In [1]:
```
# Import required packages
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
import time

import os
from keras.models import Model
from keras.optimizers import Adam
from keras.applications import vgg16, vgg19, resnet, inception_v3, MobileNetV3Small, m
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.layers import Dense, Dropout, Flatten
from keras.utils import to_categorical
from livelossplot.inputs.keras import PlotLossesCallback
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
from sklearn.metrics import confusion_matrix
from datetime import datetime as dt

from sklearn.model_selection import KFold

import warnings
warnings.filterwarnings("ignore")
```

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]:
```
# [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)
```

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 [6]:
```
# [DO NOT MODIFY THIS CELL]
def baseline_model(image):
    '''
    This is the baseline predictive model that takes in the image and returns a label
    '''
    feature1 = np.histogram(image[:,:,0],bins=bins)[0]
    feature2 = np.histogram(image[:,:,1],bins=bins)[0]
    feature3 = np.histogram(image[:,:,2],bins=bins)[0]
    feature = np.concatenate((feature1, feature2, feature3), axis=None).reshape(1,-1)
    return clf.predict(feature)
```

In [7]:
```
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)

print('Clean labels:')
print(' '.join('%5s' % classes[clean_labels[j]] for j in range(8)))
print('Predicted baseline labels:')
print(' '.join('%5s' % classes[int(baseline_model(imgs[j])[0])] for j in range(8)))
```

```
Clean labels:
 frog truck truck  deer   car   car  bird horse
Predicted baseline labels:
 frog  ship truck  frog  ship   cat  deer horse
```

2.2. Model I

- Loading images for pre-processing using the pre-trained inception V3 model. The dataset is split into train and test dataset using `train_test_split`

In [9]:
```
imgs_load = np.empty((n_img,75,75,3))
for i in range(n_img):
    img_fn = f'../data/images/{i+1:05d}.png'
    imgs_load[i,:,:,:]=cv2.resize(cv2.imread(img_fn),(75,75),interpolation=cv2.INTER_C

imgs_incep = inception_v3.preprocess_input(imgs_load)
```

In [7]:
```
# Split the data into training set (75%) and test set (25%)
imgs_incep_train, imgs_incep_test, noisy_labels_train, noisy_labels_test = train_test_
```

- We load the pre-trained model (Inception_v3) weights and include layers of fully connected network for training the model. Adam optimizer with `learning_rate=0.001` is used.

In [15]:
```
# Load inception_v3 model
def create_model_incep(input_shape, n_classes, optimizer, fine_tune):
    """
    Compiles a model integrated with inception_v3 pretrained layers

    input_shape: tuple - the shape of input images (width, height, channels)
    n_classes: int - number of classes for the output layer
    optimizer: string - instantiated optimizer to use for training. Defaults to 'RMSPr
    fine_tune: int - The number of pre-trained layers to unfreeze.
               If set to 0, all pretrained layers will freeze during training
    """
    # Pretrained convolutional layers are loaded using the Imagenet weights.
    # Include_top is set to False, in order to exclude the model's fully-connected lay
```

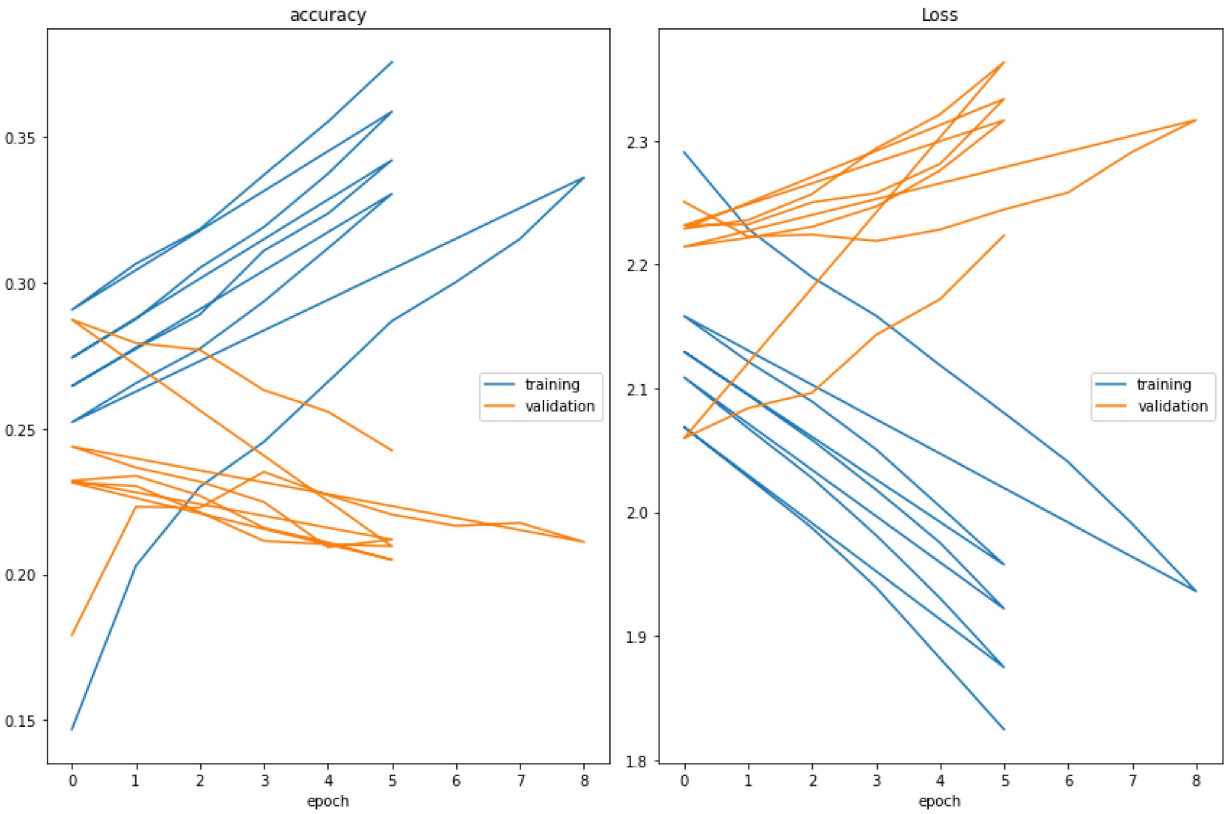
```
    conv_base = inception_v3.InceptionV3(include_top=False,
                            weights='imagenet',input_shape=input_shape)

    # Defines how many layers to freeze during training.
    # Layers in the convolutional base are switched from trainable to non-trainable
    # depending on the size of the fine-tuning parameter.
    # If the arg fine_tune is set to 0, all pre-trained layers will be frozen and left
    # Otherwise, the last n layers will be made available for training.
    if fine_tune > 0:
        for layer in conv_base.layers[:-fine_tune]:
            layer.trainable = False
    else:
        for layer in conv_base.layers:
            layer.trainable = False

    # Create a new 'top' of the model (i.e. fully-connected layers)
    # by grabbing the conv_base outputs and flattening them.
    # This is 'bootstrapping' a new top_model onto the pretrained layers.
    top_model = conv_base.output
    top_model = Flatten()(top_model)
    top_model = Dense(n_classes*8, activation='relu')(top_model)
    top_model = Dense(n_classes*4, activation='relu')(top_model)
    top_model = Dropout(0.2)(top_model)
    output_layer = Dense(n_classes, activation='softmax')(top_model)

    # Group the convolutional base and new fully-connected layers into a Model object.
    model = Model(inputs=conv_base.input, outputs=output_layer)

    # Compiles the model for training.
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model


input_shape = (75, 75,3)
n_classes=10
optim = Adam(learning_rate=0.001)
incep_model = create_model_incep(input_shape, n_classes, optim, fine_tune = 0)
```

- K-fold cross validation technique is used for evaluation. `Earlystopping` is used to optimize training time of the model.

In [12]:

```
# Train the incep model
# ModelCheckpoint callback is used to save a model or weights (in a checkpoint file) d
# EarlyStopping stops training when a monitored metric has stopped improving.
# Batch size defines the number of samples to work through before updating the interna
# The number of epochs defines the number times that the learning algorithm will work
start = time.time()

incep_checkpoint = ModelCheckpoint(filepath='../output/incep.weights.best.hdf5', save_
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True,
plot_loss = PlotLossesCallback()

n_split=5

for train_index,test_index in KFold(n_split).split(imgs_incep_train):
```

```
  x_train,x_test=imgs_incep_train[train_index],imgs_incep_train[test_index]
  y_train,y_test=noisy_labels_train[train_index],noisy_labels_train[test_index]

  incep_fit = incep_model.fit(x_train,
                              tf.one_hot(y_train,10),
                              batch_size=128, # Mini-batch gradient descent
                              epochs=20,# 20
                              validation_split=0.2,
                              callbacks=[incep_checkpoint, early_stop, plot_loss],
                              verbose=1)

  print('Model evaluation ',incep_model.evaluate(x_test,tf.one_hot(y_test,10)))

print('Model Training Time:', round((time.time() - start)/60, 2), 'mins')
```




```
accuracy
        training                (min:    0.147, max:    0.376, cur:    0.376)
        validation              (min:    0.179, max:    0.287, cur:    0.243)
Loss
        training                (min:    1.825, max:    2.291, cur:    1.825)
        validation              (min:    2.060, max:    2.363, cur:    2.224)
188/188 [==============================] - 217s 1s/step - loss: 1.8247 - accuracy: 0.
3756 - val_loss: 2.2236 - val_accuracy: 0.2425
235/235 [==============================] - 60s 252ms/step - loss: 2.1874 - accuracy:
0.2505
Model evaluation  [2.1873598098754883, 0.2505333423614502]
Model Training Time: 141.55 mins
```

In [13]:

```
# Prediction & Accuracy
start = time.time()
incep_model.load_weights('../output/incep.weights.best.hdf5')
incep_preds = incep_model.predict(imgs_incep_test)
incep_pred_classes = np.argmax(incep_preds, axis=1)
print('Model Test Time:', round((time.time() - start)/60, 2), 'mins')
```

```
incep_acc = accuracy_score(noisy_labels_test, incep_pred_classes)
print("incep Model Accuracy: {:.2f}%".format(incep_acc * 100))
```

```
391/391 [==============================] - 147s 346ms/step
Model Test Time: 2.76 mins
incep Model Accuracy: 22.76%
```

In [14]:
```
# save model
incep_model.save('saved_models/incep/incep_model')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_comp
iled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit
_compiled_convolution_op while saving (showing 5 of 94). These functions will not be
directly callable after loading.
INFO:tensorflow:Assets written to: saved_models/incep/incep_model/assets
INFO:tensorflow:Assets written to: saved_models/incep/incep_model/assets
```

In [13]:
```
def model_I(image):
    '''
    This is the MobileNetV3 predictive model that takes in the image and returns a lab
    '''
    #incep_model.load_weights('incep.weights.best.hdf5')
    incep_preds = incep_model.predict([np.expand_dims(image, axis=0)],verbose=0);
    incep_pred_classes = np.argmax(incep_preds, axis=1)
    return int(incep_pred_classes[0])
```

In [14]:
```
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)

print('Clean labels:')
print(' '.join('%5s' % classes[clean_labels[j]] for j in range(8)))
print('Predicted baseline labels:')
print(' '.join('%5s' % classes[model_I(imgs_incep[j])] for j in range(8)))
```

```
Clean labels:
 frog truck truck  deer   car   car  bird horse
Predicted baseline labels:
  dog truck plane  bird   car   car  bird horse
```

2.3. Model II

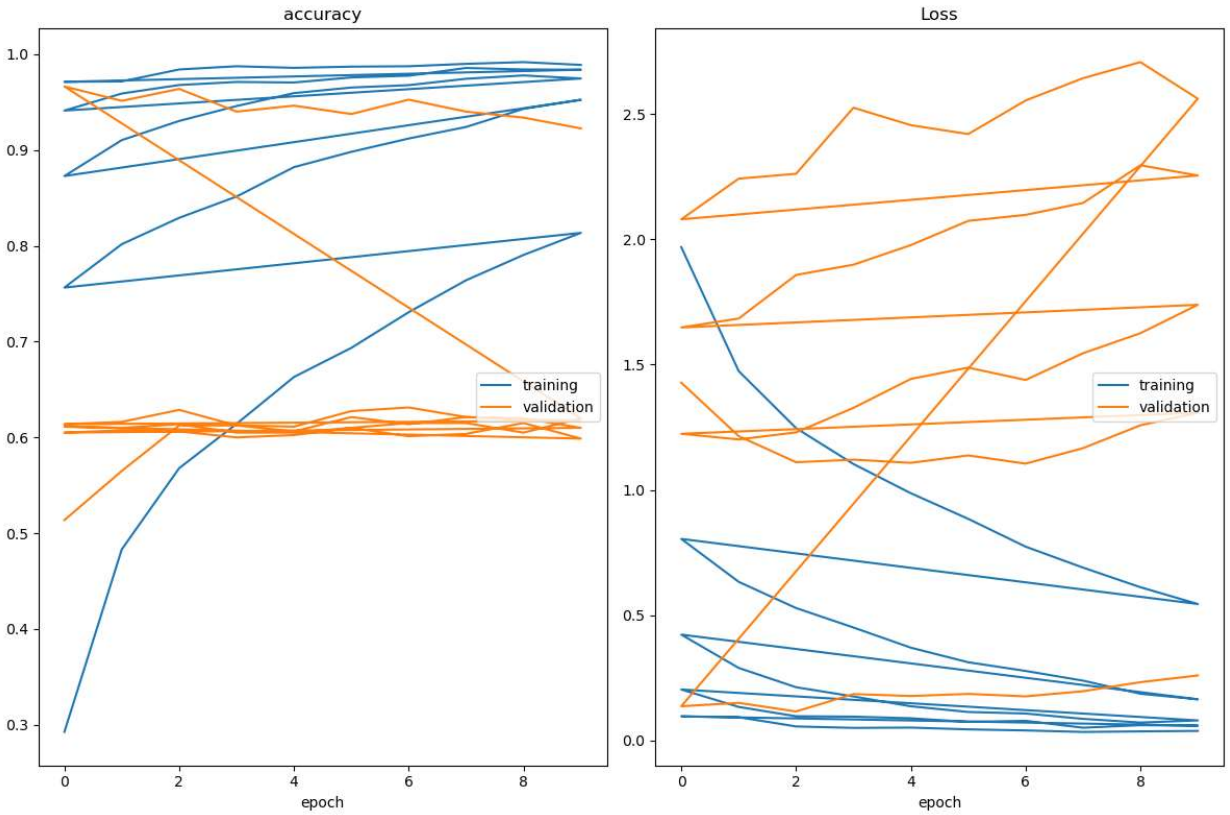
- Model II Motivation: Since the labels for 40,000 images are noisy. We use the inception_v3 model weights and pass it through a fully connected network for training. This label-cleaning model is used to predict labels for the 40,000 images. We use the `new_labels` generated to train Model-I again.

In [10]:
```
incep_ii = inception_v3.InceptionV3(include_top=False,
                                    weights='imagenet',input_shape=(75,75,3))

incep_ii.load_weights('../output/incep.weights.best.hdf5',by_name=True)

incep_ii_train_output = incep_ii.predict(imgs_incep)
```

```
1563/1563 [==============================] - 120s 76ms/step
```

In [12]:
```
x_train1 = incep_ii_train_output[:n_clean_noisy]
x_train2 = noisy_labels[:n_clean_noisy]
y_train = tf.one_hot(clean_labels,10)

# write your code here...
#input1 = tf.keras.layers.Input(shape=(1,1,2048))
#flatten1 = tf.keras.layers.Flatten()(input1)
#input2 = tf.keras.layers.Input(shape=(1,))
#merged = tf.keras.layers.Concatenate(axis=1)([flatten1,input2])
#dense1 = tf.keras.layers.Dense(128,activation='relu')(merged)
#dense2 = tf.keras.layers.Dense(64,activation='relu')(dense1)
#dropout1 = tf.keras.layers.Dropout(0.2)(dense2)
#dense4 = tf.keras.layers.Dense(10, activation='softmax')(dropout1)
#output1 = tf.keras.layers.add([dense4,input2])
#label_cleaning_model = tf.keras.models.Model([input1, input2], output1)

label_cleaning_input = np.concatenate((incep_ii_train_output.reshape(n_img, 2048), noi
```

```
label_cleaning_model = tf.keras.Sequential([
     tf.keras.layers.Dense(128, activation='relu'),
     tf.keras.layers.Dense(64, activation = 'relu'),
     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Dense(10, activation='softmax')
])

label_cleaning_model.compile(optimizer='adam',
                             loss='categorical_crossentropy',
                             metrics=['accuracy'])

#label_cleaning_model.fit((x_train1,x_train2), y_train, batch_size=32, epochs=10, vali

incep_checkpoint2 = ModelCheckpoint(filepath='lcni.weights.best.hdf5', save_best_only=
early_stop2 = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True, mod
plot_loss2 = PlotLossesCallback()

n_split=5

y_noisy1 = noisy_labels[:n_clean_noisy]

for train_index,test_index in KFold(n_split).split(label_cleaning_input[:n_clean_noisy
  x_train,x_test=label_cleaning_input[train_index],label_cleaning_input[test_index]
  #x_train2,x_test2=y_noisy1[train_index],y_noisy1[test_index]
  y_train,y_test=clean_labels[train_index],clean_labels[test_index]

  lcn_fit = label_cleaning_model.fit(x_train,
                             tf.one_hot(y_train,10),
                             batch_size=64, # Mini-batch gradient descent
                             epochs=10,
                             validation_split=0.1,
                             callbacks=[incep_checkpoint2, early_stop2, plot_loss2],
                             verbose=1)


  print('Model evaluation ',label_cleaning_model.evaluate(x_test,tf.one_hot(y_test,10)
```



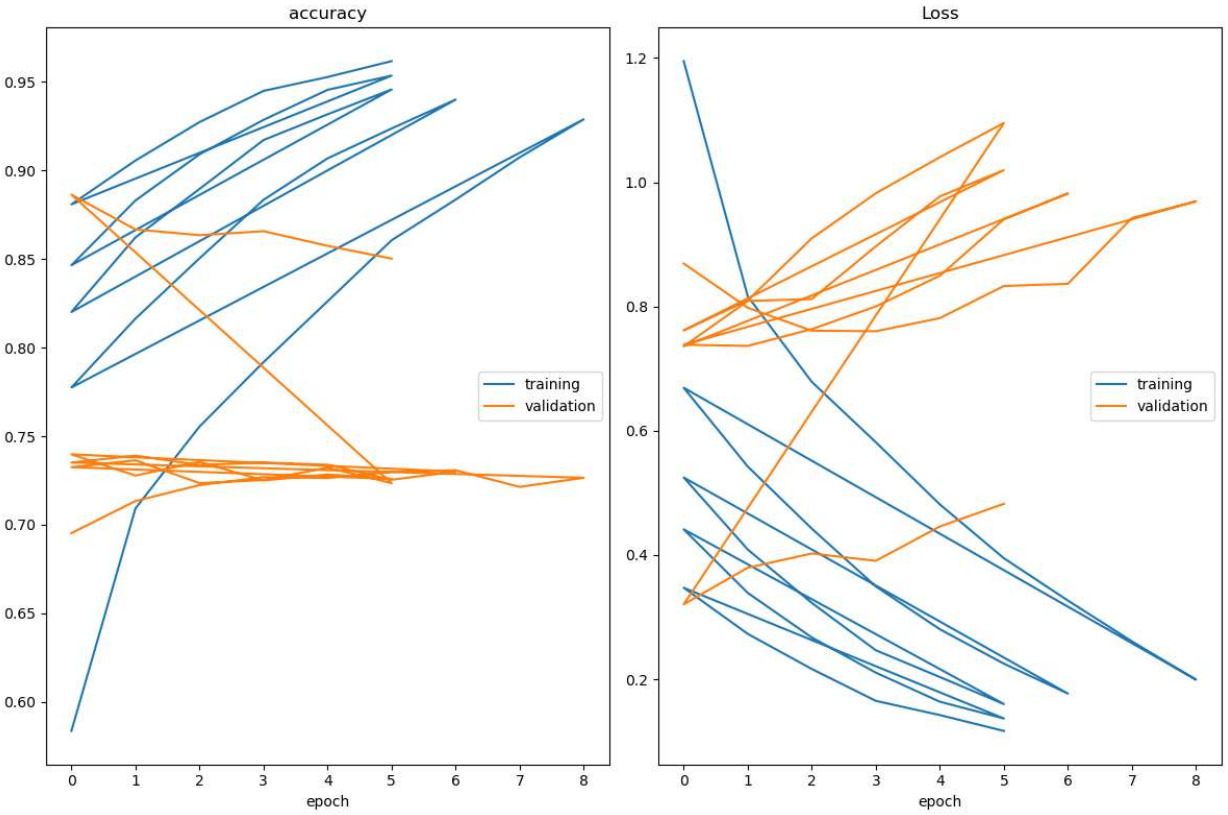

```
accuracy
        training                (min:    0.293, max:    0.992, cur:    0.989)
        validation              (min:    0.514, max:    0.966, cur:    0.923)
Loss
        training                (min:    0.034, max:    1.968, cur:    0.038)
        validation              (min:    0.116, max:    2.707, cur:    0.260)
113/113 [==============================] - 0s 4ms/step - loss: 0.0383 - accuracy: 0.9
887 - val_loss: 0.2596 - val_accuracy: 0.9225
63/63 [==============================] - 0s 897us/step - loss: 1.2492 - accuracy: 0.8
350
Model evaluation  [1.2492079734802246, 0.8349999785423279]
```

In [18]:
```
# Clean labels for all
predicted_clean_labels = label_cleaning_model.predict(label_cleaning_input[n_clean_noi
predicted_clean_labels = predicted_clean_labels.argmax(axis=-1)
new_labels = np.append(clean_labels, predicted_clean_labels)
new_labels_oh = tf.one_hot(new_labels,depth=10)

img_class_model = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

img_class_model.compile(optimizer='adam',
                            loss='categorical_crossentropy',
                            metrics=['accuracy'])
```

```
1250/1250 [==============================] - 1s 877us/step
```

In [19]:
```
start = time.time()
incep_checkpoint1 = ModelCheckpoint(filepath='incepwlcn.weights.best.hdf5', save_best_
early_stop1 = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True,
```

```
plot_loss1 = PlotLossesCallback()

for train_index,test_index in KFold(n_split).split(incep_ii_train_output):
  x_train,x_test=incep_ii_train_output[train_index],incep_ii_train_output[test_index]
  y_train,y_test=new_labels[train_index],new_labels[test_index]

  img_class_fit = img_class_model.fit(x_train,
                           tf.one_hot(y_train,10),
                           batch_size=128, # 64
                           epochs=20,
                           validation_split=0.2,
                           callbacks=[incep_checkpoint1,early_stop1, plot_loss1],
                           verbose=1)
  print('Model evaluation ',img_class_model.evaluate(x_test,tf.one_hot(y_test,10)))

print('Model Training Time:', round((time.time() - start)/60, 2), 'mins')
```




```
accuracy
        training                 (min:    0.584, max:    0.962, cur:    0.962)
        validation               (min:    0.695, max:    0.886, cur:    0.850)
Loss
        training                 (min:    0.117, max:    1.195, cur:    0.117)
        validation               (min:    0.321, max:    1.095, cur:    0.482)
250/250 [==============================] - 1s 5ms/step - loss: 0.1169 - accuracy: 0.9
618 - val_loss: 0.4825 - val_accuracy: 0.8503
313/313 [==============================] - 1s 2ms/step - loss: 0.7200 - accuracy: 0.7
579
Model evaluation  [0.720008134841919, 0.7578999996185303]
Model Training Time: 0.88 mins
```

In [23]:

```
# Prediction & Accuracy
start = time.time()
#incep_ii.load_weights('incep.weights.best.hdf5',by_name=True)
#incep_ii_preds = incep_ii.predict(x_test)
img_class_model.load_weights('../output/incepwlcn.weights.best.hdf5')
```

```
incepwlcn_ii_preds = img_class_model.predict(x_test)
incep_ii_pred_classes = np.argmax(incepwlcn_ii_preds, axis=1)
print('Model Training Time:', round((time.time() - start)/60, 2), 'mins')
incep1_acc = accuracy_score(y_test, incep_ii_pred_classes)
print("Inception Model Accuracy: {:.2f}%".format(incep1_acc * 100))
```

```
313/313 [==============================] - 0s 1ms/step
Model Training Time: 0.01 mins
Inception Model Accuracy: 67.35%
```

In [24]:
```
# save model
incep_ii.save('../output/saved_models/incep/incep_ii')
img_class_model.save('../output/saved_models/incep/img_class_model')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of 94). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: ../output/saved_models/incep/incep_ii\assets
INFO:tensorflow:Assets written to: ../output/saved_models/incep/incep_ii\assets
INFO:tensorflow:Assets written to: ../output/saved_models/incep/img_class_model\assets
INFO:tensorflow:Assets written to: ../output/saved_models/incep/img_class_model\assets
```

In [21]:
```
def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and returns
    '''
    # write your code here...
    #incep_ii.load_weights('../output/incep.weights.best.hdf5',by_name=True)
    incep_preds = incep_ii.predict([np.expand_dims(image, axis=0)],verbose=0);
    #img_class_model.load_weights('../output/incepwlcn.weights.best.hdf5')
    incep_pred_classes = np.argmax(img_class_model.predict(incep_preds,verbose=0), axi
    return int(incep_pred_classes[0])
```

In [32]:
```
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)

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

```
print('Predicted baseline labels:')
print(' '.join('%5s' % classes[model_II(imgs_incep[j])] for j in range(8)))
```

```
Clean labels:
 frog truck truck  deer   car   car  bird horse
Predicted baseline labels:
 frog truck truck  deer   car   car  bird horse
```

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 [33]:
```
n_test = 10000
test_labels = np.genfromtxt('../data/test_labels.csv', delimiter=',', dtype="int8")
test_imgs = np.empty((n_test,32,32,3))
test_imgs_cnn = np.empty((n_test,75,75,3))
for i in range(n_test):
    img_fn = f'../data/test_images/{i+1:05d}.png'
    test_imgs[i,:,:,:]=cv2.cvtColor(cv2.imread(img_fn),cv2.COLOR_BGR2RGB)
    test_imgs_cnn[i,:,:,:]=cv2.resize(cv2.imread(img_fn),(75,75),interpolation=cv2.INT

test_imgs_cnn = inception_v3.preprocess_input(test_imgs_cnn)

bl_out = np.zeros(n_test)
m1_out = np.zeros(n_test)
m2_out = np.zeros(n_test)
```

In [26]:
```
def model_I(image):
    '''
    This is the MobileNetV3 predictive model that takes in the image and returns a lab
```

```
    '''
    incep_preds = incep_model.predict([np.expand_dims(image, axis=0)],verbose=0);
    incep_pred_classes = np.argmax(incep_preds, axis=1)
    return int(incep_pred_classes[0])
```

In [27]:
```
def model_II(image):
    '''
    This function should takes in the image of dimension 32*32*3 as input and returns
    '''
    # write your code here...
    incep_preds = incep_ii.predict([np.expand_dims(image, axis=0)],verbose=0);
    incep_pred_classes = np.argmax(img_class_model.predict(incep_preds,verbose=0), axi
    return int(incep_pred_classes[0])
```

In [28]:
```
incep_model = tf.keras.models.load_model('../output/saved_models/incep/incep_model')
img_class_model = tf.keras.models.load_model('../output/saved_models/incep/img_class_m
incep_ii = inception_v3.InceptionV3(include_top=False,
                            weights='imagenet',input_shape=(75,75,3))
incep_ii.load_weights('../output/incep.weights.best.hdf5',by_name=True)
```

In [40]:
```
# [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))
    return [item for sublist in y_pred for item in sublist]
```

In [41]:
```
# [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
start = time.time()
bl_out = evaluation(baseline_model, test_labels, test_imgs)
print('Model Evaluation Time:', round((time.time() - start)/60, 2), 'mins')
```

```
              precision    recall  f1-score   support

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

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

Model Evaluation Time: 0.04 mins
```

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 [22]:
```
# [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
start = time.time()
m1_out = evaluation(model_I, test_labels, test_imgs_cnn)
print('Model Evaluation Time:', round((time.time() - start)/60, 2), 'mins')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [22], in <cell line: 7>()
      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 #incep_model.load_weights('incep.weights.best.hdf5')
      6 start = time.time()
----> 7 m1_out = evaluation(model_I, test_labels, test_imgs_cnn)
      8 print('Model Evaluation Time:', round((time.time() - start)/60, 2), 'mins')

Input In [20], in evaluation(model, test_labels, test_imgs)
      4 y_pred = []
      5 for image in test_imgs:
----> 6     y_pred.append(model(image))
      7 print(classification_report(y_true, y_pred))
      8 return y_pred

Input In [11], in model_I(image)
      2 '''
      3 This is the MobileNetV3 predictive model that takes in the image and returns
 a label prediction
      4 '''
      5 #incep_model.load_weights('incep.weights.best.hdf5')
----> 6 incep_preds = incep_model.predict([np.expand_dims(image, axis=0)],verbose=0);
      7 incep_pred_classes = np.argmax(incep_preds, axis=1)
      8 return int(incep_pred_classes[0])

File ~\anaconda3\lib\site-packages\keras\utils\traceback_utils.py:65, in filter_trace
back.<locals>.error_handler(*args, **kwargs)
     63 filtered_tb = None
     64 try:
---> 65     return fn(*args, **kwargs)
     66 except Exception as e:
     67     filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\lib\site-packages\keras\engine\training.py:2220, in Model.predict(se
lf, x, batch_size, verbose, steps, callbacks, max_queue_size, workers, use_multiproce
ssing)
   2211     except ValueError:
   2212         warnings.warn(
   2213             "Using Model.predict with MultiWorkerMirroredStrategy "
   2214             "or TPUStrategy and AutoShardPolicy.FILE might lead to "
   (...)
   2217             stacklevel=2,
   2218         )
-> 2220 data_handler = data_adapter.get_data_handler(
   2221     x=x,
   2222     batch_size=batch_size,
   2223     steps_per_epoch=steps,
   2224     initial_epoch=0,
   2225     epochs=1,
   2226     max_queue_size=max_queue_size,
   2227     workers=workers,
   2228     use_multiprocessing=use_multiprocessing,
   2229     model=self,
   2230     steps_per_execution=self._steps_per_execution,
   2231 )
   2233 # Container that configures and calls `tf.keras.Callback`s.
```

```
   2234 if not isinstance(callbacks, callbacks_module.CallbackList):

File ~\anaconda3\lib\site-packages\keras\engine\data_adapter.py:1582, in get_data_han
dler(*args, **kwargs)
   1580 if getattr(kwargs["model"], "_cluster_coordinator", None):
   1581     return _ClusterCoordinatorDataHandler(*args, **kwargs)
-> 1582 return DataHandler(*args, **kwargs)

File ~\anaconda3\lib\site-packages\keras\engine\data_adapter.py:1262, in DataHandler.
__init__(self, x, y, sample_weight, batch_size, steps_per_epoch, initial_epoch, epoch
s, shuffle, class_weight, max_queue_size, workers, use_multiprocessing, model, steps_
per_execution, distribute)
   1259     self._steps_per_execution = steps_per_execution
   1261 adapter_cls = select_data_adapter(x, y)
-> 1262 self._adapter = adapter_cls(
   1263     x,
   1264     y,
   1265     batch_size=batch_size,
   1266     steps=steps_per_epoch,
   1267     epochs=epochs - initial_epoch,
   1268     sample_weights=sample_weight,
   1269     shuffle=shuffle,
   1270     max_queue_size=max_queue_size,
   1271     workers=workers,
   1272     use_multiprocessing=use_multiprocessing,
   1273     distribution_strategy=tf.distribute.get_strategy(),
   1274     model=model,
   1275 )
   1277 strategy = tf.distribute.get_strategy()
   1279 self._current_step = 0

File ~\anaconda3\lib\site-packages\keras\engine\data_adapter.py:347, in TensorLikeDat
aAdapter.__init__(self, x, y, sample_weights, sample_weight_modes, batch_size, epoch
s, steps, shuffle, **kwargs)
    344         flat_dataset = flat_dataset.shuffle(1024).repeat(epochs)
    345     return flat_dataset
--> 347 indices_dataset = indices_dataset.flat_map(slice_batch_indices)
    349 dataset = self.slice_inputs(indices_dataset, inputs)
    351 if shuffle == "batch":

File ~\anaconda3\lib\site-packages\tensorflow\python\data\ops\dataset_ops.py:2245, in
DatasetV2.flat_map(self, map_func, name)
   2212 def flat_map(self, map_func, name=None):
   2213   """Maps `map_func` across this dataset and flattens the result.
   2214
   2215   The type signature is:
   (...)
   2243     Dataset: A `Dataset`.
   2244   """
-> 2245   return FlatMapDataset(self, map_func, name=name)

File ~\anaconda3\lib\site-packages\tensorflow\python\data\ops\dataset_ops.py:5484, in
FlatMapDataset.__init__(self, input_dataset, map_func, name)
   5482 """See `Dataset.flat_map()` for details."""
   5483 self._input_dataset = input_dataset
-> 5484 self._map_func = structured_function.StructuredFunctionWrapper(
   5485     map_func, self._transformation_name(), dataset=input_dataset)
   5486 if not isinstance(self._map_func.output_structure, DatasetSpec):
   5487   raise TypeError(
   5488       "The `map_func` argument must return a `Dataset` object. Got "
```

```
   5489       f"{_get_type(self._map_func.output_structure)!r}.")

File ~\anaconda3\lib\site-packages\tensorflow\python\data\ops\structured_function.py:
271, in StructuredFunctionWrapper.__init__(self, func, transformation_name, dataset,
 input_classes, input_shapes, input_types, input_structure, add_to_graph, use_legacy_
function, defun_kwargs)
    264       warnings.warn(
    265           "Even though the `tf.config.experimental_run_functions_eagerly` "
    266           "option is set, this option does not apply to tf.data functions. "
    267           "To force eager execution of tf.data functions, please use "
    268           "`tf.data.experimental.enable_debug_mode()`.")
    269     fn_factory = trace_tf_function(defun_kwargs)
--> 271 self._function = fn_factory()
    272 # There is no graph to add in eager mode.
    273 add_to_graph &= not context.executing_eagerly()

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\function.py:2610, in Funct
ion.get_concrete_function(self, *args, **kwargs)
   2601 def get_concrete_function(self, *args, **kwargs):
   2602   """Returns a `ConcreteFunction` specialized to inputs and execution contex
t.
   2603
   2604   Args:
   (...)
   2608       or `tf.Tensor` or `tf.TensorSpec`.
   2609   """
-> 2610   graph_function = self._get_concrete_function_garbage_collected(
   2611       *args, **kwargs)
   2612   graph_function._garbage_collector.release()  # pylint: disable=protected-ac
cess
   2613   return graph_function

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\function.py:2576, in Funct
ion._get_concrete_function_garbage_collected(self, *args, **kwargs)
   2574   args, kwargs = None, None
   2575 with self._lock:
-> 2576   graph_function, _ = self._maybe_define_function(args, kwargs)
   2577   seen_names = set()
   2578   captured = object_identity.ObjectIdentitySet(
   2579       graph_function.graph.internal_captures)

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\function.py:2760, in Funct
ion._maybe_define_function(self, args, kwargs)
   2758   # Only get placeholders for arguments, not captures
   2759   args, kwargs = placeholder_dict["args"]
-> 2760 graph_function = self._create_graph_function(args, kwargs)
   2762 graph_capture_container = graph_function.graph._capture_func_lib  # pylint: d
isable=protected-access
   2763 # Maintain the list of all captures

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\function.py:2670, in Funct
ion._create_graph_function(self, args, kwargs)
   2665 missing_arg_names = [
   2666     "%s_%d" % (arg, i) for i, arg in enumerate(missing_arg_names)
   2667 ]
   2668 arg_names = base_arg_names + missing_arg_names
   2669 graph_function = ConcreteFunction(
-> 2670     func_graph_module.func_graph_from_py_func(
   2671         self._name,
   2672         self._python_function,
```

```
   2673         args,
   2674         kwargs,
   2675         self.input_signature,
   2676         autograph=self._autograph,
   2677         autograph_options=self._autograph_options,
   2678         arg_names=arg_names,
   2679         capture_by_value=self._capture_by_value),
   2680     self._function_attributes,
   2681     spec=self.function_spec,
   2682     # Tell the ConcreteFunction to clean up its graph once it goes out of
   2683     # scope. This is not the default behavior since it gets used in some
   2684     # places (like Keras) where the FuncGraph lives longer than the
   2685     # ConcreteFunction.
   2686     shared_func_graph=False)
   2687 return graph_function

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\func_graph.py:1247, in func_graph_from_py_func(name, python_func, args, kwargs, signature, func_graph, autograph, autograph_options, add_control_dependencies, arg_names, op_return_value, collections, capture_by_value, acd_record_initial_resource_uses)
   1244 else:
   1245   _, original_func = tf_decorator.unwrap(python_func)
-> 1247 func_outputs = python_func(*func_args, **func_kwargs)
   1249 # invariant: `func_outputs` contains only Tensors, CompositeTensors,
   1250 # TensorArrays and `None`s.
   1251 func_outputs = nest.map_structure(
   1252     convert, func_outputs, expand_composites=True)

File ~\anaconda3\lib\site-packages\tensorflow\python\data\ops\structured_function.py:248, in StructuredFunctionWrapper.__init__.<locals>.trace_tf_function.<locals>.wrapped_fn(*args)
    242 @eager_function.defun_with_attributes(
    243     input_signature=structure.get_flat_tensor_specs(
    244         self._input_structure),
    245     autograph=False,
    246     attributes=defun_kwargs)
    247 def wrapped_fn(*args):  # pylint: disable=missing-docstring
--> 248   ret = wrapper_helper(*args)
    249   ret = structure.to_tensor_list(self._output_structure, ret)
    250   return [ops.convert_to_tensor(t) for t in ret]

File ~\anaconda3\lib\site-packages\tensorflow\python\data\ops\structured_function.py:177, in StructuredFunctionWrapper.__init__.<locals>.wrapper_helper(*args)
    175 if not _should_unpack(nested_args):
    176   nested_args = (nested_args,)
--> 177 ret = autograph.tf_convert(self._func, ag_ctx)(*nested_args)
    178 if _should_pack(ret):
    179   ret = tuple(ret)

File ~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py:689, in convert.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    687 try:
    688   with conversion_ctx:
--> 689     return converted_call(f, args, kwargs, options=options)
    690 except Exception as e:  # pylint:disable=broad-except
    691   if hasattr(e, 'ag_error_metadata'):

File ~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py:377, in converted_call(f, args, kwargs, caller_fn_scope, options)
    374   return _call_unconverted(f, args, kwargs, options)
```

```
    376 if not options.user_requested and conversion.is_allowlisted(f):
--> 377   return _call_unconverted(f, args, kwargs, options)
    379 # internal_convert_user_code is for example turned off when issuing a dynamic
    380 # call conversion from generated code while in nonrecursive mode. In that
    381 # case we evidently don't want to recurse, but we still have to convert
    382 # things like builtins.
    383 if not options.internal_convert_user_code:

File ~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py:458, in _c
all_unconverted(f, args, kwargs, options, update_cache)
    455   return f.__self__.call(args, kwargs)
    457 if kwargs is not None:
--> 458   return f(*args, **kwargs)
    459 return f(*args)

File ~\anaconda3\lib\site-packages\keras\engine\data_adapter.py:336, in TensorLikeDat
aAdapter.__init__.<locals>.slice_batch_indices(indices)
    333 flat_dataset = tf.data.Dataset.from_tensor_slices(first_k_indices)
    334 if self._partial_batch_size:
    335     index_remainder = tf.data.Dataset.from_tensors(
--> 336         tf.slice(
    337             indices, [num_in_full_batch], [self._partial_batch_size]
    338         )
    339     )
    340     flat_dataset = flat_dataset.concatenate(index_remainder)
    342 if shuffle == "batch":
    343     # 1024 is a magic constant that has not been properly evaluated

File ~\anaconda3\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in
filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\lib\site-packages\tensorflow\python\util\dispatch.py:1176, in add_di
spatch_support.<locals>.decorator.<locals>.op_dispatch_handler(*args, **kwargs)
   1174 # Fallback dispatch system (dispatch v1):
   1175 try:
-> 1176   return dispatch_target(*args, **kwargs)
   1177 except (TypeError, ValueError):
   1178   # Note: convert_to_eager_tensor currently raises a ValueError, not a
   1179   # TypeError, when given unexpected types.  So we need to catch both.
   1180   result = dispatch(op_dispatch_handler, args, kwargs)

File ~\anaconda3\lib\site-packages\tensorflow\python\ops\array_ops.py:1163, in slice
(input_, begin, size, name)
   1111 @tf_export("slice")
   1112 @dispatch.add_dispatch_support
   1113 def slice(input_, begin, size, name=None):
   1114   # pylint: disable=redefined-builtin
   1115   """Extracts a slice from a tensor.
   1116
   1117   See also `tf.strided_slice`.
   (...)
   1161     A `Tensor` the same type as `input_`.
   1162   """
-> 1163   return gen_array_ops._slice(input_, begin, size, name=name)
```

```
File ~\anaconda3\lib\site-packages\tensorflow\python\ops\gen_array_ops.py:9588, in _s
lice(input, begin, size, name)
   9586     pass  # Add nodes to the TensorFlow graph.
   9587 # Add nodes to the TensorFlow graph.
-> 9588   _, _, _op, _outputs = _op_def_library._apply_op_helper(
   9589       "Slice", input=input, begin=begin, size=size, name=name)
   9590 _result = _outputs[:]
   9591 if _execute.must_record_gradient():

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:779,
in _apply_op_helper(op_type_name, name, **keywords)
    777 with g.as_default(), ops.name_scope(name) as scope:
    778   if fallback:
--> 779     _ExtractInputsAndAttrs(op_type_name, op_def, allowed_list_attr_map,
    780                            keywords, default_type_attr_map, attrs, inputs,
    781                            input_types)
    782     _ExtractRemainingAttrs(op_type_name, op_def, keywords,
    783                            default_type_attr_map, attrs)
    784     _ExtractAttrProto(op_type_name, op_def, attrs, attr_protos)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:552,
in _ExtractInputsAndAttrs(op_type_name, op_def, allowed_list_attr_map, keywords, defa
ult_type_attr_map, attrs, inputs, input_types)
    546       values = ops.convert_to_tensor(
    547           values,
    548           name=input_arg.name,
    549           as_ref=input_arg.is_ref,
    550           preferred_dtype=default_dtype)
    551   else:
--> 552     values = ops.convert_to_tensor(
    553         values,
    554         name=input_arg.name,
    555         dtype=dtype,
    556         as_ref=input_arg.is_ref,
    557         preferred_dtype=default_dtype)
    558 except TypeError as err:
    559   if dtype is None:

File ~\anaconda3\lib\site-packages\tensorflow\python\profiler\trace.py:183, in trace_
wrapper.<locals>.inner_wrapper.<locals>.wrapped(*args, **kwargs)
    181   with Trace(trace_name, **trace_kwargs):
    182     return func(*args, **kwargs)
--> 183 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py:1638, in conver
t_to_tensor(value, dtype, name, as_ref, preferred_dtype, dtype_hint, ctx, accepted_re
sult_types)
   1629       raise RuntimeError(
   1630           _add_error_prefix(
   1631               f"Conversion function {conversion_func!r} for type "
   (...)
   1634               f"actual = {ret.dtype.base_dtype.name}",
   1635               name=name))
   1637 if ret is None:
-> 1638   ret = conversion_func(value, dtype=dtype, name=name, as_ref=as_ref)
   1640 if ret is NotImplemented:
   1641   continue

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\constant_op.py:343, in
_constant_tensor_conversion_function(v, dtype, name, as_ref)
```

```
    340 def _constant_tensor_conversion_function(v, dtype=None, name=None,
    341                                          as_ref=False):
    342   _ = as_ref
--> 343   return constant(v, dtype=dtype, name=name)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\constant_op.py:267, in
constant(value, dtype, shape, name)
    170 @tf_export("constant", v1=[])
    171 def constant(value, dtype=None, shape=None, name="Const"):
    172   """Creates a constant tensor from a tensor-like object.
    173
    174   Note: All eager `tf.Tensor` values are immutable (in contrast to
   (...)
    265     ValueError: if called on a symbolic tensor.
    266   """
--> 267   return  constant impl(value, dtype, shape, name, verify_shape=False,
    268                         allow_broadcast=True)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\constant_op.py:289, in
_constant_impl(value, dtype, shape, name, verify_shape, allow_broadcast)
    287 dtype_value = attr_value_pb2.AttrValue(type=tensor_value.tensor.dtype)
    288 attrs = {"value": tensor_value, "dtype": dtype_value}
--> 289 const tensor = g. create op internal(  # pylint: disable=protected-access
    290     "Const", [], [dtype_value.type], attrs=attrs, name=name).outputs[0]
    292 if op_callbacks.should_invoke_op_callbacks():
    293   # TODO(b/147670703): Once the special-op creation code paths
    294   # are unified. Remove this `if` block.
    295   callback_outputs = op_callbacks.invoke_op_callbacks(
    296       "Const", tuple(), attrs, (const_tensor,), op_name=name, graph=g)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\func_graph.py:735, in
FuncGraph._create_op_internal(self, op_type, inputs, dtypes, input_types, name, attr
s, op_def, compute_device)
    733   inp = self.capture(inp)
    734   captured_inputs.append(inp)
--> 735 return super(FuncGraph, self)._create_op_internal(  # pylint: disable=protect
ed-access
    736     op_type, captured_inputs, dtypes, input_types, name, attrs, op_def,
    737     compute_device)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py:3800, in Graph.
_create_op_internal(self, op_type, inputs, dtypes, input_types, name, attrs, op_def,
 compute_device)
   3797 # _create_op_helper mutates the new Operation. `_mutation_lock` ensures a
   3798 # Session.run call cannot occur between creating and mutating the op.
   3799 with self._mutation_lock():
-> 3800   ret = Operation(
   3801       node def,
   3802       self,
   3803       inputs=inputs,
   3804       output types=dtypes,
   3805       control inputs=control inputs,
   3806       input types=input types,
   3807       original_op=self._default_original_op,
   3808       op_def=op_def)
   3809   self._create_op_helper(ret, compute_device=compute_device)
   3810 return ret

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py:2108, in Operat
ion.__init__(***failed resolving arguments***)
```

```
   2105     control_input_ops.append(control_op)
   2107 # Initialize c_op from node_def and other inputs
-> 2108 c_op = _create_c_op(g, node_def, inputs, control_input_ops, op_def=op_def)
   2109 self._init_from_c_op(c_op=c_op, g=g)
   2111 self._original_op = original_op

File ~\anaconda3\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py:1966, in _create_c_op(graph, node_def, inputs, control_inputs, op_def, extract_traceback)
   1962   pywrap_tf_session.TF_SetAttrValueProto(op_desc, compat.as_str(name),
   1963                                          serialized)
   1965 try:
-> 1966   c_op = pywrap_tf_session.TF_FinishOperation(op_desc)
   1967 except errors.InvalidArgumentError as e:
   1968   # Convert to ValueError for backwards compatibility.
   1969   raise ValueError(e.message)

KeyboardInterrupt:
```

In [31]:

```
# [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
start = time.time()
m2_out = evaluation(model_II, test_labels, test_imgs_cnn)
print('Model Evaluation Time:', round((time.time() - start)/60, 2), 'mins')
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93      1005
           1       0.96      0.92      0.94       974
           2       0.91      0.89      0.90      1032
           3       0.92      0.83      0.88      1016
           4       0.86      0.91      0.88       999
           5       0.91      0.89      0.90       937
           6       0.89      0.96      0.92      1030
           7       0.92      0.90      0.91      1001
           8       0.98      0.90      0.94      1025
           9       0.86      0.96      0.91       981

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000

Model Evaluation Time: 22.45 mins
```

In [42]:

```
d = {'Baseline': bl_out, 'Model I': m1_out, 'Model II': m2_out }
label_prediction = pd.DataFrame(d)
label_prediction.to_csv('../output/label_prediction.csv',index = True)
```