

Do Dinh Duc

Building real time object detection iOS application using machine learning

Metropolia University of Applied Sciences

Bachelor of Engineering

Media Engineering

Bachelor thesis

2 April 2018

Author Title Number of Pages Date	Do Dinh Duc Building real time object detection iOS application using machine learning 49 pages 2 April 2018
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Professional Major	Mobile Solutions
Instructors	Petri Vesikivi, Principal Lecturer
<p>The goal of this project was to introduce and develop complete neural network models, then use them as part of a real time object detection application which supports multiple models' integrations.</p> <p>The basic neural network architectures were presented to demonstrate the hypothesis of how neurones are connected to each other. One simple example of neural network usage was included in order to show the algorithm's ability to learn by itself. Some examples of how neural network has affected your daily life will show how far this technique has come and changed human life.</p> <p>Machine learning has played an important role in modern software application, and in most cases, neural network is the heart of the algorithms. Along with it, hardware innovation and the rise of cloud computing technology are helping developers to write, run, and deploy neural network algorithms much faster with reasonable costs. As a result, real time object detection has become usable on our personal devices with great potential.</p> <p>This project describes how to build an image classification neural network and trained models out of it with different existing architectures using Turi Create, then integrate them into an iOS application with CoreML and Vision.</p>	
Keywords	neural networks, machine learning, algorithms, CoreML, Turi Create, Swift, Python, SqueezNet, ResNet50

Contents

1	Introduction	1
2	Theoretical background	2
2.1	<i>Relationship between Artificial Intelligent, Machine Learning, Deep Learning, and Neural Networks</i>	2
2.2	<i>Machine learning</i>	3
2.3	<i>Neural Networks</i>	5
2.4	<i>CoreML framework</i>	9
2.5	<i>Vision framework</i>	10
3	Methods and materials	11
3.1	<i>Neural network for classify images with TuriCreate</i>	11
3.2	<i>Lap timer iOS application</i>	18
3.2.1	<i>App design</i>	18
3.2.2	<i>Implementation without Vision</i>	21
3.2.3	<i>Implementation with Vision</i>	21
4	Results	22
4.1	<i>Collect results</i>	22
4.2	<i>Different models benchmark</i>	25
4.3	<i>Vision and non-Vision benchmark</i>	27
5	Discussion	28
6	Conclusion	29
	References	30
	Appendices	1

1 Introduction

To date, people have heard the terms “Machine learning” and “Neural networks” regularly without knowing the meaning behind them. Neural Networks, which is one of Machine Learning’s state of the art algorithms, has become the trendiest topic in software industry alongside with Big Data, Micro services, Virtual Reality, and Augmented Reality. In daily life, one can easily acknowledge the existence of neural networks in every digital service, and in most of the cases, it is known as a recommended system. For example, Spotify offers “Your daily mixes”, or “Recommended stations”, the “Recommend” section in Youtube, and “Inspired by your shopping” from Amazon which are all used to gather data while customers are using the applications, and processing them with their own neural network algorithms, and finally use those output results useful to them.

Unfortunately, due to the concept’s level of complexity, neural networks have been a modern myth for general public, even for lots of developers. Therefore, the purpose of this report is to give a glimpse of neural networks, how they are defined and introduce the simplest neural network structures.

With the help of machine learning through open source libraries such as Keras and Turi.create, and powerful interactive programming tools such as hydrogen and Jupyter lab, machine learning applications are getting easier to build. Furthermore, many advanced neural network architectures have been developing rapidly also to boost mobile applications using a machine learning model, especially in the area of real time object detection.

2 Theoretical background

2.1 Relationship between Artificial Intelligent, Machine Learning, Deep Learning, and Neural Networks

The terms Artificial Intelligent, Machine Learning, Deep Learning, and Neural Networks are often found together in lots of documents or articles, even though their meaning is often mixed up. In many cases, readers think all those terms refer to the same thing. Hence, figure 1 demonstrates the differences between Artificial Intelligent, Machine Learning, and Deep Learning.

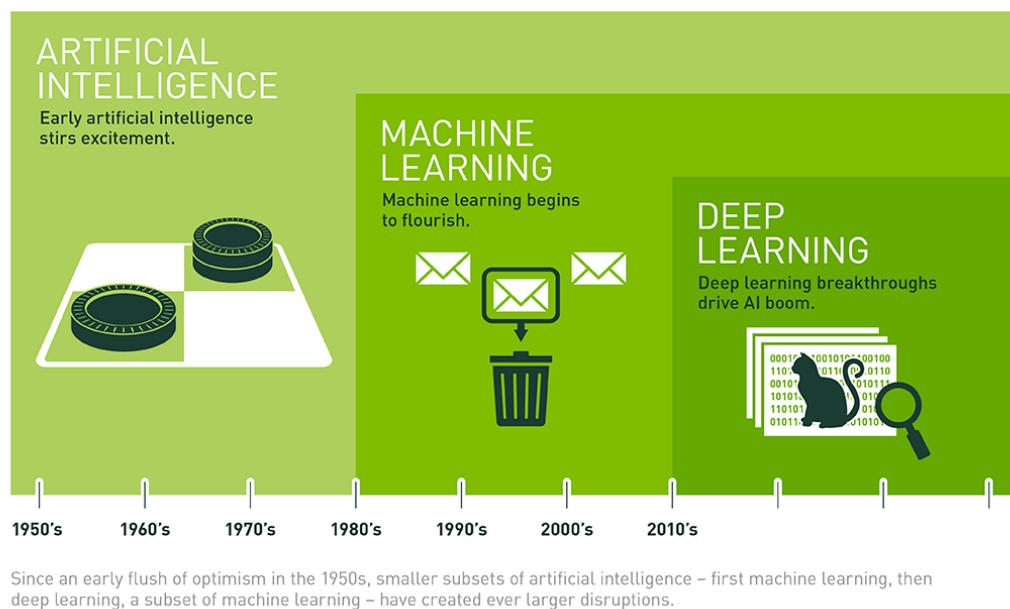


Figure 1: Artificial Intelligence, Machine Learning, and Deep Learning connections. Copied from Copeland [1].

Deep learning, which means complex neural networks, is the most widely used Machine Learning algorithm. Machine learning, on the other hand, is part of Artificial Intelligence, and a field of Computer Sciences.

2.2 Machine learning

Machine learning is a computer science field which started at the beginning of computer science history. In 1950, Alan Turing, the founder of computer science, asked the question “Can machines think?” [2], which set the very first milestone for machine learning studies. Arthur Samuel later defined machine learning “field of study that gives computers the ability to learn without being explicitly programmed” [3]. However, machine learning was finally defined by Tom M. Mitchell:

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with the experience E .” [4].

For instance, in an email application, for classify if an email should be mark as spam or not:

- Experience E is observing user label spam emails manually
- Task T is classifying emails into desired groups (spam/ not spam)
- Performance measure P is the fraction of emails correctly classified

As a result, the application’s purposes narrow down to improve the performance measure (P).

Figure 2 illustrates the differences between supervised and unsupervised learning.

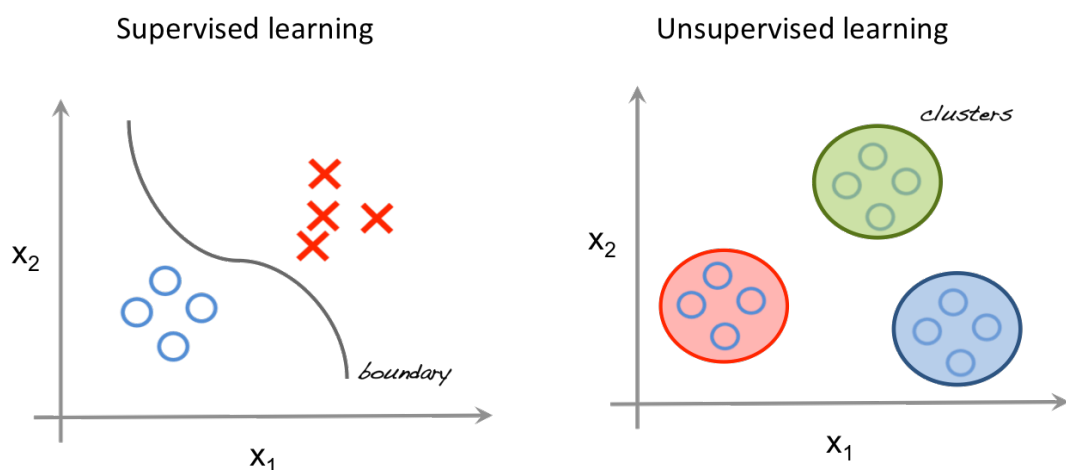


Figure 2. Supervised vs. Unsupervised learning. Copied from Cambridge Spark [5].

Machine learning algorithms are categorised into two major types, as demonstrated in figure 2:

- Supervised learning: where the output result groups are categorised with clear conditions, for example: labelling emails as spam or not, predicting if patient has cancer or not, predicting housing price, etc.
- Unsupervised learning: where the output result groups are categorised with abstract conditions, for example: grouping a set of news article from the internet into groups that shared the same story, categorising different groups of consumer customers for marketing purposes, etc.

2.3 Neural Networks

Neural Networks is seen as state-of-the-art technique for many applications. The original idea was created in the 80s: to create algorithms that are capable of mimicking animal brains, or in other words, simulating the brain's architecture. These algorithms were developed and used actively for a decade before fading out in the late 90s due to the limitation of hardware technology at that time. All of Neural Networks algorithms' levels of complexity are high and it takes a long time to execute them on a 1990s-supercomputer. However, with the current technologies, the algorithms' running time is now acceptable and can be compiled on cloud servers, which has brought the whole Neural Networks section back to life [6].

Figure 3 displays a neuron inside a brain.

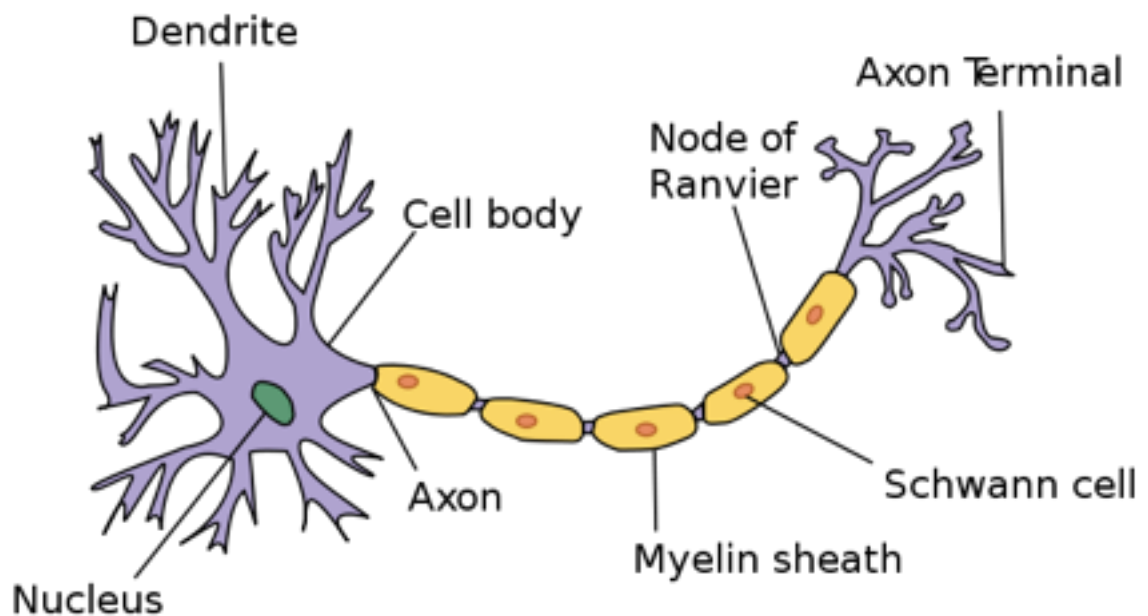


Figure 3. The structure of a typical neuron. Modified from Andrew Ng (2017) [6].

In order to copy the brain's architecture, which is based on multiple neuron units working parallel with the various inputs and outputs, Neural Networks algorithms focus on simulating a single neuron in the brain and connect them together in a network demonstrated in figure 3. A casual neuron takes electrical input, called spikes from dendrite. Then, the data signal is processed through nucleus before sending the output terminal, axon. Similarly, a neuron model has the input data, and it transfers data to the next layer to process. In the last neuron layer, called output layer, the data is fully processed and ready to be used [6].

Figure 4 and 5 illustrate simple neuron networks.

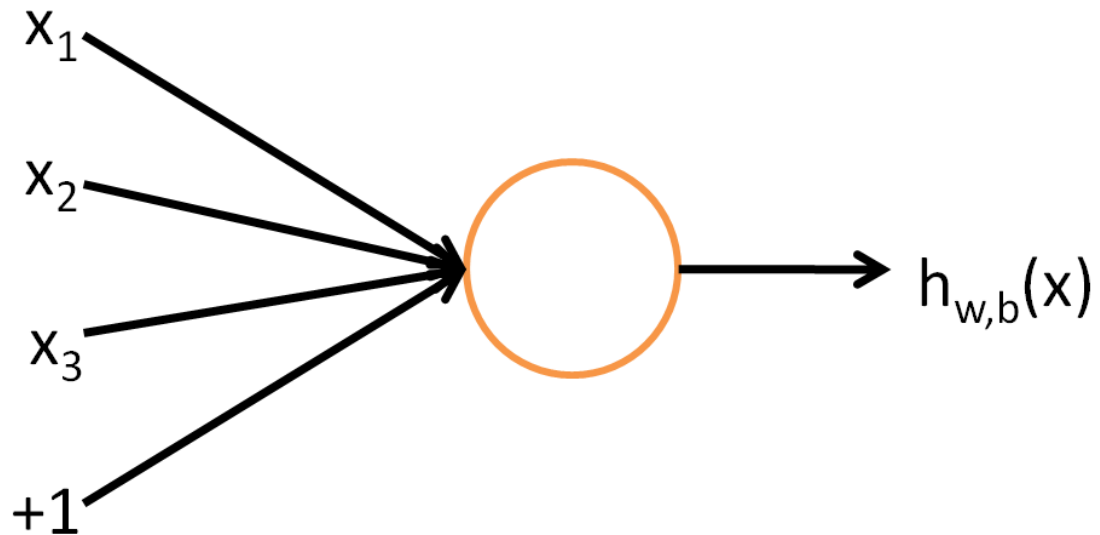


Figure 4. Neuron network with no hidden layer. Copied from Stanford UFLDL [7].

A simple neural network without hidden layer is demonstrated in figure 4. From the left of the figure, the input layer with 3 inputs (x_1, x_2, x_3), the inputs then get passed to output layer and transformed into the expected data, weights (w, b), which are also symbolised as theta θ . From the input layer, there is a plus one input in the bottom called bias unit which allows one to shift the sigmoid activation function, in other words, mutate the output results θ without changing the input parameters X [6].

Set the bias unit as x_0 , and using sigmoid logistic activation function for this network, input parameters X and weights θ and the hypothesis logistic function h are demonstrated as:

$$X = \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix} \quad \theta = \begin{matrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{matrix} \quad h_{\theta}(x) = \frac{1}{1+e^{-\theta^T x}}$$

With a single transformation using sigmoid logistic activation function $g(z)$, one can figure out the weight of each input parameter.

$$g(z) = \frac{1}{1 + e^{-z}}$$

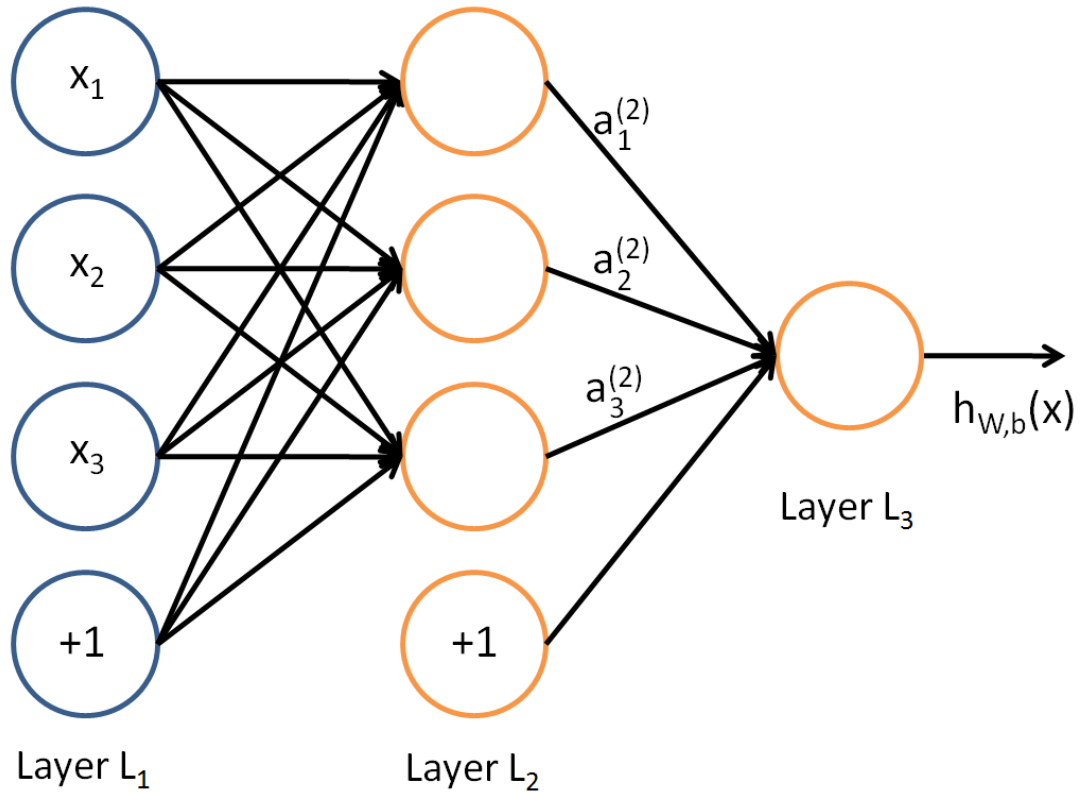


Figure 5. Basic neuron network. Copied from Stanford UFLDL [7].

Moving on to a basic neuron network is displayed in figure 5. This network is very basic, with three input units in input layer L1 (x_1 , x_2 , x_3) and the same output layer as in figure 4. The only different in figure 5 is layer L2, a hidden layer with three hidden units. The purpose of a hidden layer is re-adjusting the meaning input parameters, which is the key point of neural network and learning by themselves which parameter should be more important. In other words, with all the data gathered, the network has learnt by itself and improves the input data through each hidden layer. From the raw input data X , each activation of unit i in layer j $a_i^{(j)}$ is the result of a transformation from previous layer, and the same process only stops when it reaches the output layer [6].

In this case, with $\theta^{(i)}$ as matrix of weights controlling function mapping from layer j to layer $j+1$:

$$a_1^{(2)} = g \left(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 \right)$$

$$a_2^{(2)} = g \left(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 \right)$$

$$a_3^{(2)} = g \left(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 \right)$$

$$h_{\theta}(x) = g \left(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)} \right)$$

$$\text{And the final weight } \theta^{(2)} = [\theta_{10}^{(2)}; \theta_{11}^{(2)}; \theta_{12}^{(2)}; \theta_{13}^{(2)}]$$

Adding more hidden layers will give better optimisation weights, which also means the final weights result will be much better, also improving the learning its own features. However, on the down side, each hidden layer also increases the runtime of the algorithm significantly. Therefore, the number of hidden layers should be balanced between the output quality and the resources used.

2.4 CoreML framework

CoreML, the first ever fundamental machine learning framework from Apple, was released in 2017 and focuses mostly on image analysis, natural language processing (NLP) with a limited number of APIs and is only available for iOS 11 and above [8].

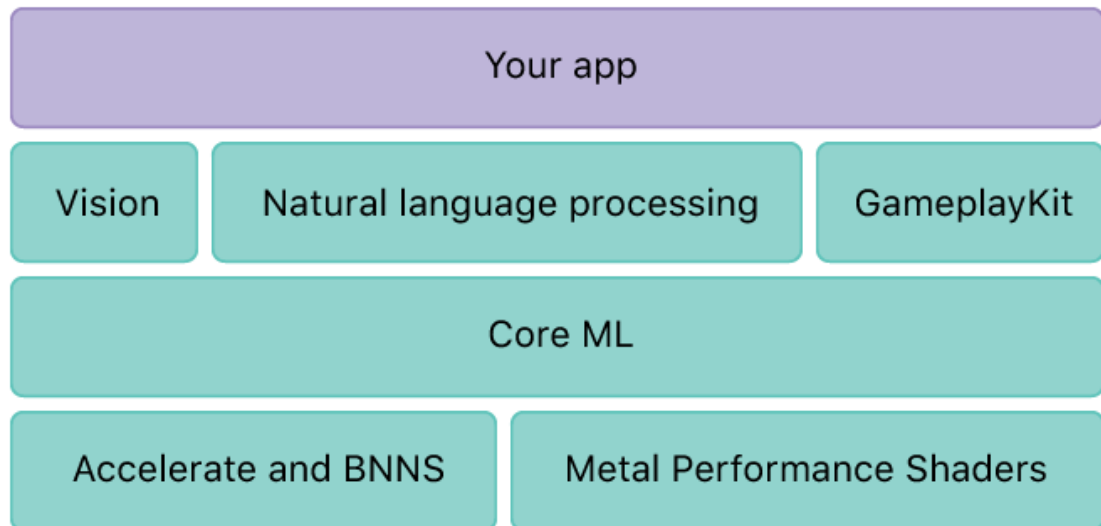


Figure 6. General machine learning application structure. Copied from Apple documentation [8].

As can be seen from figure 6, CoreML is a middle layer framework which is built on top of machine learning performance primitive frameworks such as Accelerate, BNNS, and Metal Performance Shaders, which collect and optimise raw data processed from those frameworks and send them to higher level frameworks such as Vision, NLP, or GameplayKit.

CoreML framework is a powerful machine learning framework, yet it comes with straight forward usage. In order to start integrating machine learning application, one just needs to add a Core ML model file into the project. The framework will generate the model class, which includes all utilities that help predict the output with selected input. More details on integration will be found in chapters 3.2.2 and 3.2.3.

2.5 Vision framework

Introduced and released along with CoreML in a previous Apple Worldwide Developers Conference (WWDC 2017), Vision is a high-level framework which used for processed visual data (images or videos). It is received from CoreML, using the devices' GPU for image analysis tasks [9]. Vision plays an important role in enhancing performance for real time object detection applications (displays later in section 4.3)

CoreML framework, supports multiple input data types such as numbers, text, and images, and focuses on machine learning and deep learning algorithms. Eventually, it is not optimised to work with graphic data. Vision framework, which is built on top of CoreML, has improved the image processing phase.

Vision framework's APIs usage is also straight forward. First, a Vision request will be declared with CoreML model and an option completion handler block as an input. Inside the completion block, one can catch the predicted outputs of the image inputs and then start processing the results (more details on Vision's implementation can be found in section 3.2.3). Furthermore, using call back integration would help the application process more image inputs at the same time rather than queuing inputs like CoreML's approach for better performance.

3 Methods and materials

3.1 Neural network for classify images with TuriCreate

Overview

Turi Create is an open-sourced machine learning framework developed by Apple and released in 2017. Alongside with Keras it has become one of the most powerful tools for quick and accurate solutions for machine learning models [10].

Turi Create's main purpose is focusing on developers who do not have much experience in machine learning. It has been efficient in simplifying the amount and complexity level of code one needs to get through to create a working custom machine learning model while also keeping its own scaling ability. Furthermore, no extra steps are needed to export models to "mlmodel" format, which is Apple's official supported machine learning model format for CoreML. With Apple's support, Turi Create has a good chance to become one of the major frameworks in machine learning.

Installation

- Install Python Package Index – pip

```
sudo easy_install pip
```

- Install Turi Create

```
pip install turicreate
```

- Install Atom

```
brew tap caskroom/cask && brew cask install atom
```

- Install Hydrogen

```
apm install hydrogen
```

Implementation

Create workspace

Create folder TCFruitClassification.

```
mkdir TCFruitClassification
```

```
cd TCFruitClassification
```

Create file classify.py and FruitImages folder pollute dataset with training data of Bananas and Avocados. In this implementation, images dataset from Horea will be used [11].

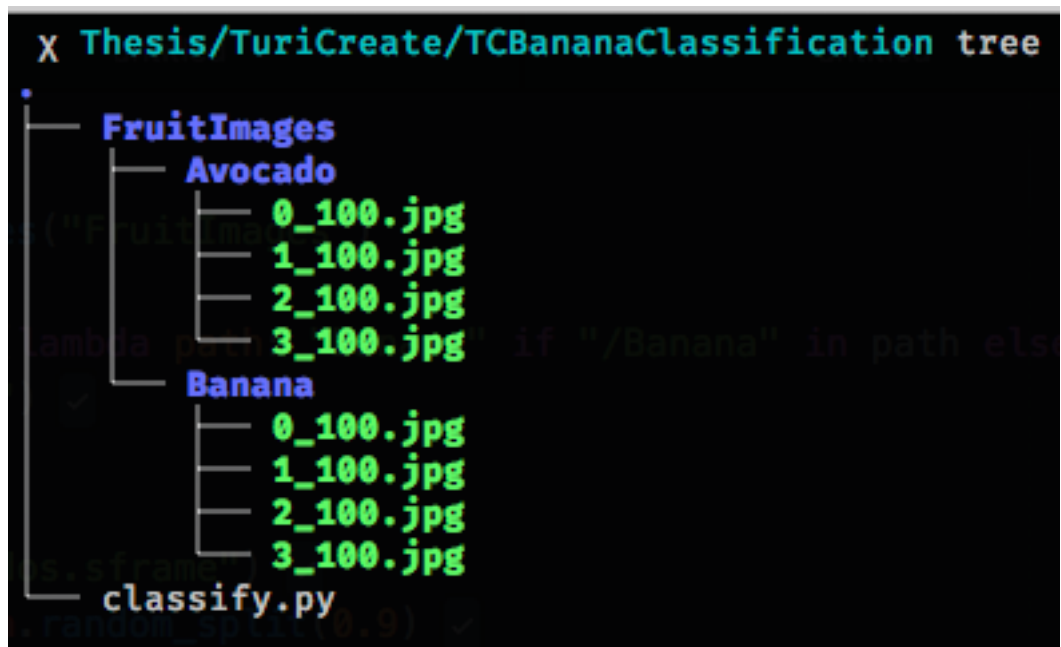


Figure 7. Tree graph of TCFruitClassification folder

All examples, training and test ones, can be put together in one place, Turi Create will then automatically separate examples to training examples and test examples.

```

1 # Import turicreate
2 import turicreate as tc
3
4 # Load images from FruitImages folder
5 data = tc.image_analysis.load_images("FruitImages", with_path=True)
6
7 # Define labels for classification
8 labels = ['Apple', 'Avocado', 'Banana', 'Plum', 'Strawberry']
9
10 def getLabel(path, labels=labels):
11     for label in labels:
12         if label in path:
13             return label
14
15 # Define data label using in exported model is fruit
16 data['fruit'] = data['path'].apply(getLabel)
17
18 # Save examples into bananas-avocados.sframe for training model
19 data.save("fruits.sframe")
20
21 # Visualised labeled data
22 data.explore()
23
24 # Fetching all data from sframe file
25 allData = tc.SFrame("fruits.sframe")
26
27 # Splitting all labeled examples into 90% of training examples and 10% of testing examples
28 trainingData, testingData = allData.random_split(0.9)
29
30 # Define 2 types of architecture available to train data
31 squeezeNetModel = "squeezenet_v1.1"
32 resNetModel = "resnet-50"
33
34 # Create model with selected architecture
35 model = tc.image_classifier.create(trainingData, target="fruit", model=squeezeNetModel, max_iterations=100)
36
37 # Save predictions to mlmodel
38 predictions = model.classify(testingData)
39
40 # Evaluate created model and print out accuracy
41 evaluations = model.evaluate(testingData)
42 print "Accuracy : %s" % evaluations['accuracy']
43 print "Confusion Matrix : \n%s" % evaluations['confusion_matrix']
44
45 # Save model
46 model.save("fruits.model")
47
48 # Export model to CoreML model
49 model.export_coreml("FruitClassifierResNet.mlmodel")
50

```

Figure 8. Implementation of TuriCreate classification fruits images

As seen in figure 8, the whole process from fetching images data (line 5) until a completed CoreML model is exported (line 49), is straight forward. The data will be resized and labelled in line 16, with five labels: “Apple”, “Avocado”, “Banana”, “Plum”, and “Strawberry”. Labelled data then will be saved to an “sframe” file in line 19.







Turi Create Visualization			
	path	image	fruit
97	/Users/doduc/workspaces/Thesi...		Avocado
98	/Users/doduc/workspaces/Thesi...		Avocado
99	/Users/doduc/workspaces/Thesi...		Avocado
100	/Users/doduc/workspaces/Thesi...		Banana
101	/Users/doduc/workspaces/Thesi...		Banana
102	/Users/doduc/workspaces/Thesi...		Banana

Figure 9. Banana and avocado images get labelled and visualised.

Figure 9 is the result of “data.explore()”, which visualises the labelled data and displays them to the user. It is a good check for a developer before generating a new model. A model will be created after this step and one will have two architecture options for initialising: ResNet50 with more accuracy results but heavier model output, and SqueezeNet which provides lower accuracy results but is a lightweight trained model. The next step is testing the model’s accuracy level with testing dataset (line 41), and if the result is acceptable (line 42, 43), model will be saved (line 46) and exported to CoreML format (line 49).

```

34
35 # Save model
36 model.save("bananas-avocados.model")
37
38 # Export model to CoreML model
39 model.export_coreml("FruitClassifierResNet.mlmodel")

```

Unsupported image format. Supported formats are JPEG and PNG file: /Users/doduc/workspaces/Thesis/TuriCreate/TCBananaClassif

Unsupported image format. Supported formats are JPEG and PNG file: /Users/doduc/workspaces/Thesis/TuriCreate/TCBananaClassif

Unsupported image format. Supported formats are JPEG and PNG file: /Users/doduc/workspaces/Thesis/TuriCreate/TCBananaClassif

Materializing SFrame...

Done.

Resizing images...

Performing feature extraction on resized images...

Completed 181/181

PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Unless an app

Logistic regression:

Number of examples : 171

Number of classes : 2

Number of feature columns : 1

Number of unpacked features : 1000

Number of coefficients : 1001

Starting L-BFGS

Iteration	Passes	Step size	Elapsed Time	Training-accuracy	Validation-accuracy
1	6	0.000421	0.120289	0.502924	0.000000
2	8	1.000000	0.169113	1.000000	1.000000
3	9	1.000000	0.201620	1.000000	1.000000
4	10	1.000000	0.236091	1.000000	1.000000
5	11	1.000000	0.271864	1.000000	1.000000
6	12	1.000000	0.306314	1.000000	1.000000
11	17	1.000000	0.477368	1.000000	1.000000

SUCCESS: Optimal solution found.

Resizing images...

Performing feature extraction on resized images...

Completed 19/19

1.0

Figure 10. TuriCreate classify neural network running result

Figure 10 shows all steps in above sourced code compiled interactively from Atom IDE with Hydrogen plugin. It gives the same experience as Jupyter Notebook. There are a

few warnings about unsupported image format from .DS_Store file generated automatically in MacOS., such as the number of feature dimensions in this problem is very large in comparison with the number of examples. Unless an appropriate regularization value is set, this model may not provide accurate predictions for a validation/test set.” This warning is a result of the neural network architecture which is used, either SqueezeNet or ResNet. It comes with multiple features for handling data, which normally provides more than 200 examples in this implementation. Therefore, one can remove this warning by providing more training dataset.

Results

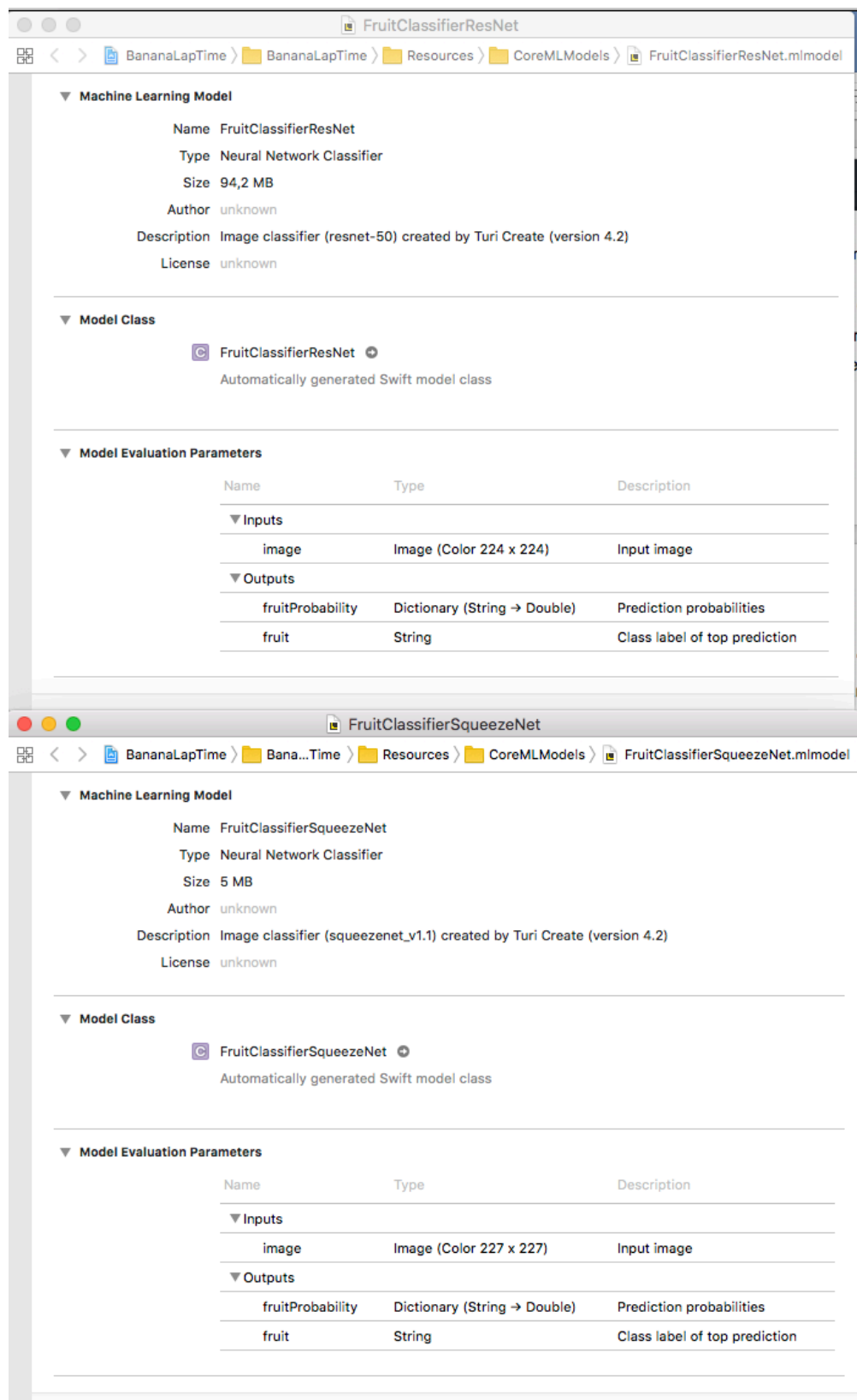


Figure 11. CoreML models for Banana-Avocado classification with ResNet and SqueezeNet architecture

As can be seen from figure 11, there is a big difference between the size of these two output models, 5 MB for SqueezeNet model and 94.2 MB for ResNet model.

3.2 Lap timer iOS application

3.2.1 App design

BananaLapTime is an iOS application for automatically recording lap times of an object using real-time object detection system with machine learning models. The application is designed with two major goals: using machine learning models to detect object on the fly and support multiple machine learning models.

The application's initial idea came from Mr. Sami Pippuri (you need the year and reference here, if you mentions a person's name!!) when he watched some races and thought about improving the timer systems used. If the timer can detect each different car in the race automatically, it will have the ability to record the lap time with one device only, instead of having multi devices to follow the whole race. With laptops, phones, and tablets, it is relative easy for one to own a smart timer on the smartphone, with acceptable accuracy. This is the reason BananaLapTime project got started.

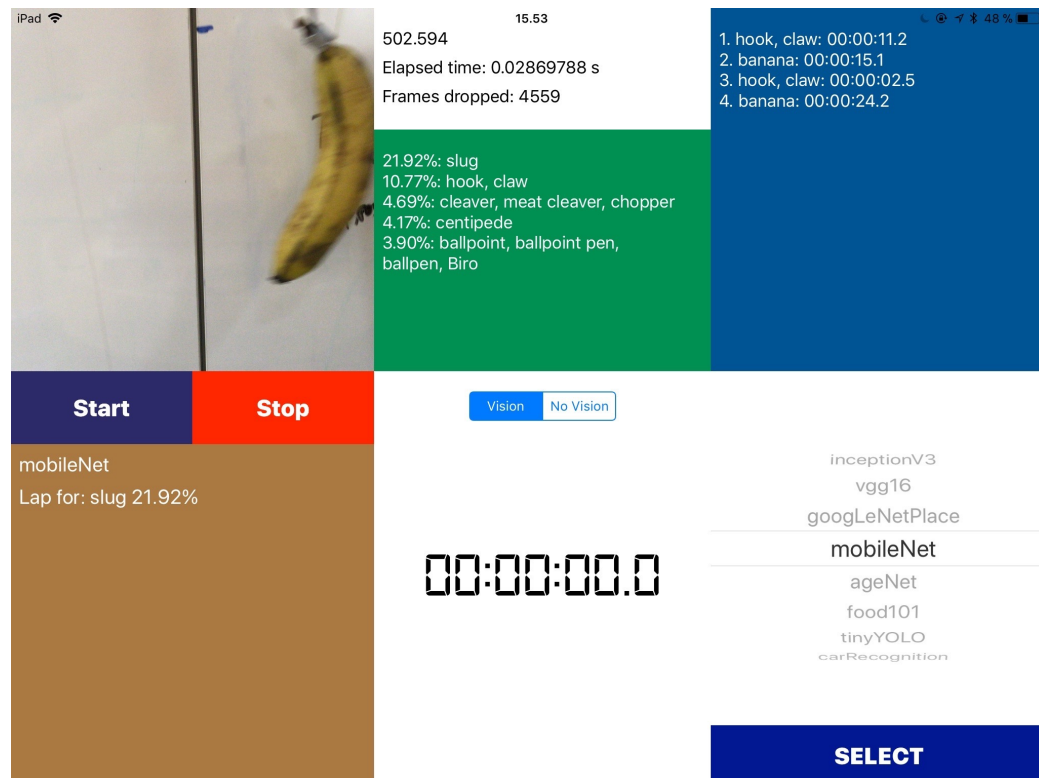


Figure 12. Screenshot of application in used.

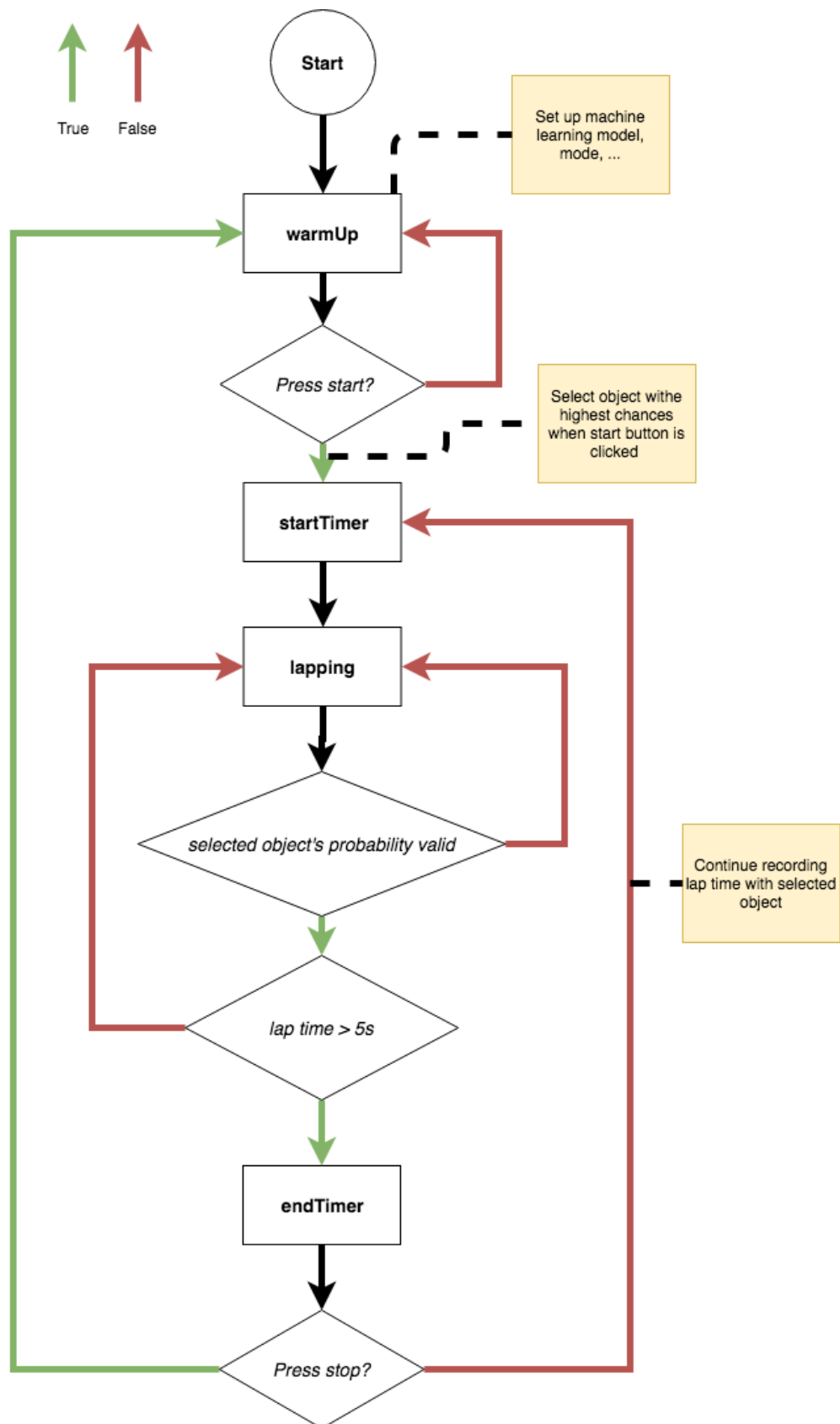


Figure 13. BananaLapTime's state machine

Figure 13 demonstrates the application's state machine with four stages: warm up, where user configures which machine learning model will be used; whether Vision will be used or not and finally, should the desired object be focused on camera frame, to give that object the highest probability on the list. The start timer, which is triggered by pressing Start button on the UI manually, goes straight to lapping state, while the timer is working. Whenever the selected object appears on the probability list with an acceptable chance (not lower than the starter chance 10%) and the record time is longer than five seconds, the state will change automatically to end timer state and finally back to the beginning of the start timer. After that it should be ready for recording another lap time of the same object. If Stop button on the UI is clicked and the clock is ticking, it will trigger the flow to stop the timer, and move the state back to warm up state.

```
enum ModelType: String {
    case inceptionV3
    case googLeNetPlace
    case mobileNet
    case vgg16
    case ageNet
    case food101
    case carRecognition
    case tinyYOLO
    case fruitResNet
    case fruitSqueezeNet
}
```

Figure 14. Enumerate to keep all available CoreML models

The second goal of the application is supporting multiple machine learning models. Although Swift, the language used for writing the application, has supported Protocol Oriented Programming (POP), each machine learning model class is generated with different property names for prediction probabilities, top prediction, or feature name. This takes more work to make all model classes to follow the same protocol. The application solution shown in the figure, create a middle layer called "ModelType" which includes all models names. Also, it uses switch loop to categorise each case to handle CoreML model process.

3.2.2 Implementation without Vision

Appendix 1 [15], shows integration without Vision mechanism: every frame captured by device's camera will be returned as a sample buffer "CMSampleBuffer". That sample buffer will be resized to suit the model's input image size and then parse into a "CVPixelBuffer" [17].

The pixel buffer image is the valid input that all CoreML models require as a prediction input. In the end of the prediction process, a "MLFeatureProvider" object will be returned. It normally holds the list of labels and their probabilities accordingly. Finally, the interface will be updated with received data.

3.2.3 Implementation with Vision

Different than non-Vision solution, every CoreML model using Vision for prediction will be required to create an asynchronous request. There will be an observer which observes the result of model prediction for each request sent. It will help the application to handle the latest frame prediction faster and synchronise the classify method implemented in non-Vision solution. Furthermore, Vision framework will process image buffers in graphics processing unit (GPU), which is dedicated to process graphics, rather than central processing unit (CPU) like non-Vision method, which is dedicated to process logic. Therefore, Vision solution gives users a better performance [15].

4 Results

4.1 Collect results

When it comes to real time object detection, there are two major parameters that would affect the outcome's quality: one frame process elapse time and the accuracy of the machine learning model (how fast can it detect the correct object). Because all machine learning models used in this project are created for demonstration purpose, none of them is well trained. This makes elapse time for detecting correct object longer (5-20s) and recognisable by human eye. The elapse time for processing one frame, on the other hand, depends on device's hardware, CPU with non-Vision implementation. Also, it can depend on GPU if Vision framework, which is normally in the range of one tenth to one hundredth of a second, is used. Therefore, a specific logging system is needed to record elapse time of one frame.

In Appendix 2 [16], logging system implementation in BananaLapTime application can be found. Logging data will be saved into comma-separated values format (csv) [13] with 3 fields: `elapse_time`, `memory_used`, and `frames_dropped` which save data of elapse time when processing one frame. The memory of the device is used at the end of each frame processing, and the number of frames dropped at the end of each frame are processing according to the logging method with two parameters: string "s" includes three values of the fields, and string "fileName" includes the log file's name. An observer of an internal value "processTime" will trigger the logging method, whenever the value is changed. The logging method will also change when the current memory used, and number of frames are dropped at that moment to be sent along with elapse time. Every recording session, which is using the same setup are the machine learning model and the Vision mode. The device will be saved in one separated log file with format "<model_name>_<timestamp>.csv".

```

1. vim banana/lpadmini4/novision/inceptionV3-538495013.csv @ Benchmark (vim)
#lapse_time,memory_used,frames_dropped
0.014892624996719,118.031,449
0.00442954165919218,117.984,449
0.00500270833435934,117.938,449
0.0248279583320254,110.375,530
0.0329202916764189,110.828,570
0.0371122499927878,110.859,590
0.0327935833338415,110.859,610
0.0393227916647447,111.109,632
0.0438630416610977,111.156,653
0.0399784583278233,111.203,675
0.0344267500040587,111.203,695
0.0364498333365191,111.188,717
0.0334037083375733,111.203,738
0.0335537916689645,111.203,759
0.0356407916697208,111.234,780
0.0342873749905266,111.203,801
0.0085887500026729,111.25,820
0.0280961666576331,111.125,838
0.0352968750084983,111.109,861
0.0334293750056531,111.109,881
0.036833999986347,111.125,902
0.0331322083366103,111.109,924
0.039776666671969,111.094,944
0.0368964999943273,111.203,965
0.034577333247053,112.906,986
0.0369965833233437,111.109,1007
0.0405993750027847,111.141,1028
0.0332219583215192,111.172,1050
0.0418812083371449,111.172,1071
0.033474458323326,111.125,1092
0.0342796666664071,111.156,1113
0.0360009166615782,111.172,1133
0.0433424166694749,111.156,1155
0.0356405000056839,111.109,1176
0.0393342916649999,111.156,1197
0.0335198333341395,112.266,1219
0.0416769166622544,111.219,1240
0.0348153749946505,111.188,1261
"banana/Ipadmini4/novision/inceptionV3-538495013.csv" 46L, 1451C

```

Figure 15. Content of file “inceptionV3-538495013.csv”

Figure 15 indicates the final result when the application is running on an iPad mini 4, trying to detect a banana, using inceptionV3, without Vision framework.

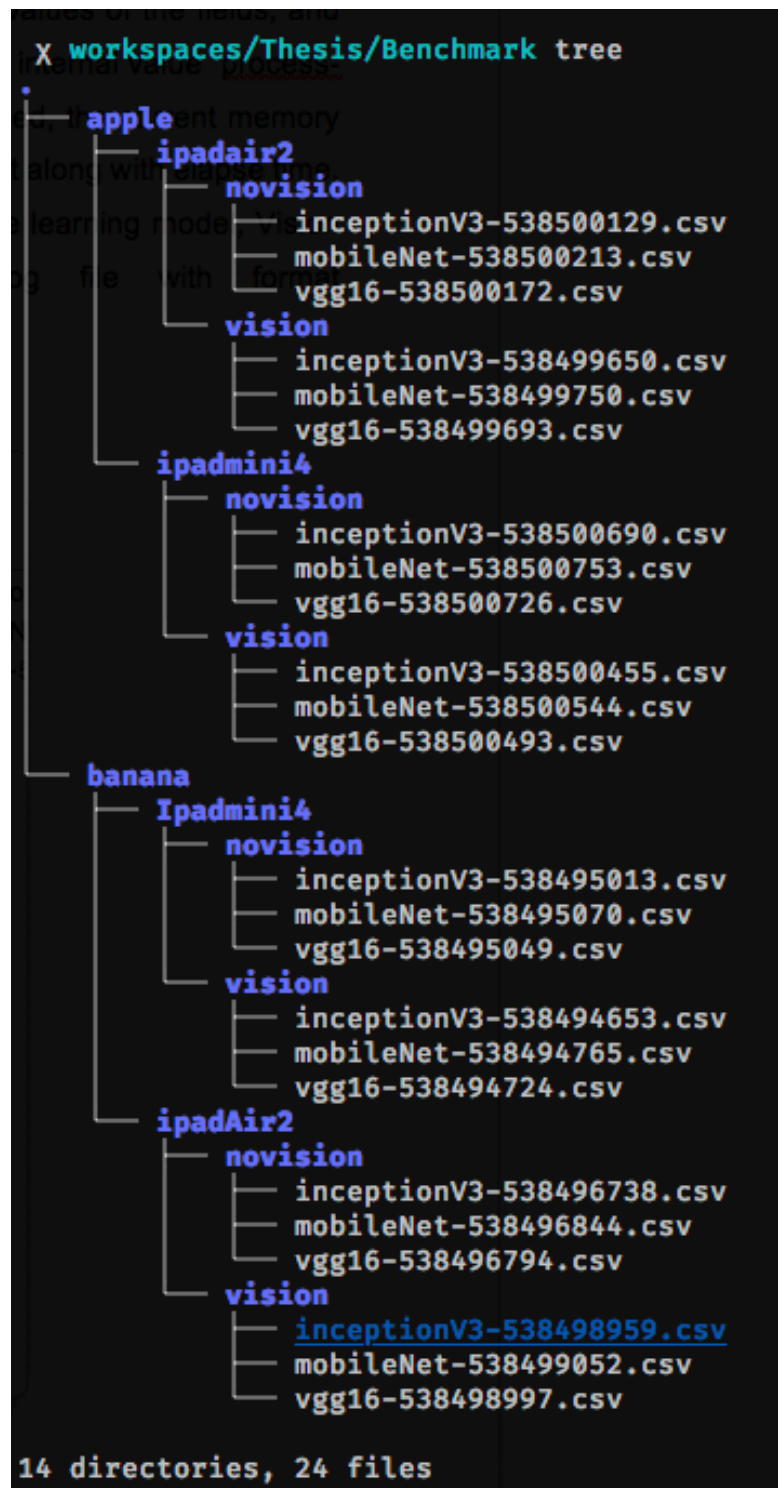


Figure 16. Tree graph of final benchmark folder

Figure 16 displays log files while testing with three models: vgg16, inceptionV3, and mobileNet with two modes: Vision and non-Vision, tested on two devices: iPad mini 4, and iPad air 2, and with two objects: an apple and a banana.

4.2 Different models benchmark

Model	non-Vision	Vision
iPad mini 4		
inception v3	0.7523052	0.032781608
vgg16	0.979963198	0.008739856
mobile net	0.145679957	0.008678492
iPad Air 2		
inception v3	0.763339933	0.026778408
vgg16	0.966867925	0.007899572
mobile net	0.127899802	0.008193072

Figure 17. Average elapse time for process one frame.

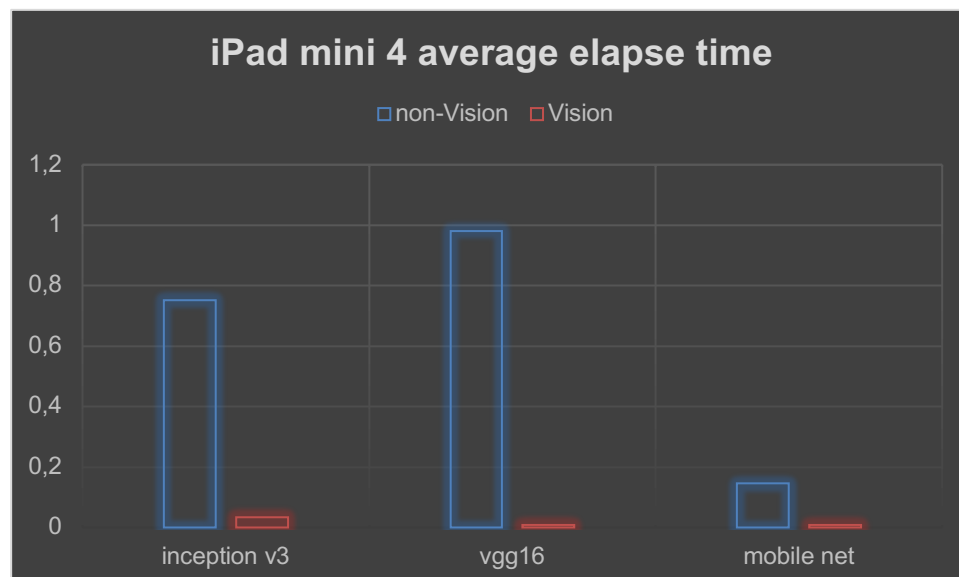


Figure 18. Bar chart of elapse time for process one frame using iPad mini 4.

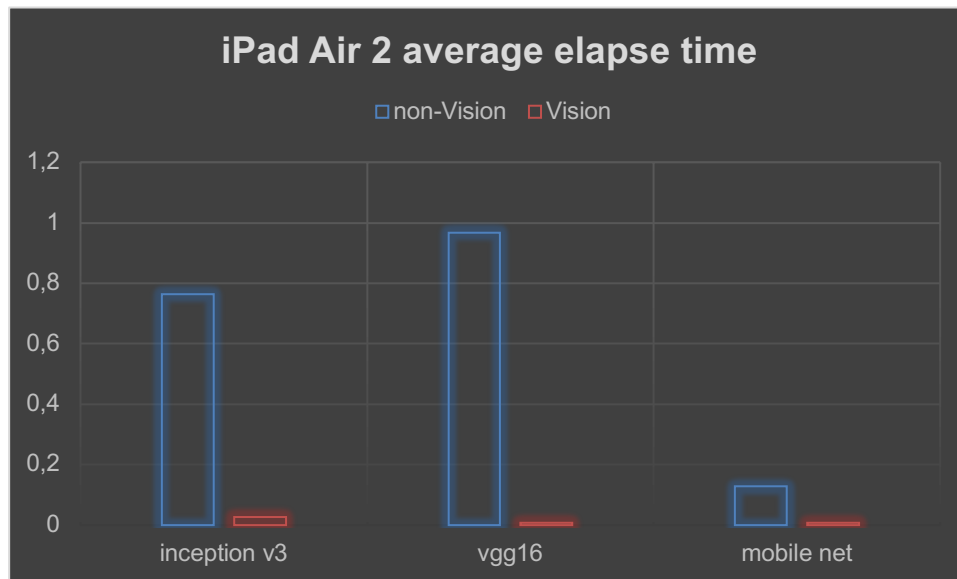


Figure 19. Bar chart of elapse time for process one frame using iPad mini 4.

Figures above have shown average elapse time for processing one frame with different machine learning models and devices. In some cases Vision was used and in some cases it was not. On paper, iPad Air 2 has a more powerful chipset, CPU, and GPU than iPad mini 4. iPad mini 4 is equipped with Apple A8 chip set, dual-core 1.5 GHz Typhoon CPU, and PowerVR GX6450 (quad-core graphics) GPU compared to Apple A8 chip set, triple-core 1.5 GHz Typhoon CPU, and PowerVR GX6450 (octa-core graphics) GPU from iPad Air 2 [14]. However, due to the small amount of data processing, there is no trace of the hardware differences.

Vgg16 model, with the largest file size (535.5MB) compared with the others with 100MB, is the most optimised model to work with GPU. However, on average, mobile net with both CPU and GPU process time is two tenths of a second, and fairly small size and better for general purposes.

4.3 Vision and non-Vision benchmark

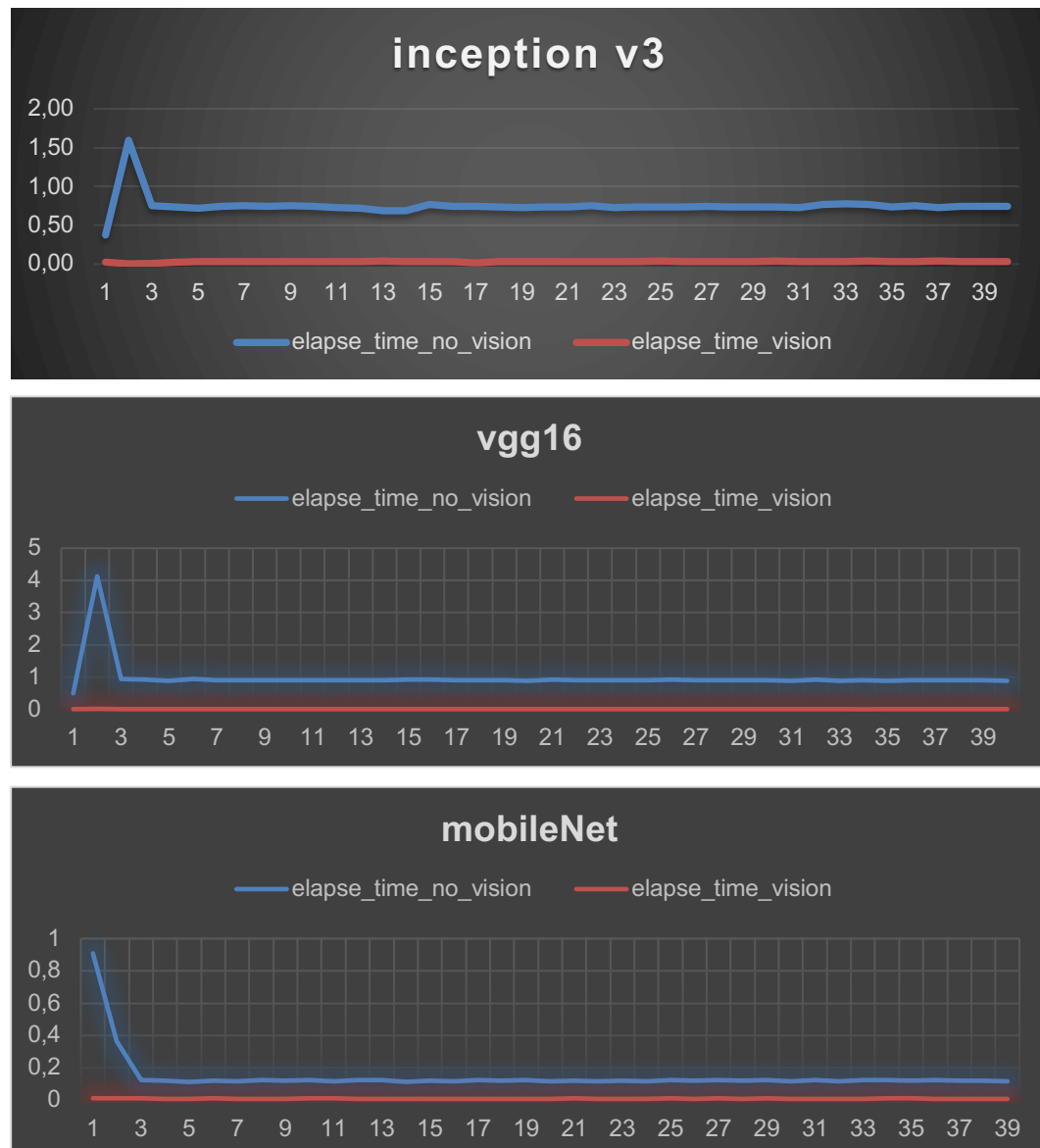


Figure 20. Line charts of elapsed time for process first 40 frames.

The performance gap appeared when the application switched to use Vision (image frames will be processed by GPU instead of CPU). The processing time has decreased around 20 times when using inception or mobile net model, and around 100 times if vgg16 model is used. Even though the gap is gigantic on numbers, all average processing times are below 1 second, which makes almost no difference while observing with human eye. The only noticeable performance is an UI jammed moment when the application switches to use non-Vision implementation from Vision implementation, which is shown as the peaks in figure 19.

5 Discussion

The application is still in early development stage, even though it can function as an automatic timer. The major problem that is blocking this project from production is the machine learning model's accuracy, which normally takes 10-20 seconds until it can detect a correct object from the start lines. The problem could get worse with high velocity objects. For now, BananaLapTime cannot work with anything that moves faster than 0.5 m/s.

Another downside is the file size of machine learning models, which is normally around 100 MB, especially with vgg16, the most accurate model, which takes 500 MB of your device's memory. Therefore, an application with CoreML support can easily go up to 200 MB or more, which become much less attractive for users.

However, machine learning algorithms are developing fast and the rise of "you only look once" (YOLOv3) in object detection field [15], and the upcoming Apple Worldwide Developers Conference (WWDC 2018) might include a breakthrough on both CoreML and Vision frameworks. There are chances that BananaLapTime application will be ready for production in autumn 2018.

6 Conclusion

Because of hardware improvement and the revolution of cloud computing, Neural Networks has resurrected and become an important part of many digital services since 2010. Currently, digital services are widely used in daily life. For instance, listening to music on Spotify, watching a movie on Netflix, everyday commuting with Whim, and renting accommodation via Airbnb, one can easily see how much neural networks has affected the services. Building deep learning applications and integrating trained models into mobile application is also getting more popular and straight forward, especially in Apple ecosystem with powerful tools such as TuriCreate, Vision, CoreML, and Hydrogen.

The first goal of the project was to build a deep learning application for real time object detection using TuriCreate with two architectures ResNet50 and SqueezeNet, and then exported the trained models. The second goal was using the trained machine learning models above, along with other existing object detection models such as VGG16, MobileNet, AgeNet, Food101, TinyYOLO, Inceptionv3, and GoogleNet, to integrate an iOS automation timer application that can detect one object and record its own the lap times. The application is using CoreML and Vision framework and it is designed to process machine learning models.

Unfortunately, all used machine learning models are unable to detect the correct object, unless it is moving slowly (less than 0.4 m/s), which makes the iOS application's main function to be record lap time, not accuracy. However, the project is still in early stages, and many parts can be improved in the future, for example: instead of using machine learning models for detection, they can be used for tracking selected objects which would enhance the final results. However, because the device's camera will have to follow the object the whole lap rather than staying still at one point, a more complex implementation will be required. Another possible improvement is using real time training with CoreML, which means the machine learning model gets trained every time the application is in used. This would result in better performance for one specific object.

The purpose of automatically recording object lap time, other than being used in races, include daily life scenarios as well. For example, the application could record a person's daily walking habits and then analyse and choose the best route for the person.

References

- 1 Michael Copeland. NVIDIA blog: What's the Different Between Artificial Intelligence, Machine Learning, and Deep Learning? [online]. <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>. Accessed 29th October 2017.
- 2 Alan M. Turing. Computing machinery and Intelligence. Mind 49: 433-460 [online]. The Imitation Game. <https://www.csee.umbc.edu/courses/471/papers/turing.pdf> Accessed 29th October 2017.
- 3 Jason B. Machine Learning: Hands-On for Developers and Technical Professionals. Canada: Wiley; 2014
- 4 Andrew N. Introduction: What is Machine Learning; 15-15
- 5 Overview of Data Science [online]. Cambridge Spark; 24 November 2016 <http://beta.cambridgespark.com/courses/jpm/01-module.html> Accessed 29th October 2017.
- 6 Andrew N. Neural Networks: Representation. Model representation I;16-16
- 7 UFLDL Tutorial Stanford: Supervised Neural Networks Multi-Layer Neural Network [online]. <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/> Accessed 29th October 2017.
- 8 Apple's documentation: CoreML framework [online] <https://developer.apple.com/documentation/coreml> Accessed 30th March 2018.
- 9 Apple's documentation: Vision framework [online] <https://developer.apple.com/documentation/vision> Accessed 30th March 2018.
- 10 Apple's Github: Turi Create project [online] <https://github.com/apple/turicreate> Accessed 30th March 2018.

- 11 Horea's Github: Fruit Image Dataset [Online] <https://github.com/Horea94/Fruit-Images-Dataset/tree/master/Training> Accessed 30th March 2018.
- 12 Andres M. Machine Learning and Optimization [online].
https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf Accessed 30th March 2018.
- 13 Edoceo: Comma Separated Values (CSV) Standard File Format [online]
<https://edoceo.com/utilitas/csv-file-format> Accessed 2nd April 2018.
- 14 GSMArena: Compare iPad mini 4 and iPad Air 2 [online] <https://www.gsmarena.com/compare.php3?idPhone1=7561&idPhone2=6742> Accessed 2nd April 2018.
- 15 Joshep R. Ali F. YOLOv3: An Incremental Improvement [online] <https://pjreddie.com/media/files/papers/YOLOv3.pdf> Accessed 2nd April 2018.
- 16 Cornell University Library [online] <https://arxiv.org/> Accessed 2nd April 2018.

List of machine learning models used

1. MobileNet
Authors: Andrew G. Menglong Z. Bo C., Dmitry K. Weijun W. Tobias W. Marco A. Hartwig A.
URL: <https://arxiv.org/abs/1704.04861v1>
2. Food101
Author: Philipp G.
URL: <http://visiir.lip6.fr/explore>
3. TinyYOLO
Authors: Joshep R. Ali F.
URL: <https://arxiv.org/abs/1612.08242>
4. VGG16
Authors: Karen S. Andrew Z. Keras Implementation: François C.
URL: <https://www.kaggle.com/keras/vgg16>
5. Inceptionv3
Authors: Christian S. Vincent V. Sergey I. Jonathon S. Zbigniew W. Keras Implementation: François C.
URL: <https://arxiv.org/abs/1512.00567>
6. GoogLeNetPlaces
Authors: B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva
URL: <http://places.csail.mit.edu/index.html>
7. AgeNet
Author: Gil L. Tal H.
URL: https://www.openu.ac.il/home/hassner/projects/cnn_agegender/

Appendices

HomeController class [17]

```
//
// ViewController.swift
// BananaLapTime
//
// Created by Do Duc on 30/11/2017.
// Copyright © 2017 Duc. All rights reserved.
//

import UIKit
import AVFoundation
import CoreML
import Vision

enum BananaState: String {
    case warmUp
    case start
    case lapping
    case end
}

final class HomeController: UIViewController {

    // MARK: - ----- IBOutlets -----
    /// ....
    @IBOutlet private weak var cameraWrapperView: UIView!
    @IBOutlet private weak var detailsLabel: UILabel!
    @IBOutlet private weak var currentObjectLabel: UILabel!
    @IBOutlet private weak var currentModelTypeLabel: UILabel!
    @IBOutlet private weak var lapTimeLabel: UILabel!
    @IBOutlet private weak var lapClockLabel: UILabel!
    @IBOutlet private weak var memoryUsageLabel: UILabel!
    @IBOutlet private weak var elapsedTimeLabel: UILabel!
    @IBOutlet private weak var framesDropLabel: UILabel!
    @IBOutlet private weak var modelSelectionPickerView: UIPickerView!
    @IBOutlet private weak var selectModelButton: UIButton!

    // MARK: - ----- Private Properties -----
    //
    private let kMinimumLapTime: TimeInterval = 3.0
    private let kLapTimerInterval: TimeInterval = 0.1 // timer only has a resolution
50ms-100ms
    private let kMemoryTimerInterval: TimeInterval = 1.0
    private let kDefaultClockText: String = "00:00:00.0"
    private var prediction: Double = 0
    private var useVision: Bool = true
    private var fileName: String = "dummy-\(Int(Date.timeIntervalSinceReference-
Date)).txt"
    private var framesDropped: Int = 0 {
        didSet {
            handlerFrameDropped()
        }
    }

    private var processTime: Double? {
        didSet {
            guard let processTime = processTime else {

```

```

        return
    }

    handlerProcessTime(processTime)
}
}
private var modelType: ModelType = .inceptionV3 {
    didSet {
        guard modelType != oldValue else {
            return
        }

        modelTypeChange()
    }
}

private var lapTimer: Timer = Timer()
private var memoryTimer: Timer = Timer()
private var startTime: Date?
private var lapRecords: [Record] = [] {
    didSet {
        guard let lapTimeLabel = lapTimeLabel else {
            return
        }

        var lapText = ""
        for (index, lapRecord) in lapRecords.enumerated() {
            lapText = lapText + "\((index+1). \((lapRecord.name): \((lapRecord.lap-
Time.clockFormat)\n"
        }

        lapTimeLabel.text = lapText
    }
}

private var selectedObject: (name: String, prediction: Double) = (name: "Unknown",
prediction: 0) {
    didSet {
        guard selectedObject.name != oldValue.name else {
            return
        }

        guard let currentObjectLabel = currentObjectLabel else {
            return
        }

        currentObjectLabel.text = "Lap for: \((selectedObject.name) \((selectedOb-
ject.prediction.percentage)%"
    }
}

private var state = BananaState.warmUp {
    didSet {
        print("[State]: \((oldValue) ==> \((state)")
        guard state != oldValue else {
            return
        }

        stateDidChange()
    }
}

var visionRequest: VNCoreMLRequest?

lazy private var inception = Inceptionv3()
lazy private var googleNet = GoogLeNetPlaces()

```

```

lazy private var vgg16 = VGG16()
lazy private var mobileNet = MobileNet()
lazy private var ageNet = AgeNet()
lazy private var food101 = Food101()
lazy private var tinyYOLO = TinyYOLO()
lazy private var carRecognition = CarRecognition()
lazy private var yolo = YOLO()
lazy private var fruitResNet = FruitClassifierResNet()
lazy private var fruitSqueezeNet = FruitClassifierSqueezeNet()

lazy private var models: [ModelType] = {
    return [ModelType.inceptionV3, ModelType.vgg16, ModelType.googleNetPlace, Model-
Type.mobileNet, ModelType.ageNet, ModelType.food101, ModelType.tinyYOLO, Model-
Type.carRecognition, ModelType.fruitResNet, ModelType.fruitSqueezeNet]
}()

// Camera related variable
lazy private var captureSession: AVCaptureSession? = {
    let session = AVCaptureSession()
    session.sessionPreset = .high
    return session
}()

private var videoPreviewLayer: AVCaptureVideoPreviewLayer? {
    guard let captureSession = self.captureSession else {
        return nil
    }
    let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)

    let screenWidth = UIScreen.main.bounds.size.width
    previewLayer.bounds = CGRect(x: 0, y: 0, width: screenWidth, height: screen-
Width)
    previewLayer.position = CGPoint.zero
    return previewLayer
}

// MARK: - ----- UIViewController life cycle -----
----
// loadView > viewDidLoad > viewWillAppear > viewWillLayoutSubviews > viewDidLay-
outSubviews > viewDidAppear
override func viewDidLoad() {
    super.viewDidLoad()
    setUpBoundingBoxes()
    setUpCamera()
    setUpMemoryDisplay()
    modelTypeChange()

    // Initialise state
    stateDidChange()
}

deinit {
    lapTimer.invalidate()
    memoryTimer.invalidate()
}

// MARK: - ----- Route Methods -----
// @IBActions, prepare(...), ...
@IBAction func startButtonClicked(_ sender: Any) {
    state = .start
}

@IBAction func stopButtonClicked(_ sender: Any) {
    state = .end
}

@IBAction func selectModelButtonClicked(_ sender: Any) {

```

```

        let selectedIndex = modelSelectionPickerView.selectedRow(inComponent: 0)
        guard selectedIndex < models.count else {
            return
        }

        modelType = models[selectedIndex]
    }

    @IBAction func visionSegmentControlValueChanged(_ sender: Any) {
        useVision.toggle()
    }

    // MARK: - ----- Public Methods -----
    //

    func classifierVGG16(image: CVPixelBuffer) {
        guard let predictedResult = try? vgg16.prediction(image: image) else {
            return
        }

        handlerPredictions(predictedResult.classLabelProbs)
    }

    func classifierGooglePlace(image: CVPixelBuffer) {
        guard let predictedResult = try? googleNet.prediction(sceneImage: image) else {
            return
        }

        handlerPredictions(predictedResult.sceneLabelProbs)
    }

    func classifierInception(image: CVPixelBuffer) {
        guard let predictedResult = try? inception.prediction(image: image) else {
            return
        }

        handlerPredictions(predictedResult.classLabelProbs)
    }

    func classifierMobileNet(image: CVPixelBuffer) {
        guard let predictedResult = try? mobileNet.prediction(image: image) else {
            return
        }

        handlerPredictions(predictedResult.classLabelProbs)
    }

    func classifierAgeNet(image: CVPixelBuffer) {
        guard let predictedResult = try? ageNet.prediction(data: image) else {
            return
        }

        handlerPredictions(predictedResult.prob)
    }

    func classifierFood101(image: CVPixelBuffer) {
        guard let predictedResult = try? food101.prediction(image: image) else {
            return
        }

        handlerPredictions(predictedResult.foodConfidence)
    }

    func classifierFruitResNet(image: CVPixelBuffer) {
        guard let predictedResult = try? fruitResNet.prediction(image: image) else {

```

```

        return
    }

    handlerPredictions(predictedResult.fruitProbability)
}

func classifierFruitSqueezeNet(image: CVPixelBuffer) {
    guard let predictedResult = try? fruitSqueezeNet.prediction(image: image) else {
        return
    }

    handlerPredictions(predictedResult.fruitProbability)
}

var boundingBoxes = [BoundingBox]()
var colors: [UIColor] = []

func classifierTinyYOLO(image: CVPixelBuffer) {
    guard let predictedResult = try? yolo.predict(image: image) else {
        return
    }

    runOnMainThread { [weak self] in
        guard let strongSelf = self else {
            return
        }

        strongSelf.show(predictions: predictedResult)
        //gridImageView.image = predictedResult.grid.image(offset: 0.0, scale: 416)
    }

    // TODO: Handler grid
    //handlerData(predictedResult: predictedResult.featureNames)
}

func show(predictions: [YOLO.Prediction]) {
    for i in 0..

```



```

    }
  }
}

func classifierCarRecognition(image: CVPixelBuffer) {
  guard let predictedResult = try? carRecognition.prediction(data: image) else {
    return
  }

  handlerPredictions(predictedResult.prob)
}

@objc func updateTimer() {
  guard let startTime = startTime else {
    return
  }

  lapClockLabel.text = startTime.timeIntervalSinceNow.clockFormat
}

// MARK: - ----- Private Methods -----
// fileprivate, private
private func setUpBoundingBoxes() {
  for _ in 0..

```

```

dataOutput.alwaysDiscardsLateVideoFrames = true
captureSession.addOutput(dataOutput)
captureSession.commitConfiguration()

videoPreviewLayer.videoGravity = AVLayerVideoGravity.resizeAspectFill
videoPreviewLayer.frame = cameraWrapperView.bounds
cameraWrapperView.layer.addSublayer(videoPreviewLayer)

// Add the bounding box layers to the UI, on top of the video preview.
/*for box in self.boundingBoxes {
    box.addToLayer(cameraWrapperView.layer)
}*/

if let captureConnection = videoPreviewLayer.connection, captureConnection.is-
VideoOrientationSupported {
    captureConnection.videoOrientation = AVCaptureVideoOrientation(rawValue:
UIApplication.shared.statusBarOrientation.rawValue) ?? .landscapeLeft
}

let queue = DispatchQueue(label: "com.banana.videoQueue")
dataOutput.setSampleBufferDelegate(self, queue: queue)
captureSession.startRunning()
}

private func setupVision(with coreMLModel: MLModel) {
    guard let visionModel = try? VNCoreMLModel(for: coreMLModel) else {
        return
    }

    visionRequest = VNCoreMLRequest(model: visionModel, completionHandler: { [weak
self] request, _ in
        let processStartTime = CACurrentMediaTime()

        defer {
            let processEndTime = CACurrentMediaTime()
            self?.processTime = processEndTime - processStartTime
        }

        guard let strongSelf = self else {
            return
        }

        if let observations = request.results as? [VNClassificationObservation] {
            strongSelf.handlerPredictions(observations)
            return
        }

        /*if let observation = request.results?.first as? VNCoreMLFeatureValueObser-
vation, let mlMultiArray = observation.featureValue.multiArrayValue {
            let boundingBoxes = strongSelf.yolo.computeBoundingBoxes(features:
mlMultiArray)
            runOnMainThread {
                strongSelf.show(predictions: boundingBoxes)
            }
        }*/

    })

    visionRequest?.imageCropAndScaleOption = .scaleFill
}

private func modelTypeChange() {
    currentModelTypeLabel.text = modelType.rawValue

```

```

        fileName = "\\(modelType.rawValue)-\\(Int(Date.timeIntervalSinceReference-
Date)).txt"
        var coreMLModel: MLModel!

        switch modelType {
        case .inceptionV3:
            coreMLModel = inception.model
        case .googleNetPlace:
            coreMLModel = googleNet.model
        case .mobileNet:
            coreMLModel = mobileNet.model
        case .vgg16:
            coreMLModel = vgg16.model
        case .ageNet:
            coreMLModel = ageNet.model
        case .carRecognition:
            coreMLModel = carRecognition.model
        case .food101:
            coreMLModel = food101.model
        case .tinyYOLO:
            coreMLModel = tinyYOLO.model
        case .fruitResNet:
            coreMLModel = fruitResNet.model
        case .fruitSqueezeNet:
            coreMLModel = fruitSqueezeNet.model
        }

        setupVision(with: coreMLModel)
    }

    private func stateDidChange() {
        switch state {
        case .warmUp:
            warmUpState()
        case .start:
            startState()
        case .lapping:
            lappingState()
        case .end:
            endState()
        }
    }

    private func warmUpState() {
        // Lap time clock stay at 0
        lapClockLabel.text = kDefaultClockText
        selectModelButton.isEnabled = true
    }

    private func startState() {
        // Start lapTimer
        startLapTimer()
        selectModelButton.isEnabled = false
    }

    private func lappingState() {
    }

    private func endState() {
        // Stop lapTimer
        stopLapTimer()
    }

    private func startLapTimer() {
        startTime = Date()
    }

```

```

        lapTimer = Timer.scheduledTimer(timeInterval: kLapTimerInterval, target: self,
selector: #selector(updateTimer), userInfo: nil, repeats: true)
        state = .lapping
    }

    private func stopLapTimer() {
        if let startTime = startTime {
            let newRecord = Record(name: selectedObject.name, lapTime: startTime.timeIn-
tervalSinceNow)
            lapRecords.append(newRecord)
            /*let alert = UIAlertController(title: "New record added", message: "Lap
time for \(selectedObject.name) (\(selectedObject.prediction.percentage)%): \(start-
Time.timeIntervalSinceNow.clockFormat)", preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "OK", style: .default, handler: { _
in

                )))
            self.present(alert, animated: true, completion: nil)*/
        }

        startTime = nil
        lapTimer.invalidate()
        state = .warmUp
    }

    private func classifierWithVision(sampleBuffer: CMSampleBuffer, model: ModelType) {
        guard let inputData = sampleBuffer.image(newWidth: model.imageSize)?.cgImage
    else {
        return
    }

    guard let visionRequest = visionRequest else {
        return
    }

    let handler = VNImageRequestHandler(cgImage: inputData, options: [:])
    try? handler.perform([visionRequest])
    }

    private func classifierWithoutVision(sampleBuffer: CMSampleBuffer, model: ModelType)
    {
        let processStartTime = CACurrentMediaTime()

        defer {
            let processEndTime = CACurrentMediaTime()
            processTime = processEndTime - processStartTime
        }

        guard let buffer = sampleBuffer.image(newWidth: model.imageSize)?.cvBuffer()
    else {
        return
    }

    switch model {
    case .inceptionV3:
        classifierInception(image: buffer)
    case .googleNetPlace:
        classifierGooglePlace(image: buffer)
    case .mobileNet:
        classifierMobileNet(image: buffer)
    case .vgg16:
        classifierVGG16(image: buffer)
    case .ageNet:
        classifierAgeNet(image: buffer)
    case .carRecognition:
        classifierCarRecognition(image: buffer)
    case .food101:

```

```

        classifierFood101(image: buffer)
    case .tinyYOLO:
        classifierTinyYOLO(image: buffer)
    case .fruitResNet:
        classifierFruitResNet(image: buffer)
    case .fruitSqueezeNet:
        classifierFruitSqueezeNet(image: buffer)
    }
}

private func handlerPredictions(_ predictedResults: [String: Double]) {

    runOnMainThread { [weak self] in
        guard let strongSelf = self else {
            return
        }

        let topFive = predictedResults.sorted(by: { $0.value > $1.value }).prefix(5)
        strongSelf.detailsLabel.text = topFive.display

        guard let topObject = topFive.first else {
            return
        }

        // Warmup handler
        if strongSelf.state == .warmUp {
            strongSelf.selectedObject = (name: topObject.key, prediction: topOb-
ject.value)
            return
        }

        // Lapping handler
        if strongSelf.state == .lapping {
            guard let startTime = strongSelf.startTime, abs(startTime.timeInter-
valSinceNow) > strongSelf.kMinimumLaptime else {
                return
            }

            for obj in topFive where obj.key == strongSelf.selectedObject.name &&
obj.value.acceptablePrediction(with: strongSelf.selectedObject.prediction) {
                strongSelf.state = .end
            }

            return
        }
    }

    private func handlerPredictions(_ predictedResults: [VNClassificationObservation]) {
        let predictions = predictedResults.reduce(into: [String: Double]()) { dict, ob-
servation in
            dict[observation.identifier] = Double(observation.confidence)
        }

        handlerPredictions(predictions)
    }

    private func handlerProcessTime(_ processTime: Double) {
        runOnMainThread { [weak self] in
            guard let strongSelf = self else {
                return
            }

            strongSelf.elapsedTimeLabel.text = String(format: "Elapsed time: %.8f s",
processTime)

```

```

        logging("\(processTime), \(getMemoryUsage()), \(strongSelf.framesDropped)",
        fileName: strongSelf.fileName)
        //print("[Process]: \(processTime) seconds")
    }

    private func handlerFrameDropped() {
        runOnMainThread { [weak self] in
            guard let strongSelf = self else {
                return
            }

            strongSelf.framesDropLabel.text = "Frames dropped:
            \(strongSelf.framesDropped)"
            //print("[Frames drop]: \(self.framesDropped)")
        }
    }
}

extension HomeViewController: AVCaptureVideoDataOutputSampleBufferDelegate {
    func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSam-
    pleBuffer, from connection: AVCaptureConnection) {

        guard useVision else {
            classifierWithoutVision(sampleBuffer: sampleBuffer, model: modelType)
            return
        }

        classifierWithVision(sampleBuffer: sampleBuffer, model: modelType)
    }

    func captureOutput(_ output: AVCaptureOutput, didDrop sampleBuffer: CMSampleBuffer,
    from connection: AVCaptureConnection) {
        framesDropped += 1
    }
}

extension HomeViewController: UIPickerViewDelegate {
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent com-
    ponent: Int) -> String? {
        guard row < models.count else {
            return nil
        }

        return models[row].rawValue
    }
}

extension HomeViewController: UIPickerViewDataSource {
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int)
    -> Int {
        return models.count
    }
}

```

Logging extension [16]

```

func logging(_ s: String, fileName: String) {
    let documents = getDocumentsDirectory()
    let pathURL = documents.appendingPathComponent(fileName)

    /// Create a new log file and insert first header row if file is not exist yet
    /// elapse_time,memory_used,frames_dropped
    if !FileManager.default.fileExists(atPath: pathURL.path) {
        FileManager.default.createFile(atPath: pathURL.path, contents: nil, attributes:
nil)
        do {
            let fileHandle = try FileHandle(forWritingTo: pathURL)
            fileHandle.write("elapse_time,memory_used,frames_dropped\n".data(using:
String.Encoding.utf8)!)
        } catch let error as NSError {
            print("[LogError]: Ooops! Something went while creating log file file wrong
\n[LogError]: \(error)")
        }
    }

    /// Insert log data to the bottom of log file
    do {
        let fileHandle = try FileHandle(forWritingTo: pathURL)
        fileHandle.seekToEndOfFile()

        fileHandle.write("\(s)\n".data(using: String.Encoding.utf8)!)
    } catch let error as NSError {
        print("[LogError] Ooops! Something went wrong while adding content \n[LogError]:
\((error)")
    }
}

func getDocumentsDirectory() -> URL {
    let paths = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
    return paths[0]
}

```

Convert to CMSampleBuffer to CVPixelBuffer [17]

```

extension CMSampleBuffer {

    func image(newWidth: CGFloat) -> UIImage? {
        guard let buffer = CMSampleBufferGetImageBuffer(self) else {
            return nil
        }

        let ciImage = CIImage(cvPixelBuffer: buffer)
        let image = UIImage(ciImage: ciImage)
        return resize(image: image, newWidth: newWidth)
    }

    func resize(image: UIImage, ratio: CGFloat) -> UIImage? {
        let newSize: CGSize = CGSize(width: image.size.width * ratio,
height: image.size.height * ratio)
        return resize(image: image, newSize: newSize)
    }

    func resize(image: UIImage, newWidth: CGFloat) -> UIImage? {
        let newSize: CGSize = CGSize(width: newWidth, height: new-
Width)
        return resize(image: image, newSize: newSize)
    }

    func resize(image: UIImage, newSize: CGSize) -> UIImage? {
        let rect = CGRect(x: 0, y: 0, width: newSize.width, height:
newSize.height)

        UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
        image.draw(in: rect)
        let newImage = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()

        return newImage
    }

}

extension UIImage {
    func cvBuffer() -> CVPixelBuffer? {
        let attrs = [kCVPixelBufferCGImageCompatibilityKey: kCFBool-
eanTrue, kCVPixelBufferCGBitmapContextCompatibilityKey: kCFBooleanT-
rue] as CFDictionary
        var pixelBuffer: CVPixelBuffer?
        let status = CVPixelBufferCreate(kCFAllocatorDefault,
Int(size.width), Int(size.height), kCVPixelFormatType_32ARGB, attrs,
&pixelBuffer)
        guard status == kCVReturnSuccess else {
            return nil
        }
    }
}

```



```

    }

    CVPixelBufferLockBaseAddress(pixelBuffer!, CVPixelBufferLock-
Flags(rawValue: 0))
    let pixelData = CVPixelBufferGetBaseAddress(pixelBuffer!)

    let rgbColorSpace = CGColorSpaceCreateDeviceRGB()
    let context = CGContext(data: pixelData, width:
Int(size.width), height: Int(size.height), bitsPerComponent: 8,
bytesPerRow: CVPixelBufferGetBytesPerRow(pixelBuffer!), space: rgbCol-
orSpace, bitmapInfo: CGImageAlphaInfo.noneSkipFirst.rawValue)

    context?.translateBy(x: 0, y: size.height)
    context?.scaleBy(x: 1.0, y: -1.0)

    UIGraphicsPushContext(context!)
    draw(in: CGRect(x: 0, y: 0, width: size.width, height:
size.height))
    UIGraphicsPopContext()
    CVPixelBufferUnlockBaseAddress(pixelBuffer!, CVPixelBuffer-
LockFlags(rawValue: 0))

    return pixelBuffer
}
}

```