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `t
f.keras.optimizers.legacy.Adam`.
```

In [30]:
```
#Fit model 1 to the noisy data, and validate the model using the data
model1.fit(imgs,noisy_labels,epochs = 10, validation_split = 0.2,callbac
print('-----Model1 run time: %s seconds-----'%(time.time()-start_time))
```

```
Epoch 1/10
1250/1250 [==============================] - 26s 20ms/step - loss: 2.60
90 - accuracy: 0.1299 - val_loss: 2.3118 - val_accuracy: 0.1614
Epoch 2/10
1250/1250 [==============================] - 24s 20ms/step - loss: 2.40
05 - accuracy: 0.1538 - val_loss: 2.2997 - val_accuracy: 0.1631
Epoch 3/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.32
36 - accuracy: 0.1743 - val_loss: 2.3011 - val_accuracy: 0.1785
Epoch 4/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.28
59 - accuracy: 0.1901 - val_loss: 2.2603 - val_accuracy: 0.1998
Epoch 5/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.25
80 - accuracy: 0.2049 - val_loss: 2.3094 - val_accuracy: 0.1772
Epoch 6/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.23
76 - accuracy: 0.2112 - val_loss: 2.2878 - val_accuracy: 0.1819
Epoch 7/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.22
71 - accuracy: 0.2197 - val_loss: 2.2239 - val_accuracy: 0.2242
Epoch 8/10
1250/1250 [==============================] - 25s 20ms/step - loss: 2.20
78 - accuracy: 0.2299 - val_loss: 2.2224 - val_accuracy: 0.2213
Epoch 9/10
1250/1250 [==============================] - 26s 20ms/step - loss: 2.19
66 - accuracy: 0.2391 - val_loss: 2.2671 - val_accuracy: 0.2029
Epoch 10/10
1250/1250 [==============================] - 29s 24ms/step - loss: 2.18
98 - accuracy: 0.2415 - val_loss: 2.2144 - val_accuracy: 0.2297
-----Model1 run time: 260.7050130367279 seconds-----
```

In [31]:
```
#Save the model for future use
model1.save('model1')
```

```
2024-03-20 13:08:21.738761: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,16,16,32]
         [[{{node inputs}}]]
2024-03-20 13:08:21.744733: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 13:08:21.749095: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 13:08:21.751188: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,4,4,200]
         [[{{node inputs}}]]
2024-03-20 13:08:22.133012: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,16,16,32]
         [[{{node inputs}}]]
2024-03-20 13:08:22.148442: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 13:08:22.164627: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 13:08:22.178024: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,4,4,200]
         [[{{node inputs}}]]
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
```

In [9]:
```
model1 = tf.keras.models.load_model('model1')
```

In [8]:
```
# load model 1 and evaluate the accuracy score
y_pred = model1.predict(x_clean_test)
y_pred = np.argmax(y_pred,axis=-1)
accuracy_score(y_pred,y_clean_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
Cell In[8], line 3
      1 # load model 1 and evaluate the accuracy score
      2 model1 = tf.keras.models.load_model('model1')
----> 3 y_pred = model1.predict(x_clean_test)
      4 y_pred = np.argmax(y_pred,axis=-1)
      5 accuracy_score(y_pred,y_clean_test)

NameError: name 'x_clean_test' is not defined
```

In [10]:
```
# Finalize model 1
def model_I(image):
    '''
    This function should takes in the image of dimension 32*32*3 as inpu
    '''
    # write your code here...
    image = tf.reshape(image,((1,)+ image.shape))
    pred = model1.predict(image)
    pred = np.argmax(pred,axis=-1)
    return pred
```

2.3. Model II

In [33]:

```
# [ADD WEAKLY SUPERVISED LEARNING FEATURE TO MODEL I]
# write your code here...

#Label Correction model on the clean data
label_corr = tf.keras.Sequential()
label_corr.add(tf.keras.layers.Conv2D(32,(3,3),activation = 'relu',input_
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',paddi
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',paddi
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(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1

#model fitting
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
label_corr.fit(x_clean_train, y_clean_train, epochs=40,batch_size=64,val
time_label = sum(timer.times)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.RM
Sprop` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer
instead, located at `tf.keras.optimizers.legacy.RMSprop`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimize
rs on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `t
f.keras.optimizers.legacy.RMSprop`.
```

```
Epoch 1/40
125/125 [==============================] - 4s 30ms/step - loss: 1.9252
- accuracy: 0.3119 - val_loss: 2.1101 - val_accuracy: 0.3470
Epoch 2/40
125/125 [==============================] - 3s 28ms/step - loss: 1.5565
- accuracy: 0.4389 - val_loss: 1.9038 - val_accuracy: 0.4265
Epoch 3/40
125/125 [==============================] - 4s 28ms/step - loss: 1.4066
- accuracy: 0.4915 - val_loss: 1.5864 - val_accuracy: 0.5020
Epoch 4/40
125/125 [==============================] - 4s 28ms/step - loss: 1.2698
- accuracy: 0.5481 - val_loss: 1.3708 - val_accuracy: 0.5305
Epoch 5/40
125/125 [==============================] - 4s 30ms/step - loss: 1.1685
- accuracy: 0.5871 - val_loss: 1.2654 - val_accuracy: 0.5565
Epoch 6/40
125/125 [==============================] - 3s 28ms/step - loss: 1.0764
- accuracy: 0.6156 - val_loss: 1.2251 - val_accuracy: 0.5765
Epoch 7/40
125/125 [==============================] - 4s 28ms/step - loss: 0.9874
- accuracy: 0.6478 - val_loss: 1.1315 - val_accuracy: 0.6035
Epoch 8/40
125/125 [==============================] - 4s 28ms/step - loss: 0.9122
- accuracy: 0.6787 - val_loss: 1.1687 - val_accuracy: 0.6055
Epoch 9/40
125/125 [==============================] - 4s 28ms/step - loss: 0.8383
- accuracy: 0.7049 - val_loss: 1.5535 - val_accuracy: 0.5115
Epoch 10/40
125/125 [==============================] - 4s 29ms/step - loss: 0.7653
- accuracy: 0.7287 - val_loss: 1.2638 - val_accuracy: 0.5895
```

In [34]:
```
# Evaluate the label correction model
y_pred = label_corr.predict(x_clean_test)
y_pred = np.argmax(y_pred,axis=-1)
accuracy_score(y_pred,y_clean_test)
```

```
63/63 [==============================] - 1s 9ms/step
```

Out[34]:
```
0.5895
```

In [35]:
```
# Use the label correction model to correct the noisy data labels before
imgs_noise = imgs[10000:]
cleaned_noise_labels = np.argmax(label_corr.predict(imgs_noise),axis=1)
labels = np.append(clean_labels,cleaned_noise_labels)
x_train, x_test, y_train, y_test = train_test_split(imgs,labels,test_siz
```

```
1250/1250 [==============================] - 6s 5ms/step
```

In [44]:
```
#Record the start time for model 2
#CNN
model2 = tf.keras.Sequential()
model2.add(tf.keras.layers.Conv2D(32,(3,3),activation = 'relu',input_sha
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 = 'relu',padding=
model2.add(tf.keras.layers.MaxPooling2D((2,2)))
model2.add(tf.keras.layers.Dropout(0.2))
model2.add(tf.keras.layers.Dense(200,activation='relu'))
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

#model fitting
early_stop = tf.keras.callbacks.EarlyStopping(patience=3)
model2.fit(x_train, y_train, epochs=10, validation_data = (x_test, y_tes
# print out the time it took to train the model
print(f'Model 2 took {sum(timer.times)+time_label} seconds to train, whi
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Ad
am` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer in
stead, located at `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimize
rs on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `t
f.keras.optimizers.legacy.Adam`.
```

```
Epoch 1/10
1407/1407 [==============================] - 30s 21ms/step - loss: 1.42
82 - accuracy: 0.5121 - val_loss: 1.1055 - val_accuracy: 0.6112
Epoch 2/10
1407/1407 [==============================] - 27s 19ms/step - loss: 1.09
06 - accuracy: 0.6181 - val_loss: 0.9522 - val_accuracy: 0.6584
Epoch 3/10
1407/1407 [==============================] - 27s 19ms/step - loss: 0.99
17 - accuracy: 0.6482 - val_loss: 0.9871 - val_accuracy: 0.6414
Epoch 4/10
1407/1407 [==============================] - 28s 20ms/step - loss: 0.94
25 - accuracy: 0.6635 - val_loss: 0.8347 - val_accuracy: 0.7062
Epoch 5/10
1407/1407 [==============================] - 31s 22ms/step - loss: 0.90
39 - accuracy: 0.6755 - val_loss: 0.8409 - val_accuracy: 0.6964
Epoch 6/10
1407/1407 [==============================] - 28s 20ms/step - loss: 0.86
69 - accuracy: 0.6876 - val_loss: 0.8653 - val_accuracy: 0.6834
Epoch 7/10
1407/1407 [==============================] - 29s 20ms/step - loss: 0.84
78 - accuracy: 0.6959 - val_loss: 0.8296 - val_accuracy: 0.6990
Epoch 8/10
1407/1407 [==============================] - 29s 21ms/step - loss: 0.82
52 - accuracy: 0.7024 - val_loss: 0.8202 - val_accuracy: 0.7040
Epoch 9/10
1407/1407 [==============================] - 28s 20ms/step - loss: 0.81
30 - accuracy: 0.7060 - val_loss: 0.8273 - val_accuracy: 0.7054
Epoch 10/10
1407/1407 [==============================] - 29s 20ms/step - loss: 0.79
41 - accuracy: 0.7138 - val_loss: 1.3474 - val_accuracy: 0.5662
Model 2 took 321.8110542297363 seconds to train, which is about 5.36351
7570495605 minutes.
```

In [45]:
```
# Save model 2 for future use
model2.save('model2')
```

```
2024-03-20 15:12:19.002491: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,16,16,32]
         [[{{node inputs}}]]
2024-03-20 15:12:19.008274: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 15:12:19.011520: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 15:12:19.013397: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,4,4,200]
         [[{{node inputs}}]]
2024-03-20 15:12:19.129155: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,16,16,32]
         [[{{node inputs}}]]
2024-03-20 15:12:20.024026: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 15:12:20.033134: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,8,8,128]
         [[{{node inputs}}]]
2024-03-20 15:12:20.046495: I tensorflow/core/common_runtime/executor.c
c:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does
not indicate an error and you can ignore this message): INVALID_ARGUMEN
T: You must feed a value for placeholder tensor 'inputs' with dtype flo
at and shape [?,4,4,200]
         [[{{node inputs}}]]
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
```

In [11]:
```
model2 = tf.keras.models.load_model('model2')
```

In [13]:
```
# load model 2 and evaluate the accuracy score

y_pred = model2.predict(x_clean_test)
y_pred = np.argmax(y_pred,axis=-1)
accuracy_score(y_pred,y_clean_test)
```

```
63/63 [==============================] - 0s 6ms/step
```

Out[13]:
```
0.533
```

In [12]:
```
# Finalize model 2
def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as inpu
    '''
    # write your code here...
    image = tf.reshape(image,((1,)+ image.shape))
    pred = model2.predict(image)
    pred = np.argmax(pred,axis=-1)
    return pred
```

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 [15]:
```
# [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 [16]:
```
# Evaluate the baseline model using clean data and record the run time
x_test = imgs[0:10000]
y_test = clean_labels
```

In [54]:

```python
# Evaluate model 1 using test data and record the run time
start = time.time()
evaluation(model_I, y_test, x_test)
end = time.time()
print("model_I took %s seconds to run" % (end-start))
```

```
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
```

In [15]:

```python
# Evaluate model 2 using test data and record the run time
start = time.time()
evaluation(model_II, y_test, x_test)
end = time.time()
print("model_II took %s seconds to run" % (end-start))
```

```
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
```

In [16]:
```
start = time.time()
evaluation(baseline_model, y_test, x_test)
end = time.time()
print("baseline model took %s seconds to run" % (end-start))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1005
           1       0.10      1.00      0.18       974
           2       0.00      0.00      0.00      1032
           3       0.00      0.00      0.00      1016
           4       0.00      0.00      0.00       999
           5       0.00      0.00      0.00       937
           6       0.00      0.00      0.00      1030
           7       0.00      0.00      0.00      1001
           8       0.00      0.00      0.00      1025
           9       0.00      0.00      0.00       981

    accuracy                           0.10     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.10      0.02     10000

baseline model took 1.6793429851531982 seconds to run

/Users/jiaqiliu/anaconda3/lib/python3.11/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/jiaqiliu/anaconda3/lib/python3.11/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/jiaqiliu/anaconda3/lib/python3.11/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [13]:
```
# Function to get predictions from a given model
n_test = 10000
test_imgs = np.empty((n_test,32,32,3))
for i in range(n_test):
    img_fn = f'../data/test_data/test_images/test{i+1:05d}.png'
    test_imgs[i,:,:,:]=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB

baseline_model_pred = []
modelI_pred = []
modelII_pred = []
# Get the predictions for each model on each image
for img in test_imgs:
    baseline_model_pred.append(baseline_model(img))
    modelI_pred.append(model_I(img))
    modelII_pred.append(model_II(img))

# Create a DataFrame with the prediction
df = pd.DataFrame({
    "Index": np.arange(len(test_imgs)),
    "Baseline": baseline_model_pred,
    "Model I": modelI_pred,
    "Model II": modelII_pred
})

# Save the DataFrame to a CSV file
df.to_csv('label_prediction.csv', index=False)
```

```
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 47ms/step

2024-03-20 18:40:28.245213: W tensorflow/tsl/platform/profile_utils/cpu
_utils.cc:128] Failed to get CPU frequency: 0 Hz

1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 11ms/step
```

In [13]:
```
# [DO NOT MODIFY THIS CELL]
# This is the code for evaluating the prediction performance on a testse
# You will get an error if running this cell, as you do not have the tes
# Nonetheless, you can create your own validation set to run the evlauat
n_test = 10000
test_labels = np.genfromtxt('../data/test_labels.csv', delimiter=',', dty
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)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[13], line 6
      1 # [DO NOT MODIFY THIS CELL]
      2 # This is the code for evaluating the prediction performance on a testset
      3 # You will get an error if running this cell, as you do not have the testset
      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):

File ~/anaconda3/lib/python3.11/site-packages/numpy/lib/npyio.py:1977, 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, ndmin, like)
   1971 with fid_ctx:
   1972     split_line = LineSplitter(delimiter=delimiter, comments=comments,
   1973                               autostrip=autostrip, encoding=encoding)
   1974     validate_names = NameValidator(excludelist=excludelist,
   1975                                    deletechars=deletechars,
   1976                                    case_sensitive=case_sensitive,
-> 1977                                    replace_space=replace_space)
   1979     # Skip the first `skip_header` rows
   1980     try:

File ~/anaconda3/lib/python3.11/site-packages/numpy/lib/_datasource.py:193, in open(path, mode, destpath, encoding, newline)
    156 """
    157 Open `path` with `mode` and return the file object.
    158
   (...)
    189
    190 """
    192 ds = DataSource(destpath)
--> 193 return ds.open(path, mode, encoding=encoding, newline=newline)

File ~/anaconda3/lib/python3.11/site-packages/numpy/lib/_datasource.py:533, in DataSource.open(self, path, mode, encoding, newline)
    530     return _file_openers[ext](found, mode=mode,
    531                               encoding=encoding, newline=newline)
    532 else:
--> 533     raise FileNotFoundError(f"{path} not found.")
```

```
FileNotFoundError: ../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.

In [ ]: