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
```

1. Load the datasets

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

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

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

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

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

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 [3]:

```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.

2.1. Baseline Model

In [4]:

```
# [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
```

In [5]:

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

In [6]:

```
score_full = clf.score(feature_mtx, target_vec)
print("accuracy", score_full)

train = feature_mtx[5000:]
train_cls = target_vec[5000:]
val = feature_mtx[:5000]
val_cls = np.array(clean_labels[:5000])

clf1 = LogisticRegression(random_state=0).fit(train, train_cls)
score_train = clf1.score(train, train_cls)
print("train accuracy", score_train)

score_val = clf1.score(val, val_cls)
print("val accuracy", score_val)
```

```
accuracy 0.14442
train accuracy 0.14453333333333335
val accuracy 0.2428
```

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 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).reshap
e(1,-1)
    return int(clf.predict(feature)[0])
```

2.2. Model I

In [8]:
```
# Splitting data into Train and Validation set

import tensorflow as tf
from sklearn.model_selection import train_test_split

X_train = tf.cast(imgs[5000:], dtype='float32')/255.0 #- 90% Training
X_val = tf.cast(imgs[:5000], dtype='float32')/255.0 #- 10% Validation
y_train = tf.one_hot(noisy_labels[5000:], depth=10)#- 90% Training
y_val = tf.one_hot(clean_labels[:5000], depth=10) #- 10% Validation

print(X_train.shape)
print(X_val.shape)
print(y_train.shape)
print(y_val.shape)
```

```
(45000, 32, 32, 3)
(5000, 32, 32, 3)
(45000, 10)
(5000, 10)
```

MobileNet v2

In [9]:
```
import keras
from keras import backend as K
from keras.layers.core import Dense, Activation
from tensorflow.keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2
from keras.applications.mobilenet import preprocess_input
import numpy as np
from IPython.display import Image
import time
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.callbacks import ReduceLROnPlateau
```

In [13]:
```
# Sophisticated Model - MobileNetv2

#imports the mobilenetv2 model and discards the last 1000 neuron layer.
mobile_model = MobileNetV2(weights='imagenet',include_top=False)

x = mobile_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512,activation='relu')(x) #we add dense layers so that the model can
learn more complex functions and classify for better results.
x = Dense(128,activation='relu')(x) #dense layer 2
x = Dense(64,activation='relu')(x) #dense layer 3
x = Dense(10,activation='softmax')(x) #final layer with softmax activation

model1 = Model(inputs=mobile_model.input, outputs=x)

for layer in model1.layers[:20]:
    layer.trainable=False
for layer in model1.layers[20:]:
    layer.trainable=True
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not
in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loade
d as the default.
```

In [14]:
```
# Defining learning rates and epochs
lr = 0.0005
epochs = 10

# defining optimizer and Compiling model
optimizer = Adam(learning_rate=lr)
model1.compile(loss= 'categorical_crossentropy', optimizer=optimizer, metrics=
['accuracy'])

# Parameter tuning and Saving the best model
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode
='min')
mcp_save = ModelCheckpoint('.mdl1_MobNet.hdf5', save_best_only=True, monitor=
'val_loss', mode='min')
reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=7,
verbose=1, epsilon=1e-4, mode='min')

#Model Training
start = time.time()
history = model1.fit(X_train, y_train, batch_size = 64 ,
                     epochs=epochs, verbose=1, validation_data=(X_val, y_val),
                     callbacks=[earlyStopping, mcp_save, reduce_lr_loss])
end = time.time()
print("Runtime of the model 1's evaluation is %d seconds." % (end - start))
```

```
WARNING:tensorflow:`epsilon` argument is deprecated and will be removed, use
`min_delta` instead.
Epoch 1/10
704/704 [==============================] - 78s 107ms/step - loss: 2.2650 - ac
curacy: 0.1802 - val_loss: 2.2154 - val_accuracy: 0.1988 - lr: 5.0000e-04
Epoch 2/10
704/704 [==============================] - 88s 125ms/step - loss: 2.2079 - ac
curacy: 0.2347 - val_loss: 2.0830 - val_accuracy: 0.3212 - lr: 5.0000e-04
Epoch 3/10
704/704 [==============================] - 85s 121ms/step - loss: 2.1804 - ac
curacy: 0.2570 - val_loss: 1.7855 - val_accuracy: 0.5026 - lr: 5.0000e-04
Epoch 4/10
704/704 [==============================] - 84s 119ms/step - loss: 2.1592 - ac
curacy: 0.2743 - val_loss: 1.6849 - val_accuracy: 0.5934 - lr: 5.0000e-04
Epoch 5/10
704/704 [==============================] - 86s 122ms/step - loss: 2.1486 - ac
curacy: 0.2832 - val_loss: 1.8186 - val_accuracy: 0.4660 - lr: 5.0000e-04
Epoch 6/10
704/704 [==============================] - 85s 120ms/step - loss: 2.1386 - ac
curacy: 0.2903 - val_loss: 1.5675 - val_accuracy: 0.6376 - lr: 5.0000e-04
Epoch 7/10
704/704 [==============================] - 76s 108ms/step - loss: 2.1344 - ac
curacy: 0.2924 - val_loss: 1.8811 - val_accuracy: 0.4026 - lr: 5.0000e-04
Epoch 8/10
704/704 [==============================] - 72s 102ms/step - loss: 2.1434 - ac
curacy: 0.2846 - val_loss: 1.6501 - val_accuracy: 0.5982 - lr: 5.0000e-04
Epoch 9/10
704/704 [==============================] - 74s 105ms/step - loss: 2.1260 - ac
curacy: 0.2982 - val_loss: 1.5918 - val_accuracy: 0.5934 - lr: 5.0000e-04
Epoch 10/10
704/704 [==============================] - 73s 104ms/step - loss: 2.1215 - ac
curacy: 0.3024 - val_loss: 1.5966 - val_accuracy: 0.6034 - lr: 5.0000e-04
Runtime of the model 1's evaluation is 799 seconds.
```

In [10]:

```python
# Loading trained model from file

model1 = keras.models.load_model(".mdl1_MobNet.hdf5")
```

In [11]:

```python
# Function to predict label for a given image using Model 1
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...
    test = tf.cast(image, dtype='float32')/255.0
    test = tf.reshape(test, [1,32,32,3])
    return np.argmax(model1(test), axis = 1)[0]
```

2.3. Model II

In [12]:
```python
# Data with only noisy labels - this will be our supporting dataset
only_noisy_set = imgs[10000:]
only_noisy_labels = noisy_labels[10000:]

# data with clean labels
clean_set = imgs[:10000]

# spilitting data with clean labels into train and validation
clean_train = clean_set[5000:]
clean_label_train = clean_labels[5000:]

clean_val = clean_set[:5000]
clean_label_val = clean_labels[:5000]

# Normalizing the data

# For clean net
X_train_cln = tf.cast(clean_train, dtype='float32')/255.0
y_train_cln = tf.one_hot(clean_label_train, depth=10)
y_train_cln = tf.keras.layers.Concatenate(axis=1)([tf.constant(float(0), shape
=(5000, 10)), y_train_cln])

# For residual net
X_train_res = tf.cast(only_noisy_set, dtype='float32')/255.0
y_train_res = tf.one_hot(only_noisy_labels, depth=10)
y_train_res = tf.keras.layers.Concatenate(axis=1)([y_train_res, tf.constant(fl
oat(0), shape=(40000, 10))])

# Combining Residual and Clean
X_train = tf.cast(np.vstack((X_train_res, X_train_cln)), dtype='float32')
y_train = tf.cast(np.vstack((y_train_res, y_train_cln)), dtype='int64')

# for validation set
X_val = tf.cast(clean_val, dtype='float32')/255.0
y_val = tf.one_hot(clean_label_val, depth=10)
y_val = tf.keras.layers.Concatenate(axis=1)([tf.constant(float(0), shape=(5000
, 10)), y_val])

print(X_train_res.shape, X_train_cln.shape)
print(y_train_res.shape, y_train_cln.shape)
print(X_train.shape, y_train.shape)
```

```
(40000, 32, 32, 3) (5000, 32, 32, 3)
(40000, 20) (5000, 20)
(45000, 32, 32, 3) (45000, 20)
```

In [13]:
```
# Defining a custom loss function
cce = tf.keras.losses.CategoricalCrossentropy(reduction = 'none')

def myLoss(yTrue, yPred):
    xrTrue = yTrue[:, :10]
    xcTrue = yTrue[:, 10:]

    xrPred = yPred[:, :10]
    xcPred = yPred[:, 10:]

    alpha = 0.1

    xr_loss = tf.experimental.numpy.nanmean(cce(xrTrue, xrPred))
    xc_loss = tf.experimental.numpy.nanmean(cce(xcTrue, xcPred))

    if xr_loss < 0:
        K.print_tensor(xr_loss)
        K.print_tensor(xrTrue)
        K.print_tensor(xrPred)

    if xc_loss < 0:
        K.print_tensor(xc_loss)
        K.print_tensor(xcTrue)
        K.print_tensor(xcPred)

    return (alpha*(xr_loss) + (xc_loss))

# Defining a custom accuracy function
def accuracy(yTrue, yPred):

    xrTrue = yTrue[:, :10]
    xcTrue = yTrue[:, 10:]

    xrMask = tf.reduce_sum(xrTrue, 1)
    xcMask = tf.reduce_sum(xcTrue, 1)

    xrPred = yPred[:, :10]
    xcPred = yPred[:, 10:]

    xrTrue = tf.cast(tf.math.argmax(xrTrue, axis = 1), tf.float32)
    xrPred = tf.cast(tf.math.argmax(xrPred, axis = 1), tf.float32)
    xcTrue = tf.cast(tf.math.argmax(xcTrue, axis = 1), tf.float32)
    xcPred = tf.cast(tf.math.argmax(xcPred, axis = 1), tf.float32)

    r_count = tf.math.reduce_sum(tf.cast((xrTrue == xrPred), tf.float32)*xrMas
k)
    c_count = tf.math.reduce_sum(tf.cast((xcTrue == xcPred), tf.float32)*xcMas
k)

    accuracy = (tf.cast(r_count, tf.float32) + tf.cast(c_count, tf.float32))/
float(len(yTrue))

    return accuracy
```

In [14]:

```
import keras
import tensorflow as tf
from keras import backend as K
from keras.layers.core import Dense, Activation
from tensorflow.keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Model
from keras.applications import imagenet_utils
from keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2
from keras.applications.mobilenet import preprocess_input
import numpy as np
from IPython.display import Image
import time
import keras.backend as K
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.callbacks import ReduceLROnPlateau
```

In [18]:
```python
#imports the mobilenetv2 model and discards the last 1000 neuron layer.
mobile_model = MobileNetV2(weights='imagenet',include_top=False)

x = mobile_model.output
x = GlobalAveragePooling2D()(x)

# Residual Net
x1 = Dense(512,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x1 = Dense(128,activation='relu')(x1) #dense layer 2
x1 = Dense(64,activation='relu')(x1) #dense layer 3

# Clean Net
x2 = Dense(512,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x2 = Dense(128,activation='relu')(x2) #dense layer 2
x2 = Dense(64,activation='relu')(x2) #dense layer 3

Xr = x1 + x2

# Residual net Output
Xr = Dense(10,activation='softmax')(Xr)

# Clean net output
Xc = Dense(10,activation='softmax')(x2)

# Combining Residual and Clean output
Xf = tf.keras.layers.Concatenate(axis=1)([Xr, Xc])

print(Xr.shape, Xc.shape, Xf.shape)
model2 = Model(inputs=mobile_model.input, outputs=Xf)

for layer in model2.layers[:20]:
    layer.trainable=False
for layer in model2.layers[20:]:
    layer.trainable=True
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
(None, 10) (None, 10) (None, 20)
```

In [20]:
```
# Defining learning rate and epochs
lr = 0.0005
epochs = 100

# Defining optimizer and Compiling Model 2
optimizer = Adam(learning_rate=lr)
model2.compile(loss= myLoss, optimizer=optimizer, metrics=[accuracy])

# Saving best model
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode
='min')
mcp_save = ModelCheckpoint('.mdl2_MobNet.hdf5', save_best_only=True, monitor=
'val_loss', mode='min')
reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=7,
verbose=1, epsilon=1e-4, mode='min')

#Model Training
start = time.time()
history = model2.fit(X_train, y_train, batch_size = 64,
                     epochs=epochs, verbose=1, validation_data=(X_val, y_val),
                     callbacks=[earlyStopping, mcp_save, reduce_lr_loss])

end = time.time()
print("Runtime of the model 2's evaluation is %d seconds." % (end - start))
```

```
WARNING:tensorflow:`epsilon` argument is deprecated and will be removed, use
`min_delta` instead.
Epoch 1/100
704/704 [==============================] - 74s 102ms/step - loss: 0.2318 - ac
curacy: 0.3594 - val_loss: 1.9516 - val_accuracy: 0.5953 - lr: 5.0000e-04
Epoch 2/100
704/704 [==============================] - 76s 109ms/step - loss: 0.2370 - ac
curacy: 0.3625 - val_loss: 1.5469 - val_accuracy: 0.6551 - lr: 5.0000e-04
Epoch 3/100
704/704 [==============================] - 79s 112ms/step - loss: 0.2292 - ac
curacy: 0.3624 - val_loss: 1.4908 - val_accuracy: 0.6754 - lr: 5.0000e-04
Epoch 4/100
704/704 [==============================] - 79s 112ms/step - loss: 0.2278 - ac
curacy: 0.3638 - val_loss: 1.4798 - val_accuracy: 0.6784 - lr: 5.0000e-04
Epoch 5/100
704/704 [==============================] - 80s 114ms/step - loss: 0.2303 - ac
curacy: 0.3636 - val_loss: 1.2718 - val_accuracy: 0.6701 - lr: 5.0000e-04
Epoch 6/100
704/704 [==============================] - 79s 113ms/step - loss: 0.2420 - ac
curacy: 0.3546 - val_loss: 1.4538 - val_accuracy: 0.6515 - lr: 5.0000e-04
Epoch 7/100
704/704 [==============================] - 76s 108ms/step - loss: 0.2231 - ac
curacy: 0.3703 - val_loss: 1.4443 - val_accuracy: 0.6812 - lr: 5.0000e-04
Epoch 8/100
704/704 [==============================] - 77s 110ms/step - loss: 0.2164 - ac
curacy: 0.3732 - val_loss: 1.5981 - val_accuracy: 0.6636 - lr: 5.0000e-04
Epoch 9/100
704/704 [==============================] - 76s 108ms/step - loss: 0.2148 - ac
curacy: 0.3739 - val_loss: 1.4772 - val_accuracy: 0.6764 - lr: 5.0000e-04
Epoch 10/100
704/704 [==============================] - 77s 110ms/step - loss: 0.2129 - ac
curacy: 0.3767 - val_loss: 1.3784 - val_accuracy: 0.6952 - lr: 5.0000e-04
Epoch 11/100
704/704 [==============================] - 77s 109ms/step - loss: 0.2154 - ac
curacy: 0.3741 - val_loss: 1.7467 - val_accuracy: 0.6555 - lr: 5.0000e-04
Epoch 12/100
703/704 [============================>.] - ETA: 0s - loss: 0.2133 - accuracy:
0.3757
Epoch 12: ReduceLROnPlateau reducing learning rate to 5.0000002374872565e-05.
704/704 [==============================] - 76s 108ms/step - loss: 0.2133 - ac
curacy: 0.3756 - val_loss: 1.4018 - val_accuracy: 0.6926 - lr: 5.0000e-04
Epoch 13/100
704/704 [==============================] - 78s 111ms/step - loss: 0.1985 - ac
curacy: 0.3866 - val_loss: 1.1523 - val_accuracy: 0.7419 - lr: 5.0000e-05
Epoch 14/100
704/704 [==============================] - 79s 112ms/step - loss: 0.1929 - ac
curacy: 0.3899 - val_loss: 1.1270 - val_accuracy: 0.7462 - lr: 5.0000e-05
Epoch 15/100
704/704 [==============================] - 77s 110ms/step - loss: 0.1914 - ac
curacy: 0.3937 - val_loss: 1.1258 - val_accuracy: 0.7486 - lr: 5.0000e-05
Epoch 16/100
704/704 [==============================] - 82s 116ms/step - loss: 0.1899 - ac
curacy: 0.3960 - val_loss: 1.1203 - val_accuracy: 0.7474 - lr: 5.0000e-05
Epoch 17/100
704/704 [==============================] - 80s 114ms/step - loss: 0.1888 - ac
```

```
curacy: 0.3966 - val_loss: 1.1477 - val_accuracy: 0.7536 - lr: 5.0000e-05
Epoch 18/100
704/704 [==============================] - 78s 111ms/step - loss: 0.1880 - ac
curacy: 0.3987 - val_loss: 1.1638 - val_accuracy: 0.7549 - lr: 5.0000e-05
Epoch 19/100
704/704 [==============================] - 75s 107ms/step - loss: 0.1877 - ac
curacy: 0.3990 - val_loss: 1.1941 - val_accuracy: 0.7516 - lr: 5.0000e-05
Epoch 20/100
704/704 [==============================] - 80s 113ms/step - loss: 0.1867 - ac
curacy: 0.4001 - val_loss: 1.2633 - val_accuracy: 0.7500 - lr: 5.0000e-05
Epoch 21/100
704/704 [==============================] - 78s 111ms/step - loss: 0.1869 - ac
curacy: 0.4007 - val_loss: 1.2453 - val_accuracy: 0.7526 - lr: 5.0000e-05
Epoch 22/100
704/704 [==============================] - 77s 110ms/step - loss: 0.1864 - ac
curacy: 0.4015 - val_loss: 1.2498 - val_accuracy: 0.7526 - lr: 5.0000e-05
Epoch 23/100
703/704 [============================>.] - ETA: 0s - loss: 0.1857 - accuracy:
0.4040
Epoch 23: ReduceLROnPlateau reducing learning rate to 5.000000237487257e-06.
704/704 [==============================] - 79s 113ms/step - loss: 0.1857 - ac
curacy: 0.4043 - val_loss: 1.3104 - val_accuracy: 0.7470 - lr: 5.0000e-05
Epoch 24/100
704/704 [==============================] - 76s 107ms/step - loss: 0.1856 - ac
curacy: 0.4048 - val_loss: 1.2914 - val_accuracy: 0.7516 - lr: 5.0000e-06
Epoch 25/100
704/704 [==============================] - 77s 110ms/step - loss: 0.1858 - ac
curacy: 0.4061 - val_loss: 1.2823 - val_accuracy: 0.7555 - lr: 5.0000e-06
Epoch 26/100
704/704 [==============================] - 81s 115ms/step - loss: 0.1851 - ac
curacy: 0.4067 - val_loss: 1.2874 - val_accuracy: 0.7557 - lr: 5.0000e-06
Runtime of the model 2's evaluation is 2025 seconds.
```

In [15]:

```
# Loading trained saved model 2 from file

model2 = keras.models.load_model(".mdl2_MobNet.hdf5", custom_objects = {'myLos
s': myLoss, 'accuracy': accuracy})
```

In [16]:

```
# Function to predict label for a given image using Model 2

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...
    test = tf.cast(image, dtype='float32')/255.0
    test = tf.reshape(test, [1,32,32,3])
    return np.argmax(model2(test)[:,10:], axis = 1)[0]
```

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 [18]:
```
# [DO NOT MODIFY THIS CELL]
def evaluation(model, 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))
    return y_pred
```

In [21]:
```
# [DO NOT MODIFY THIS CELL]
# This is the code for evaluating the prediction performance on a testset

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_imgs)
```

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

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

In [23]:
```
%%time

# Getting predictions on test set using baseline model
baseline_preds = evaluation(baseline_model, test_imgs)
```

```
Wall time: 2.63 s
```

In [24]:
```
%%time

# Getting predictions on test set using model 1
model1_preds = evaluation(model_I, test_imgs)
```

```
Wall time: 6min 9s
```

In [25]:
```
%%time

# Getting predictions on test set using model 2
model2_preds = evaluation(model_II, test_imgs)
```

```
Wall time: 5min 59s
```

In [32]:
```
# Exporting results to csv file

import pandas as pd

label_predictions = pd.read_csv('../data/label_prediction.csv')

label_predictions.columns
```

Out[32]:
```
Index(['Index', 'Baseline', 'Model I', 'Model II'], dtype='object')
```

In [34]:
```
# label_predictions = pd.DataFrame({'Index':list(range(1,10001)),
#                                   'Baseline': baseline_preds,
#                                   'Model I': model1_preds,
#                                   'Model II': model2_preds})

label_predictions['Baseline'] = baseline_preds
label_predictions['Model I'] = model1_preds
label_predictions['Model II'] = model2_preds
```

In [35]:
```
label_predictions.head()
```

Out[35]:

	Index	Baseline	Model I	Model II
0	test00001	6	3	3
1	test00002	0	1	1
2	test00003	8	8	1
3	test00004	0	8	0
4	test00005	4	6	6

In [37]:
```
# Write to csv
label_predictions.to_csv('label_predictions.csv', index = False)
```

In [ ]: