In [1]:

```
# 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
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.utils.vis_utils import plot_model
from tensorflow.keras.utils import to_categorical
from keras.models import Model
from keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50
from sklearn.model_selection import train_test_split
import time
```

In [2]:

```
from numpy.random import seed
seed(1)
tf.random.set_seed(2)
```

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 [6]:
```
# [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
noisy_labels = np.genfromtxt('../data/noisy_labels.csv', delimiter=',', dtype
```

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 [7]:

```python
# [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
```

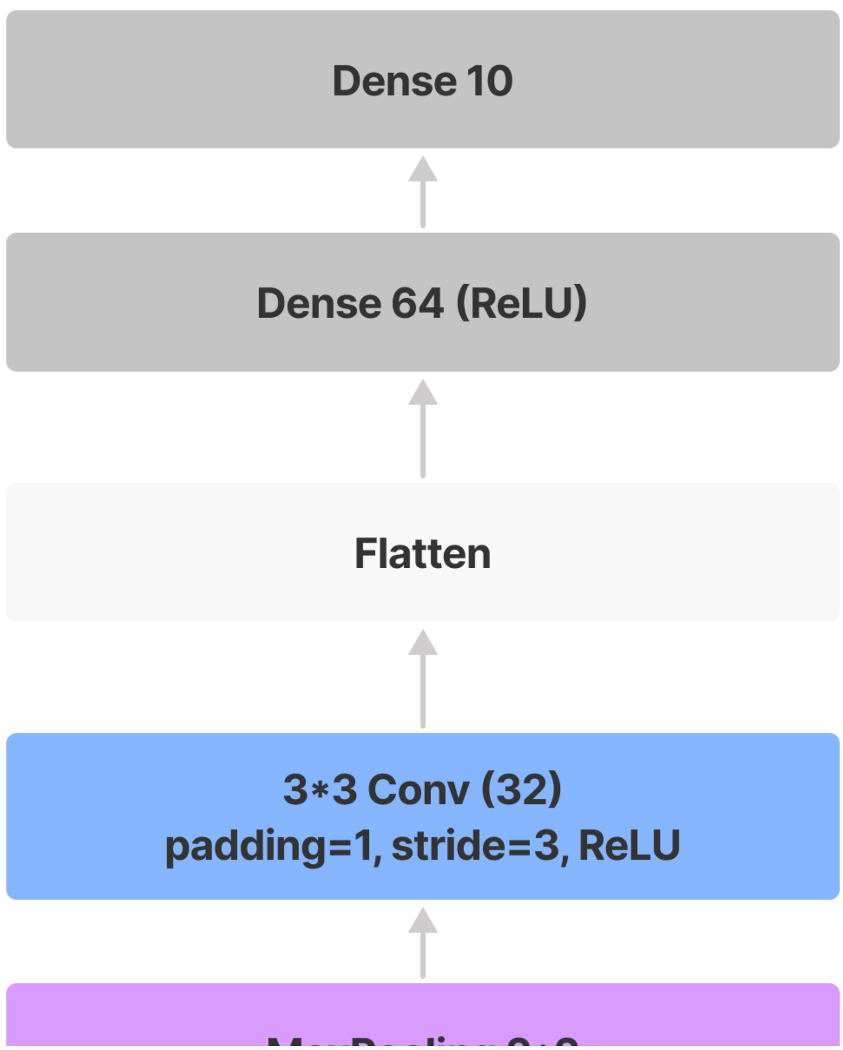
2. The predictive model

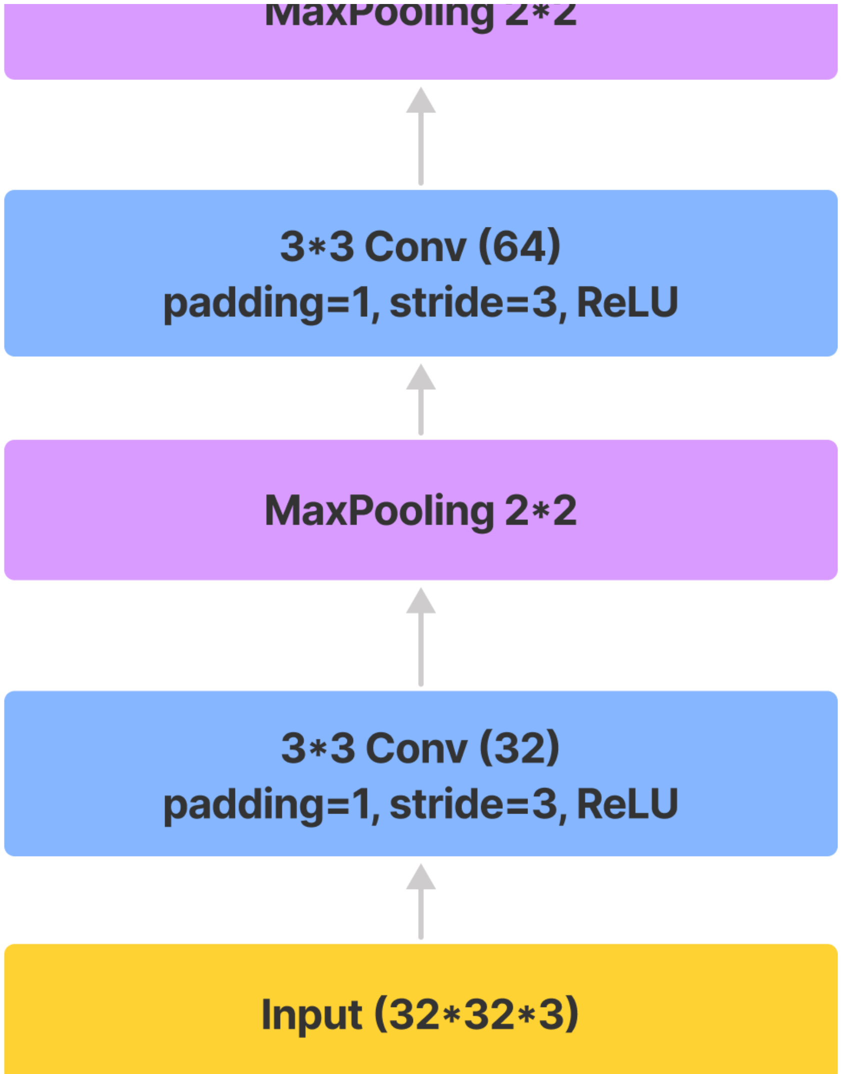
We consider a baseline model directly on the noisy dataset without any label corrections. RGB histogram features are extracted to fit a logistic regression model.

We will first discuss alternate models we tried and eventually discuss the chosen model architecture.

2.0 Models Tried

2.0.1 Convolutional Neural Network





CNN Model 1

In [51]:

```
# Import required packages
import numpy as np
import cv2
import matplotlib.pyplot as plt
import tensorflow
import tensorflow.keras as keras
from keras import layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D,BatchNormalizat
from keras.constraints import maxnorm
from keras.wrappers.scikit_learn import KerasClassifier
```

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import timeit

seed(1)
tf.random.set_seed(2)

s = time.time()

# train_valid_test split
clean_images = imgs[:10000]

clean_train_idx = np.random.choice(range(10000), 8000, replace = False)
clean_test_idx = np.setdiff1d(range(10000), clean_train_idx, assume_unique=Fa

clean_imgs_train = clean_images[clean_train_idx]
clean_imgs_test = clean_images[clean_test_idx]

clean_labels_train = clean_labels[clean_train_idx]
clean_labels_test = clean_labels[clean_test_idx]

train_images = np.concatenate((clean_imgs_train, imgs[10000:]))
train_labels = noisy_labels[np.concatenate((clean_train_idx, range(10000,5000

test_labels = clean_labels[clean_test_idx]

# Normalize x
X_train = train_images / 255
imgs_test = imgs[clean_test_idx]

#create model
model_1_cnn = Sequential()
#add model layers
model_1_cnn.add(Conv2D(32, (3,3), padding="same", activation="relu", input_sh
model_1_cnn.add(MaxPooling2D(2, 2))
model_1_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_1_cnn.add(MaxPooling2D(2, 2))
model_1_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_1_cnn.add(Flatten())
model_1_cnn.add(Dense(64, activation="relu"))
model_1_cnn.add(Dense(10))
#compile model using accuracy to measure model performance
model_1_cnn.compile(optimizer='adam', loss=keras.losses.SparseCategoricalCros
#train the model
history = model_1_cnn.fit(X_train, train_labels, epochs=10)

print("Time to train CNN Model 1: %s seconds" % (time.time() - s))

# CNN
def model_I_cnn(image):
    '''
```

```
    This function should takes in the image of dimension 32*32*3 as input and
    '''
    #predict
    X_test = np.array(image)/255
    label=np.argmax(model_1_cnn.predict(X_test),axis=1)
    return label

# test for CNN (less than 10 min)
start = timeit.default_timer()
labels_pred = model_I_cnn(imgs_test)
stop = timeit.default_timer()
print('The time to evaluate 2,000 images with CNN Model 1 is ', stop - start,

acc=np.mean(labels_pred==test_labels)
print('The accuracy of the cnn model 1 is:%3f'%(acc))
```

```
Epoch 1/10
1500/1500 [==============================] - 17s 11ms/step - loss: 2.2734 - ac
curacy: 0.1470
Epoch 2/10
1500/1500 [==============================] - 18s 12ms/step - loss: 2.2359 - ac
curacy: 0.1924
Epoch 3/10
1500/1500 [==============================] - 17s 11ms/step - loss: 2.2113 - ac
curacy: 0.2173
Epoch 4/10
1500/1500 [==============================] - 17s 12ms/step - loss: 2.1888 - ac
curacy: 0.2342
Epoch 5/10
1500/1500 [==============================] - 18s 12ms/step - loss: 2.1650 - ac
curacy: 0.2514
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 2.1359 - ac
curacy: 0.2659
Epoch 7/10
1500/1500 [==============================] - 19s 13ms/step - loss: 2.1008 - ac
curacy: 0.2801
Epoch 8/10
1500/1500 [==============================] - 20s 13ms/step - loss: 2.0563 - ac
curacy: 0.2966
Epoch 9/10
1500/1500 [==============================] - 18s 12ms/step - loss: 2.0032 - ac
curacy: 0.3161
Epoch 10/10
1500/1500 [==============================] - 19s 12ms/step - loss: 1.9386 - ac
curacy: 0.3379
Time to train CNN Model 1: 182.2326579093933 seconds
The time to evaluate 2,000 images with CNN Model 1 is  0.3186524999982794 seco
nds
The accuracy of the cnn model 1 is:0.456000
```

CNN Model 2

In [99]:

```
s = time.time()

seed(1)
tf.random.set_seed(2)

# train_valid_test split
clean_images = imgs[:10000]

clean_train_idx = np.random.choice(range(10000), 8000, replace = False)
clean_test_idx = np.setdiff1d(range(10000), clean_train_idx, assume_unique=Fa

clean_imgs_train = clean_images[clean_train_idx]
clean_imgs_test = clean_images[clean_test_idx]

clean_labels_train = clean_labels[clean_train_idx]
clean_labels_test = clean_labels[clean_test_idx]


test_labels = clean_labels[clean_test_idx]

K.clear_session()

#create model
model_2_cnn = Sequential()
#add model layers
model_2_cnn.add(Conv2D(32, (3,3), padding="same", activation="relu", input_sh
model_2_cnn.add(MaxPooling2D(2, 2))
model_2_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_2_cnn.add(MaxPooling2D(2, 2))
model_2_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_2_cnn.add(Flatten())
model_2_cnn.add(BatchNormalization())
model_2_cnn.add(Dense(64, activation="relu"))
model_2_cnn.add(Dense(10))
#compile model using accuracy to measure model performance
model_2_cnn.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalC
#train the model
history = model_2_cnn.fit(clean_imgs_train/255.0, clean_labels_train, epochs=



model_2_cnn.save('model_2_cnn.h5')

def model_II_cnn(image):
    #predict
    X_test = np.array(image)/255
    label = model_2_cnn.predict(X_test)
    label = np.argmax(np.round(label), axis=1)
    return label
```

```
label_pred1=model_II_cnn(imgs)


new_clean_idx=np.append(clean_train_idx,np.array(range(10000,50000))[label_pr
new_clean_labels=np.append(clean_labels_train,noisy_labels[new_clean_idx[8000
new_clean_imgs=imgs[new_clean_idx]

K.clear_session()

#create model
model_2_cnn = Sequential()
#add model layers
model_2_cnn.add(Conv2D(32, (3,3), padding="same", activation="relu", input_sh
model_2_cnn.add(MaxPooling2D(2, 2))
model_2_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_2_cnn.add(MaxPooling2D(2, 2))
model_2_cnn.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model_2_cnn.add(Flatten())
model_2_cnn.add(BatchNormalization())
model_2_cnn.add(Dense(64, activation="relu"))
model_2_cnn.add(Dense(10))
#compile model using accuracy to measure model performance
model_2_cnn.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalC
#train the model
history = model_2_cnn.fit(new_clean_imgs/255.0, new_clean_labels, epochs=10)

def model_II_cnn(image):
    #predict
    X_test = np.array(image)/255
    label = model_2_cnn.predict(X_test)
    label = np.argmax(np.round(label), axis=1)
    return label

print("Time to train CNN Model 2: %s seconds" % (time.time() - s))
start = timeit.default_timer()
label_pred2=model_II_cnn(clean_imgs_test)

acc=np.mean(label_pred2==clean_labels_test)
print('The accuracy of the cnn is:%3f'%(acc))

stop = timeit.default_timer()
print('Time to evalues 2,000 samples with CNN Model 2 is ', stop - start, 'se
```

```
Epoch 1/10
250/250 [==============================] - 3s 12ms/step - loss: 1.6688 - accur
acy: 0.4090
Epoch 2/10
250/250 [==============================] - 3s 12ms/step - loss: 1.2854 - accur
acy: 0.5443
Epoch 3/10
250/250 [==============================] - 3s 11ms/step - loss: 1.1047 - accur
acy: 0.6072
Epoch 4/10
```

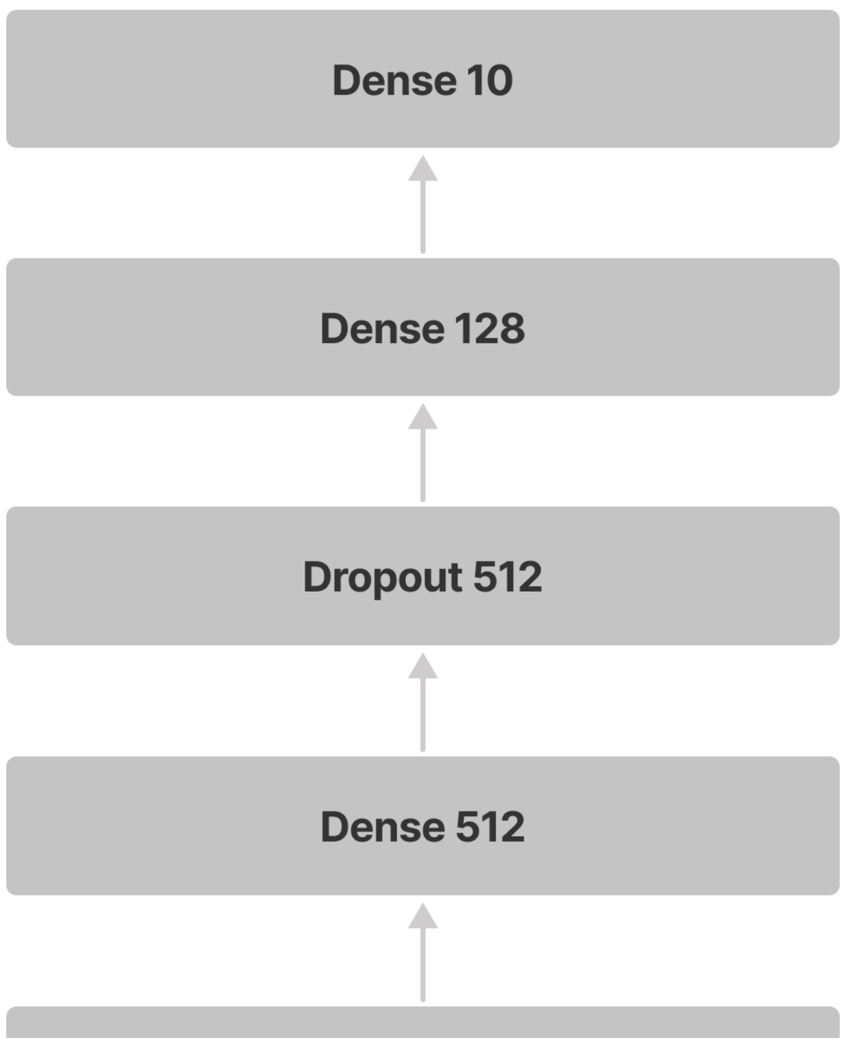
```
250/250 [==============================] - 3s 12ms/step - loss: 0.9249 - accur
acy: 0.6726
Epoch 5/10
250/250 [==============================] - 3s 11ms/step - loss: 0.7639 - accur
acy: 0.7283
Epoch 6/10
250/250 [==============================] - 3s 11ms/step - loss: 0.6433 - accur
acy: 0.7742
Epoch 7/10
250/250 [==============================] - 3s 11ms/step - loss: 0.5183 - accur
acy: 0.8148
Epoch 8/10
250/250 [==============================] - 3s 11ms/step - loss: 0.4017 - accur
acy: 0.8577
Epoch 9/10
250/250 [==============================] - 3s 11ms/step - loss: 0.3182 - accur
acy: 0.8898
Epoch 10/10
250/250 [==============================] - 3s 11ms/step - loss: 0.2449 - accur
acy: 0.9175
Epoch 1/10
558/558 [==============================] - 7s 11ms/step - loss: 1.3221 - accur
acy: 0.5369
Epoch 2/10
558/558 [==============================] - 7s 12ms/step - loss: 0.9400 - accur
acy: 0.6708
Epoch 3/10
558/558 [==============================] - 7s 12ms/step - loss: 0.7779 - accur
acy: 0.7266
Epoch 4/10
558/558 [==============================] - 7s 12ms/step - loss: 0.6714 - accur
acy: 0.7637
Epoch 5/10
558/558 [==============================] - 7s 12ms/step - loss: 0.5822 - accur
acy: 0.7962
Epoch 6/10
558/558 [==============================] - 7s 12ms/step - loss: 0.5017 - accur
acy: 0.8268
Epoch 7/10
558/558 [==============================] - 7s 12ms/step - loss: 0.4209 - accur
acy: 0.8513
Epoch 8/10
558/558 [==============================] - 7s 12ms/step - loss: 0.3604 - accur
acy: 0.8774
Epoch 9/10
558/558 [==============================] - 7s 12ms/step - loss: 0.3146 - accur
acy: 0.8889
Epoch 10/10
558/558 [==============================] - 7s 13ms/step - loss: 0.2723 - accur
acy: 0.9058
Time to train CNN Model 2: 103.55254483222961 seconds
The accuracy of the cnn is:0.541500
Time to evalues 2,000 samples with CNN Model 2 is  0.29487854200124275 seconds
```

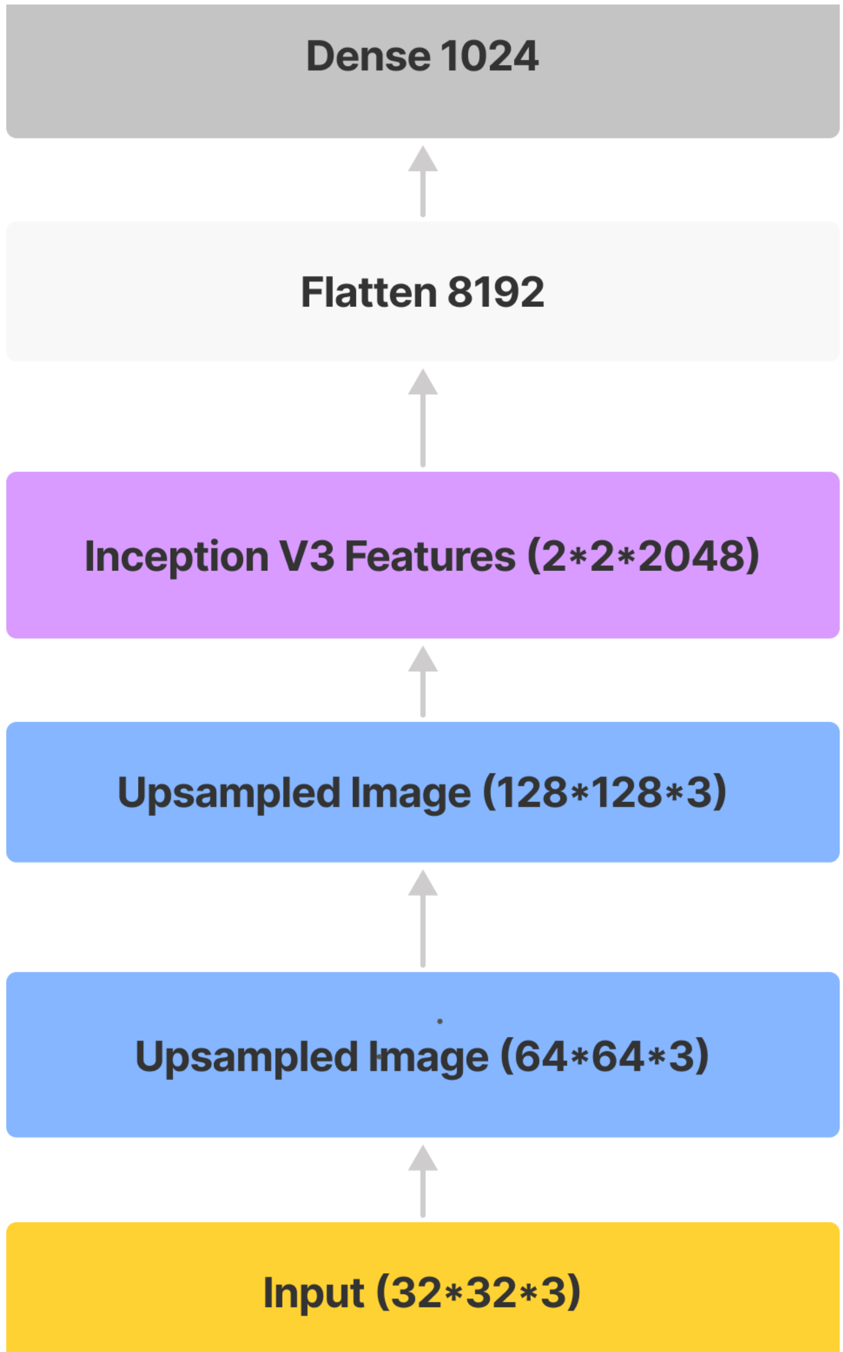
Chosen Model Architecture

Our chosen model is a **Transfer Learning** based model. The following are the high level steps in this method:

- Use InceptionV3, a pre-trained CNN model, to extract features out of our training dataset
- Pass these feature representations of the images through a neural network
- Use generated probabilities to classify each image into one of 10 categories

Hyperparameters were chosen using grid search (not shown). For details of the steps, follow the comments in sections 2.2 and 2.3.





2.1. Baseline Model

In [17]:

```
# [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 fea
    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=Non
    i += 1
```

In [18]:

```
# [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 [19]:

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

2.2. Model I

Model I assumes the provided noisy labels are clean and uses all given observations to train.

Partial Model I

"Partial Model" refers to the model where we reserve 2,000 of the clean observations for validation.

In [8]:

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

# NOTE: The model in this cell leaves some clean data for validation.
# Do not use for final testing

start_time = time.time()

seed(1)
tf.random.set_seed(2)

seed(1)

# Pick out 2,000 clean observations for validation later

clean_images = imgs[:10000]

clean_train_idx = np.random.choice(range(10000), 8000, replace = False)
clean_test_idx = np.setdiff1d(range(10000), clean_train_idx, assume_unique=Fa

clean_imgs_train = clean_images[clean_train_idx]
clean_imgs_test = clean_images[clean_test_idx]

clean_labels_train = clean_labels[clean_train_idx]
clean_labels_test = clean_labels[clean_test_idx]

train_images = np.concatenate((clean_imgs_train, imgs[10000:]))
train_labels = noisy_labels[np.concatenate((clean_train_idx, range(10000,5000

train_images = train_images / 255.0

# Begin modeling

K.clear_session()

# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o

transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False

model1_partial = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)), # This is the shape

    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
```

```
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model1_partial.compile(optimizer=Adam(learning_rate=0.0001), loss=tf.keras.lo

# Fit the model to appropriate data

history = model1_partial.fit(
    train_images,
    train_labels,
    batch_size=64,
    epochs=6
)
print("Time to train partial Model 1: %s seconds" % (time.time() - start_time



def model_I_partial(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and
    '''
    pred = np.argmax(model1_partial.predict(np.expand_dims(image/255.0, axis=

    return(pred)
```

```
Epoch 1/6
2022-03-23 05:36:58.979224: W tensorflow/core/platform/profile_utils/cpu_utils
.cc:128] Failed to get CPU frequency: 0 Hz
/opt/homebrew/lib/python3.9/site-packages/tensorflow/python/util/dispatch.py:1
082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=Tru
e`, but the `output` argument was produced by a sigmoid or softmax activation
and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
750/750 [==============================] - 343s 455ms/step - loss: 2.2704 - ac
curacy: 0.2093
Epoch 2/6
750/750 [==============================] - 301s 402ms/step - loss: 2.1833 - ac
curacy: 0.2653
Epoch 3/6
750/750 [==============================] - 313s 417ms/step - loss: 2.1468 - ac
curacy: 0.2868
Epoch 4/6
750/750 [==============================] - 307s 409ms/step - loss: 2.1200 - ac
curacy: 0.3007
Epoch 5/6
750/750 [==============================] - 323s 431ms/step - loss: 2.0943 - ac
curacy: 0.3096
Epoch 6/6
750/750 [==============================] - 318s 423ms/step - loss: 2.0637 - ac
curacy: 0.3193
Time to train partial Model 1: 1906.6354398727417 seconds
```

In [67]:

```python
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))


s = time.time()
evaluation(model_I_partial, clean_labels_test, clean_imgs_test)
time_2k = (time.time() - s)
time_1 = time_2k/2000
print("Time to predict 2000 new points with Model I: %s seconds" % time_2k)
print("Average time to predict 1 new points with Model I: %s seconds" % time_
```

```
              precision    recall  f1-score   support

           0       0.65      0.69      0.67       200
           1       0.70      0.82      0.76       182
           2       0.75      0.61      0.67       202
           3       0.66      0.52      0.58       190
           4       0.59      0.64      0.62       203
           5       0.76      0.62      0.69       191
           6       0.69      0.79      0.74       225
           7       0.65      0.79      0.72       208
           8       0.70      0.80      0.75       210
           9       0.81      0.58      0.67       189

    accuracy                           0.69      2000
   macro avg       0.70      0.69      0.68      2000
weighted avg       0.70      0.69      0.69      2000

Time to predict 2000 new points with Model I: 75.11498999595642 seconds
Average time to predict 1 new points with Model I: 0.03755749499797821 seconds
```

Full Model I

"Full Model" refers to the model where we use all 50,000 observations for training.

In [28]:

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

# NOTE: The model in this cell uses all of the clean data for training.
# This model will be used for testing on the final day.

start_time = time.time()

seed(1)
tf.random.set_seed(2)

train_images = imgs
train_labels = noisy_labels

train_images = train_images / 255.0

# Begin modeling

K.clear_session()

# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o

transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False

model1 = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)),
```

```
    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model1.compile(optimizer=Adam(learning_rate=0.0001), loss=tf.keras.losses.Spa

# Fit the model to appropriate data

history = model1.fit(
    train_images,
    train_labels,
    batch_size=64,
    epochs=6
)
print("Time to train full Model 1: %s seconds" % (time.time() - start_time))

model1.save('../output/model1')

def model_I(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and
    '''
    pred = np.argmax(model1.predict(np.expand_dims(image/255.0, axis=0)))

    return(pred)
```

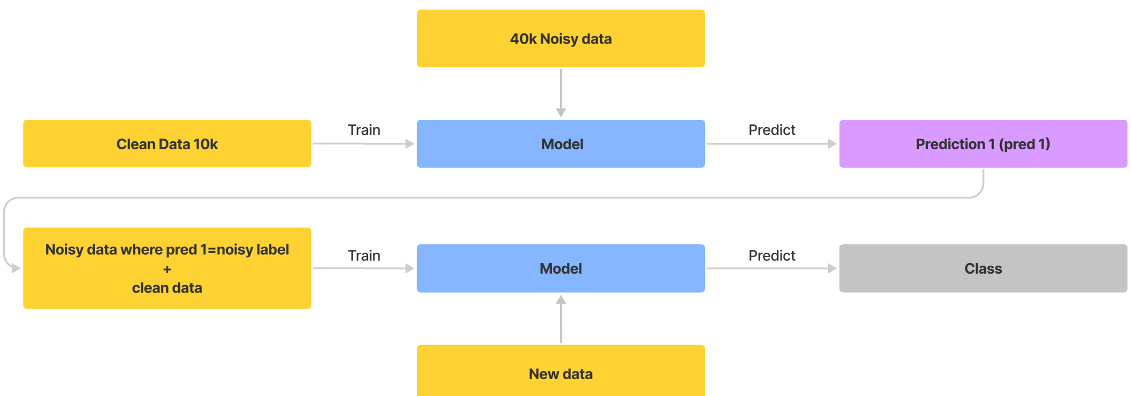
```
Epoch 1/6
/opt/homebrew/lib/python3.9/site-packages/tensorflow/python/util/dispatch.py:1
082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=Tru
e`, but the `output` argument was produced by a sigmoid or softmax activation
and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
782/782 [==============================] - 315s 401ms/step - loss: 2.2661 - ac
curacy: 0.2148
Epoch 2/6
782/782 [==============================] - 346s 442ms/step - loss: 2.1756 - ac
curacy: 0.2727
Epoch 3/6
782/782 [==============================] - 356s 455ms/step - loss: 2.1466 - ac
curacy: 0.2877
Epoch 4/6
782/782 [==============================] - 373s 477ms/step - loss: 2.1227 - ac
curacy: 0.3008
Epoch 5/6
782/782 [==============================] - 476s 608ms/step - loss: 2.0955 - ac
curacy: 0.3117
Epoch 6/6
782/782 [==============================] - 426s 545ms/step - loss: 2.0684 - ac
curacy: 0.3195
Time to train full Model 1: 2294.4149708747864 seconds
INFO:tensorflow:Assets written to: ../output/model1/assets
```

2.3. Model II

Our approach involves training 2 separate models with the same architecture as above but with different training sets.

Steps:

- Train a model using only the cleanly labeled data. Make predictions on the rest of the data.
- Of the rest of the data, keep the ones where the predictions match the noisy labels
- Use clean data + data from the previous step to train our final model



Partial Model II

In [33]:

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

# NOTE: The model in this cell leaves some clean data for validation.
# Do not use for final testing

s = time.time()

# Pick out 2,000 clean observations for validation later

seed(1)
tf.random.set_seed(2)

clean_images = imgs[:10000]

clean_train_idx = np.random.choice(range(10000), 8000, replace = False)
clean_test_idx = np.setdiff1d(range(10000), clean_train_idx, assume_unique=Fa
```

```
clean_imgs_train = clean_images[clean_train_idx] / 255.0
clean_imgs_test = clean_images[clean_test_idx]

clean_labels_train = clean_labels[clean_train_idx]
clean_labels_test = clean_labels[clean_test_idx]

train_images = np.concatenate((clean_imgs_train, imgs[10000:]))
train_labels = noisy_labels[np.concatenate((clean_train_idx, range(10000,5000

# Begin modeling

K.clear_session()

# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o

transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False

model2_1_partial = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)), # This is the shape

    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model2_1_partial.compile(optimizer=Adam(learning_rate=0.0001), loss=tf.keras.

# Fit the first model in Model II's architecture

history = model2_1_partial.fit(
    clean_imgs_train,
    clean_labels_train,
    batch_size=64,
    epochs=6,
    verbose = True
)
```

```
K.clear_session()

# Use the first model to make predictions on data we don't have clean labels

noisy_pred = model2_1_partial.predict(imgs[10000:]/255.0)
noisy_pred_label = np.argmax(noisy_pred, axis =  1)

# Keep noisy data where our predictions match the noisy label

images_to_keep = imgs[10000:50000][noisy_pred_label == noisy_labels[10000:500
images_2 = np.concatenate((clean_imgs_train, images_to_keep/ 255.0))

labels_to_keep = noisy_labels[10000:50000][noisy_pred_label == noisy_labels[1
labels_2 = np.concatenate((clean_labels_train, labels_to_keep))

# ---------------------------------------------------------------------------

# Second part of Model II


# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o

transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False

model2_2_partial = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)), # This is the shape

    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model2_2_partial.compile(optimizer=Adam(learning_rate=0.0001), loss=tf.keras.

# Fit final part of Model II

history = model2_2_partial.fit(
```

```
    images_2,
    labels_2,
    batch_size=64,
    epochs=6,
    verbose = True
)

print("Time to train partial Model 1: %s seconds" % (time.time() - s))

def model_II_partial(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and
    '''
    pred = np.argmax(model2_2_partial.predict(np.expand_dims(image/255.0, axi

    return(pred)
```

```
Epoch 1/6
/opt/homebrew/lib/python3.9/site-packages/tensorflow/python/util/dispatch.py:1
082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=Tru
e`, but the `output` argument was produced by a sigmoid or softmax activation
and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
125/125 [==============================] - 85s 662ms/step - loss: 1.2977 - acc
uracy: 0.5719
Epoch 2/6
125/125 [==============================] - 83s 661ms/step - loss: 0.8096 - acc
uracy: 0.7299
Epoch 3/6
125/125 [==============================] - 83s 661ms/step - loss: 0.6350 - acc
uracy: 0.7835
Epoch 4/6
125/125 [==============================] - 83s 662ms/step - loss: 0.4924 - acc
uracy: 0.8296
Epoch 5/6
125/125 [==============================] - 82s 660ms/step - loss: 0.3727 - acc
uracy: 0.8754
Epoch 6/6
125/125 [==============================] - 83s 662ms/step - loss: 0.2801 - acc
uracy: 0.9072
Epoch 1/6
319/319 [==============================] - 232s 715ms/step - loss: 0.8402 - ac
curacy: 0.7262
Epoch 2/6
319/319 [==============================] - 252s 790ms/step - loss: 0.4659 - ac
curacy: 0.8477
Epoch 3/6
319/319 [==============================] - 258s 809ms/step - loss: 0.3533 - ac
curacy: 0.8843
Epoch 4/6
319/319 [==============================] - 160s 500ms/step - loss: 0.2676 - ac
curacy: 0.9123
Epoch 5/6
319/319 [==============================] - 140s 440ms/step - loss: 0.2122 - ac
curacy: 0.9294
Epoch 6/6
319/319 [==============================] - 142s 445ms/step - loss: 0.1650 - ac
curacy: 0.9440
Time to train partial Model 1: 2219.1293523311615 seconds
```

In [68]:

```
s = time.time()
evaluation(model_II_partial, clean_labels_test, clean_imgs_test)
time_2k = (time.time() - s)
time_1 = time_2k/2000
print("Time to predict 2000 new points with Model II: %s seconds" % time_2k)
print("Average time to predict 1 new points with Model II: %s seconds" % time_
```

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       200
           1       0.71      0.89      0.79       182
           2       0.83      0.62      0.71       202
           3       0.70      0.60      0.65       190
           4       0.70      0.66      0.68       203
           5       0.71      0.73      0.72       191
           6       0.80      0.80      0.80       225
           7       0.75      0.82      0.78       208
           8       0.73      0.81      0.77       210
           9       0.79      0.75      0.77       189

    accuracy                           0.75      2000
   macro avg       0.75      0.75      0.74      2000
weighted avg       0.75      0.75      0.74      2000

Time to predict 2000 new points with Model II: 73.41302490234375 seconds
Average time to predict 1 new points with Model II: 0.036706512451171874 secon
ds
```

Full Model II

This is the model we propose to our clients. This takes roughly the same amount of time to evaluate a new image as the partial Model II.

In [9]:

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

# NOTE: The model in this cell uses all of the clean data for training.
# This model will be used for testing on the final day.

s = time.time()

seed(1)
tf.random.set_seed(2)

# Prepare data for first model of Model II

clean_images = imgs[:10000]

clean_images = clean_images / 255.0

# Begin modeling

K.clear_session()

# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o
transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False
```

```
model2_1 = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)), # This is the shape

    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model2_1.compile(optimizer=Adam(learning_rate=0.0001),
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logi
                 metrics=['accuracy'])

# Fit the first model in Model II's architecture

history = model2_1.fit(
    clean_images,
    clean_labels,
    batch_size=64,
    epochs=6,
    verbose = True
)

K.clear_session()

# Use the first model to make predictions on data we don't have clean labels

noisy_pred = model2_1.predict(imgs[10000:]/255.0)
noisy_pred_label = np.argmax(noisy_pred, axis =  1)

# Keep noisy data where our predictions match the noisy label

images_to_keep = imgs[10000:50000][noisy_pred_label == noisy_labels[10000:500
images_2 = np.concatenate((imgs[:10000], images_to_keep)) / 255.0

labels_to_keep = noisy_labels[10000:50000][noisy_pred_label == noisy_labels[1
labels_2 = np.concatenate((clean_labels, labels_to_keep))

# ---------------------------------------------------------------------------
```

```
# Second part of Model II


# To extract features, we keep all layers except the last one from InceptionV
# We must also freeze these layers from training, since we want the model's o

transfer_model = tf.keras.applications.InceptionV3(include_top=False, weights
transfer_model.trainable = False

model2_2 = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(32, 32, 3)), # This is the shape

    # We upsample twice so that our images are (128,128,3)
    # Upsampling is required since InceptionV3 expects input of size at least
    tf.keras.layers.UpSampling2D(size = (2,2)),
    tf.keras.layers.UpSampling2D(size = (2,2)),

    # This is the pretrained model with the layers freezed
    transfer_model,

    # Flatten to put all values in one long column
    tf.keras.layers.Flatten(),

    # Our custom neural network. Hyperparameters were chosen through grid sea
    # Random node dropouts to avoid overfitting
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model2_2.compile(optimizer=Adam(learning_rate=0.0001), loss=tf.keras.losses.S

# Fit final part of Model II

history = model2_2.fit(
    images_2,
    labels_2,
    batch_size=64,
    epochs=6,
    verbose = True
)
model2_2.save('../output/model2')
print("Time to train full Model 1: %s seconds" % (time.time() - s))

def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and
    '''
    pred = np.argmax(model2_2.predict(np.expand_dims(image/255.0, axis=0)))

    return(pred)
```

```
Epoch 1/6
157/157 [==============================] - 67s 415ms/step - loss: 1.2695 - acc
uracy: 0.5844
Epoch 2/6
157/157 [==============================] - 72s 460ms/step - loss: 0.7995 - acc
uracy: 0.7287
Epoch 3/6
157/157 [==============================] - 77s 489ms/step - loss: 0.6285 - acc
uracy: 0.7816
Epoch 4/6
157/157 [==============================] - 77s 490ms/step - loss: 0.5045 - acc
uracy: 0.8273
Epoch 5/6
157/157 [==============================] - 77s 493ms/step - loss: 0.3950 - acc
uracy: 0.8641
Epoch 6/6
157/157 [==============================] - 78s 496ms/step - loss: 0.2971 - acc
uracy: 0.9002
Epoch 1/6
349/349 [==============================] - 178s 504ms/step - loss: 0.8215 - ac
curacy: 0.7280
Epoch 2/6
349/349 [==============================] - 173s 496ms/step - loss: 0.4967 - ac
curacy: 0.8406
Epoch 3/6
349/349 [==============================] - 180s 516ms/step - loss: 0.3836 - ac
curacy: 0.8732
Epoch 4/6
349/349 [==============================] - 180s 516ms/step - loss: 0.3059 - ac
curacy: 0.8969
Epoch 5/6
349/349 [==============================] - 176s 506ms/step - loss: 0.2370 - ac
curacy: 0.9193
Epoch 6/6
349/349 [==============================] - 177s 507ms/step - loss: 0.1897 - ac
curacy: 0.9371
```

```
2022-03-23 16:47:01.177626: W tensorflow/python/util/util.cc:368] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.
INFO:tensorflow:Assets written to: ../output/model2/assets
Time to train full Model 1: 1859.8191328048706 seconds
```

Label Generation on Test Data

In [10]:

```
(x_train_tf, y_train_tf), (x_test_tf, y_test_tf) = tf.keras.datasets.cifar10.
y_test_tf = np.squeeze(y_test_tf)
```

In [ ]:

```
s = time.time()
# load the images
n_test = 10000
labels = pd.DataFrame(np.nan, index = range(n_test),columns = ["Index","Basel
for i in range(n_img):
    img_fn = f'../data/images/test{i+1:05d}.png'
    labels.iloc[i,0] = f'test{i+1:05d}'
    test_img=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB)
    labels.iloc[i,1] = int(baseline_model(test_img)[0])
    labels.iloc[i,2] = model_I(test_img)
    labels.iloc[i,3] = model_II(test_img)
print("Time to predict: %s" % (time.time()-s))
```

In [ ]:

```
labels = labels.astype({'Model II': 'int8', 'Baseline':'int8', 'Model I': 'in
```

In [29]:

```
labels.to_csv('../data/label_prediction.csv',index=False)
```

```
  Input In [29]
    labels.to_csv('../data/label_prediction.csv',index=False.
                                                             ^
SyntaxError: unexpected EOF while parsing
```

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 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="
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)
```