PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

Detecting specific foods in MRI scans using TensorFlow™ and RetinaNet implemented in Python™

Joshua Gardner

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor of Science
in Computer Science

Supervisor Dr Franck Vidal

April 2020

Acknowledgements

Special thanks to Dr Évelyne Lutton from France’s National Research Institute for Agriculture, Food and Environment (INRAE) and Prof François Boué from French National Centre for Scientific Research (CNRS) for supplying the image data. And an even more appreciated thank you to my supervisor and tutor, Dr Franck Vidal, for being extremely supportive and helpful in aiding me in my dissertation. Supplying me with his time and knowledge to help me succeed.

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Joshua Gardner

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Joshua Gardner

Abstract

Object detection has played an important role and challenge in computer vision being utilised in many diverse technologies and applications such as facial recognition, security, automated driving and medical imaging. Medical imaging has allowed us to understand how our bodies fundamentally function through non-invasive techniques, such as magnetic resonance imaging (MRI), computed tomography (CT) scans and X-Rays. It has also allowed us to study how entities in our bodies are consumed and react in us. Our lives have been improved with this understanding, diagnoses and treatments can be performed through these techniques. Analysing scans can be time consuming and often when looking for specific problems, there is potential missing other ailments.

In medical imaging, object detection has a multitude of applications; it can help quickly and accurately diagnose patients, automatically detect a multitude of conditions and illnesses such as cancers and abnormalities within organs. It can help us track objects like drugs or food in the digestive system or other pathways. With all this data being collected daily, machine and deep learning techniques could be used to quickly develop neural networks to detect them. One use of object detection is in the study of digestion of foods in the stomach using MRI scans. Through this we could learn how certain foods pass through the digestive system and how it reacts.

We will be applying object detection through the use of Neural Networks to detect food, specifically frozen peas. This to allow automation of a larger research study into the monitoring of stomach content with the aim of understanding digestion to help develop future foods to potentially aid in food crises. To do this we will be using a **Keras** implementation of **RetinaNet** developed by Fizyr as described in Focal Loss for Dense Object Detection developed

at Facebook AI Research (FAIR). Food detection in MRI scans through Neural networks has proven to be possible, as the network we trained produced a precision of **true positives of 0.81 and a total recall of 0.454**. These scores could be improved by an increase of training data sets and is possibly low due to a number of hard examples obscured in the MRI Scans which could be improved upon through image processing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Object detection has played an important role and challenge in computer vision, with massive developments in the last decade. In the field of object detection, machine learning algorithms have been developed such as Viola–Jones object detection framework and histogram of oriented gradients (HOG). Being used as implementations of facial recognition and human detection. Currently, deep learning techniques are being used over machine learning technique. Examples of deep learning include Convolutional neural network (CNN) with improvements on the core ideas like Regions with Convolutional Neural Network (R-CNN) and many other iterations. These developments are credited to the increase in performance in hardware such as dedicated graphics card, cloud computing and cost being reduced as the technology became more widespread. Object detection has many diverse applications from object tracking, character recognition, automated driving, robotics, manufacturing and medical imaging.

1.1 Medical Imaging

Medical imaging is incredibly important in medical analysis, allowing doctors to complete diagnoses of a patient's internals that are usually hidden by skin tissue and bones. Imaging techniques such as magnetic resonance imaging (MRI), X-RAY, computed tomography (CT) and Ultrasound scans are used to generate visual representations of internal structures such as organs and bones. These techniques are non-intrusive and the generated images can reveal abnormalities and illnesses such as cancers. These results can then be used to apply proper treatment and therapy for the correct cause.

Medical imaging can also be used for research purposes such as studying the human body and its functions, analysing digestion and blood flow.

Figure 1.1: Example of MRI image from the data set

1.2 Applications of object detection in medical imaging

Object Detection could automatically detect a multitude of conditions and illnesses such as cancers and abnormalities within organs. With all this data being collected daily, deep learning techniques could be used to quickly develop neural networks to detect them. A CNN may potentially, if trained well enough, be more accurate of detecting malignant tumours or any other condition than humans. If a network is trained for multitude of different diseases it could then also detect any other aliments that are present. Object detection could make treatment quicker, potentially cheaper by reducing man hours and free up researchers and hospital staffs time.

1.3 Project Goals

The larger goal of this project is to be able to analyse the digestion of foods inside the stomach to see how certain foods react and how long they take to be digested. However, in this thesis **we will be specifically looking at how to detect food in images of MRI scans of the stomach**.

To achieve this goal, we will aim to fulfil these objectives:

Using **computer vision techniques** such as **deep learning** and **neural networks** we can **train object detection models** to be able to analyse MRI scans of stomachs and hopefully detect specific foods inside it. In this project frozen peas have been used as part of a larger research project to see how small spherical foods are digested for potential research into the future of foods. We will be **implementing** this through a **combination of Google's TensorFlow** library with a **Keras** Implementation of **RetinaNet** coded in **Python**. These software libraries provide many computer vision tools specifically aimed at deep learning and developing deep neural networks capable of object detection, object classification etc. Then we will test our Neural network (NN) by detecting peas in our testing data set. We can validate these results against the actual locations in the images. We can then analyse the accuracy of our network in the form of:

Precision: where tp is the number of true positives and fp is the number of false positives

$$p = \frac{tp}{tp + fp} \tag{1.1}$$

Recall: where fn is the number of false negatives

$$p = \frac{tp}{tp + fn} \tag{1.2}$$

1.4 Contents

In Chapter 2 we will be discussing object detection and the various solutions available such as machine learning and deep learning. In Chapter 4 we will be discussing how I used these techniques to solve our problem of detecting peas in MRI scans.

Chapter 2

Object Detection

2.1 Challenges with object detection

Object Detection is a computational challenge, producing an entire research field in computer vision alone. Object detection traditionally works by the use of a sliding window [1], where features are extracted using multiple aspect ratios to analyse the region and detect objects. Figure 2.1 (reproduced from [2]) shows an example of how a sliding window is used to detect objects.

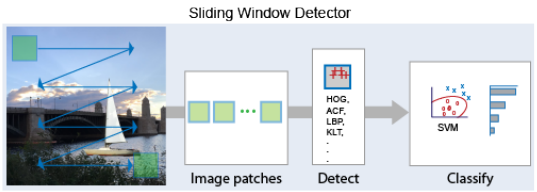


Figure 2.1: Example of a sliding window

Traditional techniques included using machine learning algorithms to detect features in images through image processing techniques. Detection methods like Histogram of oriented gradients (HOG) rely on processing techniques to identify people from their shape in the image, whereas the Viola-Jones algorithm focuses detecting specific facial features. This is not always possible for certain objects and would require us to use many different techniques and algorithms to detect several different objects. CNN solves this problem. By training a NN on a data set of training images, it can learn to detect, classify and recognise many different object types.The major drawback is that sometimes for the best and most accurate detectors there is a requirement for large amounts of training data. Which would have to be hand labelled and can be possibly hard to get. For example training CNN on

medical images like we are doing we would need access to this data, which is often private or only available as part of a larger research project. Figure 2.2 reproduced from [3] showing objects being detected with the bounding boxes surrounding them.

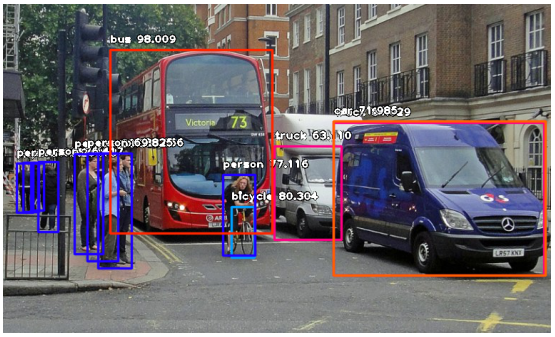


Figure 2.2: Example of an CNN object detection

2.2 Machine Learning approach

In machine learning the use of computational statistics and computer vision techniques to algorithmically detect features and objects through statistical comparisons on the pixel level. Typically through image intensity or other statistical data.

The Viola Jones object detection framework detects objects by using simple features. Originally used for facial recognition, the algorithm would find features of the face like eyebrows, lips, nose, and eyes by using Haar functions [4]. Example in figure 2.3 (reproduced after Viola et al. [4]). Haar functions are very similar to kernels used in computer vision techniques like line detection, filtering, and CNN. These functions act like a sliding window, going over regions of the image and doing some form of comparison or calculation.

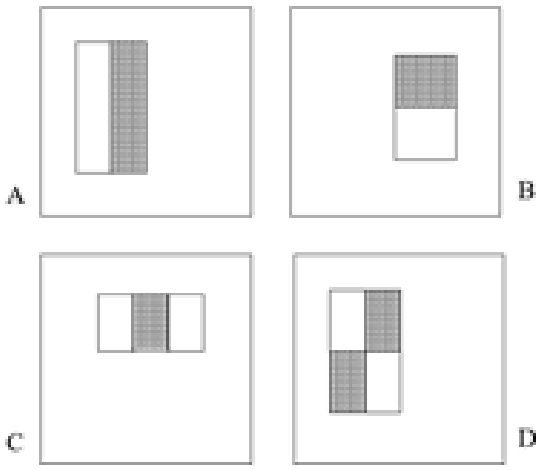


Figure 2.3: Haar function in Viola-Jones algorithm

Haar features work by computing the difference of image intensity between the sum of two sets of regions. Where ii(x,y) is the image intensity for each region, i(x',y') being the pixel values for each position in the region

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{2.1}$$

In facial recognition, this is done to detect features such as eyebrows by comparing the estimated region the eyebrows are to the skin around it. Typically done in grey scale, the eyebrow intensity would be lower, or darker in colour terms, than the surrounding skin which would be whiter or a high intensity. This same method can be applied to detecting the eyes from the cheeks, the eyes, and the bridge of the nose. All these features must be detected for the algorithm to determine that the image contains a face. See figure 2.4 (reproduced after Bukis et al [5]) for example of how the algorithm working.

HOG differs from the Viola-Jones algorithm and many deep learning methods of object detection. In that it does not try to detect or determine objects based specific features or learned weights. HOG detects a global feature; it detects the whole shape of a person or car for example. The concept being that an object's shape within an image can be determined by the distribution of intensity gradients. HOG works similarly to edge detection and will detect humans silhouette from its contours. The image is divided into small uniform

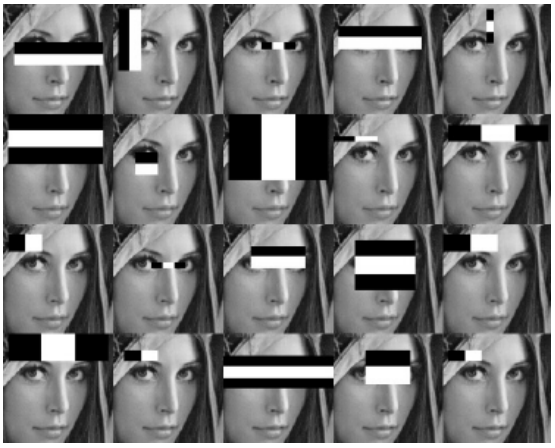


Figure 2.4: Feature detection in Viola-Jones algorithm

regions. A histogram of gradient directions is compiled for each region using the sliding window method going over these regions. A descriptor is formed from a combination of histograms [6]. HOG is best suited for human detection where people are upright and mostly visible. Figure 2.5(b,f) (reproduced by Dalal et al [6]) shows how HOG detects humans from the outline shape in the image.

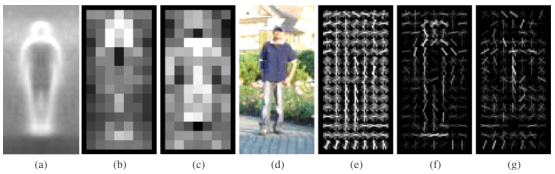


Figure 2.5: Representation of how HOG works

2.3 Deep Learning

Deep learning techniques of object detection require training a CNN to detect specific classifies on images. This requires collecting a set of data and typically hand classifying the images. For the best and most accurate results a large data set of a good ratio of easy and hard examples are required, its even good to include lots of negative examples. There are many open source data sets that exist for training networks such as Common Objects in Context (COCO) which contains a vast amount of pre-classified images [7]. Figure 2.6 (reproduced after Lin et al [7]) shows an example of their data set. COCO and other data sets such as the PASCAL Visual Object Classes Challenge (VOC), are often used as testing sets to compare object detection frame works mean

Average Precision (mAP). COCO and many other data sets are often broad and would not be useful for more specific tasks like our own.

Figure 2.6: Example of the COCO data set

With the popularisation of CNN, many research projects are being produced in this field and they often grant public access to their object detection frameworks. This gives us a wide range of trainable networks optimised for object detection, that are constantly being improved upon and bested by other network models. One of such models is R-CNN.

One major problem with Convolutional Neural Networks is that it does not know how many potential objects are in an image. Rather than trying to potentially detect for an object in every single bit of the image an extra step is implemented to reduce the area needed to be fed into the Neural network. In the work of Ross Girshick et al [8] they explained their method of Regions with Convolutional Neural Network. R-CNN is a two-stage object detector, in the first stage it proposes using a selective search to extract 2000 regions proposals and then using a greedy algorithm to recursively combine similar regions into one larger region. In the second stage a CNN is then used to compute features for each region proposal and then is classified by an Support vector machine (SVM). This is demonstrated in figure 2.7 reproduced after Ross Girshick et al [8].

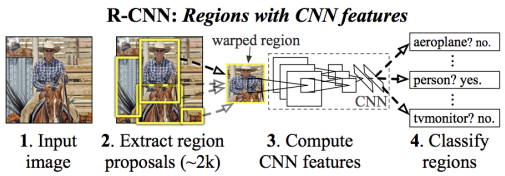


Figure 2.7: Example of how R-CNN works

However, the results from Girshick et al show that whilst producing then highly accurate results of a mAP of 53.7% on the VOC 2010 set, computing 2000 region proposals through selective search is long and slow. Each of

these proposals then must go through a ConvNet forward pass, which for 2000 images will take a considerable amount of time. The results show that RCNN takes 13 seconds per image on GPUs and 53 seconds on a CPU. This is relatively quite slow, especially when you have large amounts of testing images. The selective search algorithm is fixed, so cannot learn. Meaning the region extraction process cannot be improved upon. RCNN cannot be used for real time object detection, which while is not necessary for my project speed and efficiency is still preferred. Improvements and iterations of RCNN exist. Fast and Faster R-CNN attempt to solve the time issue with RCNN.

Ross Girshick et al [9] improved upon R-CNN with Fast R-CNN. Instead of extracting region proposals first, Fast R-CNN uses the input image and feeds it to the CNN to produce a convolutional feature map. Then in the Region of Interest (ROI) feature vector, the feature map is fed into a ROI pooling layer which identifies region proposals and reshapes them into fixed size squares. The region proposals are then fed into a fully connected layer. A softmax layer is used to predict the class of the proposed region and the bounding box. This is demonstrated in figure 2.8 reproduced after Ross Girshick et al [9].

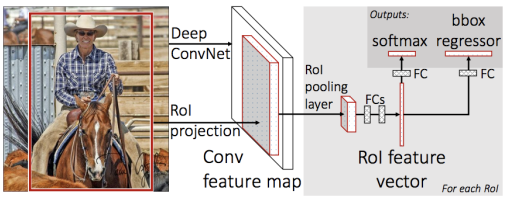


Figure 2.8: Example of how Fast R-CNN works

Fast R-CNN improves upon R-CNN as it no longer needs to feed 2000 regions into the CNN, convolution is only done once. Its also produced an increased mAP of 66.1% compared to R-CNN's 53.7% How ever region proposals are still a bottleneck for fast R-CNN as it still uses selective search. This is show as Fast R-CNN with region proposal takes 2.3 seconds per image and without it the forward pass takes 0.32 seconds. Results shown in 2.9 reproduced from Girshick et al [9]

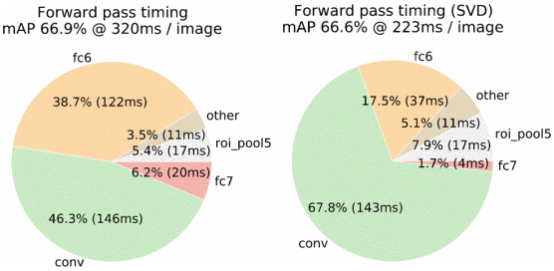


Figure 2.9: Example of how R-CNN works

Shaoqing et al [10] developed a faster real time implementation of R-CNN called Faster R-CNN. Faster R-CNN uses an object detection algorithm to replace the slow and time-consuming selective search. Like Fast R-CNN it the input image is fed into a unified network to produce a feature map but then used a separate network, called Region Proposal Network (RPN) to predict region proposals. Using a network for region proposals means that the network can learn and improve upon proposals. They are then added into a ROI pooling layer to be reshaped before they are classified, and then the bounding boxes are predicted. Figure 2.10 is reproduced from Shaoqing et al [10].

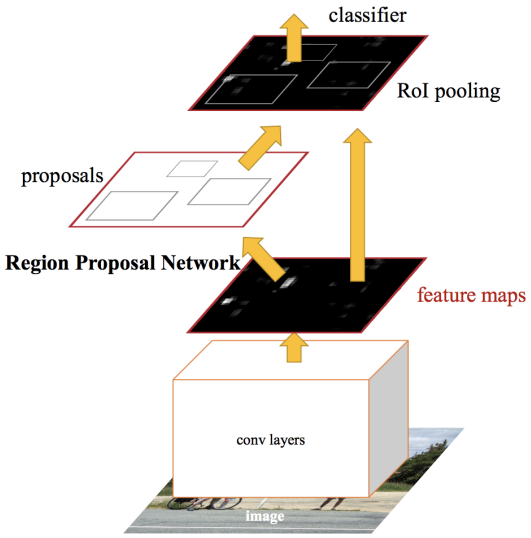


Figure 2.10: Example of how Faster R-CNN works

Faster R-CNN produces a mAP of 75.9% with a 0.198 seconds per image. This is due to fewer region proposals (300) enabling Faster R-CNN to have an almost real time detection of 17 frames per second (fps)

Another use of deep learning is using one stage object detectors, an example of this is the You Only Look Once (YOLO) model developed by Redmon

et al [11]. YOLO uses a single Fully CNN to predict bounding boxes and the class probabilities.

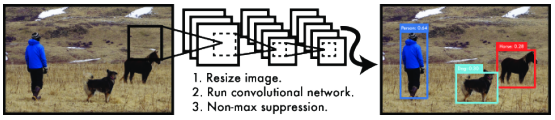


Figure 2.11: Example of how YOLO works

Generating region proposals and then classify these regions like in RCNN, is slow and hard to optimise as each individual step is a separate part. YOLO instead views object detection as a single regression problem, only looking at the image once to predict what objects are present and their locations. This unified model is extremely fast, running at 45 frames per second with no prior batch processing, and the faster version sacrifices mAP but can run up to 150 frames. It is also a lot more accurate with a mAP of 63.4%. Using a single CNN, YOLO does this by predicting areas that have high probabilities of containing objects and then predicts bounding boxes and class probabilities per box. Figures 2.11 and 2.12 show how YOLO object detection works reproduced from Redmon et al [11]

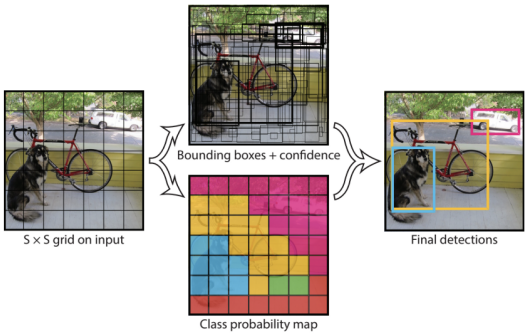


Figure 2.12: Example of how YOLO prediction predicts objects

Chapter 3

RetinaNet

Typically, one stage detectors like YOLO produce lower mAP scores compared to two stage detectors, such as R-CNN and its iterations. Research done by Tsung-Yi et al [12, 13] at Facebook AI Research (FAIR) has accumulated to produce a brand new state of the art one stage object detector capable of matching mAP scores of two-stage detectors. RetinaNet achieves this by using focal loss instead of cross entropy for learning, combined with a ResNet CNN for deep feature extraction from a Feature Pyramid Network (FPN). This has allowed RetinaNet to achieve mAP scores of 32.5-37.8%. Figure 3.1 reproduced from Tsung-Yi et al [12].

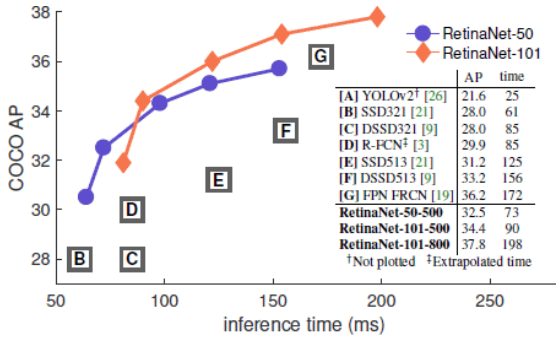


Figure 3.1: Comparison of RetinaNet mAP

3.1 Feature Pyramid Networks

One challenge in object detection is that it objects have a variety of scales. Featurised image pyramids formulate the basis of the solution to this problem. The pyramids are scale invariant meaning that objects at different scales can be shifted up and down in the pyramid model, allowing objects at a different scales to be detected. This model is still prevalent in many detection methods such as Fast and Faster R-CNN. They were heavily used in older methods like

HOG. Recent detection systems use a single feature map to quickly predict objects on a single input scale fig 3.2(b). Figure 3.2 reproduced from Tsung-Yi et al [13].

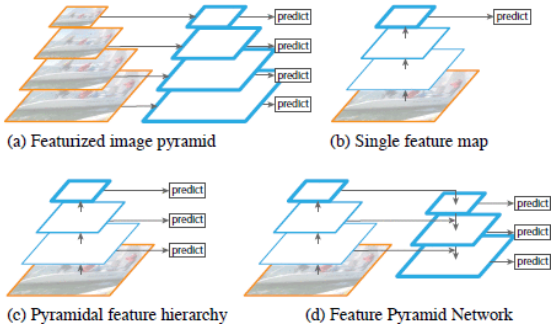


Figure 3.2: Feature Pyramids

Tsung-Yi et al [13] suggest using a feature pyramid network 3.2(d) to extract features on all scales whilst predicting objects as fast as other models (see fig 3.2(b,c)) whilst being more accurate. FPN achieves higher accuracy by using a multi-scale feature representation in which all levels are semantically strong. This is represented in figure 3.2 by the thicker blue outlines.

Using a CNN as a backbone to create an in network feature pyramid, Tsung-Yi et al have produced a feature pyramid (fig 3.2 (d)) that has rich semantics at all levels which is built quickly from a single input image scale, without sacrificing representational power, speed or memory. By combining low resolution, semantically strong features with high resolution, semantically weak features by using a top down approach and lateral connections.

3.2 Focal Loss

In two stage detectors class balance is addressed by a two stage cascade and sampling heuristics. Examples of narrowing down candidate locations and filter out background samples, R-CNN uses selective search whilst Faster R-CNN uses a RPN. Sampling heuristics are then performed to maintain a balance between the foreground and the background.

One stage detectors must process a larger set of dense candidate locations. Sampling heuristics prove to be ineffective due to there still being a class imbalance towards easily classified examples. Tsung-Yi et al [12] proposes

using a new loss function to effectively deal with class imbalance. Using a dynamically scaled cross entropy loss, where the scaling factor decreases to zero as confidence in the correct class increases see figure 3.3 (Reproduced from Tsung-Yi et al [12]). The scaling factor automatically down weights the contribution of easy examples during training to increase a focus on hard examples.

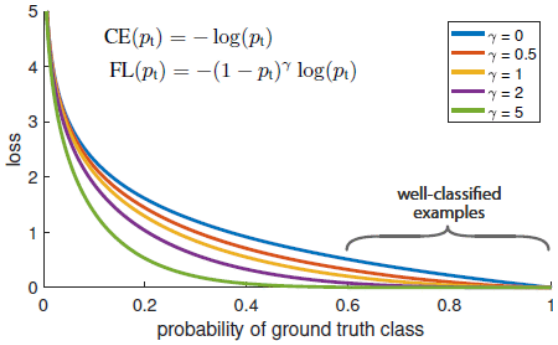


Figure 3.3: Example of focal loss

Standard cross entropy uses a weighting factor α to address class imbalance. Balanced cross entropy loss is defined as:

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t).$$

α balances the importance of positive and negative examples but does bot differentiate between easy and hard examples. Tsung-Yi et al proposes to reshape the loss function to down weight easy examples. Using a modulating factor $(1-p_t)^\gamma$ with a cross entropy, with a tune-able parameter γ to change how examples are down weighted. Balanced focal loss is then defined as:

$$\mathrm{FL}(p_t) = -\alpha_t (1-p_t)^\gamma \log(p_t).$$

3.3 Analysis on RetinaNet

The combination of using a CNN backbone with RPN to gather rich semantics and solve class balance through focal loss to focus on hard examples for training, the RetinaNet model is a fast and accurate object detection framework. With a focus on harder examples like small low resolution objects,

RetinaNet is an ideal yet simple one-stage detector. As the data set is relatively small and there are many small hard examples, I believe RetinaNet would be more than capable of effectively detecting peas and other foods in MRI scans of stomachs.

Chapter 4

Detecting Peas in MRI scans of the stomach

To implement these deep learning techniques and the RetinaNet detection framework to produce a ResNet model trained to detect the peas in MRI scans, there is essentially two major steps: Preparing the data and feeding the data to the neural network. To prepare the data we need to extract and organise the files so that it is easier to easier to access, so that we can begin training. Evaluating the data consists of detecting peas in the test images and the comparing them to the known locations to calculate the precision and recall.

Precision: where tp is the number of true positives and fp is the number of false positives

$$p = \frac{tp}{tp + fp} \tag{4.1}$$

Recall: where fn is the number of false negatives

$$p = \frac{tp}{tp + fn} \tag{4.2}$$

4.1 Preparing the data

The first step of preparing the data is to extracting it. The data collected had to be organised in data sets for the different subjects in a way more helpful for researchers. This is problematic to traverse efficiently and quickly access all the data to be trained. We must extract the data that we need

to organised directories for specific locations and rename some of files to distinguish which data is for what image, as there are multiple same named files. We use `prepareData.py` to traverse the data set and copy the files into our more organised files. All the code for this is in the appendix (chapter 7). `prepareData.py` is used through the command line issuing paramets such as the directory the data is located. This is because there is no need for an intuitive user interface to quickly traverse all the data. Figure 4.1 shows the command line interface to use `prepareData.py` and the figure 4.2 shows the results it produces.

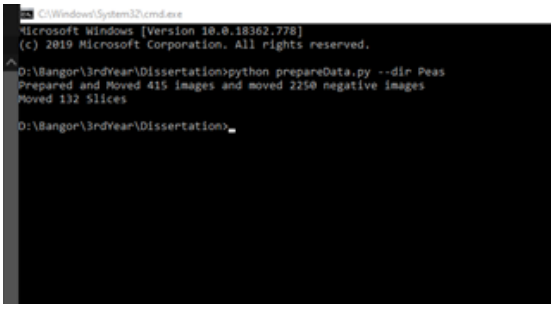


Figure 4.1: Creating Directories

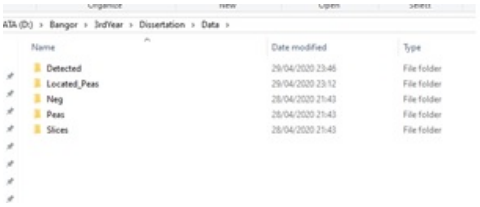


Figure 4.2: Created Directories

Preparing the data into these organised files so that we can quickly accesses the specific files we need and only these files. Each slice also contains Tab-separated values (TSV) files of the locations of each pea estimated by the researchers. These are renamed to the representative ROI image that will be used to train the neural network The images are split into training and testing images using leave p out cross validation where p is 22 out of a data set of 57 images. Instead of using Leave one out cross validation, which would ultimately produce a more accurate trained object detector, Leave p out was used due to the vast variety in the MRI scans where there is seemingly more objects in the stomach. This was done so there was a variety of testing data to see how accurate the results where compared to MRI scans with less “noise” in the ROI.

4.2 Classifying the peas

The neural network requires information of the object locations to train the model. We will need to classify each object in the image manually. To do this we use a program called `LabelImg`.Figure 4.4 shows an example of us classifying the images. This will create Extensible Markup Language (XML) files containing pixel coordinates of every bounding box of the peas in the training images. To make this easier I have programmed a script that will display the locations of the peas using the `matplotlib` library. `locatepeas.py` displays the locations on the image, using the related `tsv` file, so creating the labels would be easier. Figure 4.3 shows an example of the peas being displayed.

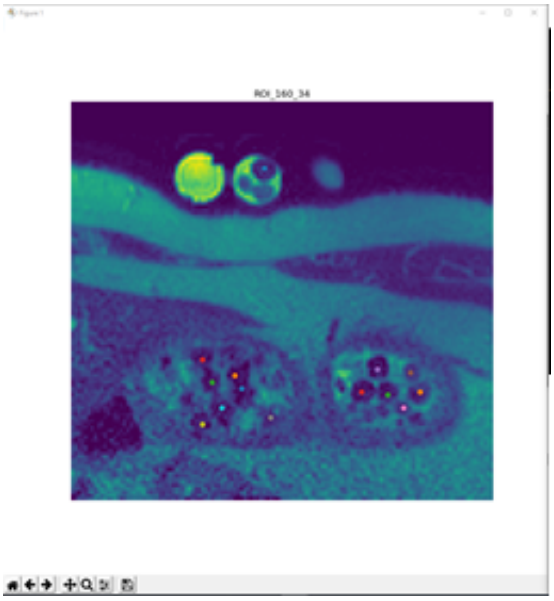


Figure 4.3: Locations of the peas in MRI Scan

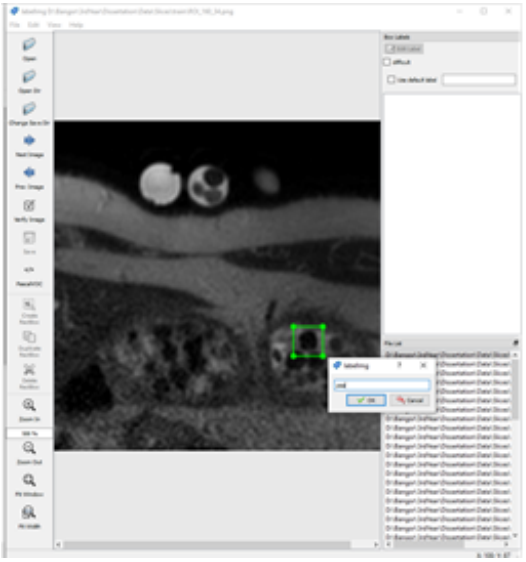


Figure 4.4: Classifying the peas in LabelImg

RetinaNet does not use XML files for training as this would require to much traversing by itself, so we need to parse the data into a more readable Comma-Separated Values (CSV) file containing the location of every pea in every file. `XML to CSV.py` will parse every XML file and add it to `train labels.csv` as well as create a `classes.csv` containing the classes and its index. As we are only detecting for one class, there is only pea at its index 0. Although this could be expanded upon if future work requires multiple types of food to be detected. `Train lables.csv` contains the image path, the 4 coordinates of the bounding box (x1, y1, x2, y2) and the class of each bounding box produced for every pea in the images.

The CSV contains the appropriate data to begin training neural network. The Keras implementation of RetinaNet provides a `train.py` file which takes the data provided and using its pre-trained ResNet models to extract the features. The model used in this project was trained in batches of 8 and 100 steps for 10 epochs. This step can take a lot of time even if using a GPU to train the neural net. The model will train the model in batches of eight, ten times. For each epoch there will be a snapshot of the trained model with the last one being the final version of the model.

Clipboard Font

POSSIBLE DATA LOSS Some features might be lost if you save this

A1 path

	A	B	C	D	E	F	G
4	D:\diss3\	48	72	56	82	pea	
5	D:\diss3\	40	54	46	62	pea	
6	D:\diss3\	30	20	37	28	pea	
7	D:\diss3\	37	20	45	27	pea	
8	D:\diss3\	50	61	56	67	pea	
9	D:\diss3\	51	26	58	34	pea	
10	D:\diss3\	53	34	59	41	pea	
11	D:\diss3\	54	64	61	72	pea	
12	D:\diss3\	38	68	46	75	pea	
13	D:\diss3\	47	53	54	59	pea	
14	D:\diss3\	61	46	67	53	pea	
15	D:\diss3\	34	47	41	55	pea	
16	D:\diss3\	43	37	49	43	pea	
17	D:\diss3\	38	40	45	46	pea	
18	D:\diss3\	30	26	38	33	pea	
19	D:\diss3\	26	17	31	23	pea	
20	D:\diss3\	34	18	39	24	pea	
21	D:\diss3\	52	26	57	33	pea	
22	D:\diss3\	53	41	58	48	pea	
23	D:\diss3\	50	51	56	58	pea	
24	D:\diss3\	51	26	58	34	pea	
25	D:\diss3\	45	36	51	42	pea	
26	D:\diss3\	27	47	33	54	pea	
27	D:\diss3\	38	38	44	45	pea	
28	D:\diss3\	93	32	101	41	pea	
29	D:\diss3\	81	36	88	44	pea	
30	D:\diss3\	89	46	96	54	pea	
31	D:\diss3\	76	43	85	51	pea	

Figure 4.5: Classification data of all peas

4.3 Testing the model

Now the model has been trained, it should be able to successfully detect peas in the MRI scans. Using RetinaNets built in functions we can pre-process test images and detect the objects in the images, returning bounding boxes and the prediction score for each pea. `detectPeas.py` will automatically generate images with the bounding boxes and scores for each pea detected and save them to the Detected Peas directory for future analysis. I have set the threshold score to be 0.6, so that only objects with a prediction score of 0.6 or above will be displayed on the image

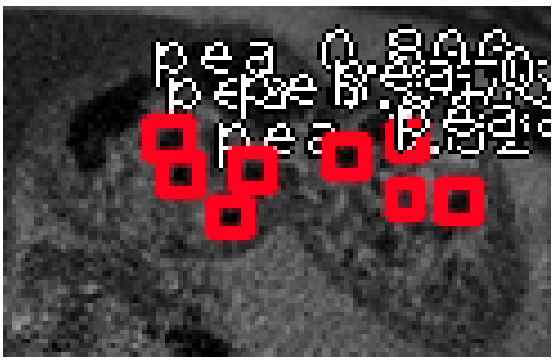


Figure 4.6: Detected Peas in a ROI

4.4 Analysing the data

We will need to analyse the results generated from the model against the actual locations in the images. This will show how accurate the model is at detecting peas in the models. Running `displayResults.py` will display both images of the detected peas and the images with all the pea locations plotted for each scan.

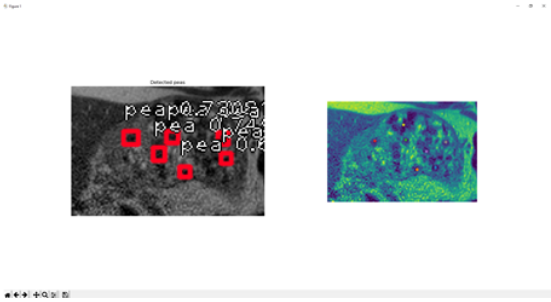


Figure 4.7: Comparison of detected peas to actual locations

Chapter 5

Results and evaluation

To work out the neural networks accuracy we must collect the positive results, the total number of actual peas and calculate the number of true positives. Then using this information, we can calculate the precision and recall.

Precision and recall are calculated mathematically through the formulas:

Precision: where `tp` is the number of true positives and `fp` is the number of false positives

$$p = \frac{tp}{tp + fp} \tag{5.1}$$

Recall: where `fn` is the number of false negatives

$$p = \frac{tp}{tp + fn} \tag{5.2}$$

Table 5.1: Results

Image	Positives	True Positives	Actual no. Peas	Precision	Recall
ROI_130_76	6	2	9	0.333	0.222
ROI_149_51	1	0	4	0	0
ROI_149_76	10	1	1	0.1	1
ROI_164_51	1	1	3	1	0.333
ROI_181_88	6	4	7	0.6	0.571
ROI_182_37	1	1	5	1	0.2
ROI_184_64	2	2	4	1	0.5
ROI_188_85	11	10	21	0.9	0.476
ROI_188_90	8	7	12	0.875	0.583
ROI_192_67	1	1	3	1	0.333
ROI_192_87	10	10	13	1	0.769
ROI_194_91	10	9	14	0.9	0.642
ROI_195_83	5	5	12	1	0.416
ROI_200_86	10	9	17	0.9	0.529
ROI_200_88	3	3	7	1	0.428
ROI_201_85	4	3	5	0.75	0.6
ROI_205_90	3	3	15	1	0.2
ROI_210_93	6	5	14	0.83	0.357
ROI_211_94	2	2	13	1	0.154
ROI_213_79	9	8	12	0.8	0.666
ROI_219_104	6	6	9	1	0.666
ROI_226_94	6	6	16	1	0.375
Total	121	98	216	0.81	0.454

In table 5.1 true positives are all the correctly detected Peas and the total positives are all the detected objects generated by the neural network. The total number of actual cases is the number of peas actually in the images.

The neural network has developed a model with a precision of positively detecting peas in MRI scans of the stomach of 0.81 and a recall of detecting all the actual peas of 0.454

The results show the model can clearly detect peas in MRI scans, however has problems with positively detecting all the peas. What causes this issue? The recall of detection is only 0.454 so only less than half the peas are being detected. This may be due to how difficult it can be for neural networks to detect hard examples. Many of these peas are around a 10 pixel resolution with the largest being 20 pixels. Another factor may be that even to the human eye many of these peas can not even be seen. Many of these peas blur into other surrounding objects in the MRI scans so the neural net would not be able to detect the peas in the image at all.

In some results the model detects many false positives that do not exist in the image at all, see figure 5.1. The model may be detecting other foods in the image as there are a lot of similarly shaped foods. There is only one pea in this MRI scan and it does detect it. Most of the MRI scans are relatively empty but this shows how many spherical objects in the stomach can produce false results. This is why I used a large testing set as the scans are varied.

To improve the object detector more data is needed as more scans will produce a better model to be trained, and should be more accurate. Possibly, retraining the model with One left out cross validation may prove better results. Consuming all the data to be trained par one to test it, preferably an MRI scan with lots of peas. However we would not know if the results are accurate in images such as fig 4.3 unless we tested against this specific image. Image ROI 149 76 (fig 5.2) produced positive results for 10 objects with only one pea being in the image, however it did detect that pea. Another method I believe that could make the model more accurate is by pre processing the

images so that they are not grey scale. This might improved accuracy as then there would difference in image intensity, which may change how the network is learning on the images.

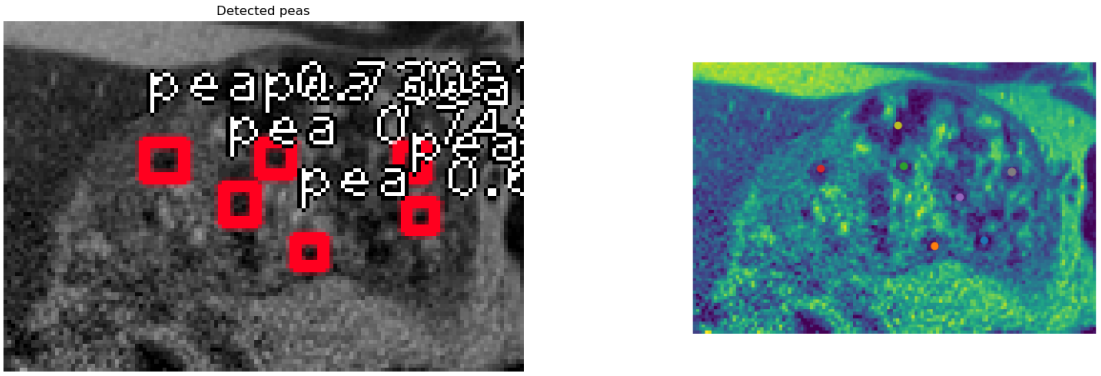


Figure 5.1: Example of the results

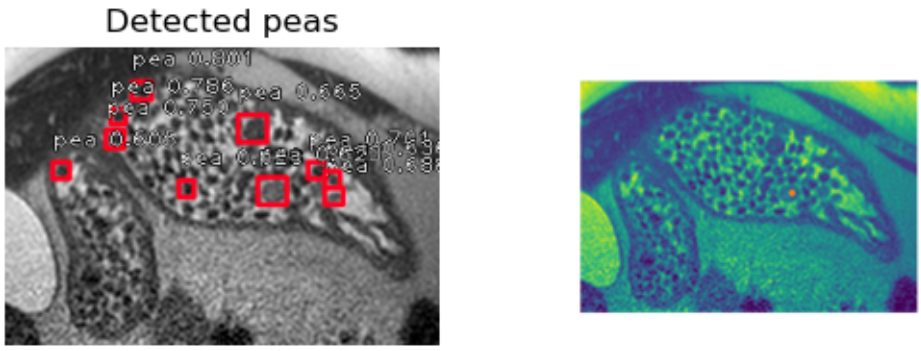


Figure 5.2: Example where Neural Net detected lots of false positives

Chapter 6

Conclusion

In conclusion, RetinaNet is capable of detecting specific foods in MRI scans of stomachs. The network produced produces a precision of 0.81 detecting a majority of true positives(See table 5.1). However its Recall value is too low at 0.45, producing too many false negatives, the network needs more training and more data to be viable to be used commercially, in industry or research at its current capabilities. By using image processing techniques and with more training specific foods could easily be detected and not just peas.

Chapter 7

Appendix

prepareData.py

```
##
#Author: Josh Gardner
#Script to parse dataset files, extracting and renaming them
#
##
import argparse # To process the command line arguments
import os # To parse the filesystem and create directories
import sys # To print the standard output of errors
import numpy as np # to load txt files
import cv2
import errno # To handles errors when file are copied and directories created
import shutil # To copy files

def createDir(aDirectory):
    try:
        os.mkdir(aDirectory)
    except OSError as e:
        if e.errno != errno.EEXIST:
            print ("Creation␣of␣the␣directory␣%s␣failed" % aDirectory, file=sys.stderr)
            raise
#moves tsv files and renames them according to related image file
def moveTSV(aName, aDir, aType):
    tsv_name = str(aName[:-4] + ".tsv")
    print(tsv_name)
    shutil.copyfile(aDir,os.path.join(args.newDir[0],"Slices",aType,tsv_name))

def prepare_data(aSource, aDir):
    pos_count = 0
    neg_count = 0
    slice_count = 0
    for dataset in os.listdir(aSource):
    #loops through all path directories extracting the data into correct directories to be used
        if os.path.isdir(os.path.join(aSource,dataset)):
            if dataset[:7] == "DATASET":
                dataset_id = dataset[7:]
                dataset_path = os.path.join(aSource,dataset)
                for slices in os.listdir(dataset_path):
                    if os.path.isdir(os.path.join(dataset_path,slices)):
                        if slices[:6] == "slice_":
                            slice_id = slices[6:]
                            slice_path = os.path.join(dataset_path,slices)
                            for labels in os.listdir(slice_path):
                                if (labels[:6] == "slice_" or labels[:3]=="ROI") and labels[-4:] == ".png":
                                    slice_count+=1
                                    if (int(dataset[7:])%2)==0: #splits data into training and data sets
                                        shutil.copyfile(os.path.join(slice_path,labels), os.path.join(aDir,"Slices","train",labels))
                                        if labels[:3] == "ROI" and os.path.exists(os.path.join(slice_path,labels,"pos","pos.tsv")) == True:
                                            moveTSV(labels,os.path.join(slice_path,labels,"pos","pos.tsv"),"train")
                                    else:
                                        shutil.copyfile(os.path.join(slice_path,labels), os.path.join(aDir,"Slices","test",labels))
```

```
                                if labels[:3] == "ROI" and os.path.exists(os.path.join(slice_path,labels,"pos","pos.tsv")) == True:
                                    moveTSV(labels,os.path.join(slice_path,labels,"pos","pos.tsv"),"test")
                            #moves and renames images of peas
                            elif os.path.isdir(os.path.join(slice_path, labels)):
                                if labels == "pos" or labels == "neg":
                                    labels_path = os.path.join(slice_path, labels)
                                    for img in os.listdir(labels_path):
                                        if img[-4:] == ".png":
                                            if labels == "pos":
                                                pos_count+=1
                                                image = cv2.cvtColor(cv2.imread(os.path.join(labels_path, img)), cv2.COLOR_BGR2GRAY)
                                                imgName="Pea_"+str(pos_count)+".png"
                                                if (int(dataset[7:])%2)==0:
                                                    cv2.imwrite(os.path.join(aDir,"Peas","train",imgName),image)
                                                else:
                                                    cv2.imwrite(os.path.join(aDir,"Peas","test",imgName),image)
                                            elif labels == "neg":
                                                neg_count+=1
                                                image = cv2.cvtColor(cv2.imread(os.path.join(labels_path, img)), cv2.COLOR_BGR2GRAY)
                                                imgName="neg_"+str(neg_count)+".png"
                                                cv2.imwrite(os.path.join(aDir,"Neg",imgName),image)

    print("Prepared and Moved", pos_count, "images and moved", neg_count, "negative images")
    print("Moved", slice_count, "Slices")


parser = argparse.ArgumentParser(description='Prepare and Copy images to new directory')
parser.add_argument('--dir', help='Where the data to copy is', nargs=1, type=str, required=True)
args = parser.parse_args()
newPath = "Data"
createDir(newPath)
createDir(os.path.join(newPath,"Peas"))
createDir(os.path.join(newPath,"Neg"))
createDir(os.path.join(newPath,"Slices"))
createDir(os.path.join(newPath,"Detected"))
createDir(os.path.join(newPath,"Located_Peas"))
createDir(os.path.join(newPath,"Located_Peas","train"))
createDir(os.path.join(newPath,"Located_Peas","test"))
createDir(os.path.join(newPath,"Peas","train"))
createDir(os.path.join(newPath,"Peas","test"))
createDir(os.path.join(newPath,"Slices","train"))
createDir(os.path.join(newPath,"Slices","test"))
prepare_data(args.dir[0],newPath)
```

pea_locator.py

```
##
# Author: Josh Gardner
# Displays and saves locations of peas in an image
##
import matplotlib.pyplot as plt
import cv2
import csv
import argparse
import os

#reads tsv file to get pea coordinates and displays them on the image.
def locatePea(name):
    img=name + ".png"
    tsv=name + ".tsv"
    im = plt.imread(os.path.join(path,img))
    plt.imshow(im)
    with open(os.path.join(path,tsv)) as tsvfile:
        reader = csv.reader(tsvfile, delimiter='\t')
        next(reader)
        for row in reader:
            x = int(row[1])
```

```
            y = int(row[2])
            plt.scatter([x], [y])
    return im

parser = argparse.ArgumentParser(description='Display the locations of peas on ROI')
parser.add_argument('--dir', help='Where the images are', nargs=1, type=str, required=True)
args = parser.parse_args()
path = args.dir[0]
#see if new folder path is test or train
if path[-5:] == "train":
    targ = path[-5:]
else:
    targ = path[-4:]
for img in os.listdir(path):
    if img[:3] == "ROI" and img[-4:] == ".png":
        im = locatePea(img[:-4])
        plt.axis('off')
        plt.savefig(os.path.join("Data","Located_Peas",targ,img))
        plt.title(img[:-4])
        plt.show()
```

xml_to_csv.py

```
##
#Author: Josh Gardner
#Parses XML data into a pandas DataFrame to be saved in a csv
#
##
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET
import argparse
import csv

def xml_to_csv(path):
    #function to parse xml files and extract the data to a dataframe
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('path').text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text),
                     member[0].text
                     )
            xml_list.append(value)
    column_name = ['path', 'x1', 'y1', 'x2', 'y2', 'class']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

path = "Data/Slices/train"
xml_df = xml_to_csv(path)
xml_df.to_csv('train_labels.csv', index=None)
with open("classes.csv", mode='w', newline='') as class_file:
    class_writer = csv.writer(class_file,delimiter=",",quotechar='"',quoting=csv.QUOTE_MINIMAL)
    class_writer.writerow(["pea", 0])

print('Successfully converted xml to csv.')
```

detectPeas.py

```
##
#Author: Joshua Gardner
#Loads model and classes to detect pea objects in images
##
import os
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
import argparse
from keras_retinanet import models
from keras_retinanet.utils.image import read_image_bgr, preprocess_image, resize_image
from keras_retinanet.utils.visualization import draw_box, draw_caption
from keras_retinanet.utils.colors import label_color

CLASSES_FILE = "classes.csv"
#sorts through snapshot folder to find latest trained model
model_path = os.path.join('snapshots', sorted(os.listdir('snapshots'),reverse=True)[0])
model = models.load_model(model_path,backbone_name ="resnet50")
model = models.convert_model(model)
labels_to_name = pd.read_csv(CLASSES_FILE, header= None).T.loc[0].to_dict()

THRES_SCORE = 0.6
#prepares image for detection model, draws boxes and predicition scores and saves the image
def detect_objects(aImg):
    imgName = aImg
    image = read_image_bgr(os.path.join(path,aImg + ".png"))
    draw = image.copy()
    draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
    image = preprocess_image(image)
    image, scale = resize_image(image)
    boxes, scores, labels = model.predict_on_batch(np.expand_dims(image,axis=0))
    boxes /=scale
    for box, score, label in zip(boxes[0], scores[0], labels[0]):
        if score < THRES_SCORE:
            break
        color = label_color(label)
        b = box.astype(int)
        draw_box(draw, b, color=color)
        caption = "{} {:.3f}".format(labels_to_name[label], score)
        draw_caption(draw, b , caption)
    cv2.imwrite(os.path.join("Data","Detected",imgName+".png"),draw)

path = "Data/Slices/train"
for img in os.listdir(path):
    if img[-4:] == ".png" and img[:1] == "R":
        detect_objects(img[:-4])
```

displayResults.py

```
##
# Author: Josh Gardner
# Displays detected results against actual locations in the images
##
import matplotlib.pyplot as plt
import os

detectPath = "Data/Detected"
locPath = "Data/Located_Peas/test"
#Display images of detected objects along side actual locations
for img in os.listdir(detectPath):
    if os.path.isfile(os.path.join(detectPath, img)):
        imgName = img[:-4]
        im1 = plt.imread(os.path.join(detectPath, img))
```

```
15          im2 = plt.imread(os.path.join(locPath, imgName+".png"))
16          fig = plt.figure()
17          a = fig.add_subplot(1, 2, 1)
18          imgplt = plt.imshow(im1)
19          plt.axis('off')
20          a.set_title("Detected␣peas")
21          a = fig.add_subplot(1,2,2)
22          imgplot = plt.imshow(im2)
23          plt.axis('off')
24          plt.show()
```

Acronyms

CT computed tomography.

MRI magnetic resonance imaging.

CSV Comma-Separated Values.

FAIR Facebook AI Research.

HOG Histogram of oriented gradients.

CNN Convolutional neural network.

NN Neural network.

R-CNN Regions with Convolutional Neural Network.

COCO Common Objects in Context.

ROI Region of Interest.

RPN Region Proposal Network.

YOLO You Only Look Once.

mAP mean Average Precision.

SVM Support vector machine.

VOC PASCAL Visual Object Classes Challenge.

fps frames per second.

FPN Feature Pyramid Network.

TSV Tab-separated values.

XML Extensible Markup Language.

Bibliography

[1] N.I Glumov, E.I. Kolomiyetz and V. V. Sergeyev. ‘Detection of objects on the image using a sliding window mode’. In: 27.4 (1995). Optics and Image Processing in Russia, pp. 241–249.

[2] *Anchor Boxes for Object Detection*. (Accessed 30/5/2020). URL: `https://uk.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html`.

[3] Moses Olafenwa. *Object Detection with 10 lines of code*. Jun 16, 2018 (Accessed 30/5/2020). URL: `https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606`.

[4] P. Viola and M. Jones. ‘Rapid object detection using a boosted cascade of simple features’. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, pp. I–I.

[5] Audrius Bukis et al. ‘Survey of face detection and recognition methods’. In: *The 6th International Conference on Electrical and Control Technologies* (May 2011).

[6] N. Dalal and B. Triggs. ‘Histograms of oriented gradients for human detection’. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 886–893 vol. 1.

[7] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2014.

[8] R. Girshick et al. ‘Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation’. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587.

[9] R. Girshick. ‘Fast R-CNN’. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448.

[10] S. Ren et al. ‘Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149.

[11] J. Redmon et al. ‘You Only Look Once: Unified, Real-Time Object Detection’. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788.

[12] T. Lin et al. ‘Focal Loss for Dense Object Detection’. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007.

[13] T. Lin et al. ‘Feature Pyramid Networks for Object Detection’. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944.

PRIFYSGOL
BANGOR
UNIVERSITY

Detecting Specific foods in MRI scans of stomachs using Python

Intro

The overall aim of this project is to train a Neural Network to detect frozen peas in MRI scan slices as part of an ongoing research project looking into digestion and the future of food alternatives. The neural network will be trained on regions of the MRI that are known to be the peas then tested on images of the whole MRI slice.

Technology

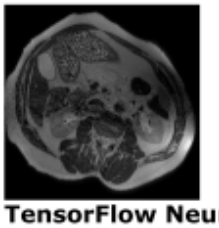
The project will be coded in Python, incorporating TensorFlow's machine learning neural network (CNN).

TensorFlow is a python library for deep learning and fast numerical computing released by google.

Python is a high-level interpreted programming language emphasizing on code readability and general-purpose use with extensive libraries.

Results

The results will show where the Neural Network has detected peas in the images MRI Scans



TensorFlow Neural Network

TensorFlow's Neural Network can be trained to recognise objects in images by feeding the neural net images of the specific objects, in this case frozen peas training it to detect for the peas in MRI slices. Neural Networks are currently being used in many different applications from Image detection, machine learning and is being used in companies such as Telsa for their autopilot systems.

Future work

As the data set being used with the experiment is quite small relative to most neural network training sets, more data could be collected to train the network to be more accurate. The network could easily be retrained for other foods for the research project if pea sized food objects are found to not be suitable.

Author: Josh Gardner
eeub10@bangor.ac.uk

Supervisor: Franck Vidal
f.vidal@bangor.ac.uk