A Project Report on

REAL TIME SIGN LANGUAGE CONVERTER

Submitted to

The Department of Information Technology
In partial fulfillment of the academic requirements of
Jawaharlal Nehru Technological University
For
The award of the degree of

Bachelor of Technology
in
Information Technology

By
M. Anirudh - 18311A1230
T. Amar - 18311A1253

Under the Guidance of
Mrs. Lalitha
Associate Professor

Sreenidhi Institute of Science and Technology
Yamnampet, Ghatkesar, R.R. District, Hyderabad - 501301

Affiliated to
Jawaharlal Nehru Technology University
Hyderabad - 500085
Department of Information Technology

Sreenidhi Institute of Science and Technology

The Department of Information Technology

CERTIFICATE

This is to certify that this project report on "**Real Time Sign Language Converter**", submitted by M. Anirudh(18311A1230), T. Amar (18311A1253) in requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in Information Technology is a bonafide work that has been carried outby them as part of their Mini Project, under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Internal guide	**Project Coordinator**	**Head of Department**
Mrs. Lalitha	Dr. K. Vijaya Lakshmi	Dr. Sunil Bhutada
Associate Professor	Professor	Professor & HOD
Department of IT	Department of IT	Department of IT

External Examiner

DECLARATION

We, **M. Anirudh(18311A1230), T. Amar(18311A1253)** students of **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY, YAMNAMPET, *GHATKESAR, of INFORMATION TECHNOLOGY solemnly declare that the project work, titled "Real Time Sign Language Converter" is submitted to SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY for partial fulfillment for the award of degree of Bachelor of technology in INFORMATION TECHNOLOGY.***

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for award of any degree.

ACKNOWLEDGEMENT

We would like to express my gratitude to all the people behind the screen who helped me to transform an idea into a real application.

We profoundly thank **Shri. K. Abhijit Kumar, Chief Executive Officer, SNIST** who has been an excellent guide and also a great source of inspiration to our work.

We profoundly thank **Dr. C.V. Tomy, Executive Director, SNIST** who has been an excellent guide and also a great source of inspiration to our work.

We would like to express my heart-felt gratitude to my parents without whom I would not have been privileged to achieve and fulfill my dreams. We are grateful to our principal, **Dr. T. Ch. Siva Reddy,** who most ably run the institution and has had the major hand in enabling me to do our project.

We profoundly thank **Dr. Sunil Bhutada ,** Head of the Department of Information Technology who has been an excellent guide and also a great source of inspiration to our work.

We would like to thank my internal guide **Mrs. Lalitha** & coordinator **Dr. K. Vijaya Lakshmi,** for her technical guidance, constant encouragement and support in carrying out my project at college.

The satisfaction and euphoria that accompany the successful completion of the task would be great but incomplete without the mention of the people who made it possible with their constantguidance and encouragement crowns all the efforts with success. In this context, we would like thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

M. ANIRUDH – 18311A1230
T. AMAR – 18311A1253

ABSTRACT:

Sign language is one of the oldest and most natural form of language for communication, but most people do not know sign language and interpreters are very difficult to come by. Sign Language Recognition is one of the most growing fields of research area. Many new techniques have been developed recently in this area. This project is used for interfacing between computer and human using hand gesture. We wish to make a windows-based application for live motion gesture recognition using webcam input in PYTHON. This project is a combination of live motion detection and gesture identification. This application uses the webcam to detect gesture made by the user and perform basic operations accordingly. The user has to perform a particular gesture. The webcam captures this and identifies the gesture, recognizes it (against a set of known gestures) and performs the action corresponding to it.

INDEX

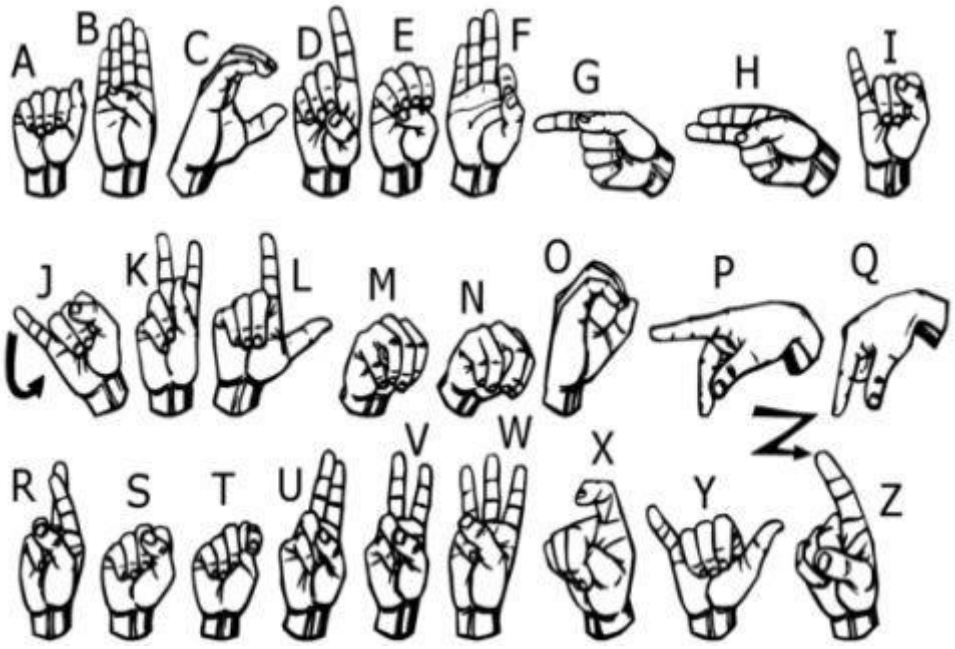
1. INTRODUCTION

The most extensively used sign language is American sign language. Because people with D&M are unable to communicate via spoken languages, sign language is their only means of communication. Communication is the process of conveying ideas and opinions using a range of means such as speech, signs, and images. Deaf and dumb people (D&M) interact with others by utilising a variety of hand gestures. Nonverbally transmitted concepts and thoughts that may be noticed with the naked eye are known as gestures. Sign language is a kind of nonverbal communication used by the deaf and dumb.

Sign language is a visual language with three main components:

Fingerspelling	Word level sign vocabulary	Non-manual features
Used to spell words letter by letter .	Used for the majority of communication.	Facial expressions and tongue, mouth and body position.

In our project we principally concentrate on producing a model which can identify Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.



1.1. Motivation:

A language connection is constructed as a sign language framework that is different from standard text for communication between normal people and D&M persons. As a result, they rely on vision-based communication to exchange ideas.

The gestures can be fluently understood by others if there is a common interface that transforms sign language to text. Numerous experiments have been conducted in order to develop a vision-based interface system that allows D&M persons to communicate without having to speak the same language.

The goal is to create a user-friendly human computer interface (HCI) that recognises our sign language. There are several sign languages used throughout the world, including American Sign Language (ASL), French Sign Language (FSL), British Sign Language (BSL), and Indian Sign Language (ISL), Japanese Sign Language and work has been done on other languages at every corner of the world.

1.2. Objective of Project

The main goal of this project is to create a real-time sign language translator that uses TensorFlow object identification and Python to convert sign language to text.

1.3. Scope of Project:

It would give an easy way for people to interact with others using sign language, so eliminating the need for a middle person to act as a translator, and it would also create a user-friendly environment by delivering text output for the input sign gesture.

2. LITERATURE SURVEY

In recent years, there has been a lot of research into hand gesture recognition. With the help of a literature review, we determined the most basic phases in hand gesture recognition:

- Acquisition of data
- Preprocessing of data
- Feature extraction
- Classification of gesture

Data acquisition:

The following are a few of the many methods for collecting data on hand gestures:

In this technology, electromechanical devices are used to offer accurate hand configuration and position. Many glove-based approaches can be used to gather information. It is, however, both costly and inconvenient to use.

A computer camera used for vision-based activities is an input device for monitoring information from hands or fingers from a different perspective. Vision-based solutions only require a camera, providing for a natural human-computer interaction without the need for further hardware. Artificial vision systems are

usually imposed in software and/or hardware to supplement organic vision in these systems.

Dealing with the vast diversity of human hand appearances generated by a high number of hand movements, variable skin-color possibilities, and variations in view points, scales, and camera speed during image acquisition is the fundamental problem with vision-based hand detection.

Data preprocessing and Feature extraction for visionbased approach:

- In the approach for hand recognition, threshold-based color recognition is paired with backdrop subtraction. We can employ Adaboost face detector to distinguish between them because both faces and hands have the same skin hue.

- To obtain the required image for training, we can alternatively employ a filter called Gaussian blur. The filter is described in and can be easily applied with the help of OpenCV, or open computer vision.

- We can extract the relevant image that needs to be taught using instrumented gloves. When compared to applying filters to data obtained by video extraction, this saves time during preprocessing and allows us to obtain more succinct and precise data.

- We attempted to hand segment an image using colour segmentation methods, but the results were poor because skin colour and tone are highly dependent on lighting conditions, according to the research article. We also decided that instead of segmenting the hand out of an arbitrary background, we would keep the hand's background a stable single colour so that we wouldn't have to segment it on the basis of colour because we have a large number of symbols to train for our project, many of which look similar to one another, such as the gesture for symbol 'V' and number '2'.

Gesture classification:

- Hidden Markov Chains HMMs are used to categorise motions (HMM). The dynamic elements of gestures are taken into account in this method. The skin-color

blobs matching to the hand are tracked into a body–facial region centred on the user's face to extract actions from a sequence of VHS pictures. The goal is to distinguish between two types of gestures: deictic movements and symbolic motions. A rapid look–up indexing mechanism is used to filter the image. Skin colour pixels are grouped into blobs after filtering. Blobs are statistical objects that are used to estimate homogenous areas based on skin colour pixels' colorimetry (Y,U,V) and position (x,y).

- To recognise stationary hand gestures, the Nave Bayes Classifier is utilised. It's a simple and efficient algorithm. After segmentation, it extracts geometric invariants from visual input to categorise distinct motions. As a result, unlike most other identification algorithms, this one does not identify persons based on their skin colour. Each frame of the video clip with a static background is used to extract the movements. The initial stage is to segment, mark, and extract geometric invariants from the objects of interest. The next stage is to identify gestures using a K closest neighbour technique with distance weighting to generate acceptable data for a locally weighted Nave Bayes classifier (KNNDW).

- Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen, graduates of Institute of Automation Technology National Taipei University of Technology Taipei, Taiwan, build a skin model to isolate the hand out of an image and also apply binary threshold to the entire image, according to their paper "Human Hand Gesture Recognition Using a Convolution Neural Network." They calibrate the threshold image around the principal axis after capturing it to centre the image around it. Using this image, they trained and forecasted the outputs of a CNN model. They trained their model on seven hand motions, and when they used it, it had a 95 percent accuracy for those movements.

3. SYSTEM ANALYSIS

This system analysis is closely associated with necessities evaluation. it's also “an explicit formal inquiry done to help a person (called the selection-maker) become aware of a higher direction of movement and make a higher selection than he would possibly otherwise have

made." Breaking down the machine into various sections to research the problem, researching project wishes, breaking down what desires to generate, and attempting to engage clients so that specific requirements may be defined are all part of this step.

3.1. Functional Requirement

Specification Following a thorough examination, the following modules have been identified as being present in the system.

TensorFlow:

Tensorflow is a numerical computing software package that is open-source. We start by defining the nodes of the computation graph, and then we do the actual calculation within a session. In Machine Learning, TensorFlow is widely utilized.

Keras:

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly make and test the neural network with minimal lines of code. It contains executions of generally used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

OpenCV:

OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision. It is substantially usedfor image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, but still bindings are available for Python, Java, MATLAB/OCTAVE.

Numpy:

NumPy is a Python package that allows you to manipulate arrays. Functions for working with matrices, Fourier transformations, and linear algebra are also included. It's an open-supply venture that you're free to participate in. Its full name is Numerical Python. In Python, we have lists that act as arrays, but they are sluggish to method. Its purpose is to make array items 50 times faster than normal Python lists. In NumPy, the

array item is called ndarray. It also includes some helpful tips that make working with it a breeze.

3.2. Performance Requirements

• All of the system's hardware specifications are described in the performance requirements. The minimal system requirement for a deep learning project is an i3 or higher processor. The 64-bit Ubuntu operating system is commonly used in deep learning studies.

3.3. Hardware Requirements:

Processor: Any Update Processer

RAM: Min 4 GB

Hard Disk: Min 50 GB

3.4. Software Requirements:

Operating System: Windows family

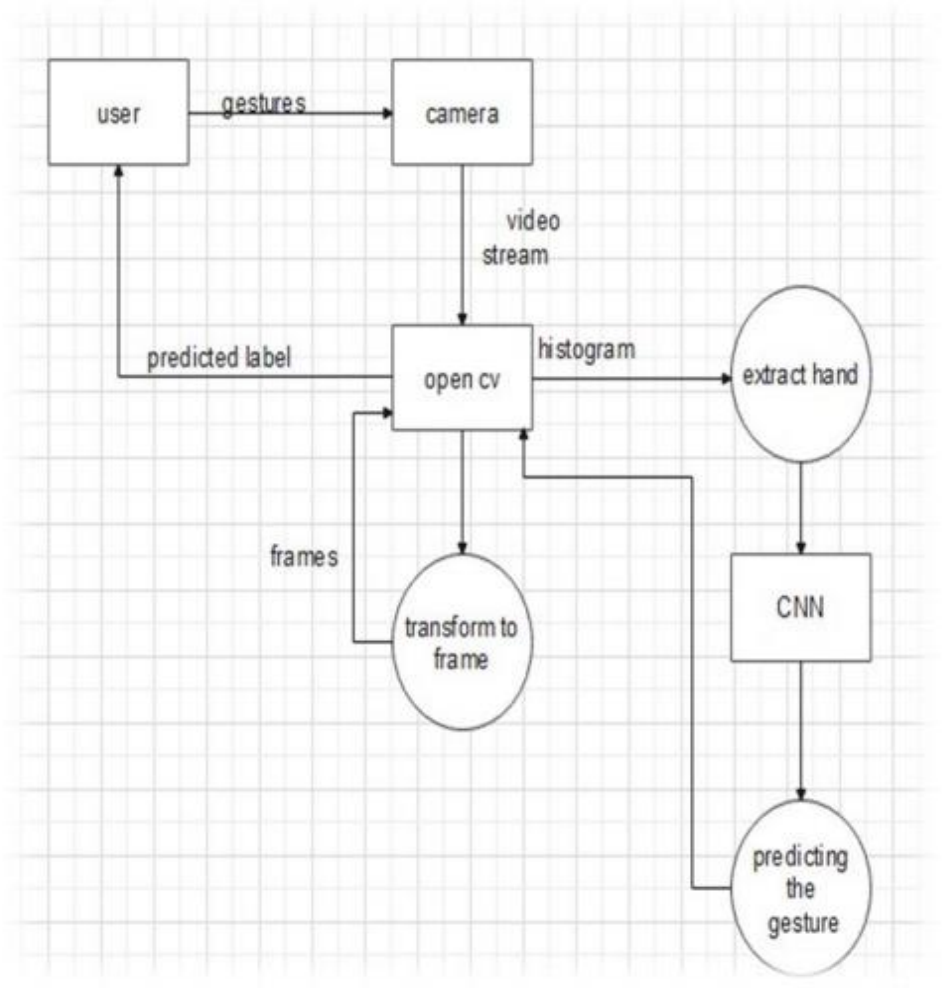
Technology: Python 3.6

IDE: Spyder, Jupyter Notebook

4. SYSTEM DESIGN

The process of arranging the architecture, components, modules, interfaces, and data of a system to achieve specific goals. It can be characterized as the application of a systems idea to improve a product. Object-oriented evaluation and design procedures are quickly becoming the most widely used methods for creating computer systems.

4.1. ARCHITECTURE



4.2. UML DIAGRAMS

UML INTRODUCTION

The Unified Modeling Language (UML) is a standard language for describing, visualizing, building, and documenting software system artefacts, as well as business modelling and non-software systems. The UML is a collection of best engineering practices for modelling large and complex systems that have been demonstrated to work. The UML is a critical component of object-oriented software development and the software development process. To express the design of software projects, the UML primarily employs graphical notations. The UML aids project teams in communicating,

exploring potential designs, and validating the software's architectural design.

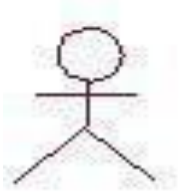
WHY IS UML USE?

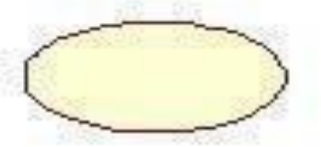
As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale.

TYPES OF UML DIAGRAM

A.USE CASE DIAGRAM

Show the relationship between the actors and the Use Cases. A use case is a collection of situations that describe a user's engagement with a technology.
The 2 important components of a use case diagram are use cases and actors.





An actor represents a user or another system that interacts with the system you're modelling. A use case is an outsider's view of the system that depicts an activity that "the user might conduct in order to achieve a task." Use cases are the most used UML diagram in projects. They're useful for identifying needs and planning projects.

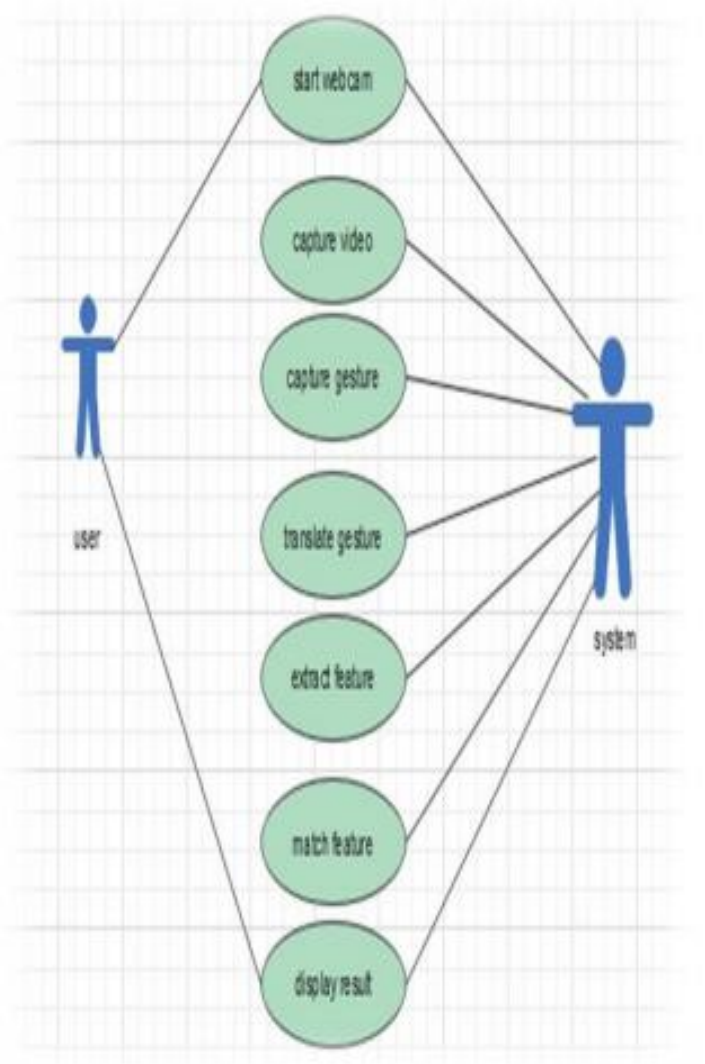
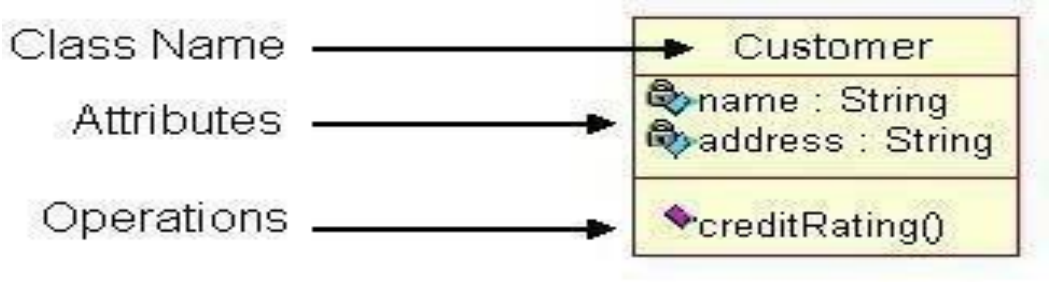


Fig: 4.1.1. Use case diagram

B. CLASS DIAGRAM

A class is a group of items that have similar characteristics, processes, relationships, and meanings. Classes that enforce one or more interface lessons are the most fundamental building blocks of any items-oriented device.



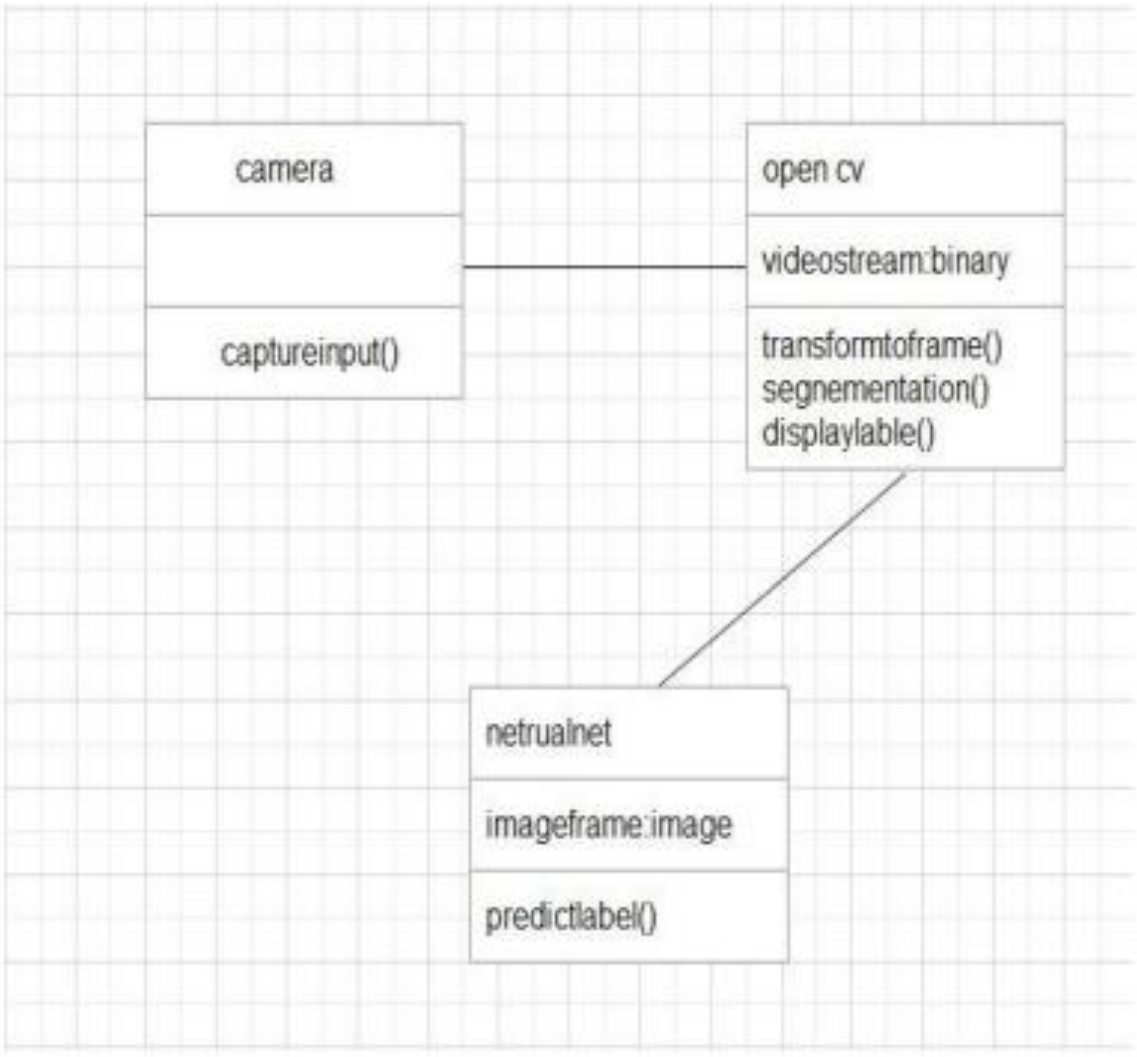


Fig: 4.1.2. class diagram

C. STATE CHART DIAGRAM

It's a type of nation chart diagram that shows how information flows from one activity to the next within a device. It is concerned with the dynamic attitude of a system. place a great emphasis on the movement of control between devices. It refers to a completely unique kingdom diagram in which the vast majority of states are action states and the vast majority of

transitions are triggered by the completion of moves within source states. Internal processing-pushed flows are the focus of this illustration. State diagrams are available in many different shapes and sizes, each with its own set of semantics.

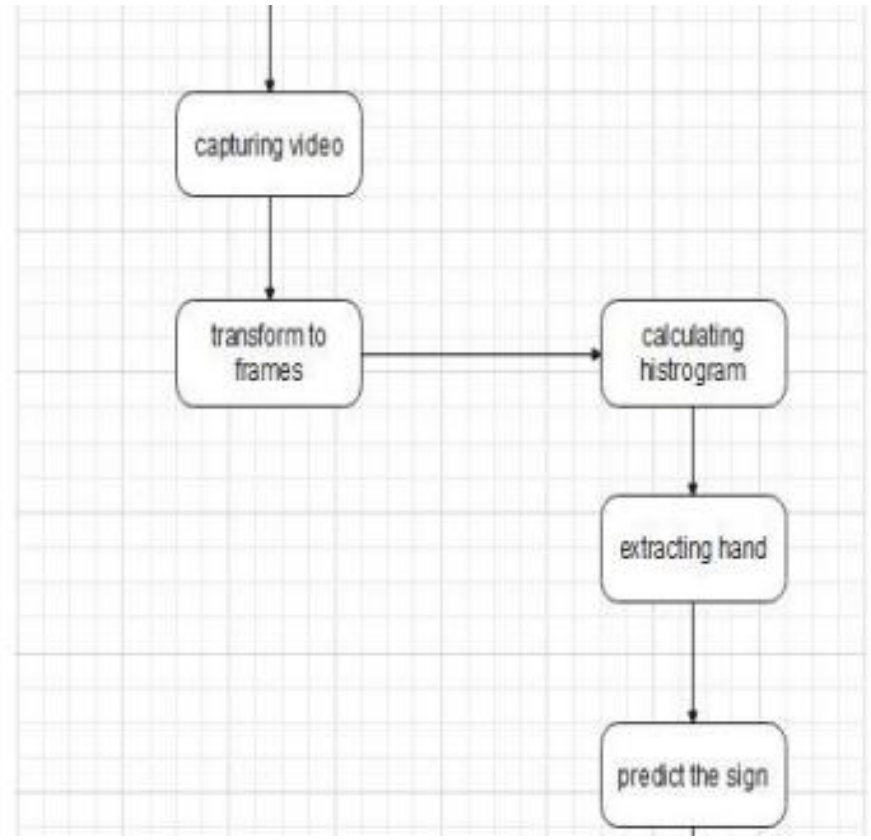


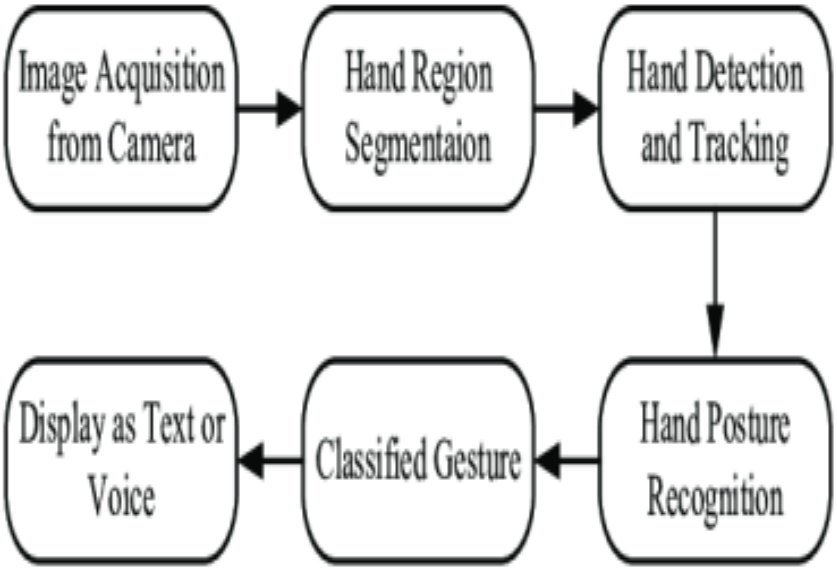
Fig: 4.1.3. state chart diagram

4.3. METHODOLOGY

Converting real time sign language into text can be classified into flow of simple steps like:

i. Recognizing a man's or woman's hand motions
ii. Developing a system learning model for picture to textual content translation
iii. Putting words together
iv. Putting sentences together
v. Creating the complete content

The flow chart of this project can be described as:



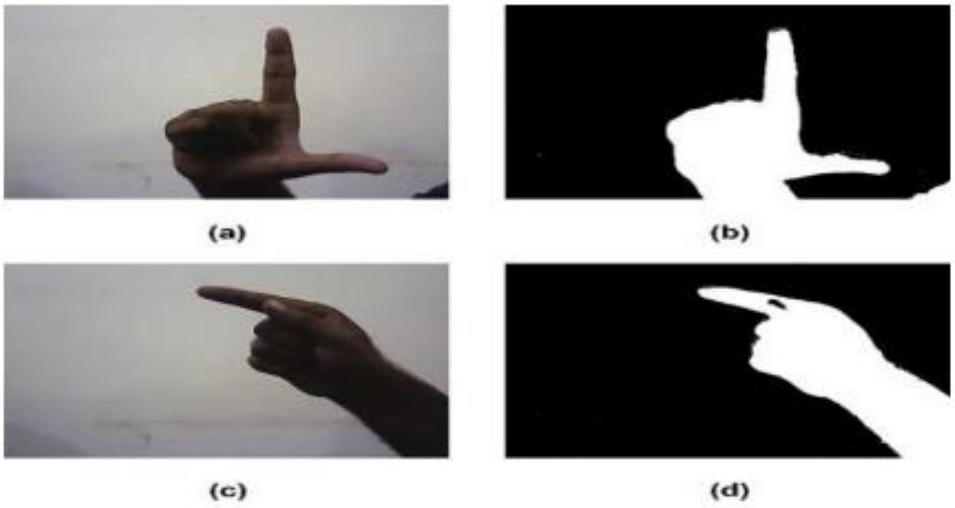
The figure above shows the phases involved in obtaining the project's goals. The detailed explanation of above mentioned steps are:

1. Image Acquisition

The camera is put to use for capturing the motions. The full signing duration is captured with the help of this OpenCV video stream. The frames are taken from the stream and converted to grayscale images with a 50*50 pixel resolution. Because the entire dataset is the same size, this dimension is consistent throughout the project.

2. Hand Region Segmentation & Hand Detection and Tracking

In the collected photographs, hand gestures are scanned. Before the image is fed into the model for prediction, this is a phase in the preprocessing process. The passages in which gestures are used have been emphasised. This increases the chances of making a successful prediction by a factor of ten.



3. Hand Posture Recognition

The preprocessed images are sent to the Keras CNN model. The model which was already trained generates the projected label. There is a probability associated with each of the gesture labels. The label which has the highest probability determines the expected label.

4. Display as Text

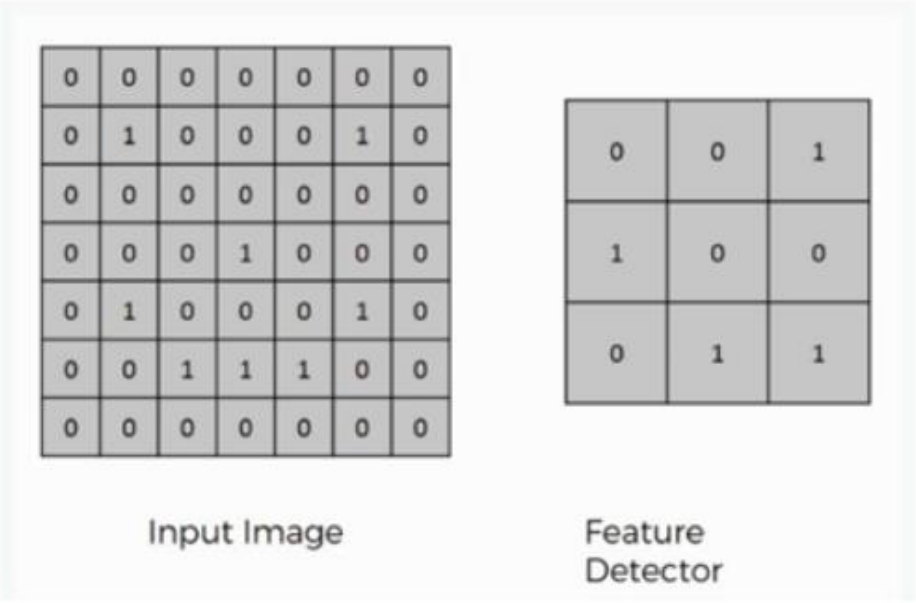
The model converts gestures, which are known, into words. The recognized words form a sentence. Therefore, forming the entire context.

5. IMPLEMENTATION

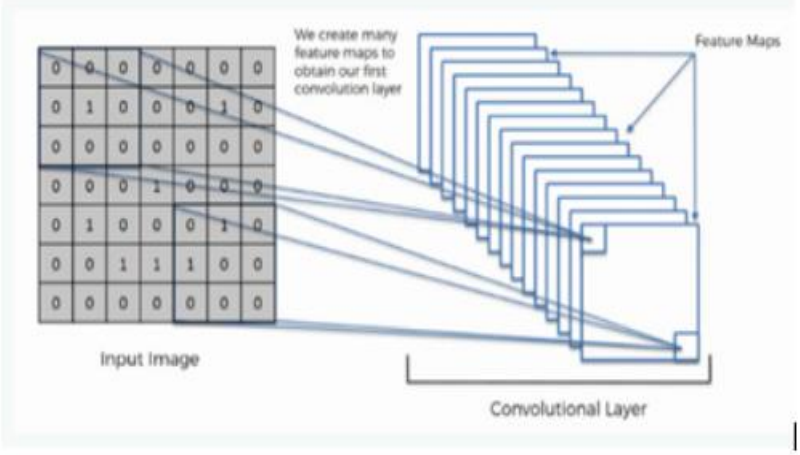
5.1 LANGUAGE OR TECHNOLOGY USED

Python: This application employs the Python programming language, which is an object-oriented programming language. Python comes with a number of application frameworks.

GitHub: For software developers, GitHub is an open-source version control and collaboration platform. GitHub, which operates on a software program as a service (SaaS) commercial enterprise model, became founded in 2008 and is predicated on Git, an open-supply code manipulation device designed by Linus Torvalds to speed up software development.



Input Image and Feature detector in Pixels Form

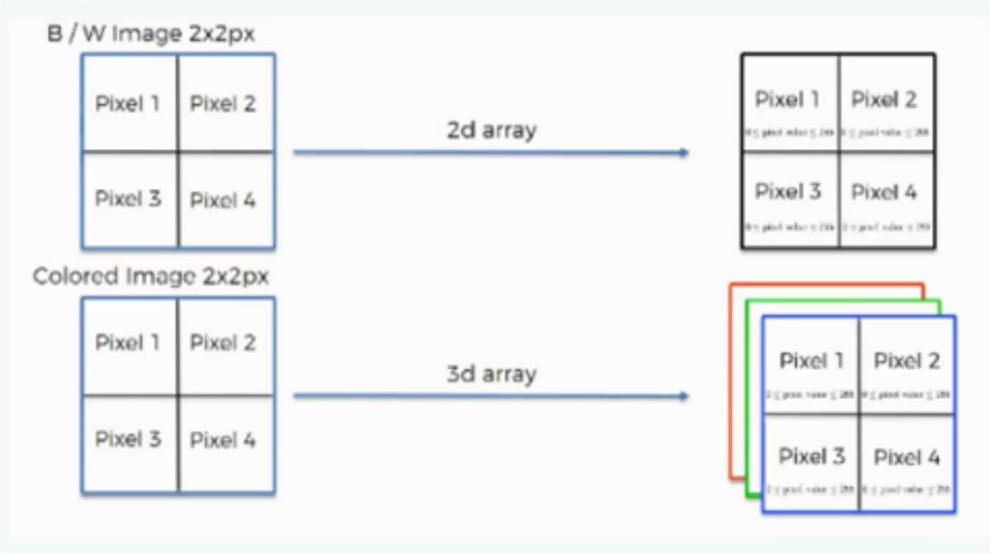


Overlapping of Feature detector

ReLU Layer

The phase's second component is the Rectified Linear Unit, or Relook.

Relu patterns and how they work in Convolutional Neural Algorithms will be discussed. Although understanding CNNs is not required, it is always a good idea to take a quick tutorial to improve your skills. e to graph the results.



Pooling Layer

This section will teach us about pooling and how it works in general.

Our nexus, on the other hand, will use a unique type of pooling: maximal pooling. We'll try a couple other approaches, such as mean (or sum) pooling.

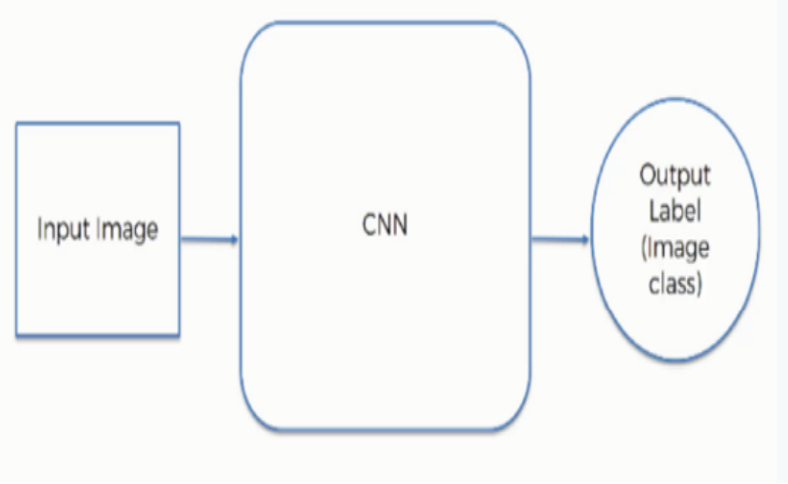
This portion will conclude with a presentation that will almost certainly include the use of a visual interactive tool to demonstrate the entire subject.

Flattening

This will be a fast description of the flatten technique and how to shift from pooling to flatten layers when working with Convolutional Neural Networks.

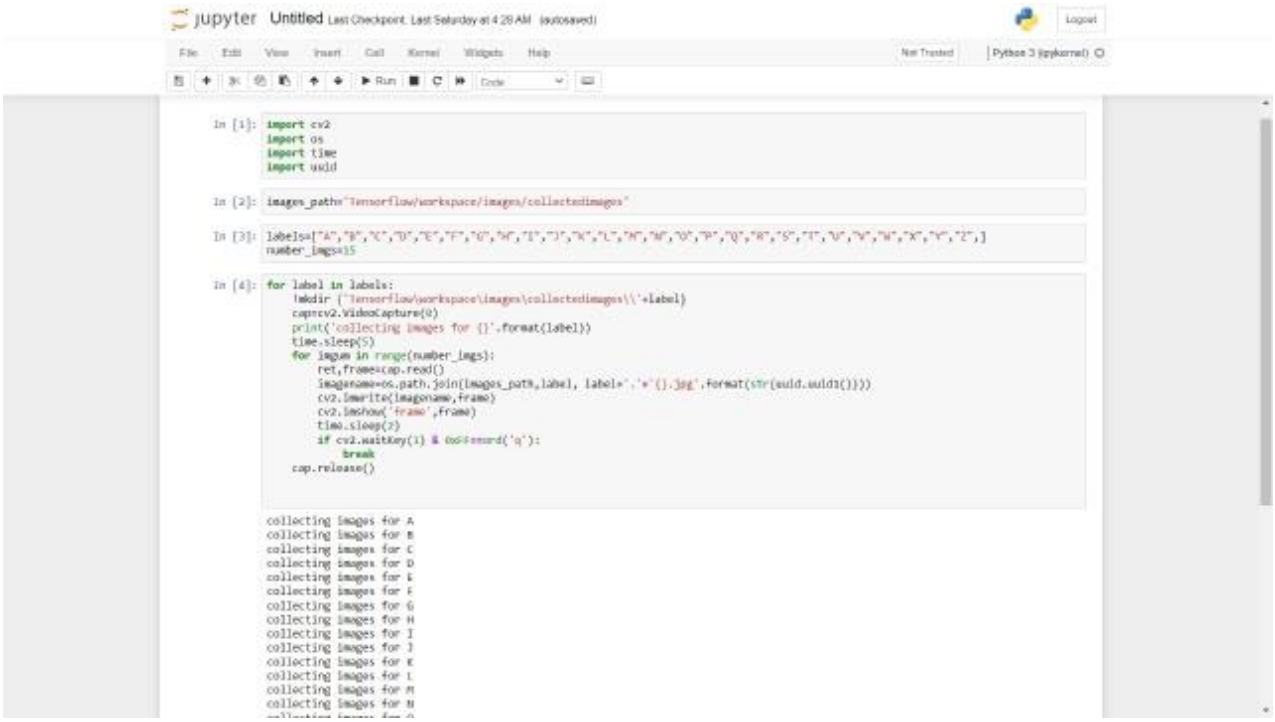
Full Connection

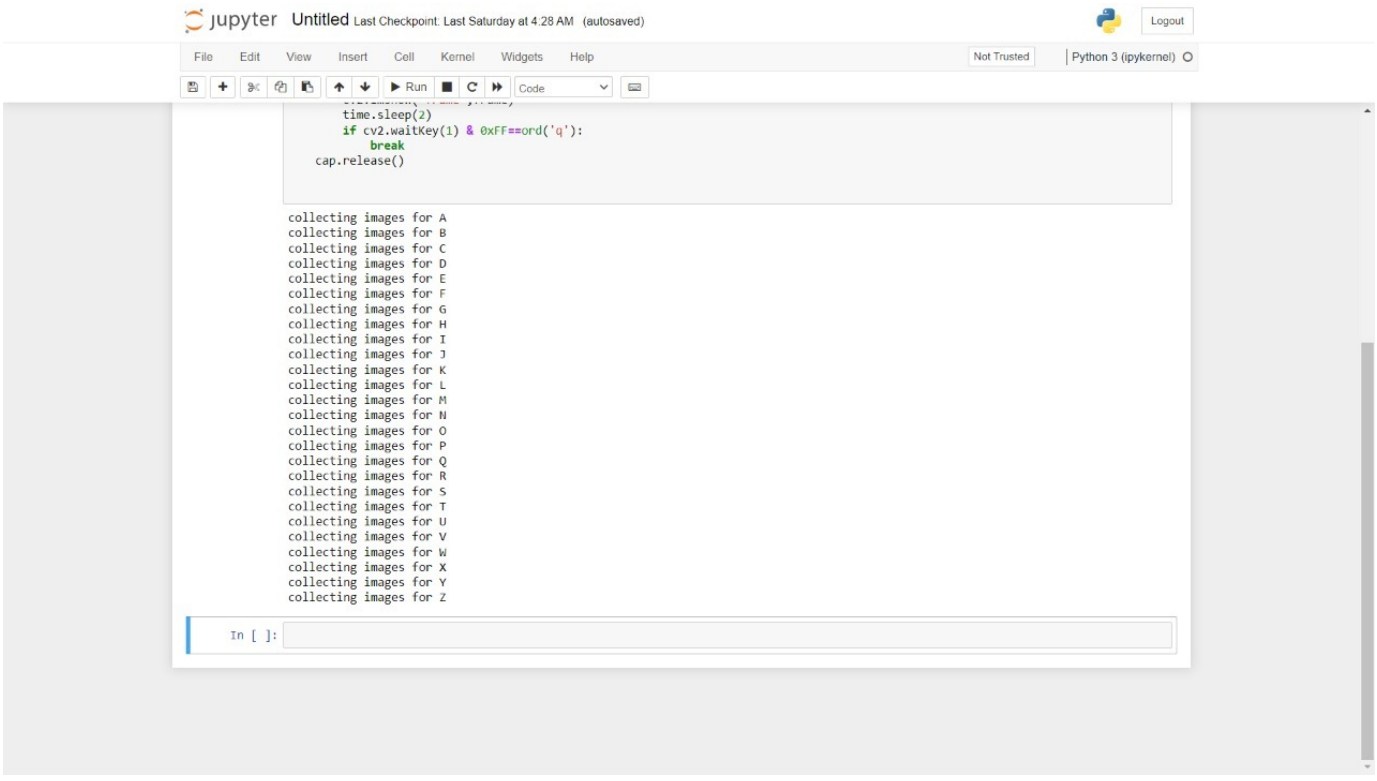
This part will bring everything we've talked about so far together. Department of Electronics and Communication Engineering 34 learning this will give us a better grasp of how Convolutional Neural Networks work and how the "neurons" that are finally generated learn to identify photographs. The basic CNN component structure is depicted in the diagram below.



5.2 CODE

- **Dataset Creation**





➢ Data Preprocessing and Labelling

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import argparse
import codecs
import os.path
import platform
import shutil
import sys
import webbrowser as wb
from functools import partial

try:
    from PyQt5.QtGui import *
    from PyQt5.QtCore import *
    from PyQt5.QtWidgets import *
except ImportError:
    # needed for py3+qt4
    # Ref:
    # http://pyqt.sourceforge.net/Docs/PyQt4/incompatible_apis.html
    # http://stackoverflow.com/questions/21217399/pyqt4-qtcore-qvariant-object-instead-of-a-string
    if sys.version_info.major >= 3:
        import sip
        sip.setapi('QVariant', 2)
    from PyQt4.QtGui import *
    from PyQt4.QtCore import *

from libs.combobox import ComboBox
from libs.default_label_combobox import DefaultLabelComboBox
from libs.resources import *
from libs.constants import *
from libs.utils import *
from libs.settings import Settings
from libs.shape import Shape, DEFAULT_LINE_COLOR, DEFAULT_FILL_COLOR
from libs.stringBundle import StringBundle
from libs.canvas import Canvas
from libs.zoomWidget import ZoomWidget
from libs.labelDialog import LabelDialog
from libs.colorDialog import ColorDialog
from libs.labelFile import LabelFile, LabelFileError, LabelFileFormat
from libs.toolBar import ToolBar
from libs.pascal_voc_io import PascalVocReader
from libs.pascal_voc_io import XML_EXT
from libs.yolo_io import YoloReader
from libs.yolo_io import TXT_EXT
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
from libs.yolo_io import TXT_EXT
from libs.create_ml_io import CreateMLReader
from libs.create_ml_io import JSON_EXT
from libs.ustr import ustr
from libs.hashableQListWidgetItem import HashableQListWidgetItem

__appname__ = 'labelImg'


class WindowMixin(object):

    def menu(self, title, actions=None):
        menu = self.menuBar().addMenu(title)
        if actions:
            add_actions(menu, actions)
        return menu

    def toolbar(self, title, actions=None):
        toolbar = ToolBar(title)
        toolbar.setObjectName(u'%sToolBar' % title)
        # toolbar.setOrientation(Qt.Vertical)
        toolbar.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        if actions:
            add_actions(toolbar, actions)
        self.addToolBar(Qt.LeftToolBarArea, toolbar)
        return toolbar


class MainWindow(QMainWindow, WindowMixin):
    FIT_WINDOW, FIT_WIDTH, MANUAL_ZOOM = list(range(3))

    def __init__(self, default_filename=None, default_prefdef_class_file=None, default_save_dir=None):
        super(MainWindow, self).__init__()
        self.setWindowTitle(__appname__)

        # Load setting in the main thread
        self.settings = Settings()
        self.settings.load()
        settings = self.settings

        self.os_name = platform.system()

        # Load string bundle for i18n
        self.string_bundle = StringBundle.get_bundle()
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        self.string_bundle = StringBundle.get_bundle()
        get_str = lambda str_id: self.string_bundle.get_string(str_id)

        # Save as Pascal voc xml
        self.default_save_dir = default_save_dir
        self.label_file_format = settings.get(SETTING_LABEL_FILE_FORMAT, LabelFileFormat.PASCAL_VOC)

        # For loading all image under a directory
        self.m_img_list = []
        self.dir_name = None
        self.label_hist = []
        self.last_open_dir = None
        self.cur_img_idx = 0
        self.img_count = len(self.m_img_list)

        # Whether we need to save or not.
        self.dirty = False

        self._no_selection_slot = False
        self._beginner = True
        self.screencast = "https://youtu.be/p0nR2YsCY_U"

        # Load predefined classes to the list
        self.load_predefined_classes(default_prefdef_class_file)

        if self.label_hist:
            self.default_label = self.label_hist[0]
        else:
            print("Not find:/data/predefined_classes.txt (optional)")

        # Main widgets and related state.
        self.label_dialog = LabelDialog(parent=self, list_item=self.label_hist)

        self.items_to_shapes = {}
        self.shapes_to_items = {}
        self.prev_label_text = ''

        list_layout = QVBoxLayout()
        list_layout.setContentsMargins(0, 0, 0, 0)

        # Create a widget for using default label
        self.use_default_label_checkbox = QCheckBox(get_str('useDefaultLabel'))
        self.use_default_label_checkbox.setChecked(False)
        self.default_label_combo_box = DefaultLabelComboBox(self,items=self.label_hist)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        self.use_default_label_checkbox.setChecked(False)
        self.default_label_combo_box = DefaultLabelComboBox(self,items=self.label_hist)

        use_default_label_qhbox_layout = QHBoxLayout()
        use_default_label_qhbox_layout.addWidget(self.use_default_label_checkbox)
        use_default_label_qhbox_layout.addWidget(self.default_label_combo_box)
        use_default_label_container = QWidget()
        use_default_label_container.setLayout(use_default_label_qhbox_layout)

        # Create a widget for edit and diffc button
        self.diffc_button = QCheckBox(get_str('useDifficult'))
        self.diffc_button.setChecked(False)
        self.diffc_button.stateChanged.connect(self.button_state)
        self.edit_button = QToolButton()
        self.edit_button.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)

        # Add some of widgets to list_layout
        list_layout.addWidget(self.edit_button)
        list_layout.addWidget(self.diffc_button)
        list_layout.addWidget(use_default_label_container)

        # Create and add combobox for showing unique labels in group
        self.combo_box = ComboBox(self)
        list_layout.addWidget(self.combo_box)

        # Create and add a widget for showing current label items
        self.label_list = QListWidget()
        label_list_container = QWidget()
        label_list_container.setLayout(list_layout)
        self.label_list.itemActivated.connect(self.label_selection_changed)
        self.label_list.itemSelectionChanged.connect(self.label_selection_changed)
        self.label_list.itemDoubleClicked.connect(self.edit_label)
        # Connect to itemChanged to detect checkbox changes.
        self.label_list.itemChanged.connect(self.label_item_changed)
        list_layout.addWidget(self.label_list)



        self.dock = QDockWidget(get_str('boxLabelText'), self)
        self.dock.setObjectName(get_str('labels'))
        self.dock.setWidget(label_list_container)

        self.file_list_widget = QListWidget()
        self.file_list_widget.itemDoubleClicked.connect(self.file_item_double_clicked)
        file_list_layout = QVBoxLayout()
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        self.file_list_widget = QListWidget()
        self.file_list_widget.itemDoubleClicked.connect(self.file_item_double_clicked)
        file_list_layout = QVBoxLayout()
        file_list_layout.setContentsMargins(0, 0, 0, 0)
        file_list_layout.addWidget(self.file_list_widget)
        file_list_container = QWidget()
        file_list_container.setLayout(file_list_layout)
        self.file_dock = QDockWidget(get_str('fileList'), self)
        self.file_dock.setObjectName(get_str('files'))
        self.file_dock.setWidget(file_list_container)

        self.zoom_widget = ZoomWidget()
        self.color_dialog = ColorDialog(parent=self)

        self.canvas = Canvas(parent=self)
        self.canvas.zoomRequest.connect(self.zoom_request)
        self.canvas.set_drawing_shape_to_square(settings.get(SETTING_DRAW_SQUARE, False))

        scroll = QScrollArea()
        scroll.setWidget(self.canvas)
        scroll.setWidgetResizable(True)
        self.scroll_bars = {
            Qt.Vertical: scroll.verticalScrollBar(),
            Qt.Horizontal: scroll.horizontalScrollBar()
        }
        self.scroll_area = scroll
        self.canvas.scrollRequest.connect(self.scroll_request)

        self.canvas.newShape.connect(self.new_shape)
        self.canvas.shapeMoved.connect(self.set_dirty)
        self.canvas.selectionChanged.connect(self.shape_selection_changed)
        self.canvas.drawingPolygon.connect(self.toggle_drawing_sensitive)

        self.setCentralWidget(scroll)
        self.addDockWidget(Qt.RightDockWidgetArea, self.dock)
        self.addDockWidget(Qt.RightDockWidgetArea, self.file_dock)
        self.file_dock.setFeatures(QDockWidget.DockWidgetFloatable)

        self.dock_features = QDockWidget.DockWidgetClosable | QDockWidget.DockWidgetFloatable
        self.dock.setFeatures(self.dock.features() ^ self.dock_features)

        # Actions
        action = partial(new_action, self)
        quit = action(get_str('quit'), self.close,
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        quit = action(get_str('quit'), self.close,
                      'Ctrl+Q', 'quit', get_str('quitApp'))

        open = action(get_str('openFile'), self.open_file,
                      'Ctrl+O', 'open', get_str('openFileDetail'))

        open_dir = action(get_str('openDir'), self.open_dir_dialog,
                          'Ctrl+u', 'open', get_str('openDir'))

        change_save_dir = action(get_str('changeSaveDir'), self.change_save_dir_dialog,
                                 'Ctrl+r', 'open', get_str('changeSavedAnnotationDir'))

        open_annotation = action(get_str('openAnnotation'), self.open_annotation_dialog,
                                 'Ctrl+Shift+O', 'open', get_str('openAnnotationDetail'))
        copy_prev_bounding = action(get_str('copyPrevBounding'), self.copy_previous_bounding_boxes, 'Ctrl+v', 'copy',
get_str('copyPrevBounding'))

        open_next_image = action(get_str('nextImg'), self.open_next_image,
                                 'd', 'next', get_str('nextImgDetail'))

        open_prev_image = action(get_str('prevImg'), self.open_prev_image,
                                 'a', 'prev', get_str('prevImgDetail'))

        verify = action(get_str('verifyImg'), self.verify_image,
                        'space', 'verify', get_str('verifyImgDetail'))

        save = action(get_str('save'), self.save_file,
                      'Ctrl+S', 'save', get_str('saveDetail'), enabled=False)

        def get_format_meta(format):
            """
            returns a tuple containing (title, icon_name) of the selected format
            """
            if format == LabelFileFormat.PASCAL_VOC:
                return '&PascalVOC', 'format_voc'
            elif format == LabelFileFormat.YOLO:
                return '&YOLO', 'format_yolo'
            elif format == LabelFileFormat.CREATE_ML:
                return '&CreateML', 'format_createml'

        save_format = action(get_format_meta(self.label_file_format)[0],
                             self.change_format, 'Ctrl+Y',
                             get_format_meta(self.label_file_format)[1],
                             get_str('changeSaveFormat'), enabled=True)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
                            get_str('changeSaveFormat'), enabled=True)

        save_as = action(get_str('saveAs'), self.save_file_as,
                         'Ctrl+Shift+S', 'save-as', get_str('saveAsDetail'), enabled=False)

        close = action(get_str('closeCur'), self.close_file, 'Ctrl+W', 'close', get_str('closeCurDetail'))

        delete_image = action(get_str('deleteImg'), self.delete_image, 'Ctrl+Shift+D', 'close', get_str('deleteImgDetail'))

        reset_all = action(get_str('resetAll'), self.reset_all, None, 'resetall', get_str('resetAllDetail'))

        color1 = action(get_str('boxLineColor'), self.choose_color1,
                        'Ctrl+L', 'color_line', get_str('boxLineColorDetail'))

        create_mode = action(get_str('crtBox'), self.set_create_mode,
                             'w', 'new', get_str('crtBoxDetail'), enabled=False)
        edit_mode = action(get_str('editBox'), self.set_edit_mode,
                           'Ctrl+J', 'edit', get_str('editBoxDetail'), enabled=False)

        create = action(get_str('crtBox'), self.create_shape,
                        'w', 'new', get_str('crtBoxDetail'), enabled=False)
        delete = action(get_str('delBox'), self.delete_selected_shape,
                        'Delete', 'delete', get_str('delBoxDetail'), enabled=False)
        copy = action(get_str('dupBox'), self.copy_selected_shape,
                      'Ctrl+D', 'copy', get_str('dupBoxDetail'),
                      enabled=False)

        advanced_mode = action(get_str('advancedMode'), self.toggle_advanced_mode,
                               'Ctrl+Shift+A', 'expert', get_str('advancedModeDetail'),
                               checkable=True)

        hide_all = action(get_str('hideAllBox'), partial(self.toggle_polygons, False),
                          'Ctrl+H', 'hide', get_str('hideAllBoxDetail'),
                          enabled=False)
        show_all = action(get_str('showAllBox'), partial(self.toggle_polygons, True),
                          'Ctrl+A', 'hide', get_str('showAllBoxDetail'),
                          enabled=False)

        help_default = action(get_str('tutorialDefault'), self.show_default_tutorial_dialog, None, 'help', get_str('tutorialDetail'))
        show_info = action(get_str('info'), self.show_info_dialog, None, 'help', get_str('info'))
        show_shortcut = action(get_str('shortcut'), self.show_shortcuts_dialog, None, 'help', get_str('shortcut'))

        zoom = QWidgetAction(self)
        zoom.setDefaultWidget(self.zoom_widget)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        zoom.setDefaultWidget(self.zoom_widget)
        self.zoom_widget.setWhatsThis(
            u"Zoom in or out of the image. Also accessible with"
            " %s and %s from the canvas." % (format_shortcut("Ctrl+[-+]"),
                                             format_shortcut("Ctrl+Wheel")))
        self.zoom_widget.setEnabled(False)

        zoom_in = action(get_str('zoomin'), partial(self.add_zoom, 10),
                         'Ctrl++', 'zoom-in', get_str('zoominDetail'), enabled=False)
        zoom_out = action(get_str('zoomout'), partial(self.add_zoom, -10),
                          'Ctrl+-', 'zoom-out', get_str('zoomoutDetail'), enabled=False)
        zoom_org = action(get_str('originalsize'), partial(self.set_zoom, 100),
                          'Ctrl+=', 'zoom', get_str('originalsizeDetail'), enabled=False)
        fit_window = action(get_str('fitWin'), self.set_fit_window,
                            'Ctrl+F', 'fit-window', get_str('fitWinDetail'),
                            checkable=True, enabled=False)
        fit_width = action(get_str('fitWidth'), self.set_fit_width,
                           'Ctrl+Shift+F', 'fit-width', get_str('fitWidthDetail'),
                           checkable=True, enabled=False)
        # Group zoom controls into a list for easier toggling.
        zoom_actions = (self.zoom_widget, zoom_in, zoom_out,
                        zoom_org, fit_window, fit_width)
        self.zoom_mode = self.MANUAL_ZOOM
        self.scalers = {
            self.FIT_WINDOW: self.scale_fit_window,
            self.FIT_WIDTH: self.scale_fit_width,
            # Set to one to scale to 100% when loading files.
            self.MANUAL_ZOOM: lambda: 1,
        }

        edit = action(get_str('editLabel'), self.edit_label,
                      'Ctrl+E', 'edit', get_str('editLabelDetail'),
                      enabled=False)
        self.edit_button.setDefaultAction(edit)

        shape_line_color = action(get_str('shapeLineColor'), self.choose_shape_line_color,
                                  icon='color_line', tip=get_str('shapeLineColorDetail'),
                                  enabled=False)
        shape_fill_color = action(get_str('shapeFillColor'), self.choose_shape_fill_color,
                                  icon='color', tip=get_str('shapeFillColorDetail'),
                                  enabled=False)

        labels = self.dock.toggleViewAction()
        labels.setText(get_str('showHide'))
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

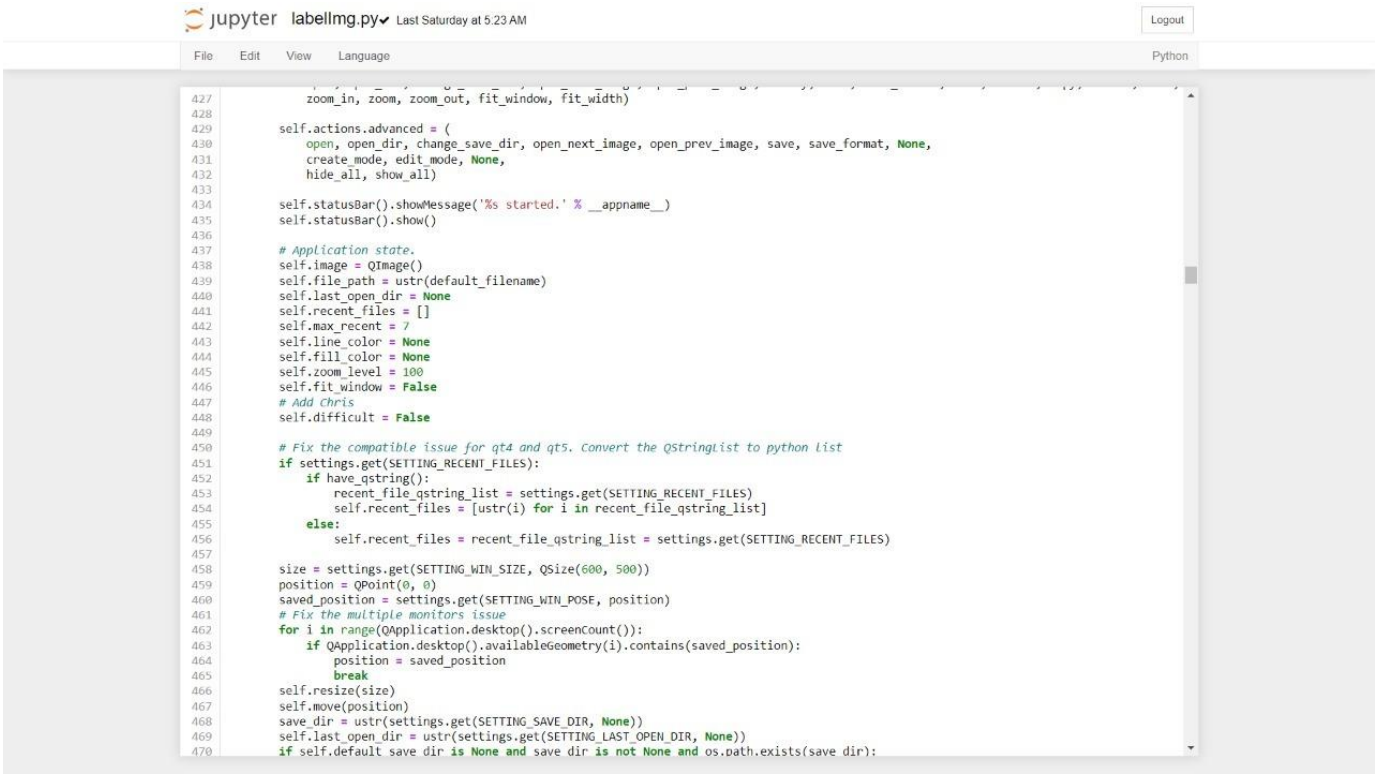
```
        labels.setText(get_str('showHide'))
        labels.setShortcut('Ctrl+Shift+L')

        # Label list context menu.
        label_menu = QMenu()
        add_actions(label_menu, (edit, delete))
        self.label_list.setContextMenuPolicy(Qt.CustomContextMenu)
        self.label_list.customContextMenuRequested.connect(
            self.pop_label_list_menu)

        # Draw squares/rectangles
        self.draw_squares_option = QAction(get_str('drawSquares'), self)
        self.draw_squares_option.setShortcut('Ctrl+Shift+R')
        self.draw_squares_option.setCheckable(True)
        self.draw_squares_option.setChecked(settings.get(SETTING_DRAW_SQUARE, False))
        self.draw_squares_option.triggered.connect(self.toggle_draw_square)

        # Store actions for further handling.
        self.actions = Struct(save=save, save_format=save_format, saveAs=save_as, open=open, close=close, resetAll=reset_all,
deleteImg=delete_image,
                              lineColor=color1, create=create, delete=delete, edit=edit, copy=copy,
                              createMode=create_mode, editMode=edit_mode, advancedMode=advanced_mode,
                              shapeLineColor=shape_line_color, shapeFillColor=shape_fill_color,
                              zoom=zoom, zoomIn=zoom_in, zoomOut=zoom_out, zoomOrg=zoom_org,
                              fitWindow=fit_window, fitWidth=fit_width,
                              zoomActions=zoom_actions,
                              fileMenuActions=(
                                  open, open_dir, save, save_as, close, reset_all, quit),
                              beginner=(), advanced=(),
                              editMenu=(edit, copy, delete,
                                        None, color1, self.draw_squares_option),
                              beginnerContext=(create, edit, copy, delete),
                              advancedContext=(create_mode, edit_mode, edit, copy,
                                               delete, shape_line_color, shape_fill_color),
                              onLoadActive=(
                                  close, create, create_mode, edit_mode),
                              onShapesPresent=(save_as, hide_all, show_all))

        self.menus = Struct(
            file=self.menu(get_str('menu_file')),
            edit=self.menu(get_str('menu_edit')),
            view=self.menu(get_str('menu_view')),
            help=self.menu(get_str('menu_help')),
            recentFiles=QMenu(get_str('menu_openRecent')),
```

jupyter labelImg.py Last Saturday at 5:23 AM

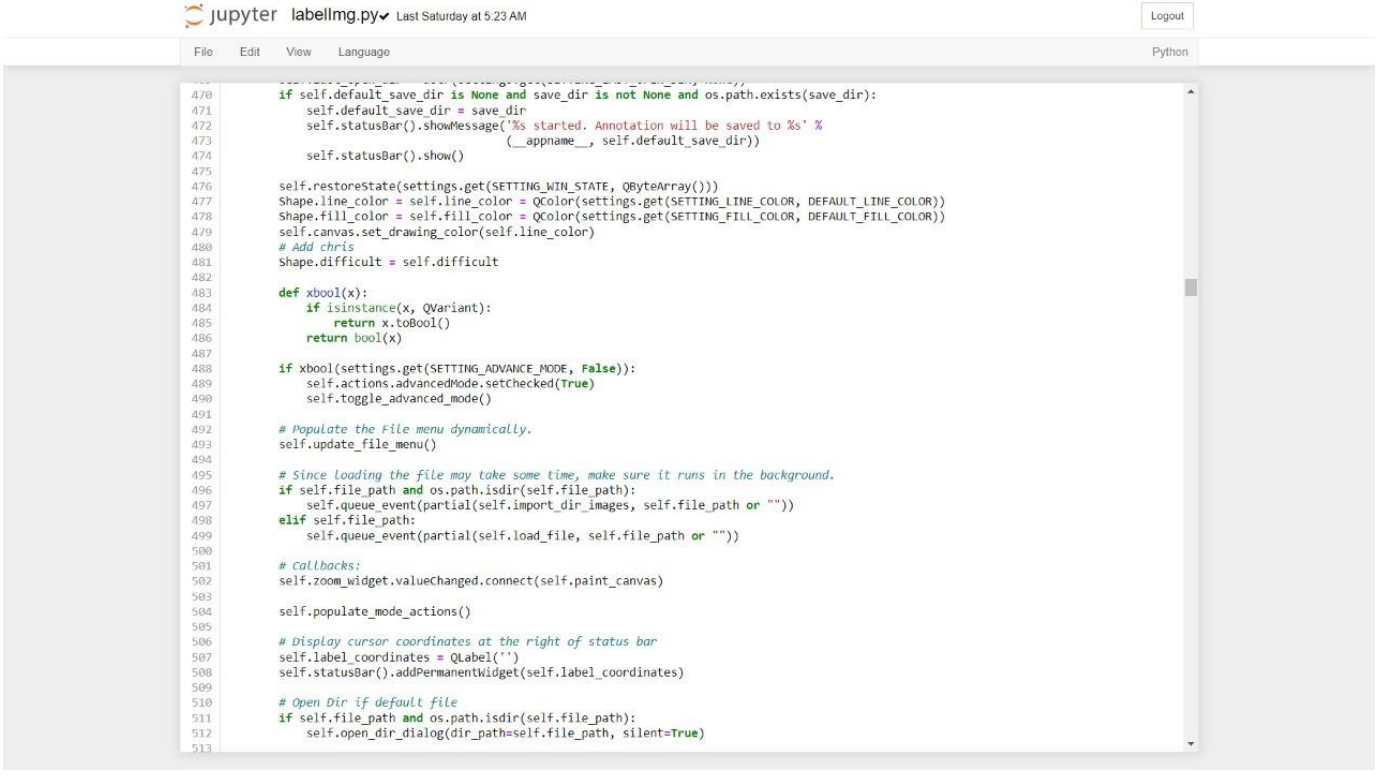
Logout

File Edit View Language Python

```
            labelList=label_menu)

        # Auto saving : Enable auto saving if pressing next
        self.auto_saving = QAction(get_str('autoSaveMode'), self)
        self.auto_saving.setCheckable(True)
        self.auto_saving.setChecked(settings.get(SETTING_AUTO_SAVE, False))
        # Sync single class mode from PR#106
        self.single_class_mode = QAction(get_str('singleClsMode'), self)
        self.single_class_mode.setShortcut("Ctrl+Shift+S")
        self.single_class_mode.setCheckable(True)
        self.single_class_mode.setChecked(settings.get(SETTING_SINGLE_CLASS, False))
        self.lastLabel = None
        # Add option to enable/disable labels being displayed at the top of bounding boxes
        self.display_label_option = QAction(get_str('displayLabel'), self)
        self.display_label_option.setShortcut("Ctrl+Shift+P")
        self.display_label_option.setCheckable(True)
        self.display_label_option.setChecked(settings.get(SETTING_PAINT_LABEL, False))
        self.display_label_option.triggered.connect(self.toggle_paint_labels_option)

        add_actions(self.menus.file,
                    (open, open_dir, change_save_dir, open_annotation, copy_prev_bounding, self.menus.recentFiles, save, save_format,
save_as, close, reset_all, delete_image, quit))
        add_actions(self.menus.help, (help_default, show_info, show_shortcut))
        add_actions(self.menus.view, (
            self.auto_saving,
            self.single_class_mode,
            self.display_label_option,
            labels, advanced_mode, None,
            hide_all, show_all, None,
            zoom_in, zoom_out, zoom_org, None,
            fit_window, fit_width))

        self.menus.file.aboutToShow.connect(self.update_file_menu)

        # Custom context menu for the canvas widget:
        add_actions(self.canvas.menus[0], self.actions.beginnerContext)
        add_actions(self.canvas.menus[1], (
            action('&Copy here', self.copy_shape),
            action('&Move here', self.move_shape)))

        self.tools = self.toolbar('Tools')
        self.actions.beginner = (
            open, open_dir, change_save_dir, open_next_image, open_prev_image, verify, save, save_format, None, create, copy, delete, None,
            zoom_in, zoom, zoom_out, fit_window, fit_width)
```




```
            zoom_in, zoom, zoom_out, fit_window, fit_width)

        self.actions.advanced = (
            open, open_dir, change_save_dir, open_next_image, open_prev_image, save, save_format, None,
            create_mode, edit_mode, None,
            hide_all, show_all)

        self.statusBar().showMessage('%s started.' % __appname__)
        self.statusBar().show()

        # Application state.
        self.image = QImage()
        self.file_path = ustr(default_filename)
        self.last_open_dir = None
        self.recent_files = []
        self.max_recent = 7
        self.line_color = None
        self.fill_color = None
        self.zoom_level = 100
        self.fit_window = False
        # Add chris
        self.difficult = False

        # Fix the compatible issue for qt4 and qt5. Convert the QStringList to python list
        if settings.get(SETTING_RECENT_FILES):
            if have_qstring():
                recent_file_qstring_list = settings.get(SETTING_RECENT_FILES)
                self.recent_files = [ustr(i) for i in recent_file_qstring_list]
            else:
                self.recent_files = recent_file_qstring_list = settings.get(SETTING_RECENT_FILES)

        size = settings.get(SETTING_WIN_SIZE, QSize(600, 500))
        position = QPoint(0, 0)
        saved_position = settings.get(SETTING_WIN_POSE, position)
        # Fix the multiple monitors issue
        for i in range(QApplication.desktop().screenCount()):
            if QApplication.desktop().availableGeometry(i).contains(saved_position):
                position = saved_position
                break
        self.resize(size)
        self.move(position)
        save_dir = ustr(settings.get(SETTING_SAVE_DIR, None))
        self.last_open_dir = ustr(settings.get(SETTING_LAST_OPEN_DIR, None))
        if self.default_save_dir is None and save_dir is not None and os.path.exists(save_dir):
```




```
        if self.default_save_dir is None and save_dir is not None and os.path.exists(save_dir):
            self.default_save_dir = save_dir
            self.statusBar().showMessage('%s started. Annotation will be saved to %s' %
                                         (__appname__, self.default_save_dir))
            self.statusBar().show()

        self.restoreState(settings.get(SETTING_WIN_STATE, QByteArray()))
        Shape.line_color = self.line_color = QColor(settings.get(SETTING_LINE_COLOR, DEFAULT_LINE_COLOR))
        Shape.fill_color = self.fill_color = QColor(settings.get(SETTING_FILL_COLOR, DEFAULT_FILL_COLOR))
        self.canvas.set_drawing_color(self.line_color)
        # Add chris
        Shape.difficult = self.difficult

        def xbool(x):
            if isinstance(x, QVariant):
                return x.toBool()
            return bool(x)

        if xbool(settings.get(SETTING_ADVANCE_MODE, False)):
            self.actions.advancedMode.setChecked(True)
            self.toggle_advanced_mode()

        # Populate the File menu dynamically.
        self.update_file_menu()

        # Since loading the file may take some time, make sure it runs in the background.
        if self.file_path and os.path.isdir(self.file_path):
            self.queue_event(partial(self.import_dir_images, self.file_path or ""))
        elif self.file_path:
            self.queue_event(partial(self.load_file, self.file_path or ""))

        # Callbacks:
        self.zoom_widget.valueChanged.connect(self.paint_canvas)

        self.populate_mode_actions()

        # Display cursor coordinates at the right of status bar
        self.label_coordinates = QLabel('')
        self.statusBar().addPermanentWidget(self.label_coordinates)

        # Open Dir if default file
        if self.file_path and os.path.isdir(self.file_path):
            self.open_dir_dialog(dir_path=self.file_path, silent=True)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
    def keyReleaseEvent(self, event):
        if event.key() == Qt.Key_Control:
            self.canvas.set_drawing_shape_to_square(False)

    def keyPressEvent(self, event):
        if event.key() == Qt.Key_Control:
            # Draw rectangle if Ctrl is pressed
            self.canvas.set_drawing_shape_to_square(True)

    # Support Functions #
    def set_format(self, save_format):
        if save_format == FORMAT_PASCALVOC:
            self.actions.save_format.setText(FORMAT_PASCALVOC)
            self.actions.save_format.setIcon(new_icon("format_voc"))
            self.label_file_format = LabelFileFormat.PASCAL_VOC
            LabelFile.suffix = XML_EXT

        elif save_format == FORMAT_YOLO:
            self.actions.save_format.setText(FORMAT_YOLO)
            self.actions.save_format.setIcon(new_icon("format_yolo"))
            self.label_file_format = LabelFileFormat.YOLO
            LabelFile.suffix = TXT_EXT

        elif save_format == FORMAT_CREATEML:
            self.actions.save_format.setText(FORMAT_CREATEML)
            self.actions.save_format.setIcon(new_icon("format_createml"))
            self.label_file_format = LabelFileFormat.CREATE_ML
            LabelFile.suffix = JSON_EXT

    def change_format(self):
        if self.label_file_format == LabelFileFormat.PASCAL_VOC:
            self.set_format(FORMAT_YOLO)
        elif self.label_file_format == LabelFileFormat.YOLO:
            self.set_format(FORMAT_CREATEML)
        elif self.label_file_format == LabelFileFormat.CREATE_ML:
            self.set_format(FORMAT_PASCALVOC)
        else:
            raise ValueError('Unknown label file format.')
        self.set_dirty()

    def no_shapes(self):
        return not self.items_to_shapes
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
    def toggle_advanced_mode(self, value=True):
        self._beginner = not value
        self.canvas.set_editing(True)
        self.populate_mode_actions()
        self.edit_button.setVisible(not value)
        if value:
            self.actions.createMode.setEnabled(True)
            self.actions.editMode.setEnabled(False)
            self.dock.setFeatures(self.dock.features() | self.dock_features)
        else:
            self.dock.setFeatures(self.dock.features() ^ self.dock_features)

    def populate_mode_actions(self):
        if self.beginner():
            tool, menu = self.actions.beginner, self.actions.beginnerContext
        else:
            tool, menu = self.actions.advanced, self.actions.advancedContext
        self.tools.clear()
        add_actions(self.tools, tool)
        self.canvas.menus[0].clear()
        add_actions(self.canvas.menus[0], menu)
        self.menus.edit.clear()
        actions = (self.actions.create,) if self.beginner()\
            else (self.actions.createMode, self.actions.editMode)
        add_actions(self.menus.edit, actions + self.actions.editMenu)

    def set_beginner(self):
        self.tools.clear()
        add_actions(self.tools, self.actions.beginner)

    def set_advanced(self):
        self.tools.clear()
        add_actions(self.tools, self.actions.advanced)

    def set_dirty(self):
        self.dirty = True
        self.actions.save.setEnabled(True)

    def set_clean(self):
        self.dirty = False
        self.actions.save.setEnabled(False)
        self.actions.create.setEnabled(True)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
599
600     def toggle_actions(self, value=True):
601         """Enable/Disable widgets which depend on an opened image."""
602         for z in self.actions.zoomActions:
603             z.setEnabled(value)
604         for action in self.actions.onLoadActive:
605             action.setEnabled(value)
606
607     def queue_event(self, function):
608         QTimer.singleShot(0, function)
609
610     def status(self, message, delay=5000):
611         self.statusBar().showMessage(message, delay)
612
613     def reset_state(self):
614         self.items_to_shapes.clear()
615         self.shapes_to_items.clear()
616         self.label_list.clear()
617         self.file_path = None
618         self.image_data = None
619         self.label_file = None
620         self.canvas.reset_state()
621         self.label_coordinates.clear()
622         self.combo_box.cb.clear()
623
624     def current_item(self):
625         items = self.label_list.selectedItems()
626         if items:
627             return items[0]
628         return None
629
630     def add_recent_file(self, file_path):
631         if file_path in self.recent_files:
632             self.recent_files.remove(file_path)
633         elif len(self.recent_files) >= self.max_recent:
634             self.recent_files.pop()
635         self.recent_files.insert(0, file_path)
636
637     def beginner(self):
638         return self._beginner
639
640     def advanced(self):
641         return not self.beginner()
642
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
642
643     def show_tutorial_dialog(self, browser='default', link=None):
644         if link is None:
645             link = self.screencast
646
647         if browser.lower() == 'default':
648             wb.open(link, new=2)
649         elif browser.lower() == 'chrome' and self.os_name == 'Windows':
650             if shutil.which(browser.lower()):  # 'chrome' not in wb._browsers in windows
651                 wb.register('chrome', None, wb.BackgroundBrowser('chrome'))
652             else:
653                 chrome_path="D:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe"
654                 if os.path.isfile(chrome_path):
655                     wb.register('chrome', None, wb.BackgroundBrowser(chrome_path))
656             try:
657                 wb.get('chrome').open(link, new=2)
658             except:
659                 wb.open(link, new=2)
660         elif browser.lower() in wb._browsers:
661             wb.get(browser.lower()).open(link, new=2)
662
663     def show_default_tutorial_dialog(self):
664         self.show_tutorial_dialog(browser='default')
665
666     def show_info_dialog(self):
667         from libs.__init__ import __version__
668         msg = u'Name:{0} \nApp Version:{1} \n{2} '.format(__appname__, __version__, sys.version_info)
669         QMessageBox.information(self, u'Information', msg)
670
671     def show_shortcuts_dialog(self):
672         self.show_tutorial_dialog(browser='default', link='https://github.com/tzutalin/labelImg#Hotkeys')
673
674     def create_shape(self):
675         assert self.beginner()
676         self.canvas.set_editing(False)
677         self.actions.create.setEnabled(False)
678
679     def toggle_drawing_sensitive(self, drawing=True):
680         """In the middle of drawing, toggling between modes should be disabled."""
681         self.actions.editMode.setEnabled(not drawing)
682         if not drawing and self.beginner():
683             # Cancel creation.
684             print('Cancel creation.')
685             self.canvas.set_editing(True)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
            print('Cancel creation.')
            self.canvas.set_editing(True)
            self.canvas.restore_cursor()
            self.actions.create.setEnabled(True)

    def toggle_draw_mode(self, edit=True):
        self.canvas.set_editing(edit)
        self.actions.createMode.setEnabled(edit)
        self.actions.editMode.setEnabled(not edit)

    def set_create_mode(self):
        assert self.advanced()
        self.toggle_draw_mode(False)

    def set_edit_mode(self):
        assert self.advanced()
        self.toggle_draw_mode(True)
        self.label_selection_changed()

    def update_file_menu(self):
        curr_file_path = self.file_path

        def exists(filename):
            return os.path.exists(filename)
        menu = self.menus.recentFiles
        menu.clear()
        files = [f for f in self.recent_files if f !=
                 curr_file_path and exists(f)]
        for i, f in enumerate(files):
            icon = new_icon('labels')
            action = QAction(
                icon, '&%d %s' % (i + 1, QFileInfo(f).fileName()), self)
            action.triggered.connect(partial(self.load_recent, f))
            menu.addAction(action)

    def pop_label_list_menu(self, point):
        self.menus.labelList.exec_(self.label_list.mapToGlobal(point))

    def edit_label(self):
        if not self.canvas.editing():
            return
        item = self.current_item()
        if not item:
            return
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
            return
        text = self.label_dialog.pop_up(item.text())
        if text is not None:
            item.setText(text)
            item.setBackground(generate_color_by_text(text))
            self.set_dirty()
            self.update_combo_box()

    # Tzutalin 20160906 : Add file list and dock to move faster
    def file_item_double_clicked(self, item=None):
        self.cur_img_idx = self.m_img_list.index(ustr(item.text()))
        filename = self.m_img_list[self.cur_img_idx]
        if filename:
            self.load_file(filename)

    # Add chris
    def button_state(self, item=None):
        """ Function to handle difficult examples
        Update on each object """
        if not self.canvas.editing():
            return

        item = self.current_item()
        if not item:  # If not selected Item, take the first one
            item = self.label_list.item(self.label_list.count() - 1)

        difficult = self.diffc_button.isChecked()

        try:
            shape = self.items_to_shapes[item]
        except:
            pass
        # Checked and Update
        try:
            if difficult != shape.difficult:
                shape.difficult = difficult
                self.set_dirty()
            else:  # User probably changed item visibility
                self.canvas.set_shape_visible(shape, item.checkState() == Qt.Checked)
        except:
            pass

    # React to canvas signals.
    def shape_selection_changed(self, selected=False):
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

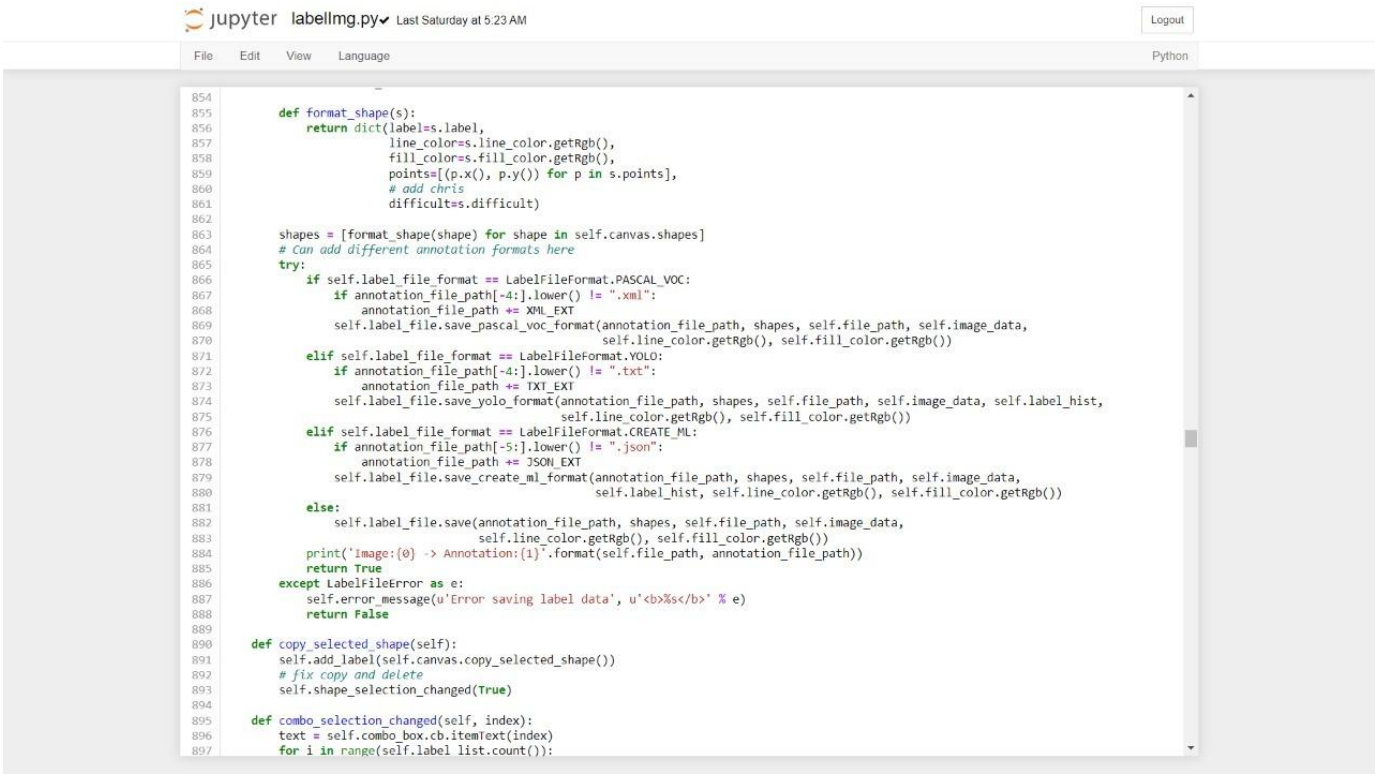
```
    # React to canvas signals.
    def shape_selection_changed(self, selected=False):
        if self._no_selection_slot:
            self._no_selection_slot = False
        else:
            shape = self.canvas.selected_shape
            if shape:
                self.shapes_to_items[shape].setSelected(True)
            else:
                self.label_list.clearSelection()
        self.actions.delete.setEnabled(selected)
        self.actions.copy.setEnabled(selected)
        self.actions.edit.setEnabled(selected)
        self.actions.shapeLineColor.setEnabled(selected)
        self.actions.shapeFillColor.setEnabled(selected)

    def add_label(self, shape):
        shape.paint_label = self.display_label_option.isChecked()
        item = HashableQListWidgetItem(shape.label)
        item.setFlags(item.flags() | Qt.ItemIsUserCheckable)
        item.setCheckState(Qt.Checked)
        item.setBackground(generate_color_by_text(shape.label))
        self.items_to_shapes[item] = shape
        self.shapes_to_items[shape] = item
        self.label_list.addItem(item)
        for action in self.actions.onShapesPresent:
            action.setEnabled(True)
        self.update_combo_box()

    def remove_label(self, shape):
        if shape is None:
            # print('rm empty label')
            return
        item = self.shapes_to_items[shape]
        self.label_list.takeItem(self.label_list.row(item))
        del self.shapes_to_items[shape]
        del self.items_to_shapes[item]
        self.update_combo_box()

    def load_labels(self, shapes):
        s = []
        for label, points, line_color, fill_color, difficult in shapes:
            shape = Shape(label=label)
            for x, y in points:
```

jupyter labelImg.py Last Saturday at 5:23 AM

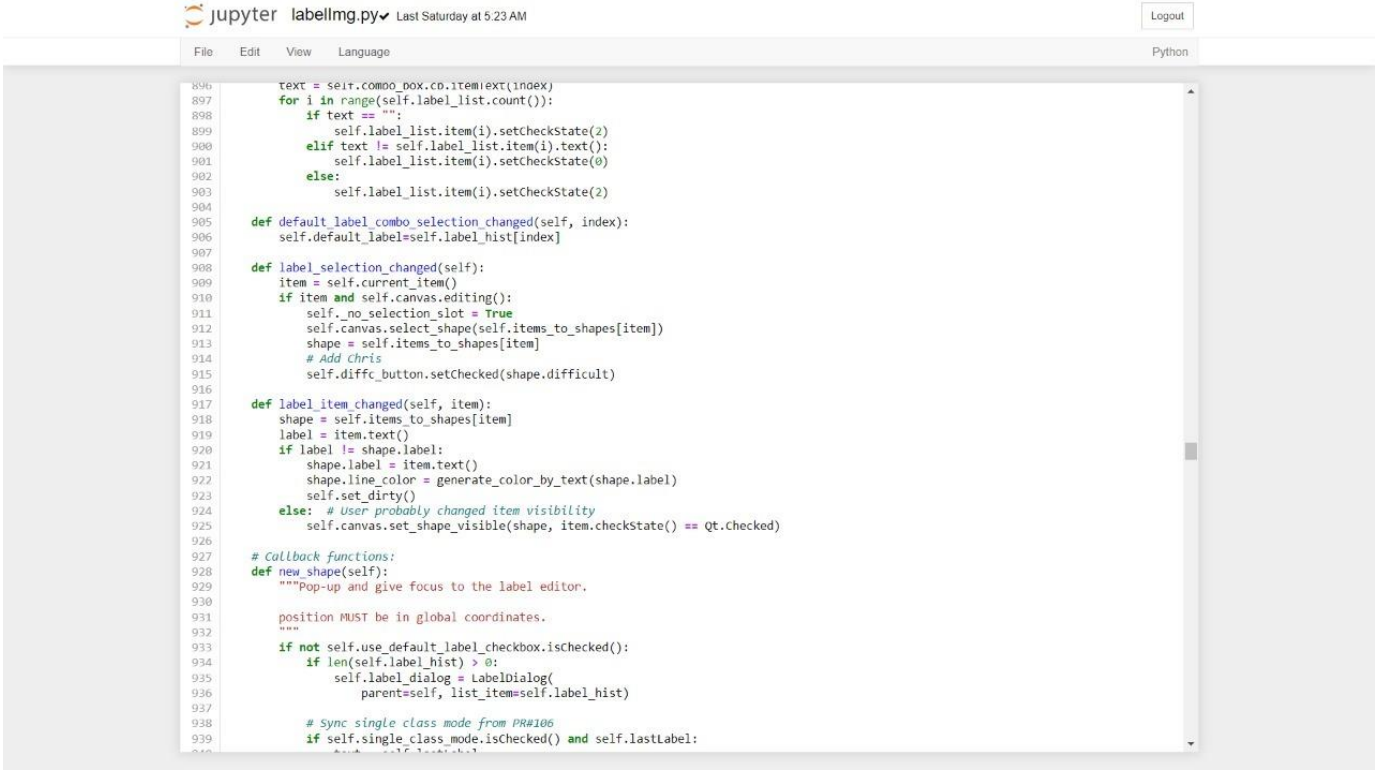
Logout

File Edit View Language Python

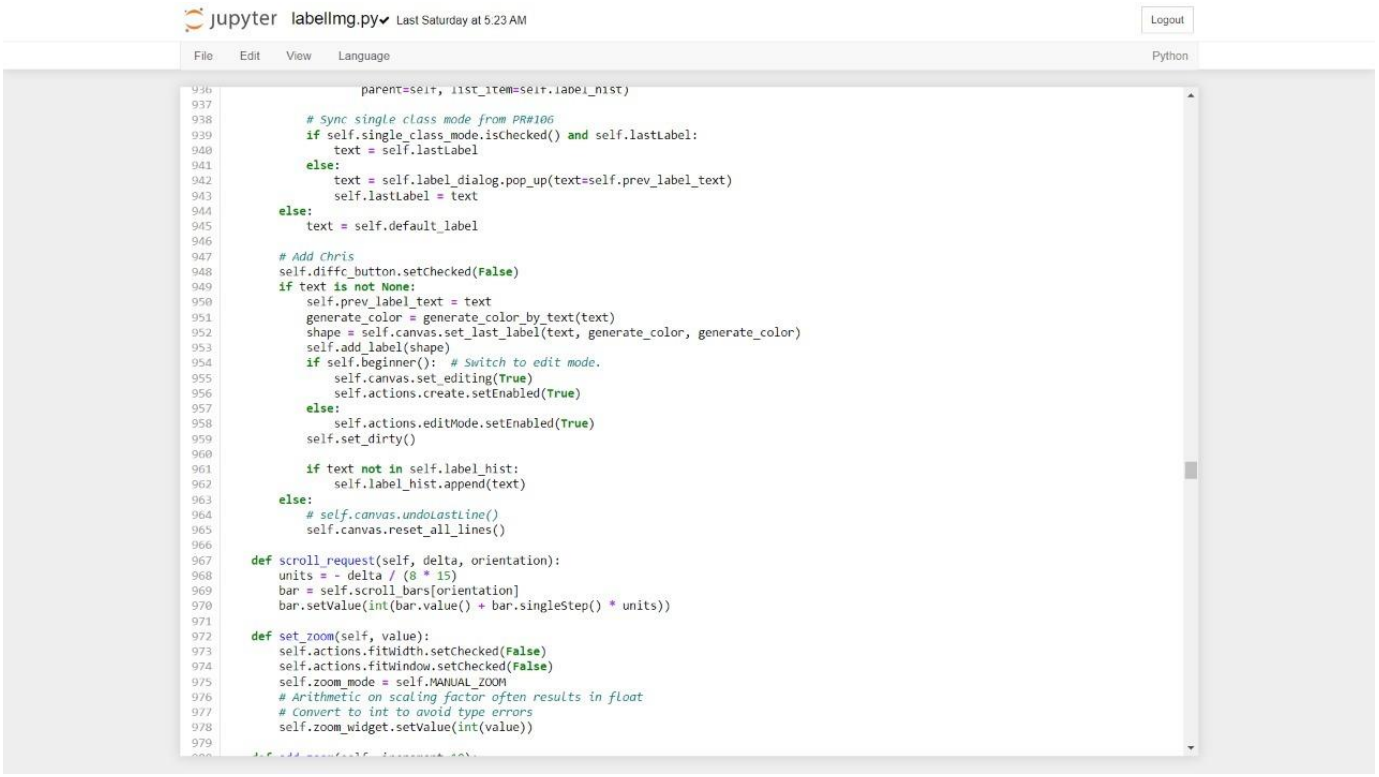
```
            shape = Shape(label=label)
            for x, y in points:

                # Ensure the labels are within the bounds of the image. If not, fix them.
                x, y, snapped = self.canvas.snap_point_to_canvas(x, y)
                if snapped:
                    self.set_dirty()

                shape.add_point(QPointF(x, y))
            shape.difficult = difficult
            shape.close()
            s.append(shape)

            if line_color:
                shape.line_color = QColor(*line_color)
            else:
                shape.line_color = generate_color_by_text(label)

            if fill_color:
                shape.fill_color = QColor(*fill_color)
            else:
                shape.fill_color = generate_color_by_text(label)

            self.add_label(shape)
        self.update_combo_box()
        self.canvas.load_shapes(s)

    def update_combo_box(self):
        # Get the unique labels and add them to the Combobox.
        items_text_list = [str(self.label_list.item(i).text()) for i in range(self.label_list.count())]

        unique_text_list = list(set(items_text_list))
        # Add a null row for showing all the labels
        unique_text_list.append("")
        unique_text_list.sort()

        self.combo_box.update_items(unique_text_list)

    def save_labels(self, annotation_file_path):
        annotation_file_path = ustr(annotation_file_path)
        if self.label_file is None:
            self.label_file = LabelFile()
            self.label_file.verified = self.canvas.verified
```



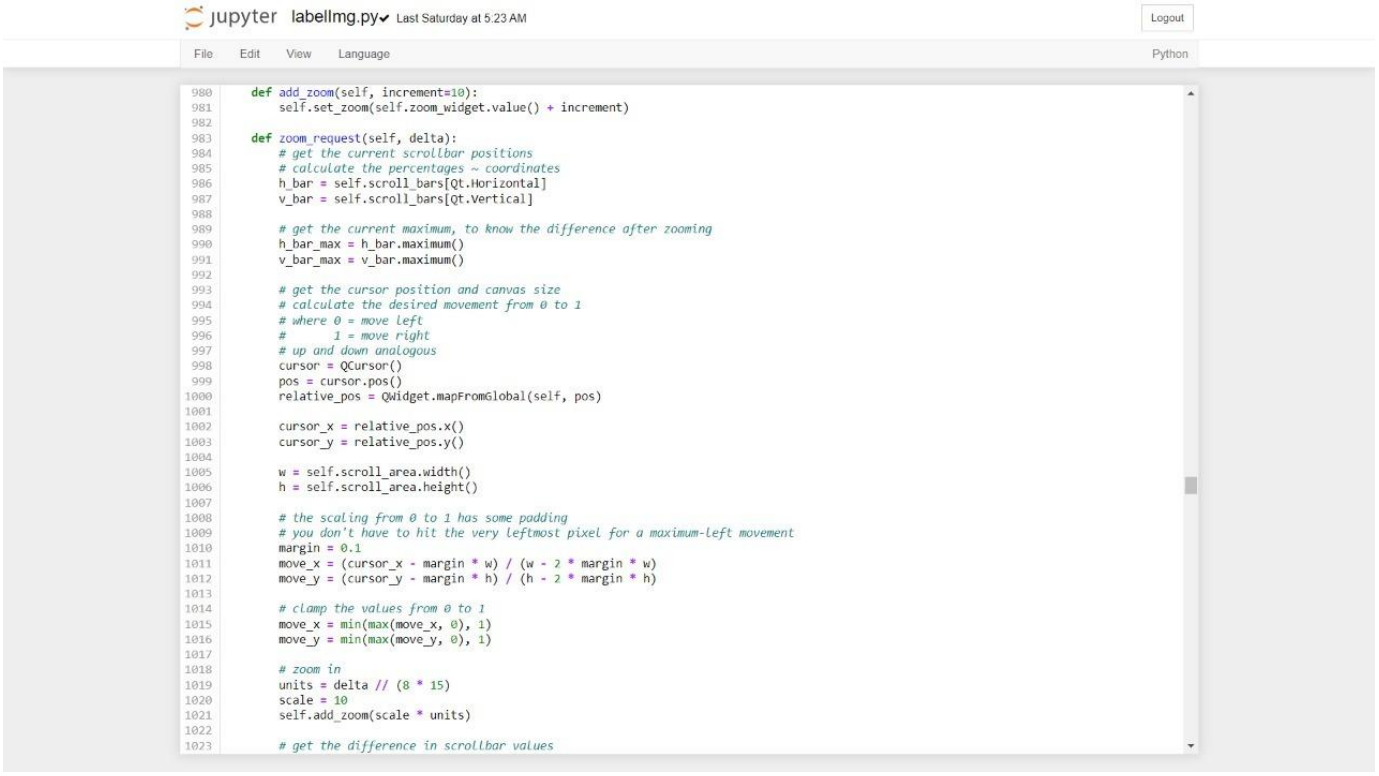

```
        def format_shape(s):
            return dict(label=s.label,
                        line_color=s.line_color.getRgb(),
                        fill_color=s.fill_color.getRgb(),
                        points=[(p.x(), p.y()) for p in s.points],
                        # add chris
                        difficult=s.difficult)

        shapes = [format_shape(shape) for shape in self.canvas.shapes]
        # Can add different annotation formats here
        try:
            if self.label_file_format == LabelFileFormat.PASCAL_VOC:
                if annotation_file_path[-4:].lower() != ".xml":
                    annotation_file_path += XML_EXT
                self.label_file.save_pascal_voc_format(annotation_file_path, shapes, self.file_path, self.image_data,
                                                       self.line_color.getRgb(), self.fill_color.getRgb())
            elif self.label_file_format == LabelFileFormat.YOLO:
                if annotation_file_path[-4:].lower() != ".txt":
                    annotation_file_path += TXT_EXT
                self.label_file.save_yolo_format(annotation_file_path, shapes, self.file_path, self.image_data, self.label_hist,
                                                 self.line_color.getRgb(), self.fill_color.getRgb())
            elif self.label_file_format == LabelFileFormat.CREATE_ML:
                if annotation_file_path[-5:].lower() != ".json":
                    annotation_file_path += JSON_EXT
                self.label_file.save_create_ml_format(annotation_file_path, shapes, self.file_path, self.image_data,
                                                      self.label_hist, self.line_color.getRgb(), self.fill_color.getRgb())
            else:
                self.label_file.save(annotation_file_path, shapes, self.file_path, self.image_data,
                                     self.line_color.getRgb(), self.fill_color.getRgb())
            print('Image:{0} -> Annotation:{1}'.format(self.file_path, annotation_file_path))
            return True
        except LabelFileError as e:
            self.error_message(u'Error saving label data', u'<b>%s</b>' % e)
            return False

    def copy_selected_shape(self):
        self.add_label(self.canvas.copy_selected_shape())
        # fix copy and delete
        self.shape_selection_changed(True)

    def combo_selection_changed(self, index):
        text = self.combo_box.cb.itemText(index)
        for i in range(self.label_list.count()):
```




```
        text = self.combo_box.cb.itemText(index)
        for i in range(self.label_list.count()):
            if text == "":
                self.label_list.item(i).setCheckState(2)
            elif text != self.label_list.item(i).text():
                self.label_list.item(i).setCheckState(0)
            else:
                self.label_list.item(i).setCheckState(2)

    def default_label_combo_selection_changed(self, index):
        self.default_label=self.label_hist[index]

    def label_selection_changed(self):
        item = self.current_item()
        if item and self.canvas.editing():
            self._no_selection_slot = True
            self.canvas.select_shape(self.items_to_shapes[item])
            shape = self.items_to_shapes[item]
            # Add Chris
            self.diffc_button.setChecked(shape.difficult)

    def label_item_changed(self, item):
        shape = self.items_to_shapes[item]
        label = item.text()
        if label != shape.label:
            shape.label = item.text()
            shape.line_color = generate_color_by_text(shape.label)
            self.set_dirty()
        else:  # User probably changed item visibility
            self.canvas.set_shape_visible(shape, item.checkState() == Qt.Checked)

    # Callback functions:
    def new_shape(self):
        """Pop-up and give focus to the label editor.

        position MUST be in global coordinates.
        """
        if not self.use_default_label_checkbox.isChecked():
            if len(self.label_hist) > 0:
                self.label_dialog = LabelDialog(
                    parent=self, list_item=self.label_hist)

            # Sync single class mode from PR#106
            if self.single_class_mode.isChecked() and self.lastLabel:
```




```
                    parent=self, list_item=self.label_hist)

            # Sync single class mode from PR#106
            if self.single_class_mode.isChecked() and self.lastLabel:
                text = self.lastLabel
            else:
                text = self.label_dialog.pop_up(text=self.prev_label_text)
                self.lastLabel = text
        else:
            text = self.default_label

        # Add Chris
        self.diffc_button.setChecked(False)
        if text is not None:
            self.prev_label_text = text
            generate_color = generate_color_by_text(text)
            shape = self.canvas.set_last_label(text, generate_color, generate_color)
            self.add_label(shape)
            if self.beginner():  # Switch to edit mode.
                self.canvas.set_editing(True)
                self.actions.create.setEnabled(True)
            else:
                self.actions.editMode.setEnabled(True)
            self.set_dirty()

            if text not in self.label_hist:
                self.label_hist.append(text)
        else:
            # self.canvas.undoLastLine()
            self.canvas.reset_all_lines()

    def scroll_request(self, delta, orientation):
        units = - delta / (8 * 15)
        bar = self.scroll_bars[orientation]
        bar.setValue(int(bar.value() + bar.singleStep() * units))

    def set_zoom(self, value):
        self.actions.fitWidth.setChecked(False)
        self.actions.fitWindow.setChecked(False)
        self.zoom_mode = self.MANUAL_ZOOM
        # Arithmetic on scaling factor often results in float
        # Convert to int to avoid type errors
        self.zoom_widget.setValue(int(value))
```




```
    def add_zoom(self, increment=10):
        self.set_zoom(self.zoom_widget.value() + increment)

    def zoom_request(self, delta):
        # get the current scrollbar positions
        # calculate the percentages ~ coordinates
        h_bar = self.scroll_bars[Qt.Horizontal]
        v_bar = self.scroll_bars[Qt.Vertical]

        # get the current maximum, to know the difference after zooming
        h_bar_max = h_bar.maximum()
        v_bar_max = v_bar.maximum()

        # get the cursor position and canvas size
        # calculate the desired movement from 0 to 1
        # where 0 = move left
        #       1 = move right
        # up and down analogous
        cursor = QCursor()
        pos = cursor.pos()
        relative_pos = QWidget.mapFromGlobal(self, pos)

        cursor_x = relative_pos.x()
        cursor_y = relative_pos.y()

        w = self.scroll_area.width()
        h = self.scroll_area.height()

        # the scaling from 0 to 1 has some padding
        # you don't have to hit the very leftmost pixel for a maximum-left movement
        margin = 0.1
        move_x = (cursor_x - margin * w) / (w - 2 * margin * w)
        move_y = (cursor_y - margin * h) / (h - 2 * margin * h)

        # clamp the values from 0 to 1
        move_x = min(max(move_x, 0), 1)
        move_y = min(max(move_y, 0), 1)

        # zoom in
        units = delta // (8 * 15)
        scale = 10
        self.add_zoom(scale * units)

        # get the difference in scrollbar values
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        # get the difference in scrollbar values
        # this is how far we can move
        d_h_bar_max = h_bar.maximum() - h_bar_max
        d_v_bar_max = v_bar.maximum() - v_bar_max

        # get the new scrollbar values
        new_h_bar_value = int(h_bar.value() + move_x * d_h_bar_max)
        new_v_bar_value = int(v_bar.value() + move_y * d_v_bar_max)

        h_bar.setValue(new_h_bar_value)
        v_bar.setValue(new_v_bar_value)

    def set_fit_window(self, value=True):
        if value:
            self.actions.fitWidth.setChecked(False)
        self.zoom_mode = self.FIT_WINDOW if value else self.MANUAL_ZOOM
        self.adjust_scale()

    def set_fit_width(self, value=True):
        if value:
            self.actions.fitWindow.setChecked(False)
        self.zoom_mode = self.FIT_WIDTH if value else self.MANUAL_ZOOM
        self.adjust_scale()

    def toggle_polygons(self, value):
        for item, shape in self.items_to_shapes.items():
            item.setCheckState(Qt.Checked if value else Qt.Unchecked)

    def load_file(self, file_path=None):
        """Load the specified file, or the last opened file if None."""
        self.reset_state()
        self.canvas.setEnabled(False)
        if file_path is None:
            file_path = self.settings.get(SETTING_FILENAME)
        #Deselect shape when loading new file
        if self.canvas.selected_shape:
            self.canvas.selected_shape.selected = False
            self.canvas.selected_shape = None
        # Make sure that filePath is a regular python string, rather than QString
        file_path = ustr(file_path)

        # Fix bug: An  index error after select a directory when open a new file.
        unicode_file_path = ustr(file_path)
        unicode_file_path = os.path.abspath(unicode_file_path)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

```
        # Fix bug: An  index error after select a directory when open a new file.
        unicode_file_path = ustr(file_path)
        unicode_file_path = os.path.abspath(unicode_file_path)
        # Tzutalin 20160906 : Add file list and dock to move faster
        # Highlight the file item
        if unicode_file_path and self.file_list_widget.count() > 0:
            if unicode_file_path in self.m_img_list:
                index = self.m_img_list.index(unicode_file_path)
                file_widget_item = self.file_list_widget.item(index)
                file_widget_item.setSelected(True)
            else:
                self.file_list_widget.clear()
                self.m_img_list.clear()

        if unicode_file_path and os.path.exists(unicode_file_path):
            if LabelFile.is_label_file(unicode_file_path):
                try:
                    self.label_file = LabelFile(unicode_file_path)
                except LabelFileError as e:
                    self.error_message(u'Error opening file',
                                       (u"<p><b>%s</b></p>"
                                        u"<p>Make sure <i>%s</i> is a valid label file.")
                                       % (e, unicode_file_path))
                    self.status("Error reading %s" % unicode_file_path)
                    return False
                self.image_data = self.label_file.image_data
                self.line_color = QColor(*self.label_file.lineColor)
                self.fill_color = QColor(*self.label_file.fillColor)
                self.canvas.verified = self.label_file.verified
            else:
                # Load image:
                # read data first and store for saving into label file.
                self.image_data = read(unicode_file_path, None)
                self.label_file = None
                self.canvas.verified = False

            if isinstance(self.image_data, QImage):
                image = self.image_data
            else:
                image = QImage.fromData(self.image_data)
            if image.isNull():
                self.error_message(u'Error opening file',
                                   u"<p>Make sure <i>%s</i> is a valid image file." % unicode_file_path)
                self.status("Error reading %s" % unicode_file_path)
```

jupyter labelImg.py Last Saturday at 5:23 AM

Logout

File Edit View Language Python

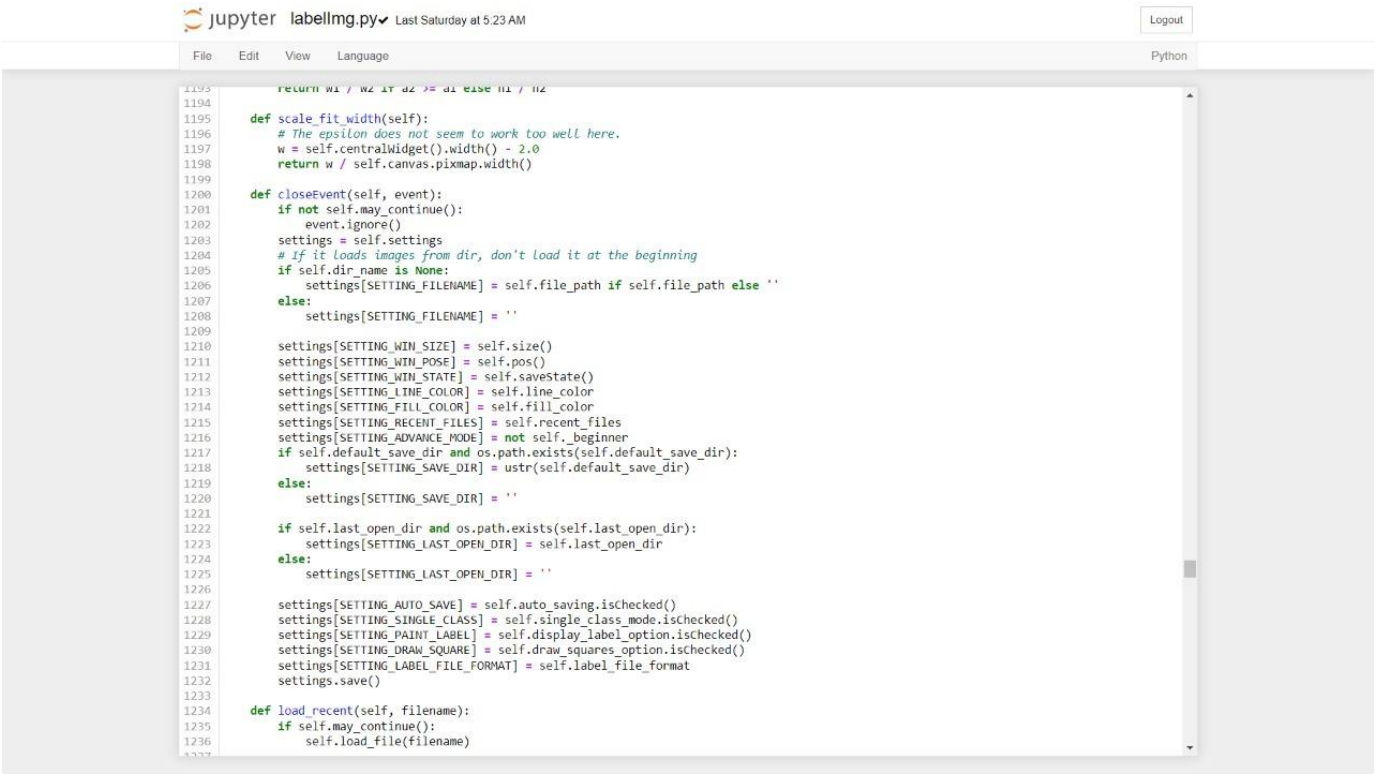
```
                self.status("Error reading %s" % unicode_file_path)
                return False
            self.status("Loaded %s" % os.path.basename(unicode_file_path))
            self.image = image
            self.file_path = unicode_file_path
            self.canvas.load_pixmap(QPixmap.fromImage(image))
            if self.label_file:
                self.load_labels(self.label_file.shapes)
            self.set_clean()
            self.canvas.setEnabled(True)
            self.adjust_scale(initial=True)
            self.paint_canvas()
            self.add_recent_file(self.file_path)
            self.toggle_actions(True)
            self.show_bounding_box_from_annotation_file(file_path)

            counter = self.counter_str()
            self.setWindowTitle(__appname__ + ' ' + file_path + ' ' + counter)

            # Default : select last item if there is at least one item
            if self.label_list.count():
                self.label_list.setCurrentItem(self.label_list.item(self.label_list.count() - 1))
                self.label_list.item(self.label_list.count() - 1).setSelected(True)

            self.canvas.setFocus(True)
            return True
        return False

    def counter_str(self):
        """
        Converts image counter to string representation.
        """
        return '[{} / {}]'.format(self.cur_img_idx + 1, self.img_count)

    def show_bounding_box_from_annotation_file(self, file_path):
        if self.default_save_dir is not None:
            basename = os.path.basename(os.path.splitext(file_path)[0])
            xml_path = os.path.join(self.default_save_dir, basename + XML_EXT)
            txt_path = os.path.join(self.default_save_dir, basename + TXT_EXT)
            json_path = os.path.join(self.default_save_dir, basename + JSON_EXT)

            """Annotation file priority:
            PascalXML > YOLO
            """
```

jupyter labelImg.py Last Saturday at 5:23 AM

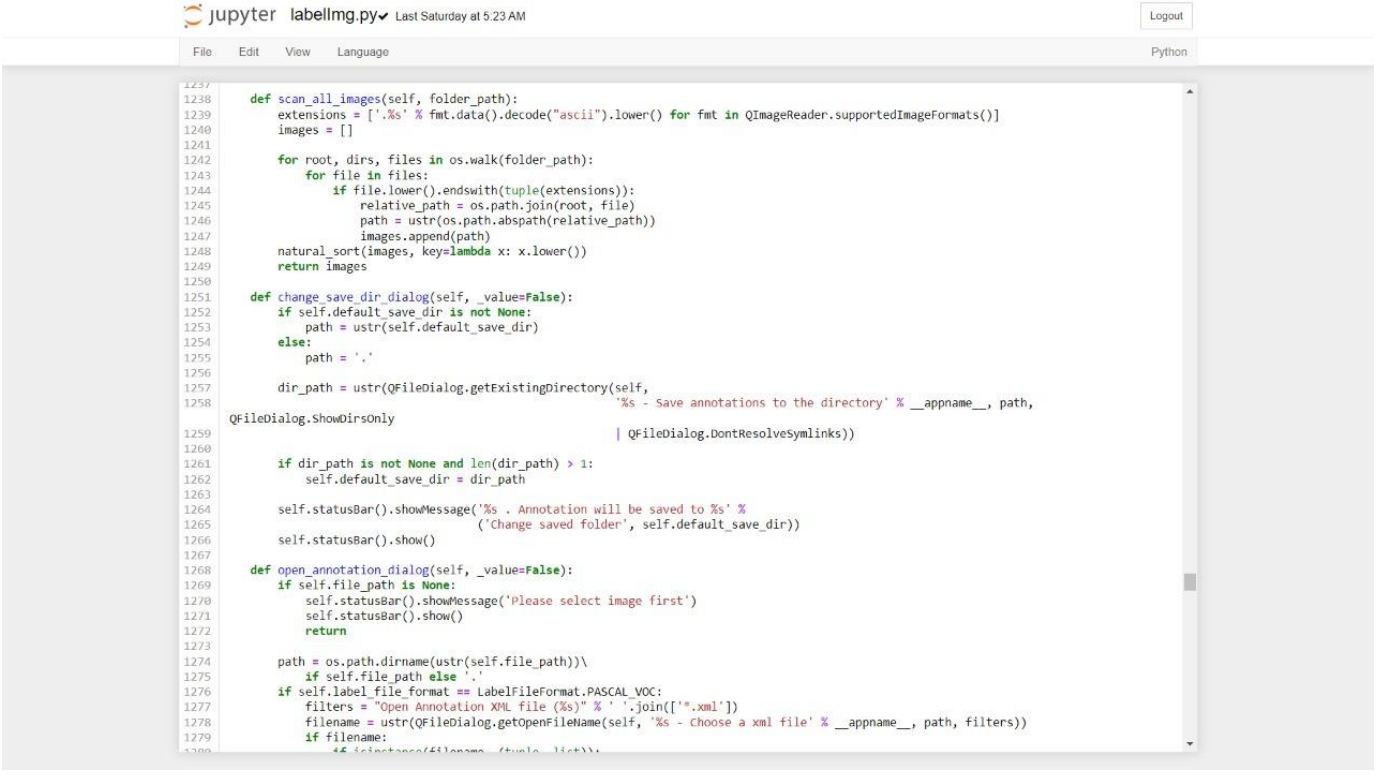
Logout

File Edit View Language Python

```
            if os.path.isfile(xml_path):
                self.load_pascal_xml_by_filename(xml_path)
            elif os.path.isfile(txt_path):
                self.load_yolo_txt_by_filename(txt_path)
            elif os.path.isfile(json_path):
                self.load_create_ml_json_by_filename(json_path, file_path)

        else:
            xml_path = os.path.splitext(file_path)[0] + XML_EXT
            txt_path = os.path.splitext(file_path)[0] + TXT_EXT
            if os.path.isfile(xml_path):
                self.load_pascal_xml_by_filename(xml_path)
            elif os.path.isfile(txt_path):
                self.load_yolo_txt_by_filename(txt_path)

    def resizeEvent(self, event):
        if self.canvas and not self.image.isNull()\
           and self.zoom_mode != self.MANUAL_ZOOM:
            self.adjust_scale()
        super(MainWindow, self).resizeEvent(event)

    def paint_canvas(self):
        assert not self.image.isNull(), "cannot paint null image"
        self.canvas.scale = 0.01 * self.zoom_widget.value()
        self.canvas.label_font_size = int(0.02 * max(self.image.width(), self.image.height()))
        self.canvas.adjustSize()
        self.canvas.update()

    def adjust_scale(self, initial=False):
        value = self.scalers[self.FIT_WINDOW if initial else self.zoom_mode]()
        self.zoom_widget.setValue(int(100 * value))

    def scale_fit_window(self):
        """Figure out the size of the pixmap in order to fit the main widget."""
        e = 2.0  # So that no scrollbars are generated.
        w1 = self.centralWidget().width() - e
        h1 = self.centralWidget().height() - e
        a1 = w1 / h1
        # Calculate a new scale value based on the pixmap's aspect ratio.
        w2 = self.canvas.pixmap.width() - 0.0
        h2 = self.canvas.pixmap.height() - 0.0
        a2 = w2 / h2
        return w1 / w2 if a2 >= a1 else h1 / h2
```

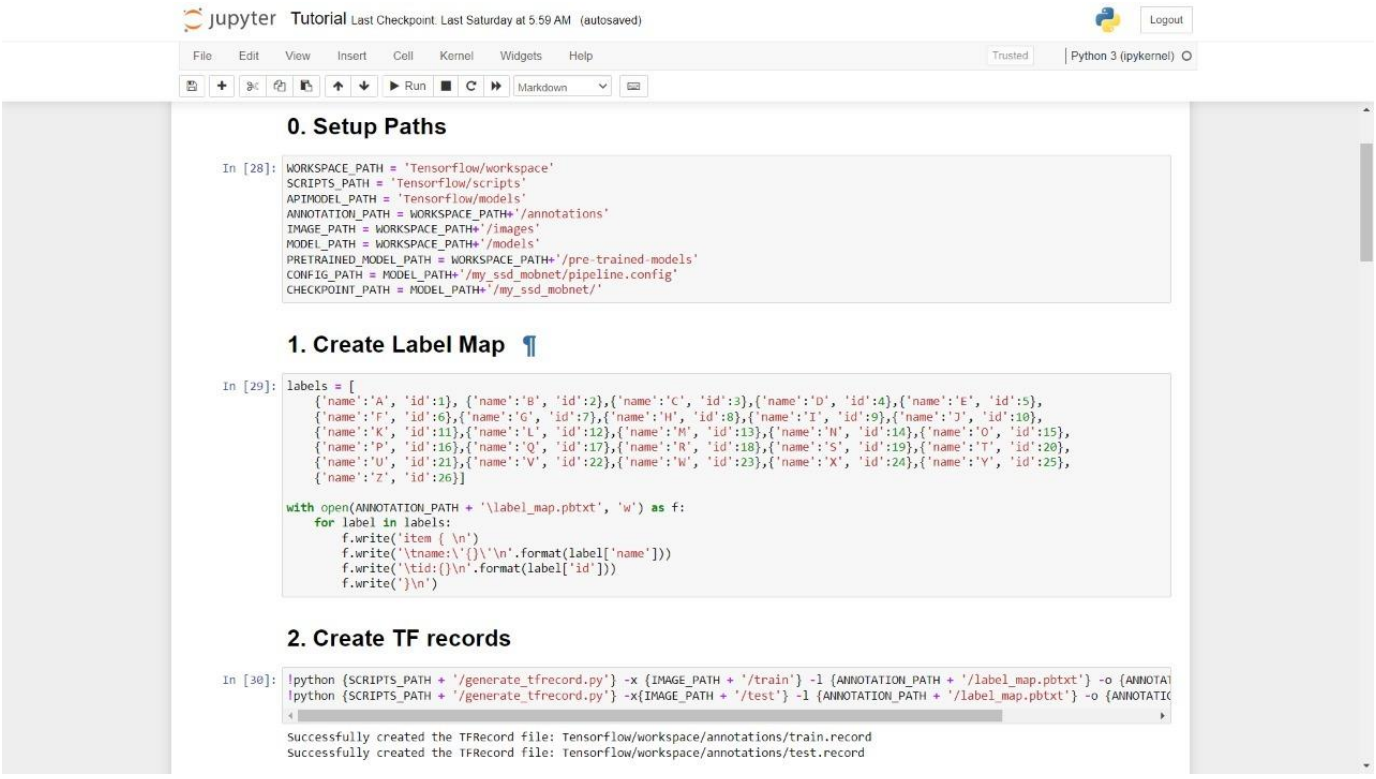


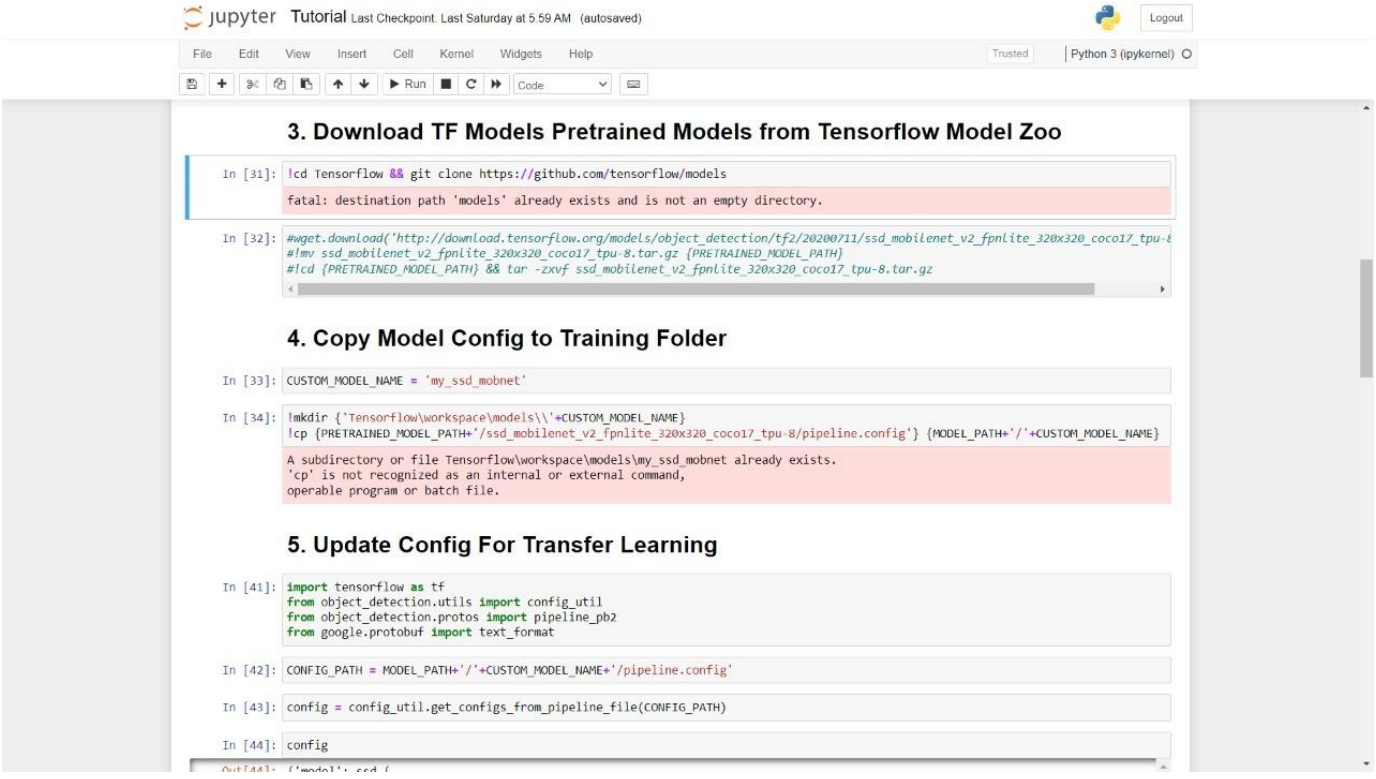

```
        return w1 / w2 if a2 >= a1 else h1 / h2

    def scale_fit_width(self):
        # The epsilon does not seem to work too well here.
        w = self.centralWidget().width() - 2.0
        return w / self.canvas.pixmap.width()

    def closeEvent(self, event):
        if not self.may_continue():
            event.ignore()
        settings = self.settings
        # If it loads images from dir, don't load it at the beginning
        if self.dir_name is None:
            settings[SETTING_FILENAME] = self.file_path if self.file_path else ''
        else:
            settings[SETTING_FILENAME] = ''

        settings[SETTING_WIN_SIZE] = self.size()
        settings[SETTING_WIN_POSE] = self.pos()
        settings[SETTING_WIN_STATE] = self.saveState()
        settings[SETTING_LINE_COLOR] = self.line_color
        settings[SETTING_FILL_COLOR] = self.fill_color
        settings[SETTING_RECENT_FILES] = self.recent_files
        settings[SETTING_ADVANCE_MODE] = not self._beginner
        if self.default_save_dir and os.path.exists(self.default_save_dir):
            settings[SETTING_SAVE_DIR] = ustr(self.default_save_dir)
        else:
            settings[SETTING_SAVE_DIR] = ''

        if self.last_open_dir and os.path.exists(self.last_open_dir):
            settings[SETTING_LAST_OPEN_DIR] = self.last_open_dir
        else:
            settings[SETTING_LAST_OPEN_DIR] = ''

        settings[SETTING_AUTO_SAVE] = self.auto_saving.isChecked()
        settings[SETTING_SINGLE_CLASS] = self.single_class_mode.isChecked()
        settings[SETTING_PAINT_LABEL] = self.display_label_option.isChecked()
        settings[SETTING_DRAW_SQUARE] = self.draw_squares_option.isChecked()
        settings[SETTING_LABEL_FILE_FORMAT] = self.label_file_format
        settings.save()

    def load_recent(self, filename):
        if self.may_continue():
            self.load_file(filename)
```

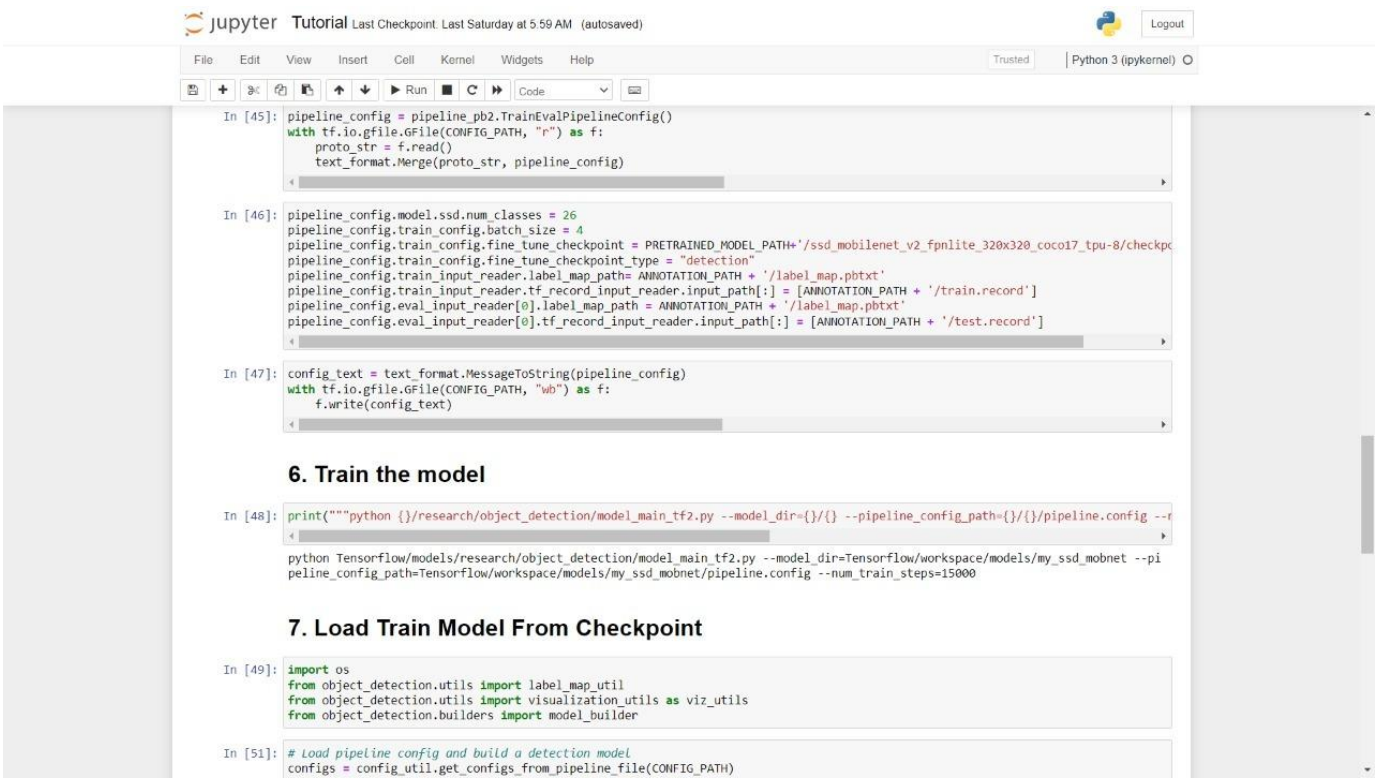


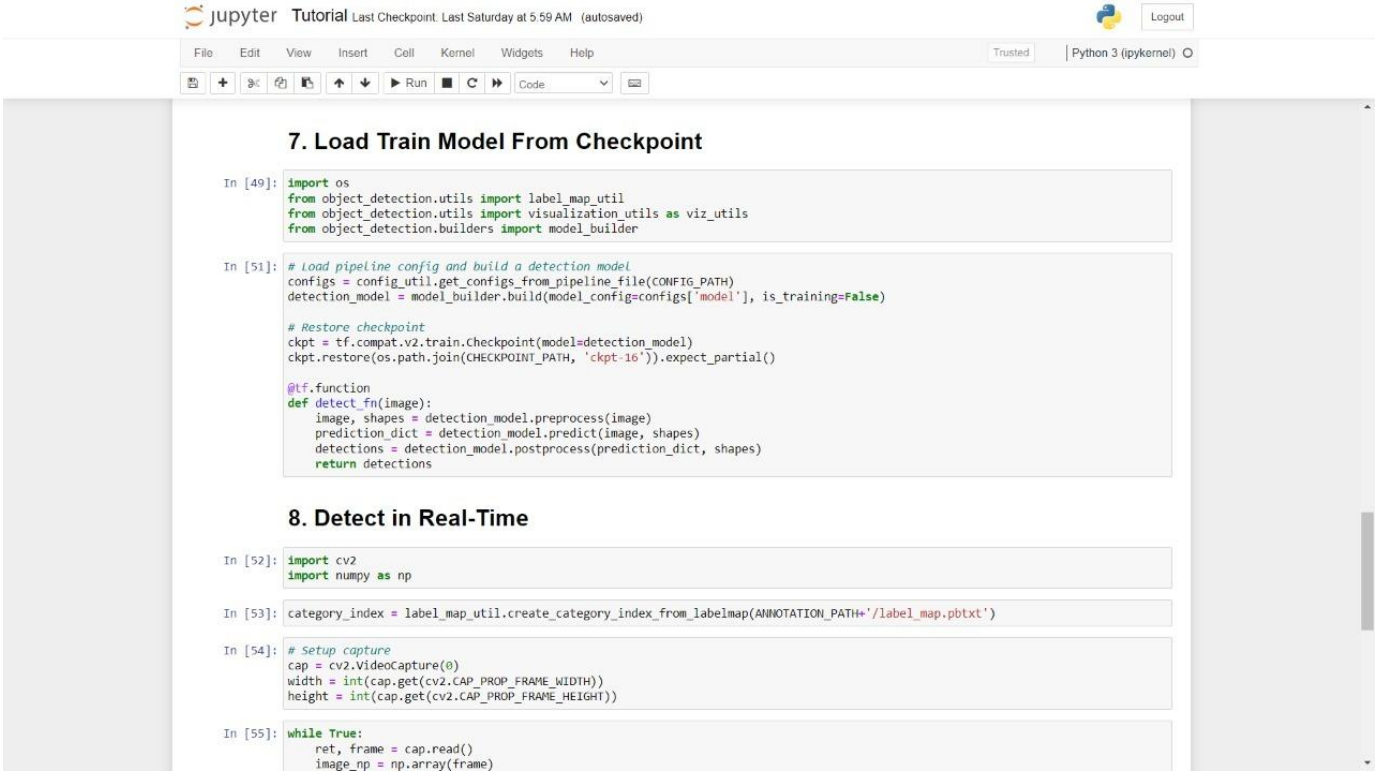

```
    def scan_all_images(self, folder_path):
        extensions = ['.%s' % fmt.data().decode("ascii").lower() for fmt in QImageReader.supportedImageFormats()]
        images = []

        for root, dirs, files in os.walk(folder_path):
            for file in files:
                if file.lower().endswith(tuple(extensions)):
                    relative_path = os.path.join(root, file)
                    path = ustr(os.path.abspath(relative_path))
                    images.append(path)
        natural_sort(images, key=lambda x: x.lower())
        return images

    def change_save_dir_dialog(self, _value=False):
        if self.default_save_dir is not None:
            path = ustr(self.default_save_dir)
        else:
            path = '.'

        dir_path = ustr(QFileDialog.getExistingDirectory(self,
                                                         '%s - Save annotations to the directory' % __appname__, path,
QFileDialog.ShowDirsOnly
                                                         | QFileDialog.DontResolveSymlinks))

        if dir_path is not None and len(dir_path) > 1:
            self.default_save_dir = dir_path

        self.statusBar().showMessage('%s . Annotation will be saved to %s' %
                                     ('Change saved folder', self.default_save_dir))
        self.statusBar().show()

    def open_annotation_dialog(self, _value=False):
        if self.file_path is None:
            self.statusBar().showMessage('Please select image first')
            self.statusBar().show()
            return

        path = os.path.dirname(ustr(self.file_path))\
            if self.file_path else '.'
        if self.label_file_format == LabelFileFormat.PASCAL_VOC:
            filters = "Open Annotation XML file (%s)" % ' '.join(['*.xml'])
            filename = ustr(QFileDialog.getOpenFileName(self, '%s - Choose a xml file' % __appname__, path, filters))
            if filename:
                if isinstance(filename, (tuple, list)):
```

➢ Algorithm, Training and Testing






jupyter Tutorial Last Checkpoint: Last Saturday at 5:59 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [45]:
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
proto_str = f.read()
text_format.Merge(proto_str, pipeline_config)
In [46]:
pipeline_config.model.ssd.num_classes = 26
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpo
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/test.record']
In [47]:
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
f.write(config_text)
6. Train the model
In [48]:
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{} --pipeline_config_path={}/{}/pipeline.config --r
python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspace/models/my_ssd_mobnet --pi
peline_config_path=Tensorflow/workspace/models/my_ssd_mobnet/pipeline.config --num_train_steps=15000
7. Load Train Model From Checkpoint
In [49]:
import os
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
In [51]:
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)


jupyter Tutorial Last Checkpoint: Last Saturday at 5:59 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
7. Load Train Model From Checkpoint
In [49]:
import os
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
In [51]:
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)
# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-16')).expect_partial()
@tf.function
def detect_fn(image):
image, shapes = detection_model.preprocess(image)
prediction_dict = detection_model.predict(image, shapes)
detections = detection_model.postprocess(prediction_dict, shapes)
return detections
8. Detect in Real-Time
In [52]:
import cv2
import numpy as np
In [53]:
category_index = label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')
In [54]:
# Setup capture
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
In [55]:
while True:
ret, frame = cap.read()
image_np = np.array(frame)

jupyter Tutorial Last Checkpoint: Last Saturday at 5:59 AM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

8. Detect in Real-Time

In [52]:
```
import cv2
import numpy as np
```

In [53]:
```
category_index = label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')
```

In [54]:
```
# Setup capture
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

In [55]:
```
while True:
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.5,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))
```

jupyter Tutorial Last Checkpoint: Last Saturday at 5:59 AM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

In [55]:
```
while True:
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.5,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(1) & 0xFF == ord('q'):
        cap.release()
        break
```

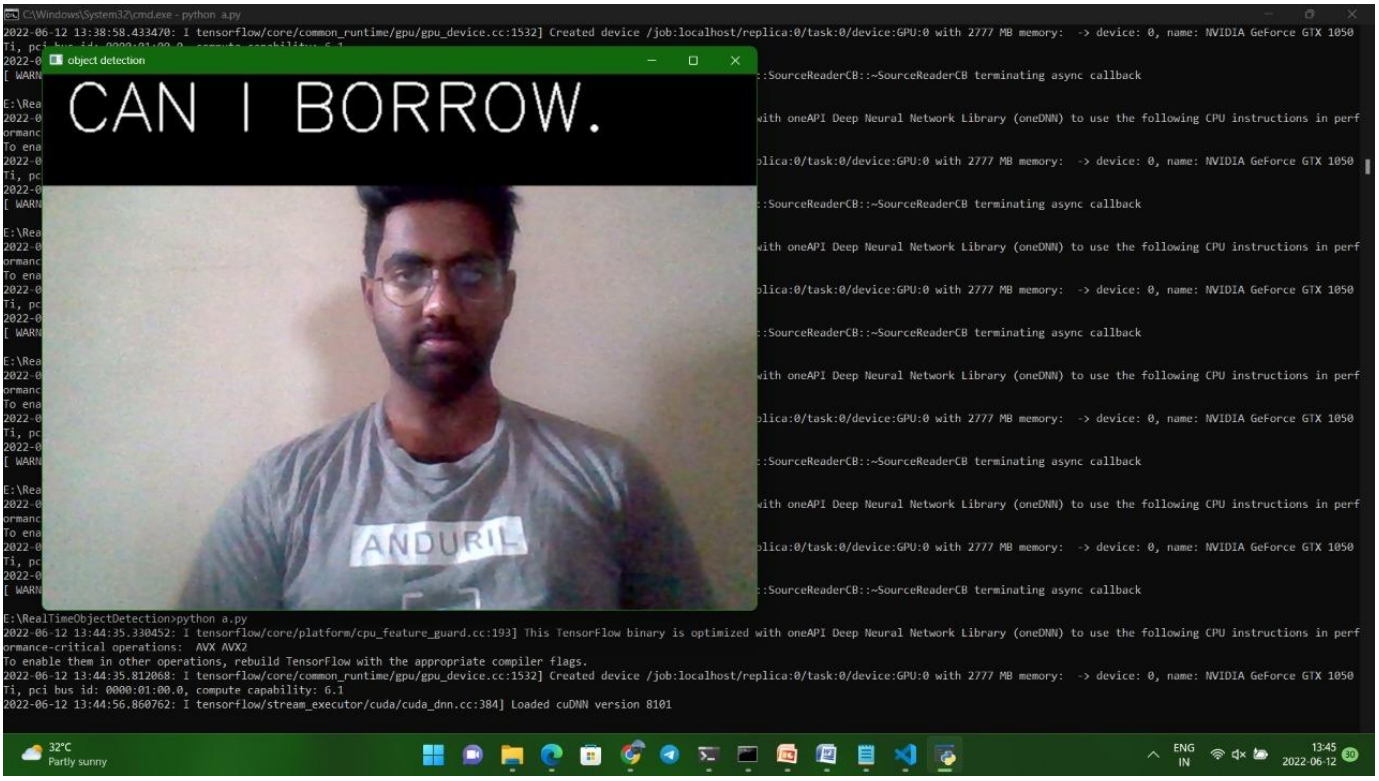
In [42]:
```
detections = detect_fn(input_tensor)
```

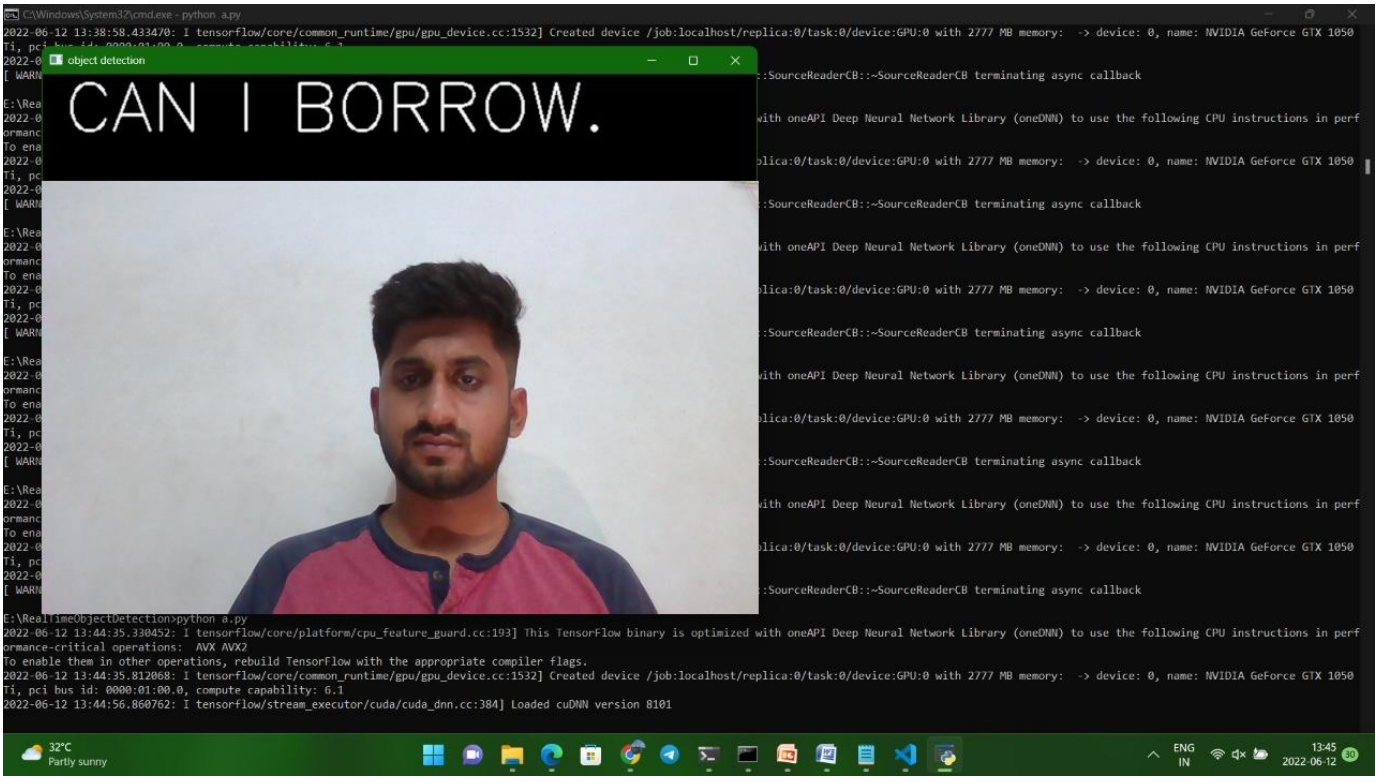
In [67]:
```
from matplotlib import pyplot as plt
```

In [ ]:

6. RESULTS

➢ OUTPUT





7. CONCLUSION AND FUTURE ENHANCEMENT

7.1. Conclusion

A functional real-time vision-based American sign language recognition system for D&M individuals has been constructed to complete this research for asl alphabets. We achieved an average accuracy of 80.0 percent on our dataset. We can improve our prediction by predicting and validating symbols that are more similar to each other after running two layers of algorithms.

If the symbols are displayed precisely, there is no background noise, and the lighting is appropriate, we can recognize almost any symbol this way.

7.2 Future Enhancements

We intend to test a variety of background subtraction techniques in order to improve accuracy even while dealing with complex backgrounds. We're also working on improving the preprocessing so that we can better predict gestures in low-light situations.

8. BIBLIOGRAPHY

References:

[1] T. Yang, Y. Xu, and "A. , Hidden Markov Model for Gesture Recognition", CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ.,Pittsburgh,PA, May 1994.

[2] Pujan Ziaie, Thomas M¨uller , Mary Ellen Foster , and Alois Knoll"A Na¨ive Bayes Munich,Dept. of Informatics VI, Robotics and Embedded Systems,Boltzmannstr. 3, DE-85748 Garching, Germany.

[3] https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

[4] Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.

[5] aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[6] http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

[7] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925.
Springer, Cham

[8] Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4),572–577 (2011)

[9] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," *2017 Nicograph International (NicoInt)*, Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9

[10] Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen ”Real-time sign language fingerspelling recognition using convolutional neural networks from depth map” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)

[11] https://opencv.org/

[12] https://en.wikipedia.org/wiki/TensorFlow

[13]https://en.wikipedia.org/wiki/Convolutional_neural_network

plag_check_report.docx

ORIGINALITY REPORT

23%	17%	5%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.jetir.org Internet Source	7%
2	samplius.com Internet Source	3%
3	Submitted to Vardhaman College of Engineering, Hyderabad Student Paper	3%
4	Submitted to Charotar University of Science And Technology Student Paper	2%
5	github.com Internet Source	2%
6	www.ijsred.com Internet Source	1%
7	Submitted to Al Musanna College of Technology Student Paper	1%
8	Submitted to Birla Institute of Technology and Science Pilani Student Paper	1%

9	Submitted to National College of Ireland Student Paper	1 %
10	Ojesh Vyas, Prateek Dembla, Rubin Jhambani, Sunidhi Manish Jain, Prashant Udawant. "Sign language illustrator", AIP Publishing, 2022 Publication	1 %
11	Submitted to Indian Institute of Information Technology, Allahabad Student Paper	1 %
12	www.slideshare.net Internet Source	<1 %
13	docplayer.info Internet Source	<1 %
14	Submitted to Strayer University Student Paper	<1 %

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		