AI-BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMA

Team ID: PNT2022TMID47799

Team Leader: S.F.Mohamed Aasiq.

Team member:A.Arul Raj.
Team member:R.Hriharasudhan.
Team member: S.Lenin.

INTRODUCTION

1.1 Project Overview:

Computer-aided diagnosis (CAD) is a computer-based system that is used in the medical imaging field to aid healthcare workers in their diagnoses. CAD has become a mainstream tool in several medical fields such as mammography and colonography. However, in dermatology, although skin disease is a common disease, one in which early detection and classification is crucial for the successful treatment and recovery of patients, dermatologists perform most noninvasive screening tests only with the naked eye. This may result in avoidable diagnostic inaccuracies as a result of human error, as the detection of the disease can be easily overlooked. Furthermore, classification of a disease is difficult due to the strong similarities between common skin disease symptoms. Therefore, it would be beneficial to exploit the strengths of CAD using artificial intelligence techniques, in order to improve the accuracy of dermatology diagnosis. This paper shows that CAD may be a viable option in the field of dermatology using state-of-the-art deep learning models.

The segmentation and classification of skin diseases has been gaining attention in the field of artificial intelligence because of its promising results. Two of the more prominent approaches for skin disease segmentation and classification are clustering algorithms and support vector machines (SVMs). Clustering algorithms generally have the advantage of being flexible, easy to implement, with the ability to generalize features that have a similar statistical variance. Trabelsi et al. experimented with various clustering algorithms, such as fuzzy c-means, improved fuzzy c-means, and K-means, achieving approximately 83% true positive rates in segmenting a skin disease. Rajab et al. implemented an ISODATA clustering algorithm to find the

optimal threshold for the segmentation of skin lesions. An inherent disadvantage of clustering a skin disease is its lack of robustness against noise. Clustering algorithms rely on the identification of a centroid that can generalize a cluster of data. Noisy data, or the presence of outliers, can significantly degrade the performance of these algorithms. Therefore, with noisy datasets, caused by images with different types of lighting, non-clustering algorithms may be preferred; however, Keke et al. implemented an improved version of the fuzzy clustering algorithm using the RGB, HSV, and LAB color spaces to create a model that is more robust to noisy data. SVMs have gained attention for their effectiveness in high-dimensional data and their capability to decipher "…subtle patterns in noisy and complex datasets". Lu et al. segmented erythema in the skin using the radial basis kernel function that allows SVMs to separate nonlinear hyperplanes. Sumithra et al. combined a linear SVM with a k-NN classifier to segment and classify five different classes of skin lesions. Maglogiannis et al. implemented a threshold on the RGB value for segmentation and used an SVM for classification. Although more robust than clustering algorithms, SVMs are more reliant on the preprocessing of data for feature extraction. Without preprocessing that allows a clear definition of hyperplanes, SVMs may also underperform.

Owing to the disadvantages of these traditional approaches, convolution neural networks (CNNs) have gained popularity because of their ability to extract high-level features with minimal preprocessing. CNNs can expand the advantages of SVMs, such as robustness in noisy datasets without the need for optimal preprocessing, by capturing image context and extracting high-level features through down-sampling. CNNs can interpret the pixels of an image within its own image-level context, as opposed to viewing each pixel in a dataset-level context. However, although down-sampling allows CNNs to view an image in its own context, it degrades the resolution of the image. Although context is gained, the location of a target is lost through down-sampling. This is not a problem for classification, but causes some difficulty for segmentation, as both the context and location of the target are essential for optimal performance. To solve this, up-sampling is needed, which works in a manner opposite to that of down-sampling, in the sense that it increases the resolution of the image. While down-sampling takes a matrix and decreases it to a smaller feature map, up-

sampling takes a feature map and increases it to a larger matrix. By learning to accurately create a higher-resolution image, CNNs can determine the location of the targets to segment. Thus, for segmentation, we use a combination of down-sampling and up-sampling, whereas for classification, we use only down-sampling. To further leverage the advantages of CNNs, skip-connections were introduced, which provided a solution to the degradation problem that occurs when CNN models become too large and complex. We implement skip-connections in both segmentation and classification models. In the segmentation model, blocks of equal feature numbers are connected between the down and up-sampling sections. In the classification model, these skip-connections exist in the form of inverted residual blocks. This allows our models to grow in complexity without any performance degradation.

In this paper, we present a method to sequentially combine two separate models to solve a larger problem. In the past, skin disease models have been applied to either segmentation or classification. In this study, we sequentially combine both models by using the output of a segmentation model as input to a classification model. In addition, although past studies of non-CNN segmentation models used innovative preprocessing methods, recent CNN developments have focused more on the architecture of the model than on the preprocessing of data. As such, we apply an innovative preprocessing method to the data of our CNN segmentation model. The methods described above lack the ability to localize and classify multiple diseases within one image; however, we have developed a method to address this problem. Our objective is two-fold. First, we show that CAD can be used in the field of dermatology. Second, we show that state-of-the-art models can be used with current computing power to solve a wider range of complex problems than previously imagined. We begin by explaining the results of our experimentation, followed by a discussion of our findings, a more detailed description of our methodology, and finally, the conclusions that can be drawn from our study.

1.2 Purpose:

The purpose of the project is design and implementation of deep learning model

deployed for detection of image processing based skin disease.

2. LITERATURE SURVEY

Paper 1: The Classification of Six Common Skin Diseases Based on

Xiangya-Derm:Development of a ChineseDatabase for Artificial Intelligence

Published year: 2021
Author: Shuang Zhao
Journal Name: Journal of medical internet research

Summary: In this study, we established a new database,Xiangya-Derm, which consists of over 150,000 clinical images of 571 different skin diseases in the Chinesepopulation. Xiang-Derm is the first integrated, normative database based on skin conditions in the Chinese population. Based on this database, we selected six commonskindiseases and proposedan AI network, Xy-SkinNet. The top 1 and top 3 diagnostic accuracies of Xy-SkinNet were higher than those of dermatologists from the Department of Dermatology. This study was an attempt at exploring AI products and services and has successfully set the stage for future development. An increasingnumber of studies are incorporating clinical images.There are already some open databases, such as AtlasDerm, Derm101, and Dermnet. Considering

differences in skin color, Xiangya-Derm can provide data for realizing AI diagnosis of skin diseases among the Chinese population. Many existing databases lack medical history information, especially information about pathological diagnosis, and, potentially, contain some misdiagnosed photos. Notably, one of the greatest advantages of Xiangya-Derm is that most images contain corresponding skin pathology results, providing the category annotation of a gold standard,which can be most effectively

applied to variousresearchstudies and in the development of AI. This feature ensures that the diagnostic information about pictures used for deep learning is accurate and reduces the diagnostic errors caused by misdiagnosis. Of course, there are also a small number of unmatched pictures in our database, which iscorrelated with the lack of corresponding dermoscopic images.In addition,XiangyaDerm provides image data with the location for all skin lesions, thus enabling researchers to apply object detection algorithms in computer vision for the automaticdiagnosis of skin diseases. Moreover, each image has a full set of clinical information about the patient, including demographic information, complaints, current medical history, past medical history, and family history. Given the complete set of big data, conducting further research on AI diagnosis using multimodal data, which is morecoincident with the real-world diagnosis process and more intuitive

for both doctors and patients, is achievable.

2.1 EXISTING SYSTEM

The color of patient skin helps doctors to determinate the type of skin lesion, if the skin lesion is diagnosed as melanoma, its color could be black, brown, pink, red, purple, blue or white,. The dermoscopy technique is high spread skin imaging way that helps in skin lesion detection. a dermatoscope device take an image, known as dermoscopic image, with a low level noise to examine the skin lesion by magnifying and filtering the infected part of skin . Another aiding way to detect the skin lesion at an early stage is the computer aided diagnosis (CAD) system. The dermoscopic images of skin lesions have been classified by Gonzalez-Castro et al. Global andlocal feature extraction method to extract a different features of an image such as color, texture, shape and domain specific features.

2.2 References

1. Kachuee, Mohammad, Shayan Fazeli, and Majid Sarrafzadeh. "Ecg heartbeat classification: A deep transferable representation." 2018 IEEE international conference on healthcare informatics (ICHI). IEEE, 2018

2.S. Zhang, W. Wang, J. Ford, and F. Makedon, “Learning from incomplete ratings using non-negative matrix factorization,” in Proc. 6th SIAM Int. Conf. Data Mining, 2006, pp. 549–553.

3.T. Hofmann and J. Puzicha, “Latent class models for collaborative filtering,” in Proc. 6th Int. Joint Conf. Artif. Intell., 1999, pp. 688–693.

4.B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl, “Item-based collaborative filtering recommendation algorithms,” in Proc. 10th Int. World Wide Web Conf., 2001, pp. 285–295

5.T. George and S. Merugu, “A scalable collaborative filtering framework based on co-clustering,” in Proc. 5th IEEE Int. Conf. Data Mining, 2005, pp. 625–628

2.3 Problem statement definition

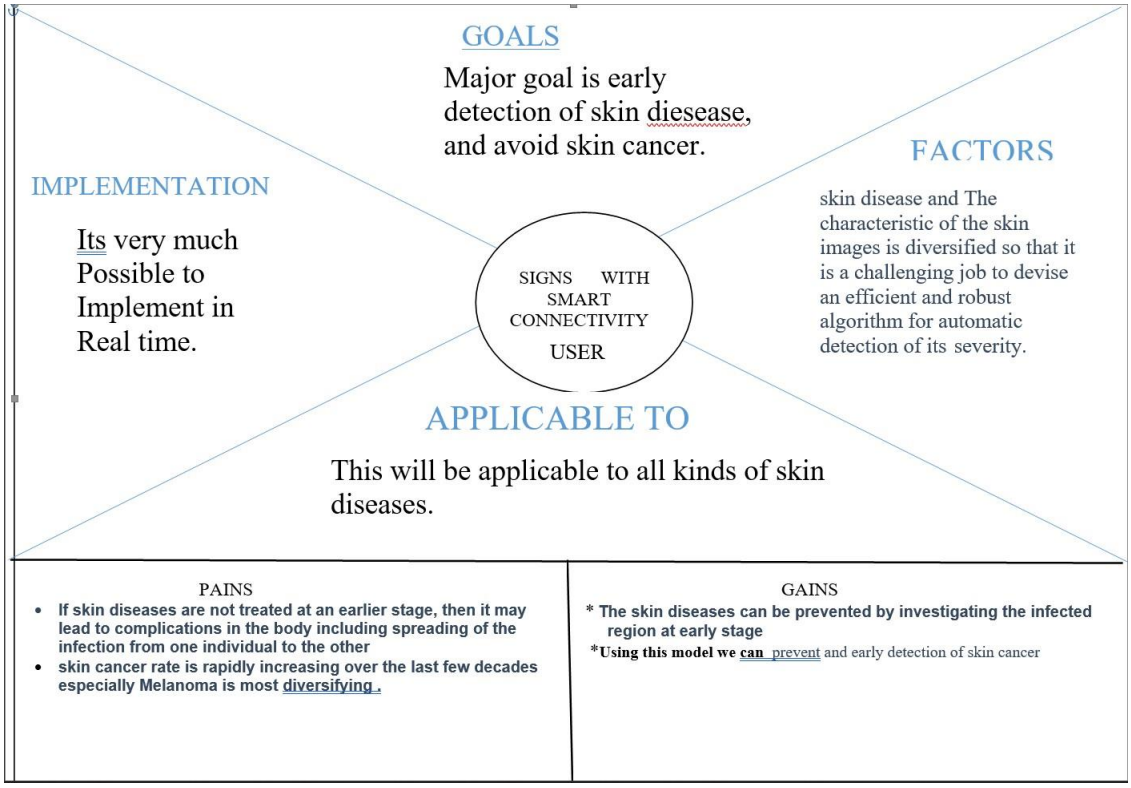
1. Cardiologists by using various values which occurred during the ECG recording can decide whether the heart beat is normal or not. Since observation of these values are not always clear, existence of automatic ECG detection system is required

2. Luz, Eduardo José da S., et al. "ECG-based heartbeat classification for arrhythmia detection: A survey." Computer methods and programs in biomedicine 127 (2016): 144-164

3. Romdhane, Taissir Fekih, and Mohamed Atri Pr. "Electrocardiogram heartbeat classification based on a deep convolutional neural network and focal loss." Computers in Biology and Medicine 123 (2020): 103866

3. IDEATION AND PROPOSED SOLUTION:

3.1 Empathy Map



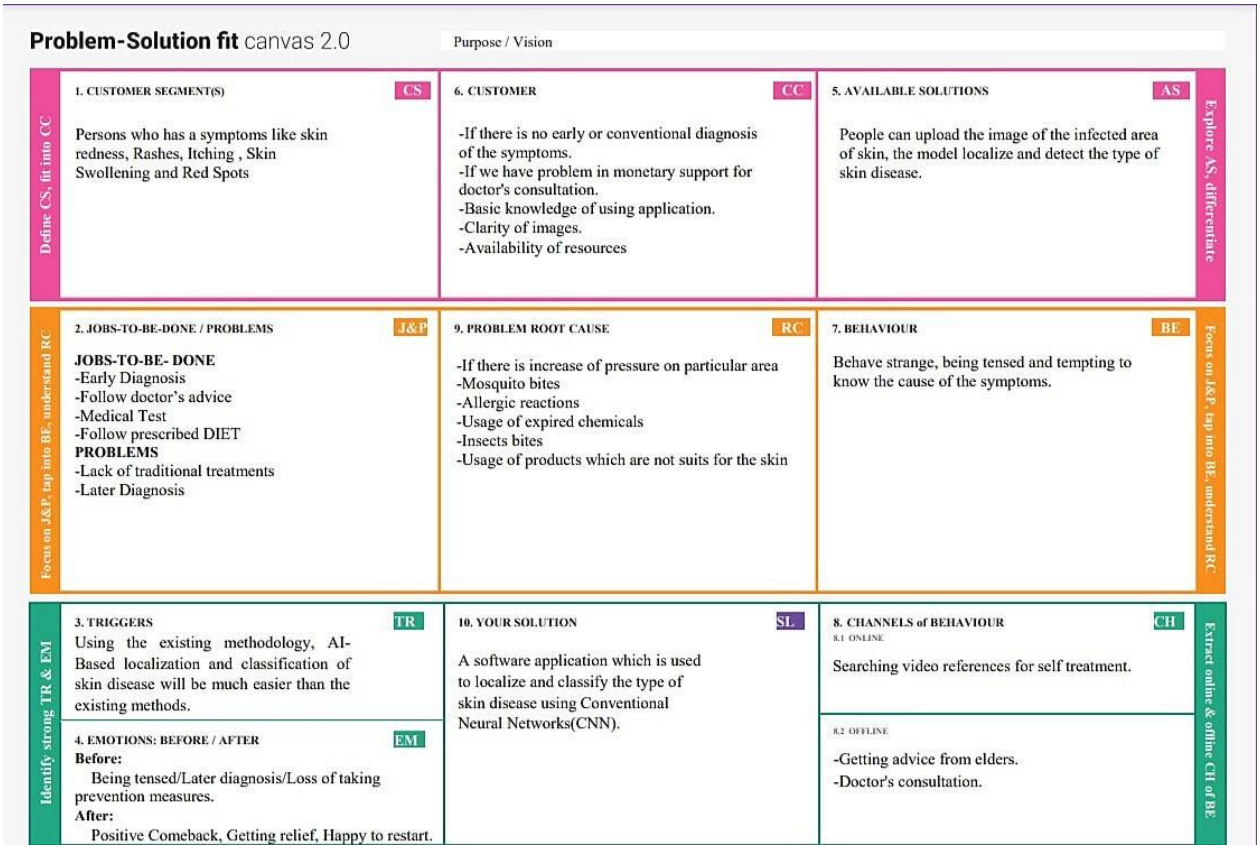
3.2 Ideation and Brainstroming:

3.3 Proposed Solution:

S.No.	Parameter	Description

1.	Problem Statement (Problem to be solved)	User is a busy worker who needs an immediate result with more accuracy for his/her skin problem but he/she has no time to visit dermatologists in-person.
2.	Idea / Solution description	The person can capture the images of skin and then the image will be sent to the trained model. The model analyses the image and detects whether the person is having skin disease or not.
3.	Novelty / Uniqueness	Images with noise have also been taken and are enhanced with effective algorithms for predicting the diseases.
4.	Social Impact / Customer Satisfaction	By just uploading the images various skin diseases can be diagnosed and this system is very efficient which serves civilians to detect the diseases earlier.
5.	Business Model (Revenue Model)	As we are planning to design a proprietary product as a solution and distribute it to users, this will serve as our return on investment.
6.	Scalability of the Solution	This system is more scalable because it takes any type of images regardless of its resolution and it provides high performance irrespective of the environment.

3.4 **Problem Solution Fit:**



4. REQUIREMENT ANALYSIS

4.1 Functional requirement:

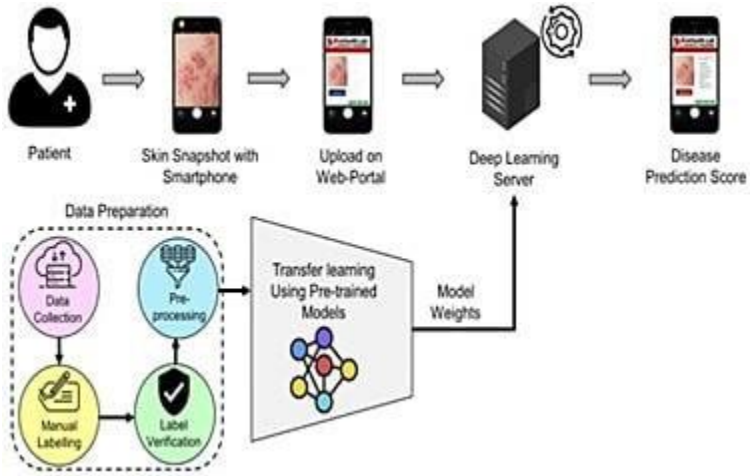
1. System : Pentium IV 2.4 GHz.
2. Hard Disk : 40 GB.
3. Floppy Drive : 1.44 Mb.
4. Monitor : 15 VGA Colour.
5. Mouse : Logitech.
6. Ram : 512 Mb.

4.2 Non-Functional requirement:

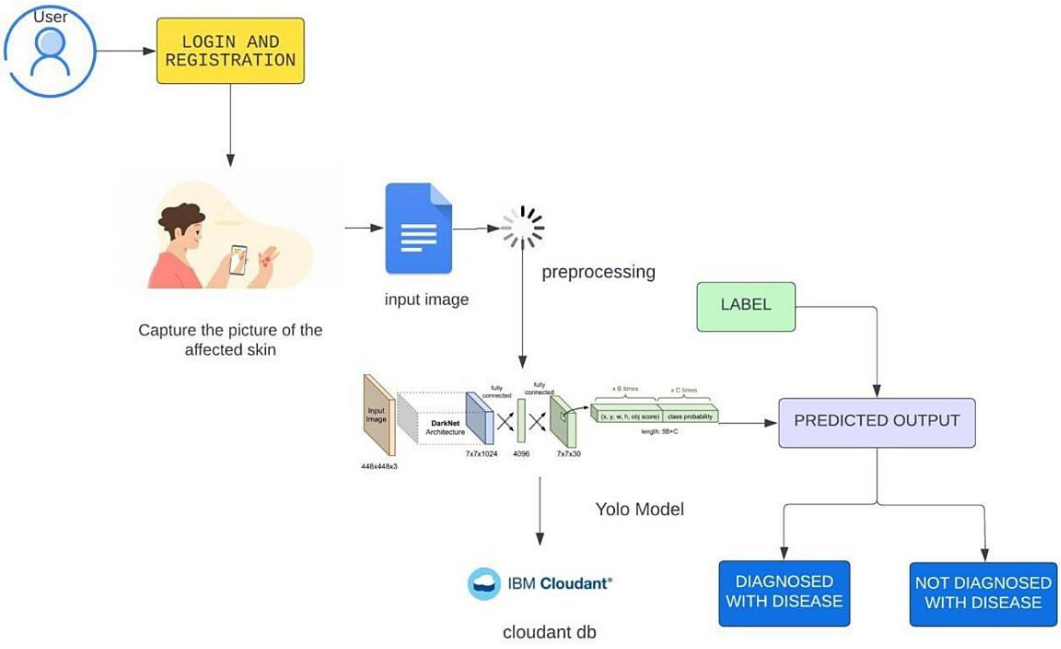
1. Python IDE (IDLE / Spyder / PyCharm)(Python 3.7)
2. Microsoft's Visual Object Tagging Tool (VoTT)
3. Python Packages need to be installed

5.PROJECT DESIGN:

5.1 Data flow diagrams:



5.2 Solution and Technology Architecture:



5.3 User Stories:

6. PROJECT PALNNING AND SCHEDULING

6.1 Sprint Delivery Schedule:

TITLE	DESCRIPTION	DATE
Literature Survey	Literature survey on the selected project & gatheringinformation by referring the,technical papers, research publications etc.	**10 OCTOBER2022**
Empathy Map for AI based localization and classification ofskin diseasewith erythema	Prepare Empathy Map Canvas to capture the user Pains&Gains, Prepare listofproblem statements	**10 OCTOBER2022**
Problem Statement	Prepare the problemstatement document	**10 OCTOBER2022**
Brainstorming IdeaGeneration Prioritization	List the by organizing the brainstorming session andprioritize the top 3 ideas based on the feasibility & importance.	**18 OCTOBER 2022**

6.2 Sprint Planning and Estimation:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	14 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022 1	19 Nov 2022	20	19 Nov 2022

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

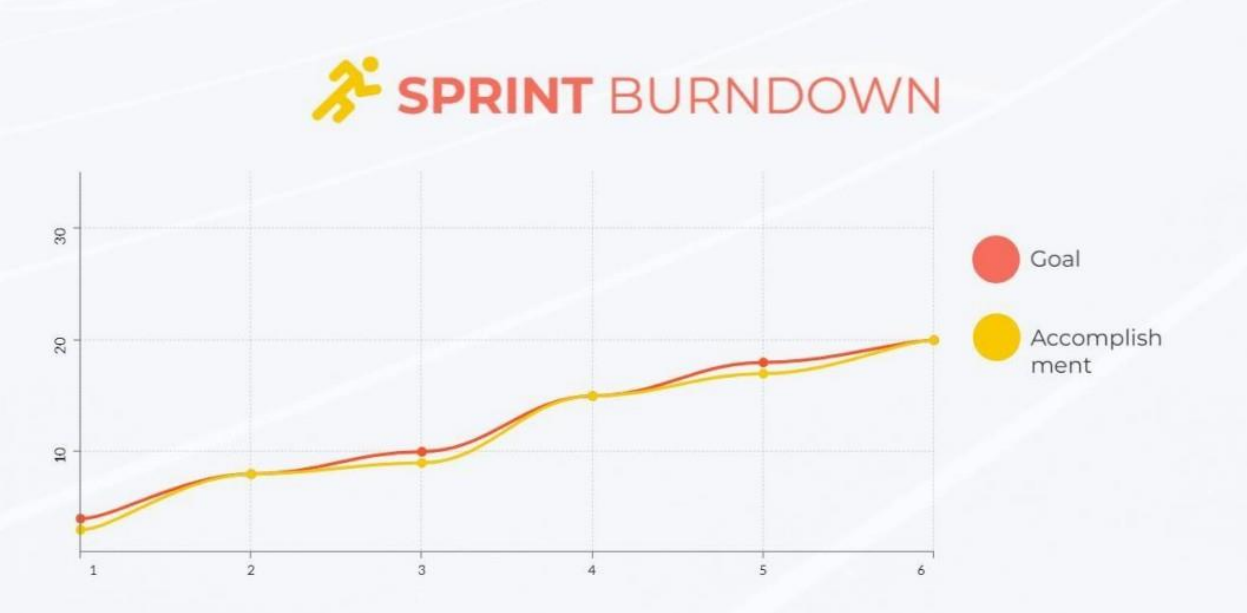
$$AV = \frac{SPRINT\ DURATION}{VELOCITY}$$

$$= \frac{20}{10}$$

$$= 2$$

Burndown Chart: A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can

be applied to any project containing measurable progress over time.



7.CODING AND SOLUTIONING:

7.1 Feature 2

Annotate Images Our detector needs some high-quality training examples before it can start learning. The images in our training folder are manually labelled using Microsoft's Visual Object Tagging Tool (VoTT). At least 100 images should be annotated for each category to get respectable results. TheVoTT csv formatted annotation data is converted to YOLOv3 format by Convert_to_YOLO_format.py file.

CODE:

```
from PIL import Image
from is import path, makedirs
import os
import re
import pandas as pd
import sys
import argparse
def get_parent_dir(n=1):
""" returns the n-the parent directory of the current
working directory """
current_path = os.path.dirname(os.path.abspath( file
))
for k in range(n):
current_path = os.path.dirname(current_path)
return current_path
sys.path.append(os.path.join(get_parent_dir(1),
"Utils"))
 from Convert_Format  import  convert_vott_csv_to_yolo
Data_Folder = os.path.join(get_parent_dir(1), "Data")
VoTT_Folder = os.path.join(
Data_Folder, "Source_Images", "Training_Images",
"vott-csv-export"
```

```
)
VoTT_csv = os.path.join(VoTT_Folder, "Annotations-export.csv")
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")
model_folder = os.path.join(Data_Folder, "Model_Weights")
classes_filename = os.path.join(model_folder, "data_classes.txt")
if name == " main ":
# surpress any inhereted default values
parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
"""
Command line options
"""
parser.add_argument(
"--VoTT_Folder",
type=str,
default=VoTT_Folder,
help="Absolute path to the exported files from the image tagging step with
VoTT. Default is "
+ VoTT_Folder,
)
parser.add_argument(
"--VoTT_csv",
type=str,
default=VoTT_csv,
help="Absolute path to the *.csv file exported from VoTT. Default is "
```

```
44+ VoTT_csv,
45)
46parser.add_argument(
47"--YOLO_filename",
48type=str,
49default=YOLO_filename,
50help="Absolute path to the file where the annotations
  in YOLO format should be
51saved. Default is "
52+ YOLO_filename,
53)
54FLAGS = parser.parse_args()
55# Prepare the dataset for YOLO
56multi_df = pd.read_csv(FLAGS.VoTT_csv)
57labels = multi_df["label"].unique()
58 labeldict = dict(zip(labels, range(len(labels))))
59multi_df.drop_duplicates(subset=None, keep="first",
  inplace=True)
60train_path = FLAGS.VoTT_Folder
61convert_vott_csv_to_yolo(
62multi_df, labeldict, path=train_path,
  target_name=FLAGS.YOLO_filename
63)
64# Make classes file
65file = open(classes_filename, "w")
66# Sort Dict by Values
67SortedLabelDict = sorted(labeldict.items(),
  key=lambda x: x[1])
68for elem in SortedLabelDict:
69 file.write(elem[0] + "\n")
70file.close()
```

7.2 Feature 2 :

Training Yolo

To prepare for the training process, convert the YOLOv3 model to the Kerasformat. The YOLOv3 Detector can then be trained by Train_YOLO.py file.\

CODE:

```
import os
import sys
import argparse
import warnings
def get_parent_dir(n=1):
""" returns the n-th parent directory of the current
working directory """
current_path = os.path.dirname(os.path.abspath( file
))
for k in range(n):
current_path = os.path.dirname(current_path)
return current_path
src_path = os.path.join(get_parent_dir(0), "src")
sys.path.append(src_path)
utils_path = os.path.join(get_parent_dir(1), "Utils")
sys.path.append(utils_path)
import numpy as np
import keras.backend as K
from keras.layers import Input, Lambda
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import (
TensorBoard,
ModelCheckpoint,
```

```
ReduceLROnPlateau,
EarlyStopping,
)
from keras_yolo3.yolo3.model import (
preprocess_true_boxes,
yolo_body,
tiny_yolo_body,
yolo_loss,
)
from keras_yolo3.yolo3.utils import get_random_data
from PIL import Image
from time import time
import tensorflow.compat.v1 as tf
import pickle
from Train_Utils import (
get_classes,
get_anchors,
create_model,
create_tiny_model,
data_generator,
data_generator_wrapper,
ChangeToOtherMachine,
)
keras_path = os.path.join(src_path, "keras_yolo3")
Data_Folder = os.path.join(get_parent_dir(1), "Data")
Image_Folder = os.path.join(Data_Folder,
  "Source_Images", "Training_Images")
VoTT_Folder = os.path.join(Image_Folder, "vott-csv-
  export")
YOLO_filename = os.path.join(VoTT_Folder,
  "data_train.txt")
Model_Folder = os.path.join(Data_Folder,
```

```
  "Model_Weights")
YOLO_classname = os.path.join(Model_Folder,
  "data_classes.txt")
log_dir = Model_Folder
anchors_path = os.path.join(keras_path, "model_data",
  "yolo_anchors.txt")
weights_path = os.path.join(keras_path, "yolo.h5")
FLAGS = None
if name == " main ":
# Delete all default flags
parser =
  argparse.ArgumentParser(argument_default=argparse.SUP
  PRESS)
"""
Command line options
"""
parser.add_argument(
"--annotation_file",
type=str,
default=YOLO_filename,
help="Path to annotation file for Yolo. Default is "
  + YOLO_filename,
)
parser.add_argument(
"--classes_file",
type=str,
default=YOLO_classname,
help="Path to YOLO classnames. Default is " +
  YOLO_classname,
)
parser.add_argument(
"--log_dir",
```

```
78 type=str,
79 default=log_dir,
80 help="Folder to save training logs and trained
   weights to. Default is "
81 + log_dir,
82 )
83 parser.add_argument(
84 "--anchors_path",
85 type=str,
86 default=anchors_path,
87 help="Path to YOLO anchors. Default is " +
   anchors_path,
88 )
89 parser.add_argument(
90 "--weights_path",
91 type=str,
92 default=weights_path,
93 help="Path to pre-trained YOLO weights. Default is "
   + weights_path,
94 )
95 parser.add_argument(
96 "--val_split",
97 type=float,
98 default=0.1,
99 help="Percentage of training set to be used for
   validation. Default is 10%.",
100 )
101 parser.add_argument(
102 "--is_tiny",
103 default=False,
104 action="store_true",
105 help="Use the tiny Yolo version for better
```

```
  performance and less accuracy.
106 Default is False.",
107 )
108 parser.add_argument(
109 "--random_seed",
110 type=float,
111 default=None,
112 help="Random seed value to make script
  deterministic. Default is 'None', i.e.
113 non-deterministic.",
114 )
115 parser.add_argument(
116 "--epochs",
117 type=float,
118 default=51,
119 help="Number of epochs for training last layers and
  number of epochs for finetuning layers. Default is
  51.",
120 )
121 parser.add_argument(
122 "--warnings",
123 default=False,
124 action="store_true",
125 help="Display warning messages. Default is False.",
126 )
127 FLAGS = parser.parse_args()
128 if not FLAGS.warnings:
129 tf.logging.set_verbosity(tf.logging.ERROR)
130 os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
131 warnings.filterwarnings("ignore")
132 np.random.seed(FLAGS.random_seed)
133 log_dir = FLAGS.log_dir
```

```
class_names = get_classes(FLAGS.classes_file)
num_classes = len(class_names)
anchors = get_anchors(FLAGS.anchors_path)
weights_path = FLAGS.weights_path
input_shape = (416, 416) # multiple of 32, height, width
epoch1, epoch2 = FLAGS.epochs, FLAGS.epochs
is_tiny_version = len(anchors) == 6 # default setting
if FLAGS.is_tiny:
model = create_tiny_model(
input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path
)
else:
model = create_model(
input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path
) # make sure you know what you freeze
log_dir_time = os.path.join(log_dir, "{}".format(int(time())))
logging = TensorBoard(log_dir=log_dir_time)
checkpoint = ModelCheckpoint(
os.path.join(log_dir, "checkpoint.h5"),
monitor="val_loss",
save_weights_only=True,
save_best_only=True,
period=5,
)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3,
verbose=1)
```

```
early_stopping = EarlyStopping(
monitor="val_loss", min_delta=0, patience=10,
  verbose=1
)
val_split = FLAGS.val_split
with open(FLAGS.annotation_file) as f:
lines = f.readlines()
# This step makes sure that the path names
  correspond to the local machine
# This is important if annotation and training are
  done on different machines (e.g.
training on AWS)
lines = ChangeToOtherMachine(lines,
  remote_machine="")
np.random.shuffle(lines)
num_val = int(len(lines) * val_split)
num_train = len(lines) - num_val
# Train with frozen layers first, to get a stable
  loss.
# Adjust num epochs to your dataset. This step is
  enough to obtain a decent model.
if True:
model.compile(
optimizer=Adam(lr=1e-3),
loss={
# use custom yolo_loss Lambda layer.
"yolo_loss": lambda y_true, y_pred: y_pred
},
)
batch_size = 32
print(
"Train on {} samples, val on {} samples, with batch
```

```
  size {}.".format(
num_train, num_val, batch_size
)
)
history = model.fit_generator(
data_generator_wrapper(
lines[:num_train], batch_size, input_shape, anchors,
  num_classes
),
steps_per_epoch=max(1, num_train // batch_size),
validation_data=data_generator_wrapper(
lines[num_train:], batch_size, input_shape, anchors,
  num_classes
),
validation_steps=max(1, num_val // batch_size),
epochs=epoch1,
initial_epoch=0,
callbacks=[logging, checkpoint],
)
model.save_weights(os.path.join(log_dir,
  "trained_weights_stage_1.h5"))
step1_train_loss = history.history["loss"]
file = open(os.path.join(log_dir_time,
  "step1_loss.npy"), "w")
with open(os.path.join(log_dir_time,
  "step1_loss.npy"), "w") as f:
for item in step1_train_loss:
f.write("%s\n" % item)
file.close()
step1_val_loss =
  np.array(history.history["val_loss"])
file = open(os.path.join(log_dir_time,
  "step1_val_loss.npy"), "w")
```

```
with open(os.path.join(log_dir_time,
  "step1_val_loss.npy"), "w") as f:
for item in step1_val_loss:
f.write("%s\n" % item)
file.close()
# Unfreeze and continue training, to fine-tune.
 # Train longer if the result is unsatisfactory.
if True:
for i in range(len(model.layers)):
model.layers[i].trainable = True
model.compile(
optimizer=Adam(lr=1e-4),        loss={"yolo_loss":
  lambday_true, y_pred: y_pred}
) # recompile to apply the change
print("Unfreeze all layers.")
batch_size = (
4 # note that more GPU memory is required after
  unfreezing the body
)
print(
"Train on {} samples, val on {} samples, with batch
  size {}.".format(
num_train, num_val, batch_size
)
)
history = model.fit_generator(
data_generator_wrapper(
lines[:num_train], batch_size, input_shape, anchors,
  num_classes
),
steps_per_epoch=max(1, num_train // batch_size),
validation_data=data_generator_wrapper(
```

```
lines[num_train:], batch_size, input_shape, anchors,
  num_classes
),
validation_steps=max(1, num_val // batch_size),
epochs=epoch1 + epoch2,
initial_epoch=epoch1,
callbacks=[logging, checkpoint, reduce_lr,
  early_stopping],
)
model.save_weights(os.path.join(log_dir,
  "trained_weights_final.h5"))
step2_train_loss = history.history["loss"]
file = open(os.path.join(log_dir_time,
  "step2_loss.npy"), "w")
with open(os.path.join(log_dir_time,
  "step2_loss.npy"), "w") as f:
for item in step2_train_loss:
f.write("%s\n" % item)
file.close()
step2_val_loss =
  np.array(history.history["val_loss"])
file = open(os.path.join(log_dir_time,
  "step2_val_loss.npy"), "w")
with open(os.path.join(log_dir_time,
  "step2_val_loss.npy"), "w") as f:
for item in step2_val_loss:
f.write("%s\n" % item)
file.close()
```

7.3 Database Schema

● Registration: When a new user registers, the backend connects to the IBM Cloudant andstores the user's credentials in the database.
● Login: To check if a user is already registered, the backend connects to Cloudant when theyattempt to log in. They are an invalid user if they are not already registered.
● IBM cloudant: Stores the data which is registered.
● app.py: Connects both Frontend and the cloudant for the verification of user credential

8.TESTING

8.1 Test case:

Test Case No.	Action	Expected Output	Actual Output	Result
1	Register for the website	Stores name, email, and password in Database	Stores name,email, and password in Database	Pass
2	Login to the website	Giving the right credentials, results in a successfull login.	Giving the right credentials, results in a successfull login.	Pass
3	Detecting the disease	It should predict the disease	It should predict the disease	Pass

8.2 User Acceptance Testing:

Section	Test cases	Not Tested	Fail	Pass
Registration	9	0	0	9

Login	40	0	0	40
Security	2	0	0	2
Disease Detection	10	0	0	10
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9. RESULTS

9.1 Performance Metrics:

S.No.	Parameter	Values
1.	Model Summary	To evaluate object detection models like R-CNN andYOLO, the mean average precision (mAP) is used. ThemAP compares the ground-truth bounding box to thedetected box and returns a score.
2.	Accuracy	Training Accuracy – 89% Validation Accuracy – 95%
3.	Confidence Score (Only Yolo Projects)	Class Detected – 93% Confidence Score – 90%

10. ADVANTAGES & DISADVANTAGES

Advantages:

> Image processing technology has enabled more efficient and accurate

treatment plans.

> It is time and money-saving process.

> Performance of the model will be good even with the higher user traffic.

> In Image processing, the pixels in the image can be manipulated to anydesired density and contrast.

> Since high pixel quality is generated, easy classification of skin disease is possible.

Disadvantages:

> AI-Models are Susceptible to security risks.

> Inaccuracies are still possible.

> Although AI has come a long way, human surveillance is still essential.

11. CONCLUSION

Even without a large dataset and high-quality images, it is possible to achievesufficient accuracy rates in this AI model. With accurate segmentation, we gain knowledge of the location of the disease, which is useful in the pre-processing of data used in classification as it allows the YOLO model to focus

on the area of interest. Our method provides a solution to classifying multiplediseases with higher quality and a larger quantity of data. With the assistanceof our AI-based methods, it saves time and money for patients.

12. FUTURE SCOPE

The future of AI in detecting skin diseases could include tasks that range fromsimple to complex—everything from answering the phone to medical record review, reading radiology images, making clinical diagnoses and treatment plans, and even talking with patients.AI is already at work, increasing convenience and efficiency, reducing costs and errors, and generally making iteasier for more patients to receive the health care they need. While AI is beingused in health care, it will become increasingly important for its potential to enhance patient engagement in their own care and streamline patient accessto care.

13. APPENDIX

SOURCE CODE:

```
import re
import numpy as np
import os
from flask import Flask, app,request,render_template
import sys
from flask import Flask, request, render_template,
redirect, url_for
import argparse
from tensorflow import keras
from PIL import Image
from timeit import default_timer as timer
```

```
import  test
import pandas as pd
 import numpy as np
import random
def get_parent_dir(n=1):
""" returns the n-th parent dicrectory of the current
working directory """
current_path = os.path.dirname(os.path.abspath( file
  ))
for k in range(n):
current_path = os.path.dirname(current_path)
return current_path
src_path
  =r'C:\Users\MadhuVasanth1606\Desktop\yolo_structure\2
  _Training\src'
print(src_path)
 utils_path =
  r'C:\Users\MadhuVasanth1606\Desktop\yolo_structure\Ut
  ils'
print(utils_path)
sys.path.append(src_path)
sys.path.append(utils_path)
import  argparse
from keras_yolo3.yolo import YOLO, detect_video
from PIL import Image
from timeit import default_timer as timer
from utils import load_extractor_model,
  load_features, parse_input, detect_object
import test
import  utils
import pandas as pd
import numpy as np
```

```
from Get_File_Paths import GetFileList
import random
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
# Set up folder names for default values
data_folder = os.path.join(get_parent_dir(n=1),
  "yolo_structure", "Data")
image_folder = os.path.join(data_folder,
  "Source_Images")
image_test_folder = os.path.join(image_folder,
  "Test_Images")
detection_results_folder = os.path.join(image_folder,
"Test_Image_Detection_Results")
 detection_results_file =
  os.path.join(detection_results_folder,
"Detection_Results.csv")
model_folder = os.path.join(data_folder,
  "Model_Weights")
model_weights = os.path.join(model_folder,
  "trained_weights_final.h5")
model_classes = os.path.join(model_folder,
  "data_classes.txt")
anchors_path = os.path.join(src_path, "keras_yolo3",
  "model_data",
"yolo_anchors.txt")
FLAGS = None
from cloudant.client import Cloudant
# Authenticate using an IAM API key
 client = Cloudant.iam('5b73f72f-2449-4298-88e8-
  3f887f8bbd2dbluemix','t3wXXORf8KoIMLzYFX2sk4e22uluSBK
  hM9-K4Q5b1zuK',
connect=True)
# Create a database using an initialized client
```

```
my_database = client.create_database('skindisease')
app=Flask( name )
#default home page or route
@app.route('/')
def index():
return render_template('index.html')
@app.route('/index.html')
def home():
return render_template("index.html")
#registration  page
@app.route('/register')
def register():
return render_template('register.html')
@app.route('/afterreg', methods=['POST'])
def afterreg():
x = [x for x in request.form.values()]
print(x)
data = {
'_id': x[1], # Setting _id is optional
'name': x[0],
'psw':x[2]
}
print(data)
query = {'_id': {'$eq': data['_id']}}
docs = my_database.get_query_result(query)
print(docs)
print(len(docs.all()))
if(len(docs.all())==0):
url = my_database.create_document(data)
#response = requests.get(url)
return render_template('register.html',
  pred="Registration Successful, please
```

```
90 login using your details")
91 else:
92 return render_template('register.html', pred="You are
   already a member, please
93 login using your details")
94 #login page
95 @app.route('/login')
96 def login():
97 return render_template('login.html')
98 @app.route('/afterlogin',methods=['POST'])
99 def afterlogin():
100                                          user =
   request.form['_id']
101                                          passw =
   request.form['psw']
102                                          print(user,pas
   sw)
103                                          query =
   {'_id': {'$eq': user}}
104                                          docs =
   my_database.get_query_result(query)
105                                          print(docs)
106                                          print(len(docs
   .all()))
107                                          if(len(docs.al
   l())==0):
108                                          return
   render_template('login.html', pred="The username is
   not found.")
109                                          else:
110                                          if((user==docs
   [0][0]['_id'] and passw==docs[0][0]['psw'])):
```

```
111                                     return
  redirect(url_for('prediction'))
112                                     else:
113                                     print('Invalid
  User')
114                                     @app.route('/l
  ogout')
115                                     def logout():
116                                     return
  render_template('logout.html')
117                                     @app.route('/p
  rediction')
118                                     def
  prediction():
119                                     return
  render_template('prediction.html')
120                                     @app.route('/r
  esult',methods=["GET","POST"])
121                                     def res():
122                                     # Delete all
  default flags
123                                     parser =
  argparse.ArgumentParser(argument_default=argparse.SUP
  PRESS)
124                                     """
125                                     Command line
  options
126                                     """
127                                     parser.add_arg
  ument(
128                                     "--
  input_path",
```

```
129                                         type=str,
130                                         default=image_
  test_folder,
131                                         help="Path to
  image/video directory. All subdirectories will be
  included. Default
132                                         is "
133                                         )
134                                         +
  image_test_folder,
135                                         parser.add_arg
  ument(
136                                         "--output",
137                                         type=str,
138                                         default=detect
  ion_results_folder,
139                                         help="Output
  path for detection results. Default is "
140                                         +
  detection_results_folder,
141                                         )
142                                         parser.add_arg
  ument(
143                                         "--
  no_save_img",
144                                         default=False,
145                                         action="store_
  true",
146                                         help="Only
  save bounding box coordinates but do not save output
  images with
147                                         annotated
```

```
    boxes. Default is False.",
148                                         )
149                                         parser.add_arg
    ument(
150                                         "--
    file_types",
151                                         "--names-
    list",
152                                         nargs="*",
153                                         default=[],
154                                         help="Specify
    list of file types to include. Default is --
    file_types .jpg .jpeg .png
155                                         .mp4",
156                                         )
157                                         parser.add_arg
    ument(
158                                         "--
    yolo_model",
159                                         type=str,
160                                         dest="model_pa
    th",
161                                         default=model_
    weights,
162                                         help="Path to
    pre-trained weight files. Default is " +
    model_weights,
163                                         )
164                                         parser.add_arg
    ument(
165                                         "--anchors",
166                                         type=str,
167                                         dest="anchors_
```

```
  path",
168                                         default=anchor
  s_path,
169                                         help="Path to
  YOLO anchors. Default is " + anchors_path,
170                                         )
171                                         parser.add_arg
  ument(
172                                         "--classes",
173                                         type=str,
174                                         dest="classes_
  path",
175                                         default=model_
  classes,
176                                         help="Path to
  YOLO class specifications. Default is " +
  model_classes,
177                                         )
178                                         parser.add_arg
  ument(
179                                         "--gpu_num",
  type=int, default=1, help="Number of GPU to use.
  Default is 1"
180                                         )
181                                         parser.add_arg
  ument(
182                                         "--
  confidence",
183                                         type=float,
184                                         dest="score",
185                                         default=0.25,
186                                         help="Thresho
  ld for YOLO object confidence score to show
```

```
  predictions. Default
187                                     is 0.25.",
188                                     )
189                                     parser.add_arg
  ument(
190                                     "--box_file",
191                                     type=str,
192                                     dest="box",
193                                     default=detect
  ion_results_file,
194                                     help="File to
  save bounding box results to. Default is "
195                                     +
  detection_results_file,
196                                     )
197                                     parser.add_arg
  ument(
198                                     "--postfix",
199                                     type=str,
200                                     dest="postfix
  ",
201                                     default="_dise
  ase",
202                                     help='Specify
  the postfix for images with bounding boxes. Default
  is "_disease"',
203                                     )
204                                     FLAGS =
  parser.parse_args()
205                                     save_img = not
  FLAGS.no_save_img
206                                     file_types =
```

```
  FLAGS.file_types
207                                         #print(input_p
  ath)
208                                         if file_types:
209                                         input_paths =
  GetFileList(FLAGS.input_path, endings=file_types)
210                                         print(input_pa
  ths)
211                                         else:
212                                         input_paths =
  GetFileList(FLAGS.input_path)
213                                         print(input_pa
  ths)
214                                         # Split images
  and videos
215                                         img_endings =
  (".jpg", ".jpeg", ".png")
216                                         vid_endings =
  (".mp4", ".mpeg", ".mpg", ".avi")
217                                         input_image_pa
  ths = []
218                                         input_video_pa
  ths = []
219                                         for item in
  input_paths:
220                                         if
  item.endswith(img_endings):
221                                         input_image_pa
  ths.append(item)
222                                         elif
  item.endswith(vid_endings):
223                                         input_video_pa
```

```
    ths.append(item)
224                                         output_path =
    FLAGS.output
225                                         if not
    os.path.exists(output_path):
226                                         os.makedirs(ou
    tput_path)
227                                         # define YOLO
    detector
228                                         yolo = YOLO(
229                                         **{
230                                         "model_path":
    FLAGS.model_path,
231                                         "anchors_path
    ": FLAGS.anchors_path,
232                                         "classes_path
    ": FLAGS.classes_path,
233                                         "score":
    FLAGS.score,
234                                         "gpu_num":
    FLAGS.gpu_num,
235                                         "model_image_s
    ize": (416, 416),
236                                         }
237                                         )
238                                         # Make a
    dataframe for the prediction outputs
239                                         out_df =
    pd.DataFrame(
240                                         columns=[
241                                         "image",
242                                         "image_path",
```

```
243                                             "xmin",
244                                             "ymin",
245                                             "xmax",
246                                             "ymax",
247                                             "label",
248                                             "confidence",
249                                             "x_size",
250                                             "y_size",
251                                             ]
252                                             )
253                                             # labels to
  draw on images
254                                             class_file =
  open(FLAGS.classes_path, "r")
255                                             input_labels =
  [line.rstrip("\n") for line in
  class_file.readlines()]
256                                             print("Found
  {} input labels: {} ...".format(len(input_labels),
  input_labels))
257                                             if
  input_image_paths:
258                                             print(
259                                             "Found {}
  input images: {} ...".format(
260                                             len(input_imag
  e_paths),
261                                             [os.path.basen
  ame(f) for f in input_image_paths[:5]],
262                                             )
263                                             )
264                                             start =
```

```
  timer()
265                                         text_out = ""
266                                         # This is for
  images
267                                         for i,
  img_path in enumerate(input_image_paths):
268                                         print(img_pat
  h)
269                                         prediction,
  image,lat,lon= detect_object(
270                                         yolo,
271                                         img_path,
272                                         save_img=save_
  img,
273                                         save_img_path=
  FLAGS.output,
274                                         postfix=FLAGS.
  postfix,
275                                         )
276                                         print(lat,lon)
277                                         y_size,
  x_size, _ = np.array(image).shape
278                                         for
  single_prediction in prediction:
279                                         out_df =
  out_df.append(
280                                         pd.DataFrame(
281                                         [
282                                         [
283                                         os.path.basena
  me(img_path.rstrip("\n")),
284                                         img_path.rstri
```

```
  p("\n"),
285                                     ]
286                                     +
  single_prediction
287                                     + [x_size,
  y_size]
288                                     ],
289                                     columns=[
290                                     "image",
291                                     "image_path",
292                                     "xmin",
293                                     "ymin",
294                                     "xmax",
295                                     "ymax",
296                                     "label",
297                                     "confidence",
298                                     "x_size",
299                                     "y_size",
300                                     ],
301                                     )
302                                     )
303                                     end = timer()
304                                     print(
305                                     "Processed {}
  images in {:.1f}sec - {:.1f}FPS".format(
306                                     len(input_imag
  e_paths),
307                                     end - start,
308                                     len(input_imag
  e_paths) / (end - start),
309                                     )
310                                     )
311                                     out_df.to_csv(
```

```
  FLAGS.box, index=False)
312                                         # This is for
  videos
313                                         if
  input_video_paths:
314                                         print(
315                                         "Found {}
  input videos: {} ...".format(
316                                         len(input_vide
  o_paths),
317                                         [os.path.basen
  ame(f) for f in input_video_paths[:5]],
318                                         )
319                                         )
320                                         start =
  timer()
321                                         for i,
  vid_path in enumerate(input_video_paths):
322                                         output_path =
  os.path.join(
323                                         FLAGS.output,
324                                         os.path.basena
  me(vid_path).replace(".", FLAGS.postfix + "."),
325                                         )
326                                         detect_video(y
  olo, vid_path, output_path=output_path)
327                                         end = timer()
328                                         print(
329                                         "Processed {}
  videos in {:.1f}sec".format(
330                                         len(input_vide
  o_paths), end - start
```

```
331                                                 )
332                                                 )
333                                                 # Close the
  current yolo session
334                                                 yolo.close_ses
  sion()
335                                                 return
  render_template('prediction.html')
336                                                 """ Running
  our application """
337                                                 if name == "
  main ":
338                                                 app.run(debug=
  True)
```