Milestone 2 Report

Team Deinonychus – Patrick Creighton, Luka Rogic, Christopher Lee, Juntong Luo

Main Goals

From our proposal, our main goals for milestone 2 are: form datasets, start the creation of all required web application pages, start adapting previous projects, and explore different machine learning models. Our progress towards these goals will be described in the project aspects that they apply to.

Face Recognition Model

Code from sklearn's code base was adapted to convert directories of images into a dataset that can be fed into the face recognition models, which is a key step in integrating the models with our application. The datasets that were manually created from the Labelled Faces in the Wild dataset were pushed to `Curated_Datasets/lfw/` to be easily accessible by the Face Recognition notebook. Detailed observations regarding the performance of the Project 1 and Project 2 models were documented in the "Face Recognition Results" PDF. The key takeaway was that the Project 2 Linear Discriminant Analysis model was the most consistent and accurate model, especially with a limited number of images per person. The quality of these images, along with the difference in facial features across people, significantly affect the models performance. With this in mind, we expect to be able to achieve 60-80% accuracy with 7 images for each of 5 people in the training dataset, or 14 images for each of 10 people. Once we start working with the actual inputs to the model (output of one of the objection detection models), we will further experiment with different models and parameters. For the next milestone, we will begin integrating the face recognition model with the application, and move on to create a low-light image enhancement model.

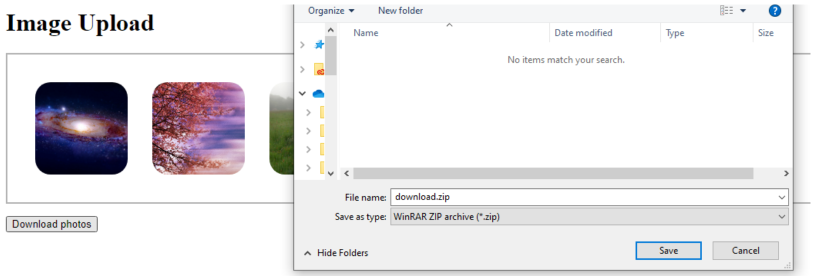
Object Detection Model(s)

We made a preliminary list of objects we want our model to identify/tag under the file Tags and Objects PDF. We also started exploring different methods for downloading images from Google's Open Images dataset. In the Tree and Skyscraper notebook, we used a python package, openimages - PyPI to download classes of images and their bounding box labels in Yolo darknet text annotation format. However, this package only includes the boxes for the requested class. For example, if you request Apple and Orange image/label files from the package, it will return two files: one called "Apple " and one called "Orange" where the Apple label files only contain labels and bounding boxes for apples even if oranges are in the image, and vice versa for the Orange label files. Thus, if two classes show up in the same image, one of them will not be labelled, which causes the model to learn incorrectly. There are several other python packages that work with the Open Images website, but all the ones we looked at have the same issue. However, the model still performs quite well. This can be seen with the "Tree_and_Skyscraper_model" notebook, which goes through getting the datasets for trees and skyscrapers, formatting the Yolov5 files, training the model on the datasets, and displaying the results at the end. For the next milestone, we will try working with the Open Image website directly to format the labels properly so our model will learn more efficiently and accurately, as well as training our model with the rest of the items listed in the Tags and Objects PDF.

We also decided to make a slight change from our proposal regarding the model structure. We were planning to use one object detection model to identify all the pre-trained COCO dataset categories and new classes we want to add. However, we realized this would be inefficient to train because in order for the model to maintain high accuracy in the COCO categories, our training dataset would have to include COCO images in addition to images for the new classes we want to add. Considering that a dataset with ~600 images already took us around ~30min to train on 50 epochs, training one that includes the COCO dataset of 100,000+ images would take a very long time, time that could be better spent experimenting with the model. For these reasons, we now plan to use two Yolov5 object detection models: one pre-trained on the COCO dataset, and one that we will train on the new classes we want to add.

Flask Web Application

In this milestone, we improved upon the foundational web application that was built in the prior milestone. The first thing we did was fix the issue that the application was only accepting JPG files to be uploaded, so now our program allows JPGs and PNGs as input and gives an error message otherwise. Next, we added the ability to download photos stored in the temporary folder of our application as a zip folder, which will be critical in returning the tagged photos from the app to users. So as of now, our application allows users to import photos from their computer and then gives them the option of downloading a copy of them (code found in main.py and templates/main.html on GitHub).



However, this is only working when the web page is run locally and not on the Google App Engine because of an issue with not having permissions to write to the zip folder for security reasons. Despite this hurdle, we are confident that there'll be an easy solution as creating zip folders is a pretty common task that we're sure many people have done before us. For the coming weeks, we plan to integrate the web page with our machine learning model by simply running our model to tag the photos when the user uploads them. As well, we want to further expand the list of features in our web page starting with an image gallery that displays photos and their tags when they are clicked on. Our ultimate goal with the webpage is to just ensure that the fundamental components are there and working, and then building upon it one by one so as to not overwhelm ourselves with the development of multiple features at once.

Python Backend

We improved our image tagger to tag images based on the output of a pre-trained YOLOv5s model. We also reorganized its code so that other python files can easily access it. The new image tagging program is in `Backend/tagger.py`. It feeds all the images in the provided folder to the YOLOv5s model, filters out labels with low confidence, and tags the original images based on the rest of the output. A separate file named `Backend/tagall.py` demonstrates the usage of it. Further, we modified our backend server to tag uploaded images with the tagger before returning them to the user. For Milestone 3, we will replace the pre-trained YOLOv5s model with our own object detection and face recognition models.