Sheffield Hallam University

College of Business, Technology and Engineering

Department of Computing
Project (Technical Computing)
[55-604708]
2020/21

Author:	**Michael Elsom**
Student ID:	**27035059**
Year Submitted:	**2021**
Supervisor:	**Babak Khazaei**
Second Marker:	**Zairul Jilani**
Degree Course:	**BEng Software Engineering**
Title of Project:	**Building a Machine Learning Domino Counter: An Investigation Into Object Detection In a Mobile Environment**

Confidentiality Required?

NO

Table of Contents

1 Introduction and Aims

Machine learning is quickly becoming a hugely popular topic in the world of computing. Also, according to a global mobile market report produced in 2019, smartphones were used by around 3.2 billion people worldwide (Newzoo, 2019). The purpose of this report is investigate machine learning for object detection and how it can be applied in a mobile environment (on a smartphone).

The objective of this project is to produce a mobile app that can take a photo or upload a photo and then use object detection to detect different domino tiles and add their values together and return that value to the end user. This will require an investigation into machine learning and how it can be applied for object detection. It will also require an investigation into the best way to deploy machine learning in a mobile environment.

The report will be split into four main sections. The first will detail research done into machine learning and object detection, there will also be research done into existing object detection apps in the mobile space. The second section will detail the design and planning of the application, including how best to deploy object detection in a mobile environment. The third section will be about the deliverable, how it was developed and how any issues were overcome. The final section will be a critical reflection on what went well, what did not go well, and what could be improved upon in future.

The driving factor behind this project is to further my understanding around machine learning and object detection and to produce a working mobile app that can be used as a sort of ‘proof of concept’ for how object detection can be applied on mobile devices.

2 Research

2.1 Machine Learning

Machine learning is the term used to describe algorithms which can find patterns in data. Data could be anything, e.g., Netflix providing recommendations based on things you have already watched. However, for this project the data we will be focusing on is images. There are three different types of machine learning: supervised, unsupervised, and reinforcement learning.

2.1.1 Supervised Learning

Supervised learning is a type of machine learning by which the algorithm is supplied with a large set of reference data to produce a model which is then used to predict whether new data is in any way related to the reference data. To put it into the context of images, you might provide the algorithm with thousands of images of dogs which are all labelled as dogs, the model produced would then be able to be given an unlabelled image and tell whether it is a dog or not.

And example of a supervised learning algorithm is a support vector machine (SVM), at their simplest the way an SVM operates is that it draws a line between two classes of data, plotted on a graph, in order to find the maximum margin between classes, the margin is determined by the perpendicular distance between the two objects from each class which are closest to the line[2]. Due to the complexity of the data contained withing images, SVM is probably not suitable for the classification of images.

A more suitable supervised learning algorithm for image classification/object detection would be *k*-nearest neighbour (kNN). This is an algorithm by which a class is assigned to an unclassified object based on how similar it is to know classes around it. For example, if a class has three objects of class A near to it and only two objects of class B then the algorithm will assign it class A. The value of k is basically a value given to the algorithm to tell it how far away from the unknown class to look for neighbours (Bzdok, Krzywinski and Altman, 2018).

2.1.2 Unsupervised Learning

Unsupervised learning, or clustering, is the process by which an algorithm looks at a large set of data and finds links between different data points and clusters them based on those links. This kind of learning is used heavily in marketing, for example to group customers based on what they purchase to issue directed advertising.

Unsupervised learning methods were not researched in detail as part of this project due to them being unsuitable for image classification and object detection.

2.1.3 Reinforcement Learning

Reinforcement learning is essentially training an artificial intelligence to perform a task in the best possible way starting from nothing. This is done by providing the AI with a reward or penalty based on any action it performs. The machine then learns by taking a trial and error approach over and over again until it can perform the set task (Błażej Osiński, 2018).

Again this type of learning is not suitable for the task at hand and as such was not researched further.

2.2 Object Recognition

Object recognition is a term that describes a grouping of computer vision related tasks that can identify what objects exist in a photograph or image.

There are three main types of object recognition: Image classification, object localisation, and object detection.

Image classification is the process by which an image with a single object is provided and the computer predicts what the object is and returns a class label.

Object localisation is where for a given object and image, the computer will detect whether that object is within the image and return the bounding box for that object, a bounding box being coordinate, width and height.

Object detection is a combination of the two above techniques, an image is processed by the computer and bounding boxes of individual objects are returned along with their class label and percentage probability (Jason Brownlee, 2019). There are two major models for object detection with machine learning, Region-Based Convolutional Neural Networks and ‘You Only Look Once’.

2.2.1 Region-Based Convolutional Neural Networks (R-CNN)

Region-Based Convolutional Neural Networks (R-CNNs) were first proposed in the paper *Rich feature hierarchies for accurate object detection and semantic segmentation* (Girshick et al., 2014) and are comprised of three modules. The first module generates region proposals, that is it scans the image and detects potential regions which could contain important data and assigns each region a bounding box. The second module is the Convolutional Neural Network (CNN), this is a deep learning algorithm specifically designed for image analysis, that is used to extract a feature from each individual region. The third module uses an image classifier to assign the feature to a known class.

One of the major downsides to this method described in the original paper is the time it takes to process a single image; this is due to the fact that the first module generates around 2000 region proposals per image and each of these regions are passed through the CNN.

2.2.2 You Only Look Once (YOLO)

You Only Look Once (YOLO) models differ from the region-based convolutional neural networks described above by the fact that instead of predicting the regions and then running the neural network on all regions, the YOLO models run the neural network just once against the entire image and generate the bounding boxes and classes on the fly.

First described in the paper *You Only Look Once: Unified, Real-Time Object Detection (Redmon et al., 2016)*, It works by splitting the image into a grid. Each grid cell predicts a number of bounding boxes, how confident it is that it contains an object and how confident it is the bounding box is correct. While YOLO is less accurate than R-CNN, it is considerably faster, according to the paper cited above it can process images in real time up to 45 frames per second.

2.2.3 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detectors (SSDs) are a method for object detection that only use a single neural network much like YOLO described above (Liu et al., 2016). When training an SSD model, it only needs the input image and ground truth boxes for each object within the image. Much like YOLO, for an unknown image, the model breaks the images up into a fixed-size collection of bounding boxes and the scores each of these boxes with the likelihood it contains an instance of an object class. It then runs through additional layers to further refine where the objects are within the bounding boxes to produce the final full bounding boxes for each individual object. At training time these bounding boxes are compared with the truth boxes and the others discarded, then when it comes to an unknown image the model can give percentage predictions on how likely it is that the bounding box is correct.

According to the paper cited above, SSDs are both faster and more accurate than YOLO models. The SSD has a mean average precision (mAP) of 74.3% and runs at 59 frames per second (FPS) on an NVIDIA Titan X compared with the YOLO model which has a mAP of 63.4% at 45 FPS.

2.3 Existing Applications

2.3.1 Google Lens

Google Lens is a multi-feature object detection app for Android and iPhone, it uses computer vision and object recognition, combined with Google search algorithms, to provide several services such as: plant and animal classification, translating text in images, formula solving, and others.

2.3.2 TruPicShop

TruPicShop is an app that allows a user to take a picture of an item of clothing and then uses image recognition to show a price comparison from different online stores. It also shows the user a list of similar items that might be of interest. Users can also share results with friends.

3 Planning and Design

3.1 Hardware considerations – Development and Deployment

Development will be done on a windows PC and the final app will be deployed to an android device since these are the two environments that I have access to.

Training object detection models can be very resource intensive, therefore the PC required to develop said models must be powerful enough to train the models in an acceptable timeframe. However most modern hardware should be up to the task with no issues.

The android device will need to have a camera in order to take the pictures that will be evaluated; however I will also be planning to implement image upload functionality for cases where no camera is available. Most modern android devices should be sufficient to run evaluate an image against the model as the plan is to use a method designed for mobile devices.

3.2 Machine learning libraries

Developing a machine learning model from scratch would require huge amounts of time and the use of some very complex mathematics and is therefore well outside the scope of this report. Therefore, there was a need to investigate existing machine learning libraries and libraries for implementing object detection.

3.2.1 PyTorch

PyTorch is a machine learning library developed by Facebook. Written in Python it "enables fast, flexible experimentation and efficient production through a user-friendly front-end" (https://pytorch.org/features/). The main benefit to PyTorch is that it has a relatively simple API for working with Neural Networks as opposed to other libraries out there. PyTorch also has an experimental mobile API that provides the pre-processing and integration tasks required for adding machine learning to Android and iOS which would be for this project.

3.2.2 Mlpack

Mlpack is a machine learning library built in C++, although it also has support for Python, it implements several standard machine learning algorithms such as the nearest neighbour algorithm described in section two and it also contains a reinforcement learning framework. It markets itself as having fast and scalable implementations of cutting-edge machine learning algorithms (Curtin et al., 2018).

However, at this time the documentation on how to use mlpack is sparse and therefore it would be unsuitable to use this library for this project since it is unclear as to how much effort will be required to harness this library for object detection.

3.2.3 OpenCV

OpenCV or Open-Source Computer Vision Library, first developed at Intel in 1999, is a machine learning library mainly aimed at computer vision, as the name implies (https://opencv.org/). It includes a library for object detection which would be relevant to this project. Written in C++, it has APIs that provide cross-platform support for Python, Java, and MATLAB. It supports GPU acceleration provided by NVIDIAs CUDA and OpenCL.

OpenCV also provides an SDK for integration of the Java API into android applications which would be useful. However, the official documentation for setting up the android SDK is quite out of date so would take some time to get it set up in a more modern development environment.

3.2.4 TensorFlow

TensorFlow (https://www.tensorflow.org/), developed by Google, is an open-source machine learning library. It provides high-level APIs to easily build and train models. Models can be built and trained in either Python, using TensorFlow, or JavaScript, using TensorFlow.js. There is also an API for deploying trained models to mobile (Android, iOS) and embedded devices (Raspberry Pi) called TensorFlow Lite. However TensorFlow Lite cannot be used to train models, only to deploy them.

TensorFlow also provide a library to easily convert models trained using the core library to models usable in TensorFlow.js and TensorFlow Lite. This means that the model can be trained in a desktop environment and then converted for use on the selected platform.

Another advantage to TensorFlow is that it provides a number of official and community produced pre-trained models which can be used as a starting point rather than creating a model from the ground up. The main advantage of TensorFlow for this project though is the large amount of easy to understand documentation on how to set everything up and use the APIs for many different use cases.

TensorFlow also has an API specifically developed for the training and deployment of object detection models (https://github.com/tensorflow/models/tree/master/research/object_detection) (Huang et al., 2017). This API can be used to easily produce a TensorFlow model from a collection of labelled images and then export this model for use in Python. The python model can then be run through the TensorFlow converter to produce a model usable by TensorFlow.js or TensorFlow Lite. This API works using a number of machine learning algorithms provided by TensorFlow such as SSD and R-CNN described in section two. This API would be very useful to train the model required for the deliverable.

3.3 Mobile Development

As stated in section 3.1, mobile development will solely focus on producing the deliverable for the Android platform. This is due to the fact that Android is the platform that I have main access to and also I don’t have a Mac which is required for iOS development. However, I do know that there are options out there that allow for platform agnostic mobile development so I decided it would be a good idea to research the options available to me.

3.3.1 Android Studio

By far the most popular choice for Android development would be Android Studio (https://developer.android.com/studio). Android Studio is an IDE that provides a robust set of tools for developing and deploying Android apps. It comes with a build in WYSIWYG editor that allows you to visualise your apps while in development and code behind can be written in both Java and Kotlin with plugins to allow running of C++ code. It allows you to build and deploy directly to a physical device and also provides the option to download and install a number of different versioned emulators to test apps directly on your PC. This is a decent option, however, I have only briefly done android development before so would need time to learn. The positive side to this is that the documentation and tutorials provided for new android developers are very good.

3.3.2 Xamarin

Xamarin is a platform created by Microsoft for the development of cross-platform mobile apps, both Android and iOS, using .NET and C# (https://dotnet.microsoft.com/apps/xamarin). It is an extension of the .NET platform which is a huge C# library with many different features. Xamarin also features a built-in designer for Android which works similar to the one found in Android Studio. One of the major advantages to Xamarin is that you can write your code once and it can be deployed to both Android and iOS, meaning that if I decided to port the app to iOS in the future it would not be much of an issue.

3.3.3 Ionic

Ionic is another cross-platform development solution (https://ionicframework.com/). However, Ionic uses Android and iOS native wrappers to deploy apps developed using modern web frameworks with any native functionality not supported by the web frameworks, for example the camera, provided by Ionic plugins. The frameworks available are React, Vue, and Angular. The disadvantage to this is that Ionic requires the native IDEs to deploy the apps, such as Android Studio for Android, meaning it would not be as easy to deploy to iOS. However, I have experience developing Angular applications in Ionic so it would make sense to use this option.

Another advantage to using Ionic would be that it should be possible to implement TensorFlow.js within the application. The advantage to this over using TensorFlow Lite in a native application, is that in the event of a failed calculation we could ask the user to provide some feedback and train the model on the fly, which is only possible with TensorFlow and TensorFlow.js.

3.4 UI Design

The UI design will be straight-forward, there will need to be a button for the user to take a new photo to be evaluated which will open the devices camera. There will also need to be a display for the previous result. I used figma (https://www.figma.com/) to do a basic wireframe of how the UI should look.

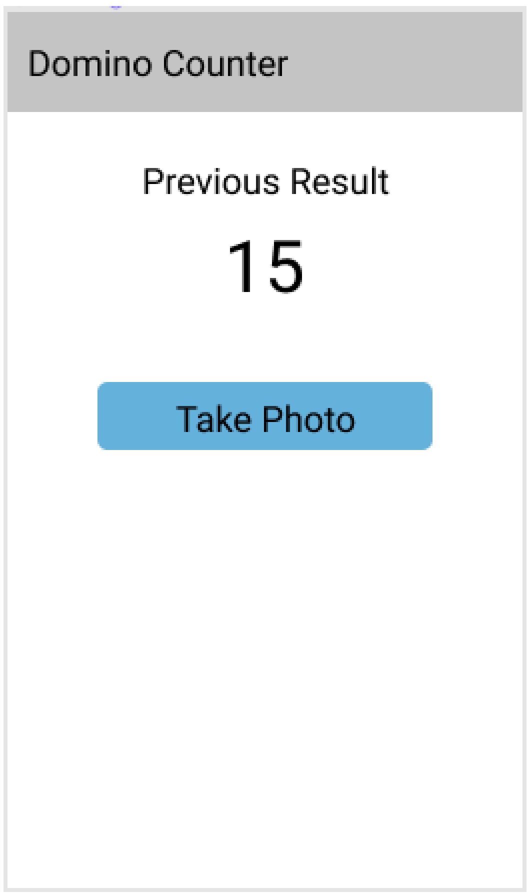


Fig 1: Wireframe of UI Design

The UI will be produced using HTML and CSS, for a clean modern look I have decided to use bootstrap to style the UI (https://getbootstrap.com/).

3.5 Software Engineering Approaches

3.5.1 Project Management

I had initially intended to take an agile approach when it came to managing the different tasks required for this project. My plan was to split break down what I needed to do for the project into easy to manage sections, including research and planning, and then further break down these sections into smaller sub-sections. However, none of this planning was ever formally documented, I kept the plan only in my head. This worked for the most part during the research and planning stages, however, when it came to the development stage of the project this agile approach very much changed into a more free-fall approach to software development. Please see section 5 critical reflection for more discussion on this topic.

3.5.2 Version Control

Version control is an important aspect of any software development projects, within teams it allows for easy collaboration on a project with a single source of 'truth', that is the central repository that everyone will pull from or merge to. However, even in a solo project it is still a good idea to utilise some form of cloud-based version control. It serves as a back-up in the event of catastrophic hardware failure, allowing the developer to continue working on another machine. If correctly used it also provides a record of what changes have been made to the system and why. Finally it provides a way for the developer to revert to previous versions of the system should they make any changes that break it.

I decided to use Git for this project since it's a form of version control I have used extensively before and there are many online providers for hosting private repositories. In this case github.com was chosen as the provider as they provide free private repositories for students.

3.5.3 Library Management

As the project will rely on a number of third-party libraries there will be a need to have something in place in order to keep track of what libraries are being used. A good way to do this is through the use of package managers. Package managers allow software developers to easily install libraries and keep them up to date. You can also use them to install old versions of libraries for example if you have some code or another library that relies on deprecated functionality. Since the training of the machine learning model will be using Python, pip is the recommended package manager (https://pypi.org/project/pip/). The front-end application being developed in Ionic for Angular will use the node package manager (NPM)

Another useful tool for developing machine learning applications in Python is Anaconda (https://www.anaconda.com/products/individual). Anaconda allows the user to setup virtual Python environments, each with their own list of installed packages allowing the user to easily switch between projects which may require either different packages or different versions of the same package.

Certain libraries, such as the TensorFlow Object API described above, are not available through package managers and as such will need to be installed and updated manually. This is not too much of an issue as these kinds of libraries are few and far between.

3.5.4 Testing Strategy

In order to test the deliverable, there are certain metrics that can be used to evaluate how accurate an object detection model is when applied to an unknown. The two metrics are Recall and Precision [18]: Precision is the ratio of correct predictions, or true positives (TP), to the total number of positive predictions, true positives plus false positives (FP). Given by the formula:

$$P = \frac{TP}{TP + FP}$$

Fig 2. Formula to determine the precision of an object-detection model

Recall is the ratio of correct predictions, or true positives, to the total number of actual objects in the image set. Any objects that were not detected are called false negatives (FN). This is given by the formula:

$$R = \frac{TP}{TP + FN}$$

Fig 3. Formula to determine the recall of an object-detection model

A small program will need to be written that can evaluate images and draw bounding boxes provided by the model in order to count true positives, false positives, and false negatives.

The testing strategy for the model when deployed on the front end will be to use it and check if the value returned to the user is the value expected from the dominos that are in the photo.

As the app is only small, I do not think that automated testing is relevant at this point in development. However, it might be something to consider in the future should more functionality be added.

4. Deliverable

4.1 Initial Plan

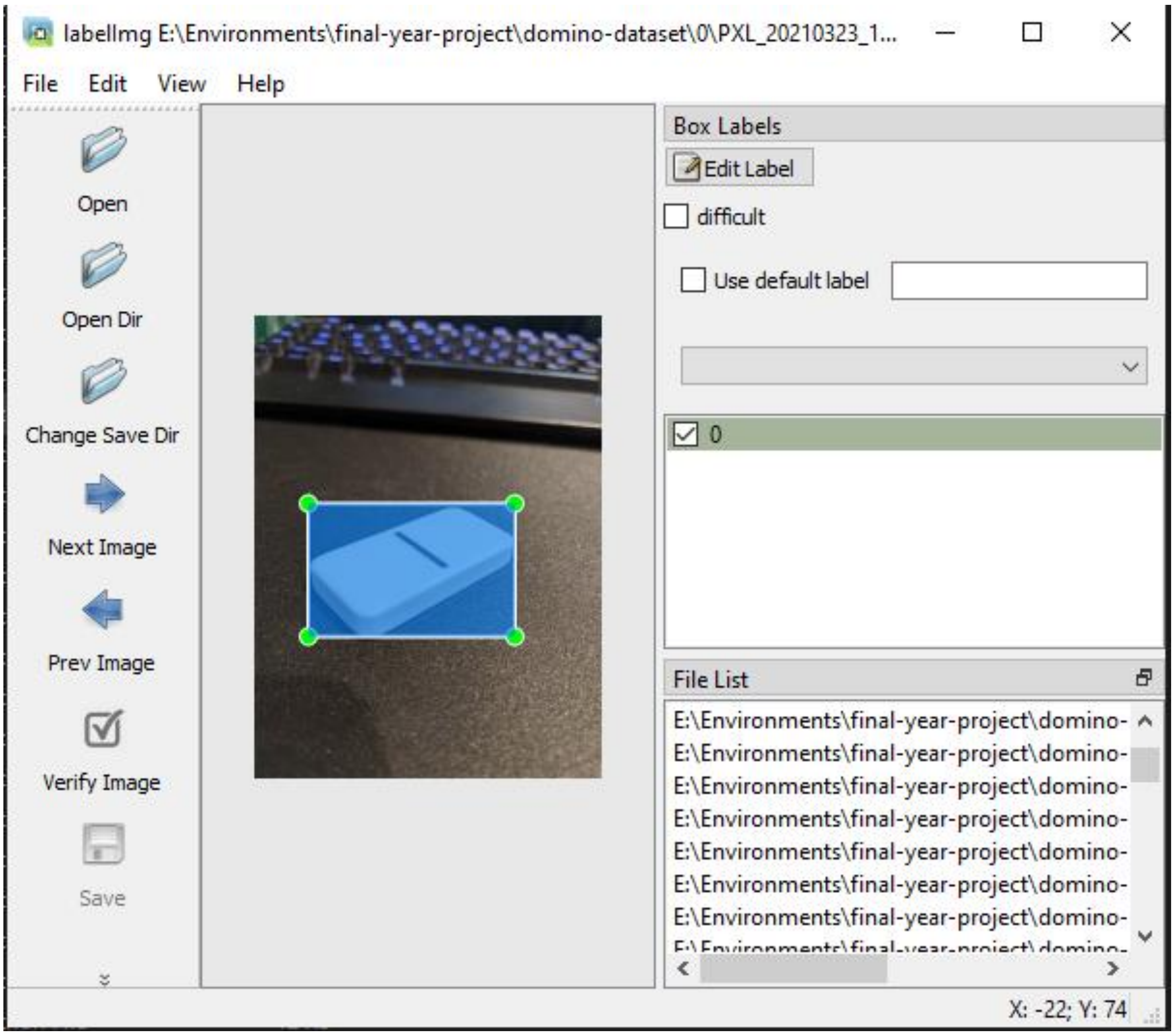
The initial plan for the development of the deliverable was to train an object detection model using TensorFlow and the TensorFlow Object Detection API. This model would then be converted into a TensorFlow.js model using the TensorFlow.js converter. An Ionic android app would then be developed that would allow the user to take photos with the devices camera to be evaluated by the converted model.

It soon became apparent however that creating a model and training it from scratch would be far too difficult to carry out in the timeframe of the project due to my inexperience in the subject area. However, there was a way forward. TensorFlow allows for a concept known as 'transfer learning', whereby an existing publicly available model is retrained with a new data set. Many of these models are packaged with the TensorFlow Object Detection API. It was decided that the model to be used would be the SSD version of MobileNet V2 which trades off accuracy for speed. The decision was made to prioritise speed over accuracy as it is being deployed to a mobile device the end user will expect a quick response from the evaluation.

4.2 Development

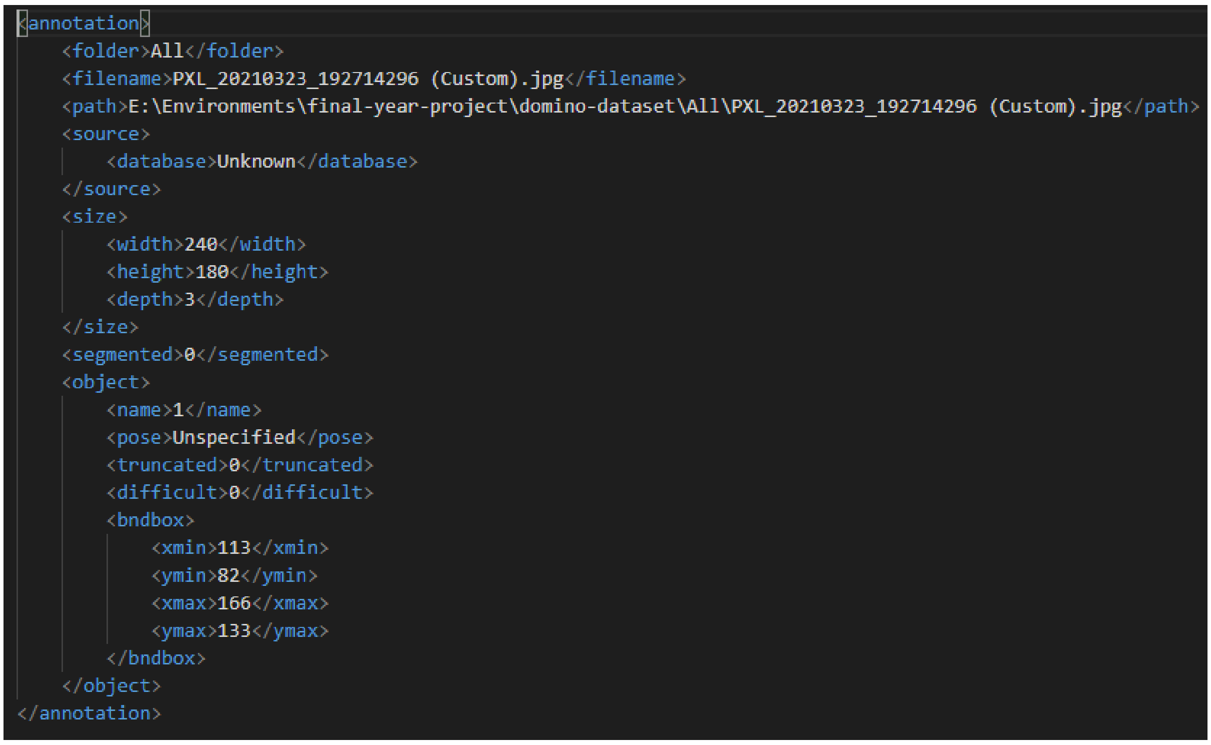
The first part of the development was to create a functional object detection model that could detect different dominos. To do this a dataset was required. Over 500 pictures of the dominos from double zero to double six were taken in various orientations. The way the TensorFlow Object API reads in images for training is that it requires each image to have a corresponding XML file with details of any objects present in the image and the coordinates and size of their bounding boxes.

Initially the task seemed daunting to individually create a file for each image and input the bounding box coordinates. It was decided that it would be worth investigating if there was a program that could do the hard work. Upon investigation a program called labelImg was found (https://github.com/tzutalin/labelImg). This program allows the user to draw boxes around objects in an image and assign them classes. The user can then save the output and generate an xml file that can be used by the object detection API.



E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-
E:\Environments\final-year-project\domino-

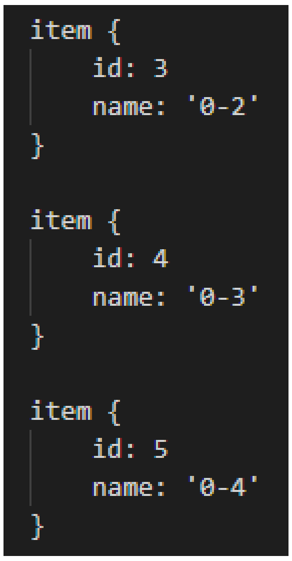
Fig 4. An example of LabelImg being used to draw a bounding box on the double zero



```xml
<annotation>
    <folder>All</folder>
    <filename>PXL_20210323_192714296 (Custom).jpg</filename>
    <path>E:\Environments\final-year-project\domino-dataset\All\PXL_20210323_192714296 (Custom).jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>240</width>
        <height>180</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>1</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>113</xmin>
            <ymin>82</ymin>
            <xmax>166</xmax>
            <ymax>133</ymax>
        </bndbox>
    </object>
</annotation>
```

Fig 5. An example of the output xml created by LabelImg

It was then a case going through all the images and labelling them. At this point the plan was to label the full dominos to create twenty-eight classes from ‘0-0’ to ‘6-6’. The images along with their xml files were placed into a training folder ready to be read into the object detection trainer. A pbtxt file called label_map containing the names of all the classes was also created and these were given an ID from 1 to 28. The purpose of this is to give all the classes a numerical value as that is what the output will be when the model is being evaluated.



```
item {
    id: 3
    name: '0-2'
}

item {
    id: 4
    name: '0-3'
}

item {
    id: 5
    name: '0-4'
}
```

Fig 6. An example of classes defined in the pbtxt file

Now that the images had all been labelled and the classes defined they needed to be combined into a TFRecord format to be used to train the model using the TensorFlow Object Detection API. TFRecord is a format used by TensorFlow to store a sequence of binary records. Creating a program to generate a TFRecord file from scratch would have been too difficult in the time frame, especially so due to a lack of experience coding in Python. Upon investigation a tutorial was found which explained how to use the TensorFlow Object Detection API, this tutorial also provided a helper script written in Python to generate TFRecords (https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html). What this script does is it goes through each image and encodes the jpg file and gets its width and height, it then uses the xml that is called the same thing as the jpg to get information on any objects within the image and any bounding box information. All of this is then serialised to a string which is added to the TFRecord.

Once the TFRecord is generated the model needed to be set up. Taking an existing model as described in section 4.1, SSD MobileNet V2, the pipeline.config was extracted into a new folder that would contain our new model once the training was complete. The pipeline.config contains the variables that the API uses to train the model. Most of them can stay the same as this is the difference between retraining a model and creating once from scratch. The fine_tune_checkpoint variable needs to point to the existing models checkpoint so that our training uses this as a starting point. There are also variables to point to the TFRecord and the label_map. Finally there is a variable that indicates to the model how many classes it should be training for, in this case twenty-eight, and also the dimensions of the image it is expecting.

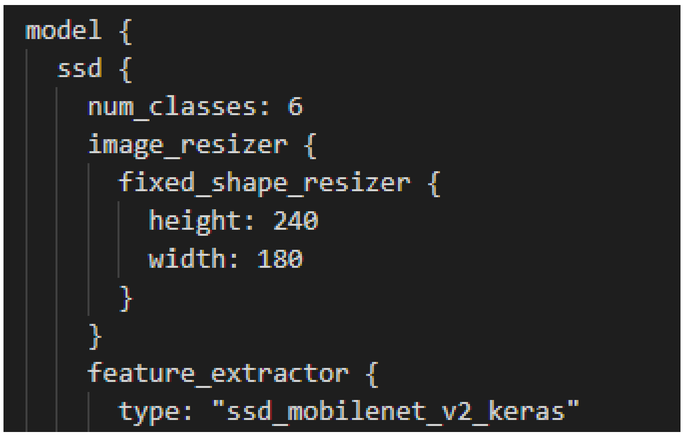

```
model {
  ssd {
    num_classes: 6
    image_resizer {
      fixed_shape_resizer {
        height: 240
        width: 180
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_keras"
```

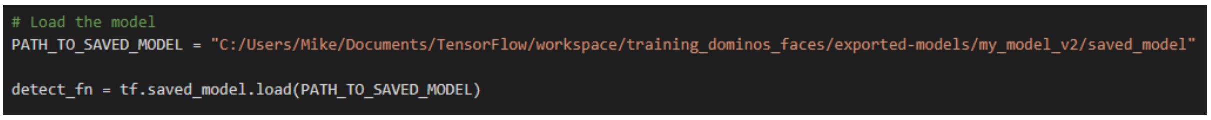
Fig 7. An example of the structure of the pipeline.config

After the pipeline.config was edited, and all the variables were correct it was time to run the object detection API script to train the model. This was just a case of setting it going and waiting for it to complete. The training runs in steps with each step being a random batch of images processed by the trainer. The size of this batch can be changed in the pipeline.config file however I decided to leave it at the default value of eight. Each step took around 1 second and the model is set to run for 50,000 steps when training. By default, the script outputs every 100 steps letting the user know how many seconds per step and what the loss is for the model. Loss was not a very useful metric for this run as no test images had been setup for the model to compare against.

After completion of the model it needs to be exported into a format that is useable. The TensorFlow object detection API comes with a Python script that takes the path to the checkpoint file created from the training described above and a type we want to convert it to, in this case an image tensor. This exporter outputs the model in a Saved Model format which can be then used by TensorFlow to evaluate other images.

Now it was time to evaluate the model. A python script was written that does the following:

1. Loads the saved model that has just been exported

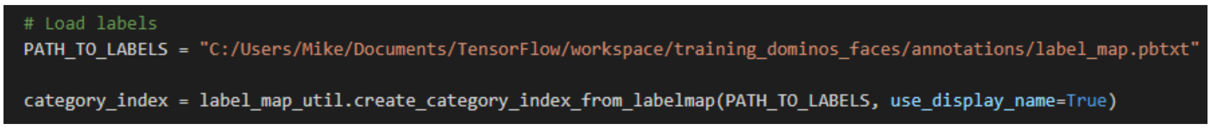


```
# Load the model
PATH_TO_SAVED_MODEL = "C:/Users/Mike/Documents/TensorFlow/workspace/training_dominos_faces/exported-models/my_model_v2/saved_model"

detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)
```

Fig 8. Code Snippet of loading the model

2. Loads the label map defined earlier

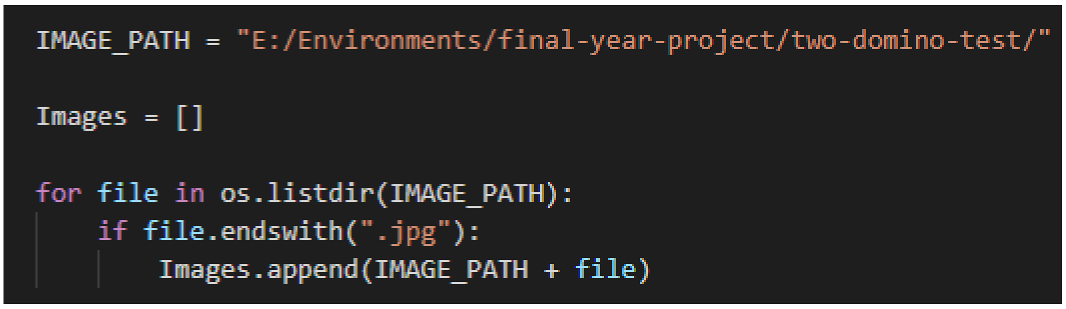


```
# Load labels
PATH_TO_LABELS = "C:/Users/Mike/Documents/TensorFlow/workspace/training_dominos_faces/annotations/label_map.pbtxt"

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

Fig 9. Code snippet of loading the label map

3. Build an array of images based on the folder they are in



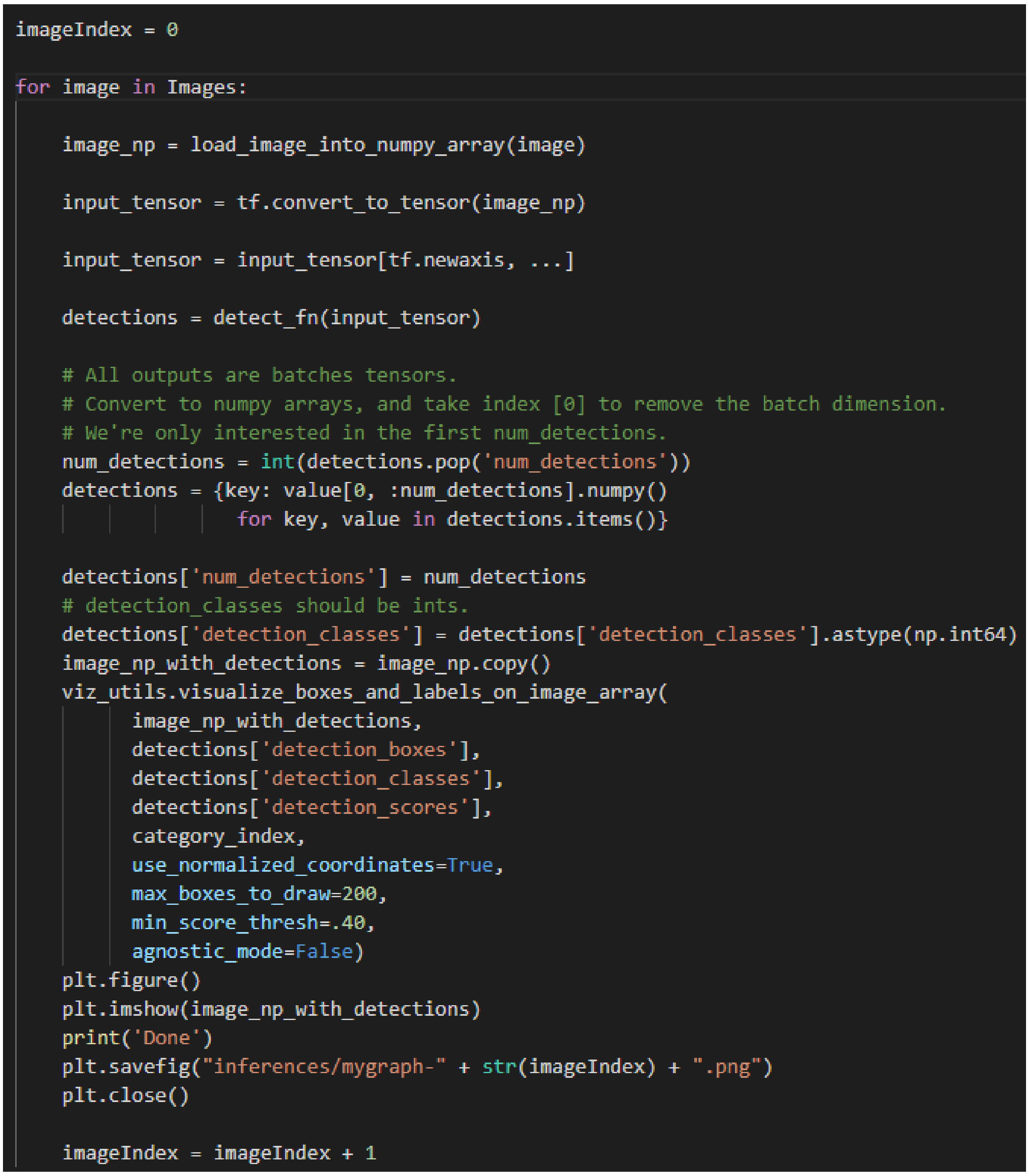
```
IMAGE_PATH = "E:/Environments/final-year-project/two-domino-test/"

Images = []

for file in os.listdir(IMAGE_PATH):
    if file.endswith(".jpg"):
        Images.append(IMAGE_PATH + file)
```

Fig 10. Code snippet of building the image array

4. Iterate over the image array and evaluate each image against the model and use the object detection API to create a new image with labelled bounding boxes drawn on



```python
imageIndex = 0

for image in Images:

    image_np = load_image_into_numpy_array(image)

    input_tensor = tf.convert_to_tensor(image_np)

    input_tensor = input_tensor[tf.newaxis, ...]

    detections = detect_fn(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    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)
    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'],
          detections['detection_scores'],
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=200,
          min_score_thresh=.40,
          agnostic_mode=False)
    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
    plt.savefig("inferences/mygraph-" + str(imageIndex) + ".png")
    plt.close()

    imageIndex = imageIndex + 1
```

Fig 11. Code snippet of the evaluation of the images

However, after feeding some images into this script it soon became clear that this first approach to classifying dominos was not going to work. The output came out with multiple classes being estimated for each image with not a single one having a probability of higher than 10%. There were two possible reasons for this outcome, it was either the initial dataset was just not big enough or the fact that all dominos are very similar in structure; they are all white rectangles with a black line down the middle, this could cause difficulties for the model to tell them apart. In fact, it was probably a combination of the two, however with no desire to go and take another 500 pictures of dominos and label them all up it was decided to try another approach.

For the next approach I needed to think of a way to discern between dominos in a way that could tell the dominos apart. Since the objective of the project was to count dots I though why not label up each individual dot as a class called 'dot' and train the model to only detect a single class.

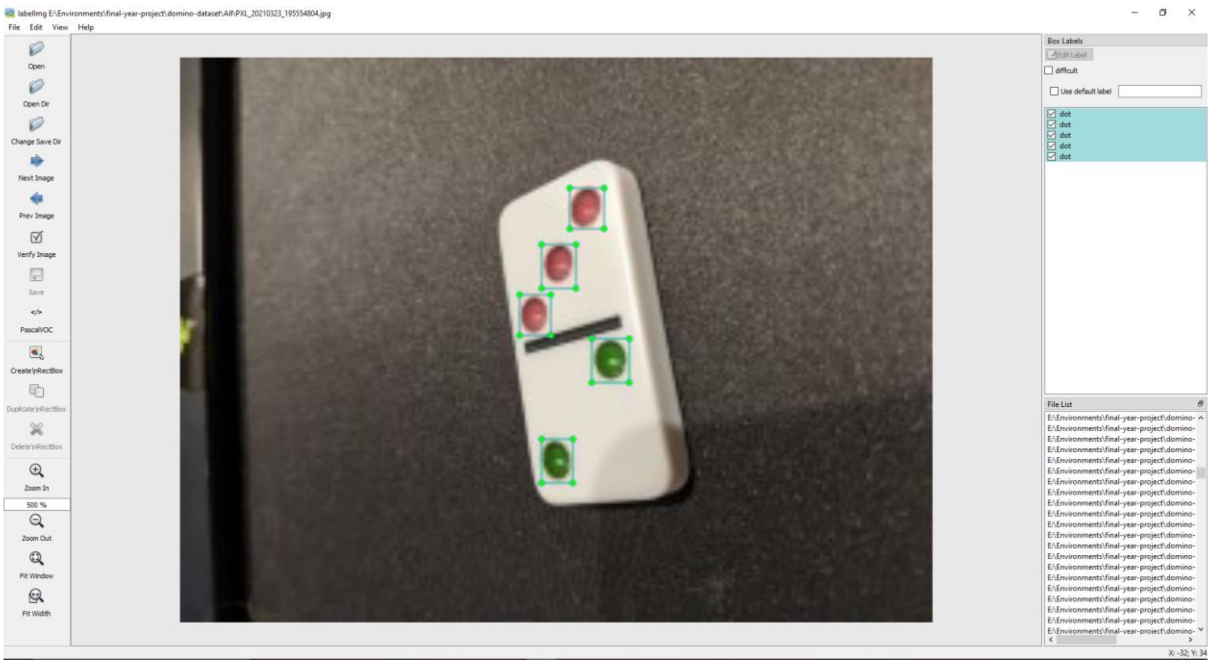


Fig 12: Labelling the dots in LabelImg

The training was then set up in the same way as before and left to run overnight. However, this approach did not work either. This time there were lots of false negatives, it is possible that due to the small size of the dots the model quickly became over-tuned and therefore would find dots just in the image noise. Tweaking the pipeline.config in order to find the balance between over-tuned and under-tuned would have taken too long so I decided to try a different approach.

For the third approach I needed to think of a feature of the domino which would be discernible from other parts of the domino but not too small that it would be lost in the image noise. I decided to try training the model on the individual faces, the reasoning for this is that they are easier to tell apart than entire dominos and with only six classes the data set for each class would be much larger than when there were twenty-eight classes in the first instance. Also, with each face taking up a much larger area than individual dots it was less likely the model would easily become over-tuned.

Two more changes were made for the third approach, the first was to split the images and their xml files out into training data, as before, and testing data in a 9:1 split of training:testing. The testing images were also serialised into a TFRecord file which could be used while training to get a more accurate loss value every 100 steps. Which as it turns out was around just below 1 which is a good value. Loss is calculated using a complex formula but at its bare bones is a combination of the localisation and classification losses. The localisation loss being how different the bounding box is from what the ground truth is and classification losses are how close to the actual classification does the model think it is, again this uses some very complex mathematics to work out.

The next change was to look at the learning rate of the model in the pipeline.config, a high learning rate can lead to higher loss values, so I decided to reduce the learning rate slightly. And while this means that the model would take a bit longer to train, it also means that it is more likely to have more accurate outputs.

The model was then set to train again overnight and exported as before. Upon running the testing script this time against a few examples it was much more successful.

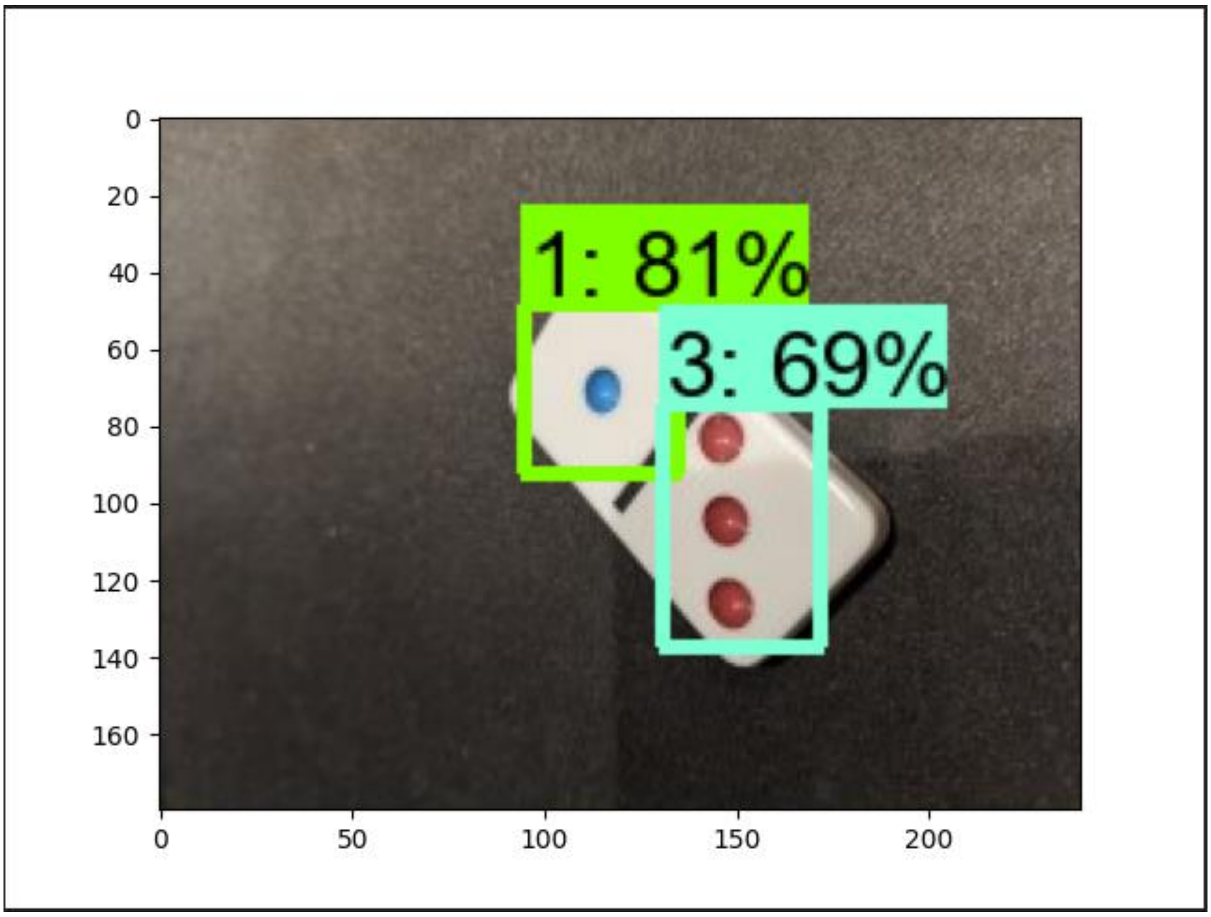


Fig 13. A successful result when evaluating the model

Now that it had been proven that this attempt was evaluating a few images correctly it was time to get some proper metrics. Fifty-three pictures of individual dominos were taken and put in a folder to be run through the evaluation script. The results were then examined to get a count of the three things needed to work out precision and recall; true positives, false positives, and false negatives. The formulae presented in section 3.5.4 of this report were then used to calculate recall and precision for the model when evaluated against a single domino. The value for recall was 98.9% and the value for precision was 96.8% which is very good considering the size of the data set. It was decided it would be a good idea to evaluate the model with two dominos in the picture. Twenty-eight pictures containing two dominos were taken and evaluated given a recall value of 88.9% and a precision value of 77.4%. This is less than for a single domino and is probably due to the fact that the faces take up less of the picture than when it was a single domino and is such it is harder for the model to detect. A solution to this would be to carry on training the model with a bigger dataset containing images of dominos at different distances. However, at this point time had run out for working on the model and I felt these values were good enough to move forward with the mobile app. The saved model was converted to a TensorFlow.js model using the converter provided by TensorFlow to be used in the mobile app.

The first step for the mobile application was to build a simple UI with a button to open the camera, after a photo had been taken the UI would output the result on the user's screen. It was also decided it would be a good idea to add a button to upload an image in the event of the device not having access to a camera for whatever reason. In order to style the UI bootstrap, which is a CSS framework, was used as it is something I have experience with, and it gives a clean, modern look to the UI.

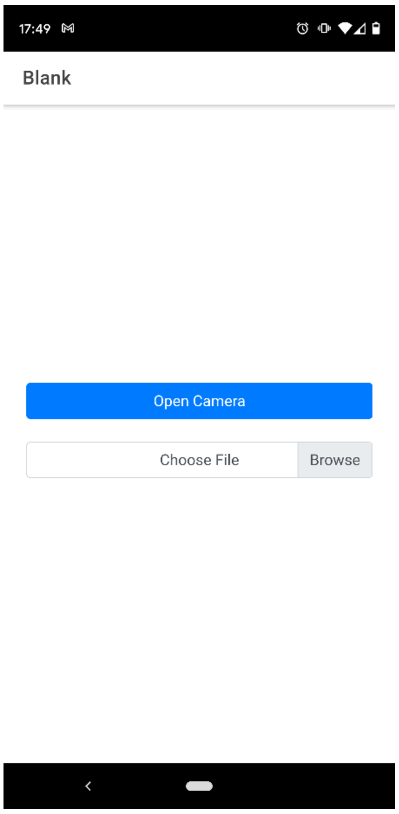
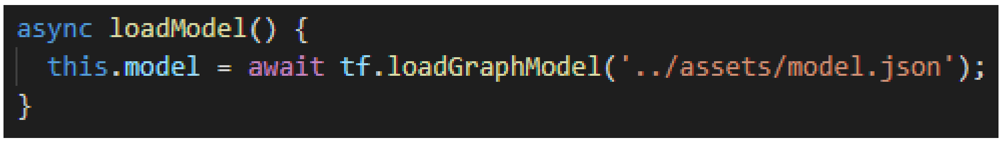


Fig 14. The final UI as displayed on an Android device

The next step was to link up the buttons, firstly the button to open the camera was setup. Ionic comes with a plugin to allow the opening of the native camera app and then return the image in a number of formats so this was used to implement the camera. File upload was handled using a standard approach to uploading files to the web with use of the file type input.

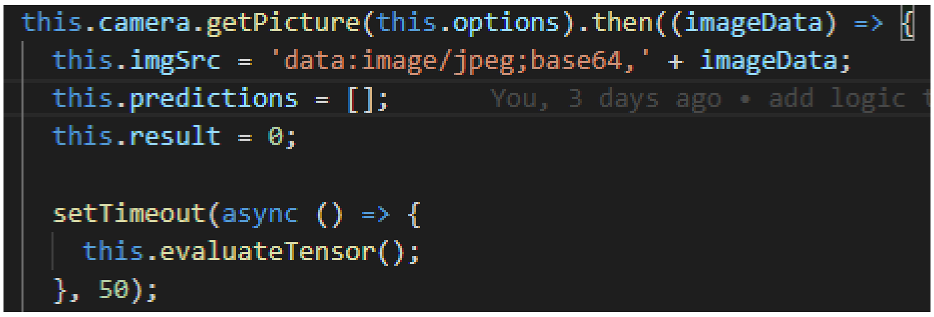
Then it was time to implement TensorFlow.js, the library was imported and used to load the model on app start.



```
async loadModel() {
  this.model = await tf.loadGraphModel('../assets/model.json');
}
```

Fig 15: Code snippet of loading the model

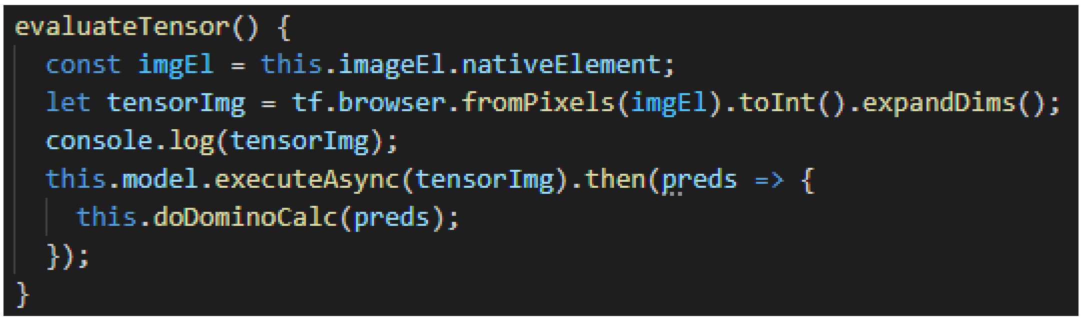
In order to use the model to evaluate an image, the image first needed to be converted to a Tensor, which is basically all the images pixels serialised in a way that TensorFlow can interpret. The easiest way to do this was to convert the image to a base64 string and then display it in an image tag on the screen. The TensorFlow method 'browser.fromPixels', which takes the image tag element as a parameter, can then used to construct the Tensor from the image that is displayed on screen.



```
this.camera.getPicture(this.options).then((imageData) => {
  this.imgSrc = 'data:image/jpeg;base64,' + imageData;
  this.predictions = [];        You, 3 days ago • add logic t
  this.result = 0;

  setTimeout(async () => {
    this.evaluateTensor();
  }, 50);
```

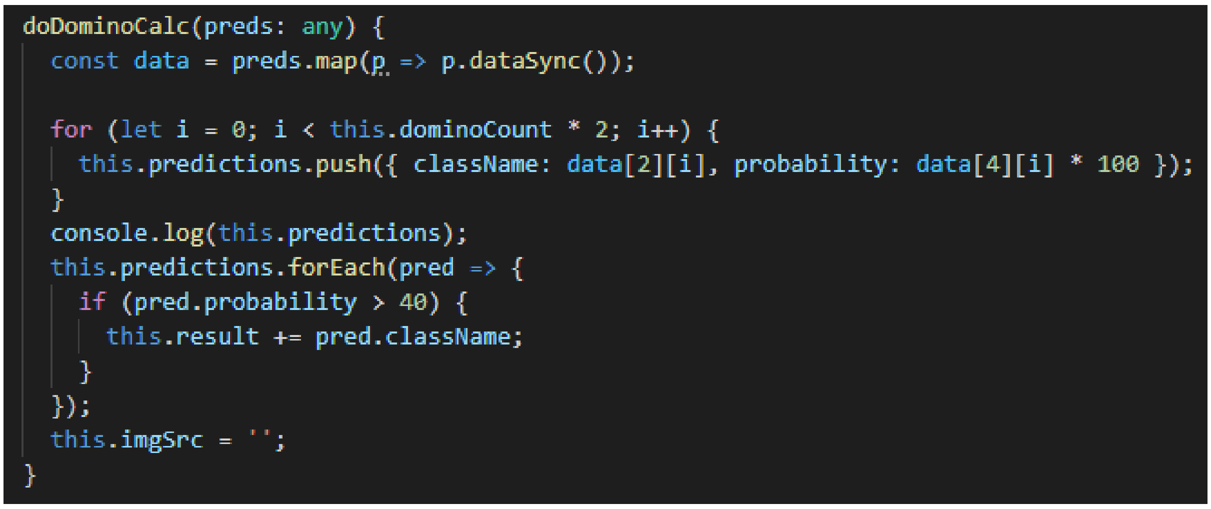
Fig 16: Code snippet of setting the image source and then calling evaluateTensor



```
evaluateTensor() {
  const imgEl = this.imageEl.nativeElement;
  let tensorImg = tf.browser.fromPixels(imgEl).toInt().expandDims();
  console.log(tensorImg);
  this.model.executeAsync(tensorImg).then(preds => {
    this.doDominoCalc(preds);
  });
}
```

Fig 17: The evaluateTensor method

The above method is where the Tensor is built and then passed into executeAsync on the model. This method evaluates the image against the model and returns a set of predictions that then need to be translated into something that is readable.



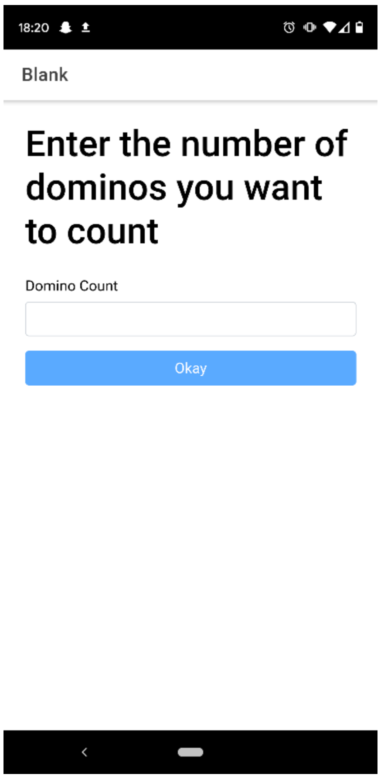
```
doDominoCalc(preds: any) {
  const data = preds.map(p => p.dataSync());

  for (let i = 0; i < this.dominoCount * 2; i++) {
    this.predictions.push({ className: data[2][i], probability: data[4][i] * 100 });
  }
  console.log(this.predictions);
  this.predictions.forEach(pred => {
    if (pred.probability > 40) {
      this.result += pred.className;
    }
  });
  this.imgSrc = '';
}
```

Fig 18: The doDominoCalc method

The first line of the doDominoCalc is where the prediction result is translated into something readable. The predictions are mapped using their dataSync method to create an array of results. The array has 8 items each of which is another array. The important parts are the items at index 2 and 4, the data in index two contains the class names for each prediction whereas the data in index 4 contains the probabilities of those predictions. Other data includes the bounding box coordinates amongst other things however these were not important at this time.

Now, an issue arose here due to the fact that the model was set up to allow for a maximum of 100 predictions, which it will always do even if the vast majority of these predictions have probabilities of less than 1%. As such a method was required to filter out the low probabilities and also to only add up a maximum number of classes based on the number of dominos. The preferred way to do this would be to train a separate model that could detect how many dominos were in the picture and get the value from there. However, there was no time to train a new model and as such it was decided to update the app's landing page to prompt the user to input the number of dominos they want to evaluate before allowing them to take or upload a picture.



Enter the number of dominos you want to count

Fig 19: The new landing screen to enter a domino count

As can be seen in the code snippet this domino count was then used to only push the two times the number of dominos results to the predictions array. This array was then iterated through to get the final count, notice there is still a probability threshold of 40%, this is in the event that one or more of the dominos have a zero face.

It was now time to test the deliverable in real world situations. The initial testing was done on the same surface as the training photos were taken with between 1 and 3 dominos. It appears that the app is decently successful at detecting and calculating the correct scores for the dominos in the photo.

However, upon testing the app on dominos that were on a different kind of surface it soon became apparent that the model was not quite as well trained as I first thought. While there did not appear to be too many false positives from looking at the results given it was clear that it was struggling to detect all the faces resulting in the calculation being short by one or two faces. Again, I believe this is something that could be rectified by having a much larger dataset than was used to train the model. With dominos on a variety of surfaces and at a number of different distances.

4.3 Completeness of the Deliverable

Overall, the deliverable is fairly complete; it is an Android app that is relatively successful at counting dominos on certain surfaces. There are certain things missing from the deliverable that I would have liked to implement such as a separate model for counting the number of dominos in the image to take away the need for the user to input this value. It would also be good to try and refine the model using a much larger dataset in order to make it more accurate. Another thing that was proposed in the design section which I did not have time to implement was allowing the user to say whether the model had got the answer right or not. This would have required a bit more research on how to update TensorFlow.js models on the fly and also some adjustments to show the user the predicted bounding boxes and allow then to draw missing ones on the screen.

5. Critical Reflection

5.1 Specification and Planning

One of the major areas that I could have improved upon was the original specification, when I produced the specification it was very bare bones and basically just said this is what I am going to do with no real justification as to why I was doing it. This led to a slow start to the project as I did not really have a starting point to being planning what research I needed to do on the subject matter which led to me losing time. I could have improved this by having much more fleshed out specification with justifications like the introduction section in this report.

Another area that was lacking was the project management side of things, as discussed earlier in this report, all the planning with regards to the tasks that needed to be done, not only during research and design, but also during actual development were only every in my head. As such this made time management difficult because it became hard to remember how much time I had spent on certain tasks and how much time I needed to give myself to complete others. It would have been much better to use some sort of project management software such as Trello or JIRA to better visualise and plan out the task I needed to do. It was also very easy to lose track of what I was working on at the time in the event of having a week or two off to focus on other things, another thing which would have been solved through use of something like Trello as it allows the user to keep track of what is done, what is being done, and what needs to be done.

One of the areas that went well I think was the research parts of the project, I kept good notes on the areas I had been researching along with links and references so that they could be easily integrated into the report at a future time. The same can be said when it came to selecting the appropriate tools for producing the deliverable. While there are definitely areas for improvement, for example, I could have done much more involved research into the inner workings of machine learning algorithms, I think that the level of research I did do was enough to gain a good surface level understanding of the concepts required to create the deliverable.

I feel that the actual development of the deliverable went fairly well, there were some failures along the way as discussed in section 4, however I quickly overcame these and took the development down different avenues as needed. As far as the completeness of the deliverable goes, as discussed in section 4.3, I think one of the major reasons for running out of time and not being able to finish the last couple of things was down to poor project management.

5.2 Professionalism, Ethics, and Personal Development

In terms of ethical concerns, there is not really anything to consider for this project. The only concern would be if you were using image recognition to detect faces or something along those lines. However, since this project is just focussing on dominos there is no issues.

From a professional development standpoint, I feel this project has really highlighted the importance of project and time management for me. I thought that I could just crack on with the project with no thought as to what I needed to do, this led to many instances where I found it really hard to motivate myself to get any work done and I think this is mainly down to the fact that I had no real plan of what I should be working on at the time. This led to losing time on the project and ultimately was a detriment to the full potential of the final deliverable. Going forward, I will strive to implement formal time management practices in any projects I undertake and really take on board their importance in future workplaces.

In terms of personal development, I have gained much deeper understanding of machine learning than I had before commencing this project. Although it is still a surface level understanding I think it is enough for me to be confident taking my understanding further should the opportunity arise in future endeavours. I have also learned how to develop an app using web technologies and take that app from development to deployment. And while it was only a small app, the knowledge I have gained will be invaluable in a world where there is a growing focus on the mobile stage.

6. Conclusion

In conclusion, I believe I have fulfilled the aims of the project for the most part. I have trained a machine learning object detection model to classify the faces of dominos. I have created a mobile app that makes use of this model to allow users to use their devices camera to count dominos, albeit with varying degrees of accuracy. Although there are some parts of the app that I had hoped to develop further before running out of time, I believe I have gained a much deeper understanding of machine learning for object detection and how it can be applied in the mobile space. I have also matured professionally with a much greater respect for proper time management.

7. References

Błażej Osiński (2018). *What is reinforcement learning? The complete guide - deepsense.ai*. [online] deepsense.ai. Available at: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

Bzdok, D., Krzywinski, M. and Altman, N. (2018). Machine learning: supervised methods. *Nature Methods*, 15(1), pp.5–6.

Curtin, R., Edel, M., Lozhnikov, M., Mentekidis, Y., Ghaisas, S. and Zhang, S. (2018). *mlpack 3: a fast, flexible machine learning library Software • Review • Repository • Archive*. [online] . Available at: https://www.mlpack.org/static/pub/2018mlpack.pdf [Accessed 14 Apr. 2021].

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)*. [online] . Available at: https://arxiv.org/pdf/1311.2524.pdf.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K. (2017). *Speed/accuracy trade-offs for modern convolutional object detectors*. [online] . Available at: https://arxiv.org/pdf/1611.10012.pdf [Accessed 28 Aug. 2019].

Jason Brownlee (2019). *A Gentle Introduction to Object Recognition With Deep Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning/.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. (2016). *SSD: Single Shot MultiBox Detector*. [online] . Available at: https://arxiv.org/pdf/1512.02325.pdf.

Newzoo. (2019). *Newzoo Global Mobile Market Report 2019 | Light Version | Newzoo*. [online] Available at: https://newzoo.com/insights/trend-reports/newzoo-global-mobile-market-report-2019-light-version/.