**BuildingIdentifier (ML/AI) Summary**

Purpose: the purpose of this summary is to describe the various files in the BuildingIdentifier folder, and how they work and what they do as best I can remember. This project has various pieces located in various places, with a lot of information contained both on paper and in the mind.

Summary: the intent of this project was to be able to train a machine learning model to learn and classify images of buildings into one of 6 construction materials. We tried various approaches in order to achieve the most efficient model that we could, moving from a virtual machine to Mr. Boe’s Desktop that has a dedicated GPU, and reached a maximum accuracy of 64% before we ran into some memory issues. Then, we started looking for methods online, and found Google’s autoML, which should be able to yield us much higher percentages and accuracy/precision/recall, while being easier to maintain and deploy.

Generally speaking, images were annotated using the ImageAnnotator software. They were then trained on using the BuildingIdentifier.py script, as well as an attempt to use KerasTuner using the KerasTuner.py and KerasTunerO.py scripts. Many changes and processing of data that needed to be done was done using the ProcessData.py and FilterImages.py scripts.

Files

* Potential Datasets – this folder contains some potential datasets, both in the form of actual images, and also in the form of links/connections to those images or their sites, or just containing labels that relate to the images. Some of these were used and some were not, as specified in the ImageAnnotator software and in the Datasets.txt.
* QC\_Boe and QC\_Taha – these contain output data from the ImageAnnotator (in the form of 3 cols with numbers in them, the first is the primary construction material, the second is the secondary construction material, and the third is the building use, followed by 3 more columns with those same values but as Strings), as well as output data from QCing the data (running a certain number of images through again to check for human errors and variance), and a comparison between outputs of the same person as well as both people.
* Street\_View\_Test – folder that contains the basis from which this idea formed, and also has some scripts and images that could be useful when deploying this software in the future. Mr. Boe made this one so he has more knowledge on it.
* BuildingCategories.xlsx – the 22 building use categories we decided on.
* Datasets.txt – info on the various datasets I looked at and generally why I chose some over others. Also contains some more info on dataset selection and other things to consider.
* ModelTrainingNotes.xlsx – This is the record of model training runs we performed on Mr. Boes desktop using the trained images in the ConstructionMaterialDataset, and any modifications and parameter changes we used. The top runs are also mentioned here and could be used for Vision ML with Google so that you don’t have to start from scratch as to what to set parameters to.
* Output.csv – file that contains output from annotation, as mentioned above
* outputoriginalQC.csv – 100 quality checked images (redone basically)
* outputQC.csv – 966 QC’d/redone images.
* Output-tboe.csv – some annotated images by Mr. Boe
* WikiNotes.docx – Notes from my Teams Wiki. It contains information on various useful things:
  + Intro and general overview of the project
  + How the data was chosen and the images were split up
  + Notes on the development of the model
  + Different image augmentation options, which can be used to increase the total dataset size and increase the variety of the image training data.
  + Some more information on KerasTuner
  + Info on 3rd party outsourcing options
* ImageAnnotation folder
  + Annotated – the folder containing all of the annotated images
    - This, the Temp, and the ToAnnotate folder are all used by the ImageAnnotator. It takes images from the ToAnnotate folder, and when working on them puts them in the Temp folder, and then when images are annotated they are moved to the Annotated folder.
  + Temp – as explained above, images temporarily placed here during annotation
  + ToAnnotate – images that still need to be annotated, as explained above
  + ConstructionMaterialDataset – the main dataset of images, split into train and validation datasets, with each one split into the 6 classes/labels, each class/label folder contains the images that have been annotated as that class/label. The BuildingIdentifier.py method 1 (you’ll see in the comments of that script) takes images from a folder with this structure, and determines what the image labels are based on the file structure, and then trains the model using that. This is all to emphasize that the file structure here is important, although it is not the only way for images to be input into the model trainer. If someone wanted to change the way data is input, they would also have to change a few things in BuildingIdentifier.py.
    - Also contains folders that have images that were determined to be not useful to the dataset, and they are already/automatically ignored by BuildingIdentifier.py.
  + ImageSet154x154 – same as the above dataset, except all images were resized to 154x154. This was done because we had memory issues with the original and modified sizes of the images (256x256 and 415x415 respectively), although the smaller size does result in a loss in effectiveness.
  + KerasTunerTest – Contains training images split into the same file structure as mentioned above, but used as the test dataset for the KerasTuner scripts.
  + KerasTunerTestImages – all of the annotated images from the valid ConstructionMaterialDataset folders (meaning from the 6 folders in validation I believe), but modified for use for KerasTuner. I believe these were then split into the various folders in the KerasTunerTest.
  + KerasTunerTrain – same as above but for the train dataset
  + KerasTunerTrainImages – same as above but for the training dataset
  + SubsetofImagesReAnnotateforGooglesVision – folder of images that were reannotated and then uploaded to googles Vision ML online in order to test it out. I believe about 25% of the total images, so around 2300 images.
  + BuildingIdentifier.py – this is the main script for this project, it has two methods (you’ll see one commented out). The first is the original and the main way of training a model. It reads in data using the file structure mentioned above, performs image augmentation and other data preprocessing methods, then builds and compiles the model, and then runs it.
    - Method 2 – it pulls an already built model from online in order to use its training weights and layers, since those models have been trained on datasets far larger than this one. It removes its top layer as that’s the one with the least use for us. This is called Transfer Learning. It then freezes some of those layers, mainly the top ones, and then trains the rest of that models layers on our data, thus making it more specific to our data and usually more effective. This is called Fine Tuning.
  + FilterImages.py – I believe this was just used to take images and put them in there proper place in the proper order corresponding to the correct labels (which were also in the proper and same order) without having to do it manually, all for the purpose of the KerasTuner.
  + KerasTuner.py and KerasTunerO.py – KerasTuner is a method of automating tuning a model. What it does is allow you to set up your parameters for model training, and it will run a model training run, and then normally you would have to manually check it and adjust parameters, but KerasTuner will automatically update those/change them for and then move right to the next run. You determine the range and choices that it runs through, but it is very useful because you can leave it to run for a week or so, and it will go through a large variety of combinations of model parameters, without much help or input by you, and you can then have it document its progress, as well as have it save the best model that it trained.
    - Both of these files are attempts at doing that, and very likely would have worked had it not been for the memory issues we faced (wasn’t really able to test them). One of them is less resource intensive and has a smaller model that does less iterations/combinations, whereas the other one does far more and is larger.
  + ProcessData.py – this script contains various image processing functions, which I used to automate things so I wouldn’t have to do them manually.
    - The first function reads the output.csv file, and moves images from the Annotated folder to their respective place in the train or validation folders.
    - The second moves images back from Temp to ToAnnotate, which occurs when someone quits an Annotation session early or some other error occurs, leaving unannotated images in the Temp folder.
    - The third removes any duplicates from the ToAnnotate folder
    - The fourth checks if an image has been represented and thus annotated in the output.csv file, and if not, it moves them back to the ToAnnotate folder.
    - The fifth clears the training and validation folders in the ConstructionMaterialsDataset.
    - The sixth counts the images in the train and validation folders.
    - I’m not 100% sure what the seventh does, but I believe it checks the output.csv against a QC output.csv file and outputs the results from that.
    - 8th converts images in the KerasTunerTrain and KerasTunerTest folders into .png
    - 9th not a 100% sure here either exactly what’s done but I know it has to do with the KerasTuner and ordering images and labels in the same order.
    - 10th function moves images and then resizes them, I believe this was also done for KerasTuner
  + test\_images.csv, test\_labels.csv, train\_images.csv, train\_labels.csv – all of these are for KerasTuner, and have to do with correctly ordering the training and test images and training and test labels correctly so that it works properly. If not using KerasTuner, these can certainly be deleted.
  + VisionInput.csv – a csv file that corresponds to data in a certain format on Googles Vision ML. The data must be stored in one of their buckets, and then you must upload a .csv file that references the urls to those data, which the system then uses to do the training. I uploaded data in my usual file structure (split by construction material), and then used a script to create this and the below CSV file. This one was for the whole annotated dataset that was contained in the ConstructionMaterialDataset.
  + VisionInput2.csv – just like above. This one was for the smaller, reannotated dataset, which was around 2300 images, and performed much better than the larger one (probably due to better quality data).

NOTE: some of the image folders mentioned above are not contained in the ImageAnnotation folder that is within BuildingIdentifier project folder, due to the fact that they were too big to move. I moved them instead to the O drive, and also have them shared with Mr. Boe from my OneDrive.