Deep Learning Action Recognition
"Drinking–Cooking"

Mehdi Gorjian

1 Topic

This is a Deep Learning Action Recognition project which was trained on a GPU server to recognize "Drinking Activity".

2 Dataset

I have implemented different sources to prepare the dataset such as Kinetics, ActivityNet and Youtube. The files were videos with the approximate length of 10 sec.
I wrote a code to extract the frames from the videos, put them in a specific folder and rename them. I extracated 3907 images for drinking and 7831 images for cooking as the training dataset with Data Augmentation.

3 DNN Model

3.1 Architecture

I got the benefit of utilizing Transfer Learning - Fine Tuning from ResNet50 trained on the "ImageNet". Then applied a CNN to it in order to be trained on my custom dataset. For prediction part, I took advantage of the temporal nature of videos, specifically the assumption that subsequent frames in a video will have similar semantic contents. (I used 128 frames in the sequence.)

```
baseModel = ResNet50(include_top=False, weights="imagenet", input_tensor=Input(shape
    =(224, 224, 3)), input_shape=(244,244,3))
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten()(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(lb.classes_), activation="softmax")(headModel)
model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
    layer.trainable = False
```

3.2 Input Tensor Shape

Drinking Tensor Shape:
$X\_Train = (5320, 224, 224, 3)$
$X\_Test = (1773, 224, 224, 3)$
Cooking Tensor Shape:
$X\_Train = (5873, 224, 224, 3)$
$X\_Test = (1958, 224, 224, 3)$

3.3 Layer Structure

Drinking Layer Structure:

- AveragePooling2D Layer: Pool(7,7)
- Flatten Layer
- Dense Layer: 512 + Regularization l1
- Dropout Layer: 50%
- Softmax Prediction

Cooking Layer Structure:

- AveragePooling2D Layer: Pool(7,7)
- Flatten Layer
- Dense Layer: 512 + Regularization l1
- Dropout Layer: 50%
- Softmax Prediction

4 Hyper–parameters

4.1 List and Range of Hyper–parameters

Batch Size	32, 64 & 128
Epoch	10 - 50
Dropout	0.1 - 0.5

4.2 Optimal Hyper–parameters

Batch Size	32
Epoch	35
Dropout	0.5

4.3 Improvement and Performance

- Keras callbacks can help to fix bugs more quickly, and to build better models. They can help to visualize how the model's training is going, and can even help prevent overfitting by implementing checkpoints or customizing the learning rate on each iteration. A callback is a set of functions to be applied at given stages of the training procedure.
- I used a ModelCheckPoint callback to save the model after each epoch and to check the performance of the model. Before utilizing the checkpoint, I set the epochs to 50, however the checkpoint technique, illustrated that there is no improvement after epoch 10.
- I implemented EarlyStopping callback to adjust the optimized Epochs numbers.
- Adding (l1) Regularizaion to the FC layer. It had a good effects on improvement to the project.
- Adding ModelCheckpoint
- Adding RemoteMonitor
- Adding ReduceLROnPlateau
- Meanwhile, I got the benefits of TensorBoard visualization tool to track the training process in live.
- Reducing prediction flickerring
- Changing prediction to show "computing..." text when prediction is out of the boundary
- Revising the dataset: adding and removing some photos and balance them
- Bug fixing

5 Annotated Code

5.1 Training Code

```python
# USAGE
# TO RUN THE CODE: python train.py --data dataset --model output/activity.model --
    label_bin output/lb.pickle --plot output/fig_v1.png --epochs 50
# set the matplotlib backend so figures can be saved in the background
import matplotlib
matplotlib.use("Agg")
# import the necessary packages
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.pooling import AveragePooling2D
from keras.applications import ResNet50
from keras.layers.core import Dropout
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Input
from keras.models import Model
from keras.optimizers import SGD
```

```python
from keras import regularizers
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping, TensorBoard, History, ModelCheckpoint,
    RemoteMonitor, ReduceLROnPlateau
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import warnings
import pickle
import time
import cv2
import os
BATCH_SIZE = 32
# construct the argument parser and parse the arguments
parse = argparse.ArgumentParser()
parse.add_argument("-d", "--data", required=True,
    help="path to input dataset")
parse.add_argument("-m", "--model", required=True,
    help="path to output serialized model")
parse.add_argument("-l", "--label_bin", required=True,
    help="path to output label binarizer")
parse.add_argument("-e", "--epochs", type=int, default=25,
    help="# of epochs to train our network for")
parse.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
args = parse.parse_args()

# initialize the set of labels from the spots activity dataset we are
# going to train our network on
LABELS = {"cooking", "drinking"}

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args.data))

data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]
    info_file_name = imagePath.split(os.path.sep)[-1]

    # if the label of the current image is not part of of the labels
    # are interested in, then ignore the image
    if label not in LABELS:
        continue

    # load the image, convert it to RGB channel ordering, and resize
    # it to be a fixed 224x224 pixels, ignoring aspect ratio
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```python
    image = cv2.resize(image, (224, 224))
    print('[INFO] processing image: ', info_file_name)
    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)

# convert the data and labels to NumPy arrays
data = np.array(data)
labels = np.array(labels)
# print('labels before one-hot: ', labels)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# print('labels after one-hot: ', labels)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
print('[INFO] splitting data...')

trainX, testX, trainY, testY = train_test_split(data, labels,
    test_size=0.25, stratify=labels, random_state=42)

print('[INFO] creating generator object...')
# initialize the training data augmentation object
trainAug = ImageDataGenerator(
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")

# initialize the validation/testing data augmentation object (which
# we'll be adding mean subtraction to)
valAug = ImageDataGenerator()

# define the ImageNet mean subtraction (in RGB order) and set the
# the mean subtraction value for each of the data augmentation
# objects
mean = np.array([123.68, 116.779, 103.939], dtype="float32")
trainAug.mean = mean
valAug.mean = mean

# train and test generators
train_generator = trainAug.flow(trainX, trainY, batch_size=BATCH_SIZE)
validation_generator = valAug.flow(testX, testY)
# load the ResNet-50 network, ensuring the head FC layer sets are left
# off
# ignore warnings
warnings.filterwarnings("ignore")

# using tensorboard
tensorboard = TensorBoard(log_dir=f'logs/{time.time()}', batch_size=BATCH_SIZE)

# baseModel = ResNet50(include_top=False, weights="imagenet",
```

```python
    # input_tensor=Input(shape=(224, 224, 3)))

baseModel = ResNet50(include_top=False, weights="imagenet", input_tensor=Input(shape=(224, 224, 3)), input_shape=(244,244,3))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten()(headModel)
headModel = Dense(512, activation="relu", kernel_regularizer=regularizers.l1(0.01))(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(lb.classes_), activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the training process
for layer in baseModel.layers:
    layer.trainable = False

# compile our model (this needs to be done after our setting our
# layers to being non-trainable)
print("[INFO] compiling model...")
opt = SGD(lr=1e-4, momentum=0.9, decay=1e-4 / args.epochs)
model.compile(loss="categorical_crossentropy", optimizer=opt,
    metrics=["accuracy"])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)
es_callbacks = EarlyStopping(monitor='val_loss', mode='min', restore_best_weights=True, patience=10)
check_point = ModelCheckpoint(filepath='weights/weights.hdf5',  save_best_only=True, monitor='val_loss', mode='min', verbose=1)

# train the head of the network for a few epochs (all other layers
# are frozen) -- this will allow the new FC layers to start to become
# initialized with actual "learned" values versus pure random
print("[INFO] training head...")
H = History()
# R = RemoteMonitor(root='http://localhost:9000', path='output_temp/', field='data')
model.fit_generator(train_generator, steps_per_epoch=len(trainX) // BATCH_SIZE, validation_data=validation_generator, validation_steps=len(testX) // BATCH_SIZE, epochs=args.epochs, callbacks=[H, tensorboard, check_point, es_callbacks, reduce_lr])

# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=BATCH_SIZE)

print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1), target_names=lb.classes_))

# plot the training loss and accuracy
plt.style.use("ggplot")
```

```
plt.figure()
plt.plot(H.history["loss"], label="train_loss")
plt.plot(H.history["val_loss"], label="val_loss")
plt.plot(H.history["accuracy"], label="train_acc")
plt.plot(H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss | Accuracy")
plt.legend(loc="lower left")
plt.savefig(args.plot)
# serialize the model to disk
print("[INFO] serializing network...")
model.save(args.model)

# serialize the label binarizer to disk
f = open(args.label_bin, "wb")
f.write(pickle.dumps(lb))
f.close()
print('[INFO] model saved!')

# -------------------------------------------------TESTING RUNS
#RUN ON THE FLOYDHUB: python train.py --data /floyd/input/dlaction --model output/activity.model --label_bin output/lb.pickle --plot output/fig_v1.png --epochs 50

# TO DEBUG ON THE LOCAL SYSTEM: python train.py --data dataset_temp --model output_temp/activity.model --label_bin output_temp/lb.pickle --plot output_temp/_plot1.png --epochs 1
```

5.2 Prediction Code

```
# USAGE
# python predict_video.py --model model/activity.model --label-bin model/lb.pickle --input example_clips/drink1.mp4 --output output/drink1_128avg.avi --action drinking --size 128

# import the necessary packages
from keras.models import load_model
from collections import deque
from utils.json_export import to_json_file
from utils.fig_export import fig_plot
import numpy as np
import argparse
import pickle
import json
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="path to trained serialized model")
ap.add_argument("-l", "--label-bin", required=True,
    help="path to  label binarizer")
ap.add_argument("-i", "--input", required=True,
    help="path to our input video")
ap.add_argument("-o", "--output", required=True,
    help="path to our output video")
ap.add_argument("-s", "--size", type=int, default=128,
```

```
    help="size of queue for averaging")
ap.add_argument("-a", "--action", required=True, help="choose a predictive action from the list [drinking, cooking]")
args = ap.parse_args()

# load the trained model and label binarizer from disk
print("[INFO] loading model and label binarizer...")
model = load_model(args.model)
lb = pickle.loads(open(args.label_bin, "rb").read())

# initialize the image mean for mean subtraction along with the
# predictions queue
mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")
Q = deque(maxlen=args.size)

# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture(args.input)
writer = None
(W, H) = (None, None)


# list of lists [time(milsec), binary_label]
timeStamps = []
# loop over frames from the video file stream
while True:
    # read the next frame from the file
    (grabbed, frame) = vs.read()

    # if the frame was not grabbed, then we have reached the end
    # of the stream
    if not grabbed:
        break

    # if the frame dimensions are empty, grab them
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # clone the output frame, then convert it from BGR to RGB
    # ordering, resize the frame to a fixed 224x224, and then
    # perform mean subtraction
    output = frame.copy()
    # output = cv2.resize(frame.copy(), (244,244))
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (224, 224)).astype("float32")
    frame -= mean

    # make predictions on the frame and then update the predictions
    # queue
    preds = model.predict(np.expand_dims(frame, axis=0))[0]
    Q.append(preds)

    # perform prediction averaging over the current history of
    # previous predictions
    results = np.array(Q).mean(axis=0)
    i = np.argmax(results)
    label = lb.classes_[i]

```

```python
    if label == args.action:
        # get the frame time
        ts = round(vs.get(cv2.CAP_PROP_POS_MSEC), 15)
        # appending time, label to the list
        timeStamps.append([round(ts, 16), round(max(results), 5)])

    # draw the activity on the output frame
        text = "activity: {}".format(label)
        cv2.putText(output, text, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,0.6, (0, 255, 0), 2)
    else:
        cv2.putText(output, 'computing...', (10, 25), cv2.FONT_HERSHEY_SIMPLEX,0.6, (0, 0, 255), 2)
    # cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,
        # 1.25, (0, 255, 0), 2)

    # check if the video writer is None
    if writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args.output, fourcc, 30,
            (W, H), True)

    # write the output frame to disk
    writer.write(output)

    # show the output image
    cv2.imshow("Output", output)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# release the file pointers
print("[INFO] labled video saved to the folder...")
writer.release()
vs.release()

# creating json file as: timeLabel.json
print("[INFO] writing json time-label...")
to_json_file(args.action, timeStamps)

print("[INFO] saving time-label figure to the folder...")
x_p = [item[0] for item in timeStamps]
y_p = [item[1] for item in timeStamps]
x_np_param = np.array(x_p)
y_np_param = np.array(y_p)

fig_plot(x_np_param, y_np_param, args.action)

print("[INFO] DONE...")






# python predict_video.py --model model/model_v5/activity.model --label-bin model/
```

```
    model_v5/lb.pickle --input example_clips/v5/cook5.mp4 --output output/output_v5/
    c5_v5.avi --action cooking --size 128
```

6 Running Instruction

6.1 Install Dependencies

- keras
- numpy
- opencv

7 Running the Prediction Code

```

python predict_video.py --model model/activity.model --label-bin model/lb.pickle
--input "YOUR VIDEO PATH" --output output/"ARBITRARY NAME".avi
--action drinking --size 128
```

7.1 Video Instruction Link

Playlist link *>> HERE <<*

or

`https://www.youtube.com/watch?v=LGWF0j00oMI&list=PLHQn6Zmgbs7lScSfC6COPhtF7Hu57teD0`

8 Performance, Loss and Accuracy Figures

8.1 Previous Attempts

[INFO] evaluating network...

	precision	recall	f1-score	support
cooking	0.87	0.98	0.92	1958
drinking	0.95	0.71	0.81	977
accuracy			0.89	2935
macro avg	0.91	0.84	0.86	2935
weighted avg	0.90	0.89	0.88	2935

(a) Performance after 50 epochs

[INFO] evaluating network...

	precision	recall	f1-score	support
cooking	0.85	0.98	0.91	1958
drinking	0.94	0.66	0.77	977
accuracy			0.87	2935
macro avg	0.89	0.82	0.84	2935
weighted avg	0.88	0.87	0.87	2935

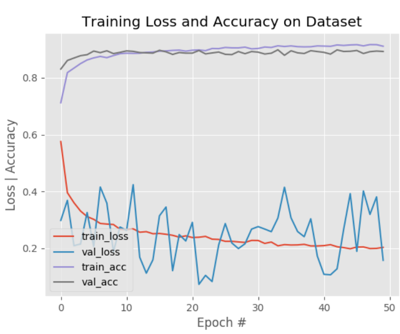
(b) Performance after 10 epochs

```
Epoch 00008: val_loss improved from 0.20288 to 0.19947, saving model to weights/weights.hdf5
Epoch 9/50
275/275 [==============================] - 131s 478ms/step - loss: 0.2744 - accuracy: 0.8798
 - val_loss: 0.3043 - val_accuracy: 0.8867

Epoch 00009: val_loss did not improve from 0.19947
Epoch 10/50
275/275 [==============================] - 131s 476ms/step - loss: 0.2661 - accuracy: 0.8822
 - val_loss: 0.0519 - val_accuracy: 0.8825

Epoch 00010: val_loss improved from 0.19947 to 0.05192, saving model to weights/weights.hdf5
Epoch 11/50
275/275 [==============================] - 131s 476ms/step - loss: 0.2638 - accuracy: 0.8844
 - val_loss: 0.1969 - val_accuracy: 0.8856

Epoch 00011: val_loss did not improve from 0.05192
Epoch 12/50
275/275 [==============================] - 130s 475ms/step - loss: 0.2654 - accuracy: 0.8812
 - val_loss: 0.5451 - val_accuracy: 0.8794

Epoch 00012: val_loss did not improve from 0.05192
Epoch 13/50
275/275 [==============================] - 131s 475ms/step - loss: 0.2547 - accuracy: 0.8903
 - val_loss: 0.3427 - val_accuracy: 0.8898
```

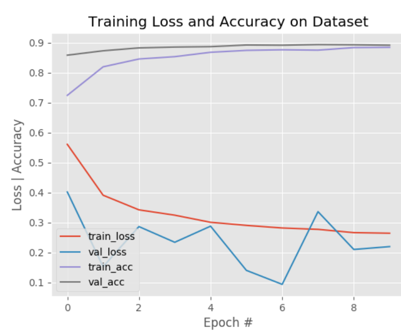
(c) Training procedure between epoch 8–12

```
Epoch 00045: val_loss did not improve from 0.05192
Epoch 46/50
275/275 [==============================] - 130s 471ms/step - loss: 0.1999 - accuracy: 0.9116
 - val_loss: 0.4214 - val_accuracy: 0.8753

Epoch 00046: val_loss did not improve from 0.05192
Epoch 47/50
275/275 [==============================] - 131s 475ms/step - loss: 0.2029 - accuracy: 0.9144
 - val_loss: 0.1752 - val_accuracy: 0.8915

Epoch 00047: val_loss did not improve from 0.05192
Epoch 48/50
275/275 [==============================] - 130s 473ms/step - loss: 0.2001 - accuracy: 0.9147
 - val_loss: 0.7645 - val_accuracy: 0.8805

Epoch 00048: val_loss did not improve from 0.05192
Epoch 49/50
275/275 [==============================] - 130s 474ms/step - loss: 0.1965 - accuracy: 0.9185
 - val_loss: 0.2798 - val_accuracy: 0.8595

Epoch 00049: val_loss did not improve from 0.05192
Epoch 50/50
275/275 [==============================] - 130s 474ms/step - loss: 0.2057 - accuracy: 0.9112
 - val_loss: 0.2782 - val_accuracy: 0.8794

Epoch 00050: val_loss did not improve from 0.05192
```

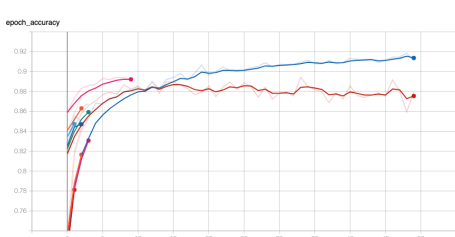
(d) Training procedure between epoch 45–50



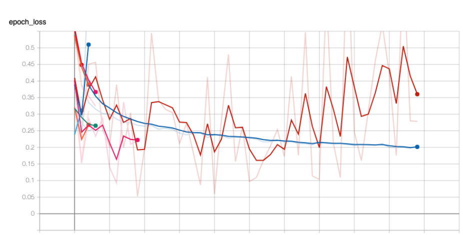
(e) Loss — Acc after 50 Epochs



(f) Loss–Acc after 10 Epochs



(g) Loss–Acc logs performance

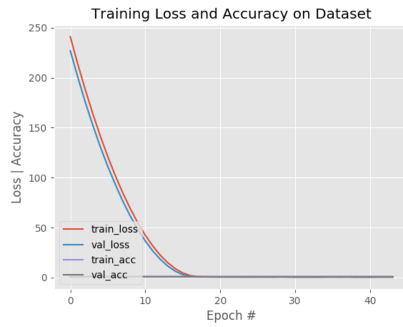


(h) Epoch–Loss logs performance

8.2 Latest Improvements

```
[INFO] evaluating network...
              precision    recall  f1-score   support

     cooking       0.87      0.94      0.90      1958
    drinking       0.85      0.71      0.78       977

    accuracy                           0.86      2935
   macro avg       0.86      0.83      0.84      2935
weighted avg       0.86      0.86      0.86      2935
```

(a) Performance–Final Project



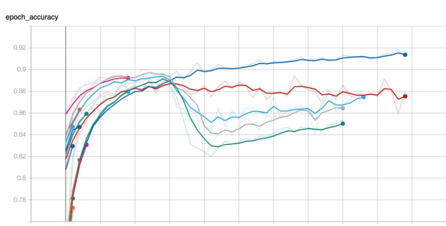
(b) Loss–Acc–Final Project

```
Epoch 00030: val_loss did not improve from 0.44479
Epoch 31/50
275/275 [==============================] - 131s 476ms/step - loss: 0.5372 - accuracy: 0.8398
 - val_loss: 0.4174 - val_accuracy: 0.8753

Epoch 00031: val_loss improved from 0.44479 to 0.41739, saving model to weights/weights.hdf5
Epoch 32/50
275/275 [==============================] - 133s 482ms/step - loss: 0.5356 - accuracy: 0.8425
 - val_loss: 0.4753 - val_accuracy: 0.8553

Epoch 00032: val_loss did not improve from 0.41739
Epoch 33/50
275/275 [==============================] - 132s 481ms/step - loss: 0.5325 - accuracy: 0.8390
 - val_loss: 0.4421 - val_accuracy: 0.8622

Epoch 00033: val_loss did not improve from 0.41739
Epoch 34/50
275/275 [==============================] - 128s 467ms/step - loss: 0.5309 - accuracy: 0.8371
 - val_loss: 0.3705 - val_accuracy: 0.8650

Epoch 00034: val_loss improved from 0.41739 to 0.37049, saving model to weights/weights.hdf5
Epoch 35/50
275/275 [==============================] - 131s 476ms/step - loss: 0.5289 - accuracy: 0.8362
 - val_loss: 0.4811 - val_accuracy: 0.8643
```

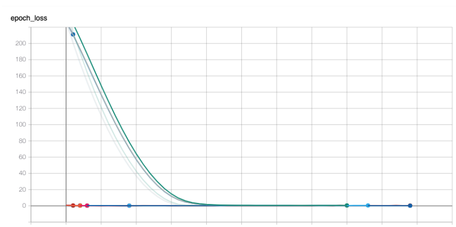
(c) Training procedure between epoch 30–34–Final Project

```
Epoch 00040: val_loss did not improve from 0.37049
Epoch 41/50
275/275 [==============================] - 128s 467ms/step - loss: 0.5073 - accuracy: 0.8533
 - val_loss: 0.4072 - val_accuracy: 0.8674

Epoch 00041: val_loss did not improve from 0.37049
Epoch 42/50
275/275 [==============================] - 131s 476ms/step - loss: 0.5037 - accuracy: 0.8492
 - val_loss: 0.5217 - val_accuracy: 0.8719

Epoch 00042: val_loss did not improve from 0.37049
Epoch 43/50
275/275 [==============================] - 132s 479ms/step - loss: 0.5063 - accuracy: 0.8440
 - val_loss: 0.4921 - val_accuracy: 0.8784

Epoch 00043: val_loss did not improve from 0.37049
Epoch 44/50
275/275 [==============================] - 131s 475ms/step - loss: 0.5041 - accuracy: 0.8449
 - val_loss: 0.5585 - val_accuracy: 0.8774

Epoch 00044: val_loss did not improve from 0.37049
```

(d) Training procedure between epoch 40–44–Final Project



(e) Loss–Acc logs performance–Final Project



(f) Epoch–Loss logs performance–Final Project