Identification and Classification in Food Images

Tom Barrett

February 12, 2018

Abstract

Final Year Project focused on identification and classification of food images. A background of previous work done on the subject is outlined, followed by experiments in creating neural networks to perform classifications and segmentations. These experiments will be applied to food images and evaluated.

Contents

Chapter 1

Introduction

1.1 Overview

This project explores the use of identification and classification of food images for use in a calorie measurement android application. Food calorie consumption is a huge problem in the modern world. Over 25% of the population in Ireland is obese and this figure is likely to rise over the coming years. A mobile application that could help keep track of a user's calorie intake by taking pictures of their meals would be a great help. The area of Machine Vision is a very difficult topic to address as it is a very hard task for computers to undertake. We, as humans, take vision for granted as we can soon see, from the study of Machine Vision, that there are many difficult steps that have to be made for full identification and classification of an image.

When looking into calorie measurement using an image, there are three questions that have to be answered:

- Where are the Regions of Interest (ROI) in this food image?
- What food types are in these ROI's?
- What is the portion size of each food type?

Figure 1.1: Pre-segmented Image

Figure 1.2: Segmented Image

In this project, the main focus will be on the first two questions, 'Where are the Regions of Interest (ROI) in this food image?' and 'What food types are in these ROI's'. The first step is normally achieved through image segmentation. Image segmentation is the process in which you divide an image into multiple segments as per Figure 1.2 and 1.1.

Many researchers in various machine vision labs have attempted to solve this problem using different methodologies. There has been promising results from some papers but these are mostly under highly constrained circumstances. When mixed foods are introduced to the problem, many of the methods fail. Convolutional Neural Networks (CNN) have had very promising results in the field of image classification in the recent years but to get to the classification step, image segmentation is first needed, otherwise known as image identifi-

cation. Where CNN's have been shown to work best is in the area of face detection.

Extensive research on many different methods of image segmentation has been carried out but it seems that CNN's have had the best results for multiple objects in one image and therefore, this method will be applied to for many foods in an image. This is because we will rarely want to classify an image that only has one food item in it. Therefore, the one shot approach may not be the most successful by which we build a classifier that takes in an image and gives back the most likely food in that image. In contrast to this, there could be an element of sub-sampling of the image to rectify this.

The system proposed to solve the problem statement would be able to integrate with an Android mobile phone application. The idea is, that when a user is about to eat their meal, they can simply take a picture of their meal for computation. From here, the application would take the image, find the objects (ROI) in the image and take note of them. Concurrently, the application would attempt to classify each object detected. Once this is done, the size of each food type would be measured and through this an overall calorie count would be displayed for the user. This could be logged for user metrics. The full system may not be possible to implement due to time constraints so therefore, classification will be the furthest step looked in to.

1.2 Objectives

Primary Objectives

Use Convolutional Neural Networks for Food Image Classification

There has been a large paradigm shift in food image recognition in recent years to using convolutional neural networks. This paradigm will be used to answer the problem statement.

Tune the CNN, replicating previous work

There are many different approaches to food identification and classification, many of which we will see in Chapter 2. An approach will be selected that has shown promising results in the past and replicate them. In addition to this, there are many different network architectures and parameters that can be adjusted when building CNNs and I hope to tune these in order to get the best outcome possible.

Develop a mobile application to leverage the CNN

Another objective for this project is to develop an application that can be used for dietary assessment. This application would be able to take an image and then identify and classify the foods within the image. Size estimation will not be explored for this project.

Secondary Objectives

Understanding of Convolutional Neural Networks

In the project, Convolutional Neural Networks (CNN's) will be used for object identification in Food Images. I will be using a machine learning library for this due to time constraints but it is a key objective to develop a deep understanding of CNN's as they are quite pivotal in the current Machine Vision Industry and bio-inspired systems are very interesting.

Learn about different image identification and classification techniques

Although, CNN's will be used for implementation, other methods of identification and classification will not be ignored. It is very important to learn about other methods as different methods are better suited for some situations and it would be best to know about these methods due to the inevitability of their use.

1.3 Methodology

The following methodology were adapter for this project:

Define the research question

The first step to this project was to define the research question. The general area of 'Food Identification and Classification' was known at conception but the scope of this is too broad for an FYP. Therefore it was decided that the research question would be to look at the food identification and classification aspect.

Literature Review

Once the research question was defined, finding related work was the next milestone. There are many attempts at Food Image Classification and these were not difficult to find, using Google Scholar, but many of these papers glossed over the segmentation aspect and relied on third parties for this step. Because of this, quite a few references to different papers had to be followed that focused solely on image segmentation and classification.

Explore different image identification methods

Various image identification methods were collected from the literature review that was carried out, so there were many options to evaluate. Convolutional Neural Networks were the clear choice due to recent popularity, so more traditional methods of identification using colour and texture was not so strenuously explored.

Select an image identification method

Research technologies and develop skills in these technologies

As Convolutional Neural Network (CNN) are used in this project, many resources have been leveraged to enrich understanding of the process. Tensorflow is the main resource utilised in creating a CNN so on line tutorials for this technology were greatly beneficial. A Deep Learning Course on Udacity was also used to enhance understanding and skills.

Build a prototype of the application

Compare and analyse results to other implementations

1.4 Overview of Report

This report is broken down into various main headings:

Introduction

This section is to give on overview of what this project is about, how the project will be approached and why it being carried out.

Background

Some information on the background of the subject that is focused on will be outlined here.

Experiments

In this part of the report various experiments using tensorflow will be carried out.

Empirical Studies with Food Images

This section will analyse the results obtained and compare them against various metrics and other implementations.

Discussion and Conclusion

In this section, results from the empirical studies will be dicussed and a conclusion will be outlined.

1.5 Motivation

I find the topic of Computer Vision a very interesting one. It excites me, to be able to 'teach' a machine how to see as we do. For this reason, I really wanted to learn about Neural Networks and this was a large motivator for this project.

Once I had a topic that I wanted to research, I needed a focus or problem statement for this research. I find that it is much more rewarding to work on something that positively impacts both myself and other people so I decided that I wanted to research something that fit this requirement.

Food calorie consumption is a very big problem in the modern world. Over 25 percent of the population in Ireland are obese. A mobile application that could help keep track of a user's calorie intake by taking a picture of their meals would be a big help to combat this problem. This problem statement works very well for me because of it's application use and because of its complexity. Identifying and recognising food is much more difficult than say recognising faces as it has no uniform shape. Therefore, this problem would also be very beneficial to developing skills in the computer vision area.

I would like to develop these real world skills so that I can partake in Computer Vision projects in industry or to do further research in academia. This is because machine learning has really taken off in the last few years and is used by many in industry. While machine learning as a whole has become very popular, computer vision is probably the most prominent that has come out of it. Face detection, for one, is being researched extensively for use in personalised advertising and also secure access to devices and systems.

Chapter 2

Background

2.1 Introduction to Machine Learning

In Mitchell [15], Machine Learning is defined as "the question of how to construct computer programs that automatically improve with experience". Machine Learning has blossomed in recent years with applications across multiple domains using vastly different paradigms and technologies.

There are many ways in which Machine Learning can be used in the modern world, many of which are being utilised to great affect. Some of these applications, are image recognition, natural language processing, medical diagnosis and many more. There may be fear that Machine Learning will start to take away many jobs from humans but this may not be the case. There are many practical uses such as security and safety that could be leveraged such as face detection for security in airports or autonomous cars.

One of the most exciting avenues in Machine Learning, in my opinion, is Computer Vision. Computer Vision is the process of extracting high-dimensional data from an image to produce useful information, which in terms of classification usually results in labelling. It can be used in many areas to improve our lives. As mentioned earlier, autonomous cars are only possible when a machine can determine what objects are around it. Computer Vision can allow a machine to recognise skin diseases in an image. The applications are nearly

limitless and that is without taking into account other uses.

2.2 Neural Computing

The main area of my focus for this project is in Artificial Neural Networks (ANN). This is because I have researched extensively into ConvolutionalNeural Networks which are based on ANN's.

Artificial Neural Networks

An Artificial Neural Network is a bio-inspired system that is used to model the human brain in how it learns from experience. The ANN uses this model to build a very complex web of connected units called artificial neurons. These nuerons are connected by certain weights which determines the processing capacity of the network and these weights are created by learning a dataset.(Malachy) An ANN has a set of inputs that take in a value, sometimes from network outputs and produce a single result or classification. While an ANN is bio-inspired from the human brain, there are many elements of the brain that are not present in ANN and many new elements in ANN that are not modelled from the human brain.

Before I can talk about Convolution Neural Networks which are vital the image processing, I will have to talk about the perceptron learning algorithm, the multi layer percepton, and backpropogation.

Perceptron Learning - Artificial Neuron

In our Artificial Neural Network a Perecptron is an Artificial Neuron. It is called an Artifical Neuron because it is a bio-inspired neuron which models a neuron in the human brain in terms of inputs and output.

In Perceptron learning, we can take two inputs which are put towards an activation function with a bias attached as seen in 2.1. These inputs are multiplied by the weights that connect the input to the activation function and

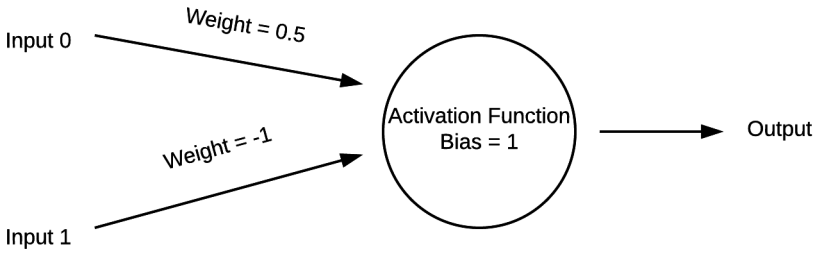


Figure 2.1: Perceptron

depending on the result, the activation function may fire an output. These inputs are either 1 or -1.

The Perceptron Training Rule is the means by which weights are selected to produce the correct output during training. As in Mitchell [15], a common way to train a perceptron is t start with random weights and change them during training as per the training rule. This rule follows the formula in Figure 2.3 , where xi is the input and 2.4 is valid. In 2.4, "t is the target output for the current training example, o is the output generated by the perceptron, and n is the positive constant called the learning rate" Mitchell [15]. This Perceptron Training Rule assumes that there are two sets of instances, a positive and negative set, and that they are linearly separable, as in Figure 2.2.

A perception is trained using supervised learning. When the perceptron classifies a results, it is told if it is correct or not. If the result is incorrect, weights are changed in value so that this error can be reduced Luger [13].

The one major problem with perceptron learning and that is that it can't solve the problem if there is not a clear linear separation between the classes. There is a way in which we can attempt to solve this, through the delta rule. The delta rule utilizes gradient descent to find the best weight for the training samples Mitchell [15]. We will discuss gradient descent in the next section.

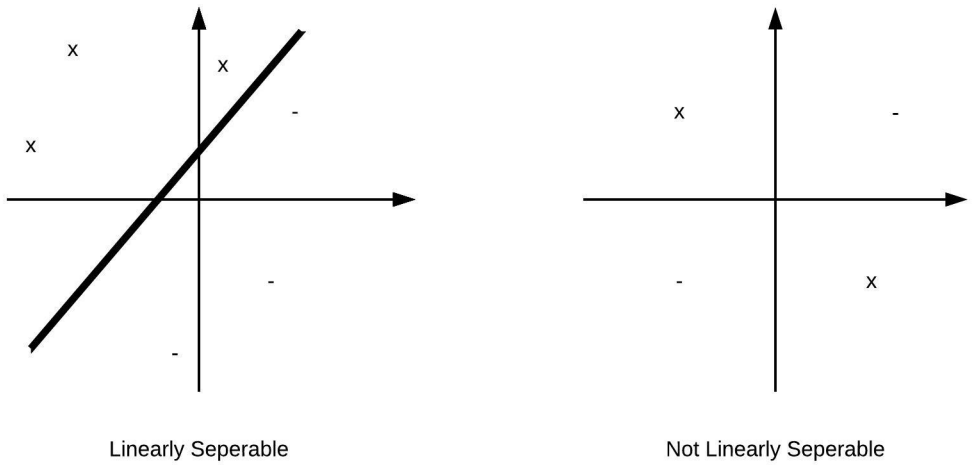


Figure 2.2: Linearly Separable, adapted from Mitchell [15]

$$w_i \leftarrow w_i + \Delta w_i$$

Figure 2.3: The Perceptron Training Rule which changes weights, sourced from Mitchell [15]

$$\Delta w_i = \eta(t - o)x_i$$

Figure 2.4: The Perceptron Training Rule condition, sourced from Mitchell [15]

Multi Layered Perceptron

Multi Layer Perceptrons (MLP) are made up of multiple layers of perceptrons connected together. Firstly, we have an input layer, followed by one of more hidden layers and then finally an output layer. Any Neural Network with more than three hidden layers is categorised as a deep layer.

The input layer of your network consists of the data you feed into the network in order to classify it. The input layer passes this data to a hidden layer whose purpose is to transform this data into something that the output layer can understand. The output layer normally consists of a class prediction.

Multi Layer Perceptrons are a class of feed forward Artificial Neural Networks. These means that the output of each perceptron feeds into an input in the next layer of the network.

There is one large problem with MLP's and this is why Convolutional Neural Networks (CNN) were created. If you attempting to classify images with an MLP then each pixel in that image would have to be a separate input. This created a massive amount of neurons through all the layers and this isn't feasible. CNN's solve this problem which we will discuss later.

Gradient Descent

Gradient Descent is an algorithm used to find the optimal weights to produce the smallest prediction error. It is used to overcome problems of non linearly separable classes. Gradient descent search selects a random weight value and then modifies it gradually to minimize the error. "At each step, the weight vector is altered in the direction that produces the steepest descent along the surface" Mitchell [15]. This step is iterated until the lowest value is met.

One problem with Gradient Descent is that if we look at 2.6, we may never get to the optimal point, point B. This is because we will find point A without too many problems but when the weights change we will get too high a slope of error and therefore will never reach point B.

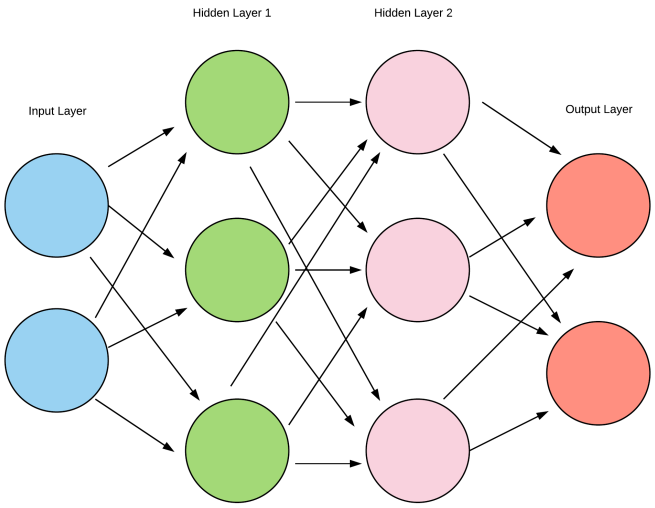


Figure 2.5: Multi Layer Perceptron

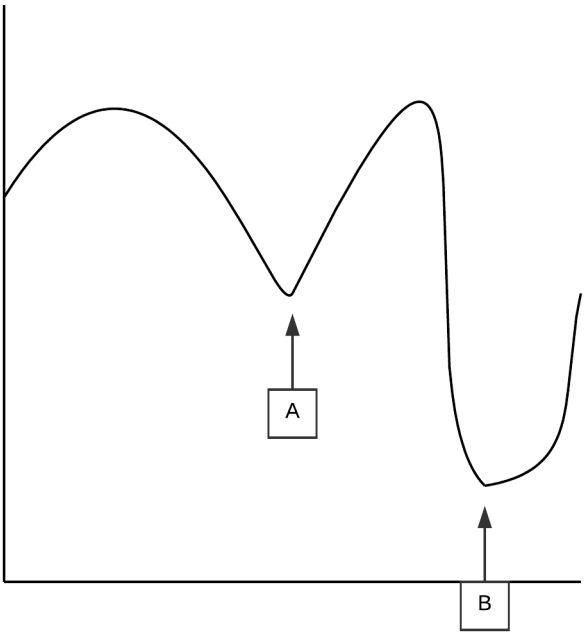


Figure 2.6: Gradient Descent

Another variation of Gradient Descent is Stochastic Gradient Descent (SGD). SGD is different because it updates "weights incrementally, following the calculation of the error of each individual example" Mitchell [15].

Backpropagation

"The Backpropagation algorithm learns the weights of a multilayer network, given a network with a fixed set of units and interconnections" Mitchell [15]. Backpropgation attempts to minimise the mean squared error between the target output and the output of a network.

Backpropagation works by starting at the output layer of the network and going back through previous hidden layers, updating weights as it goes.

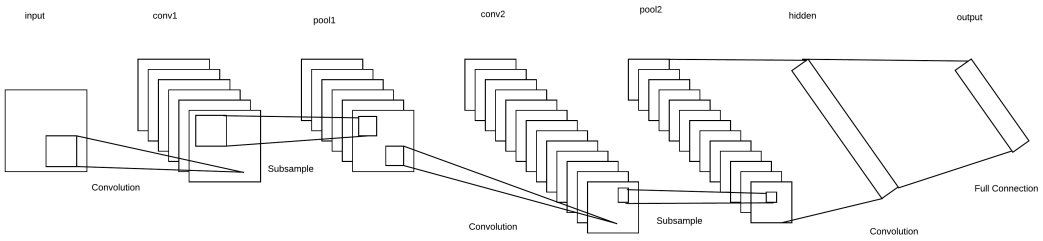


Figure 2.7: CNN Architecture

2.3 Convolutional Neural Networks Overview

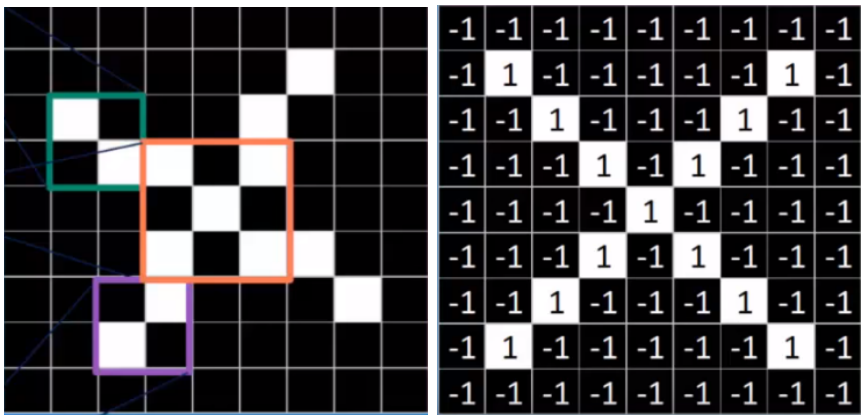
Convolutional Neural Networks (CNN's) are essentially a Multi Layered Percetron with a special structure. CNN's have one major difference from a MLP, they have extra layer of convolution and pooling. The architecture of a convolution network can be seen in Figure 2.7.

Figure 2.8a show an image that we want to compare against Figure 2.8b. For humans, it is quite easy to determine that these images are very similar but for a computer this task is surprisingly difficult.

So what a CNN does, to combat this problem, is to take a small feature from Figure 2.8a and compare it to a subsection of Figure 2.8b. The CNN multiplies the feature and a section of Figure 2.8b, adds up the results and divides by 9. This then gives a decimal value of how likely it is that the feature is in the part of the image, as seen in Figure 2.9b. This is called filtering. The Convolutional layer is composed of carrying out this filtering for every single possible location in Figure 2.8b.

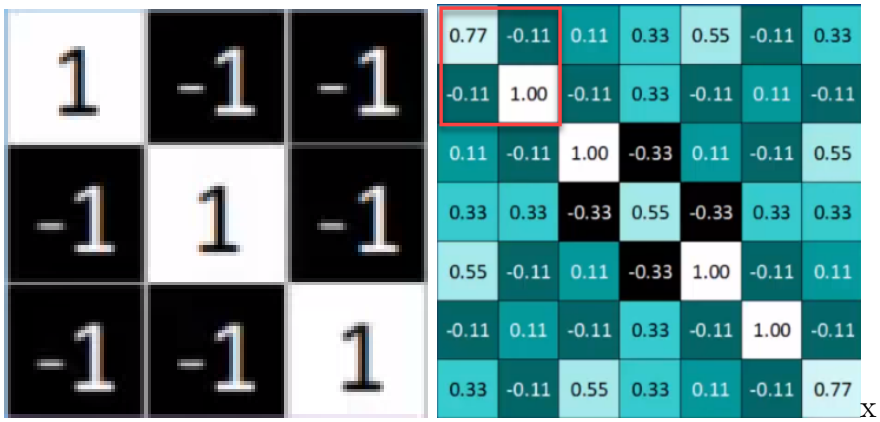
Next is the Pooling Layer, what this layer does, is it takes the convoluted layer output, you can use Figure 2.9b as reference, and from a user defined size ie. 2x2, gets either the highest decimal value (Max pooling) or the average value (Mean pooling) and records that as the new value for the section. This is then applied to the entire image. As we can see in Figure 2.10 we know have a much smaller image stack in which to classify, thus making the computation easier.

Figure 2.8: Image filtering



(a) Image to Classify (b) Image to Compare

Figure 2.9: Image Convolution



x

(a) Image Feature to Search (b) Convoluted Image

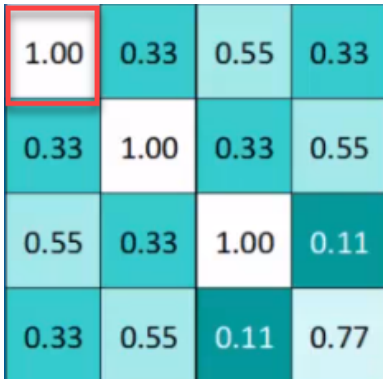


Figure 2.10: Pooled Image

In between the Convolution and Pooling layer, there is sometimes a Normalization layer. This Normalization layer creates Rectified Linear Units (RLU's). In other words, if we take Figure 2.9b, it changes all minus values to zero.

There are some problems with CNN's however. One of the main problems is that you need a very large dataset in order to produce am accurate model and the training can be very time consuming without a GPU.

2.4 Convolution Neural Networks Extended

Fully Convolutional Networks

A Fully Convolutional Network is one that does not have a fully connected layer and in a fully connected layers place is another convolution layer.

2.5 Support Vector Machine

A Support Vector Machine (SVM) is a machine learning algorithm that has been very popular before the power of a CNN was realised.

A SVM works by creating an n-dimensional space, with n as the number of inputs you have *Support Vector Machines for Machine Learning* [22]. The

SVM algorithm finds the hyperplane that splits this space. This hyperplane can then be used for classification.

2.6 Dietary Assessment using Computer Vision

Convolution Neural Networks

Many reserachers have used convolutional neural networks for image classification with various network architectures and many have used a food image dataset. I will be looking at some of these papers below.

Deep Learning Based Food Recognition

One paper focused on a deep learning approach to food image recognition based their neural network architecture on Inception-ResNet and Inception V3. They also used the Food-101 dataset Mao, Yu, and Wang [14]. For this system, Googles Tensorflow was used for image preprocessing. Preprocessing was needed as the environmental background is different in many food images. Because of these "Grey World method and Histogram equalization" Mao, Yu, and Wang [14] were used.

AWS GPU instances were used for training and the results on completion were quite impressive with a Top-1 Accuracy of 72.55% and a Top-5 Accuracy of 91.31% Mao, Yu, and Wang [14]

Food Image Recognition Using Deep Convolutional Neural Network With Pre-Training and Fine-Tuning

Another research team in Japan researched this topic. They were aware of how difficult the problem was and therefore employed many techniques to solve the problem such as "pre-training with the large-scale ImageNet data, fine-tuning

and activateion features extracted from the pre-trained DCNN" Yanai and Kawano [24].

In conclusion, they found that the "fine-tuned DCNN which was pre-trained with 2000 categories" Yanai and Kawano [24] from ImageNet was the best method. A DCNN is a Deep Convolution Neural Network. The achieved results of 78.77% for Top-1 Accuracy in the UECFOOD100 dataset.

Food Detection and Recognition Using Convolutional Neural Network

Kagaya, Aizawa, and Ogawa [11] also employed the use of convolutional neural networks for image detection. They used a CNN for the "tasks of food detection and recognition through parameter optimization" Kagaya, Aizawa, and Ogawa [11].

They found that a CNN is much better suited to the task than a Support Vector Machine (SVM). They achieved an overall classification accuracy of 93.8% against their baseline accuracy of 89.7% Kagaya, Aizawa, and Ogawa [11]. This accuracy was calculated using a dataset that they created specifically for this task. When they had completed the task they analysed the trained convolutional kernels and came to an interesting conclusion. They found that "color features are essential to food image recognition" Kagaya, Aizawa, and Ogawa [11].

DeepFood: Deep Learning-based Food Image Recognition for Computer-aided Dietary Assessment

The last paper that I will look at, oriented around using a convolutional neural network for food image recognition, focused on developing a dietary assessment application for use on a smartphone. They used the UEC-256 and Food-101 datset for their experiments and achieved impressive results.

They used a convolutional neural network but "with a few major optimizations, such as optimized model and an optimized convolution technique" Liu et al.

Table 2.1: DeepFood Results

	Top-1	Top-5
UEC-256	54.7%	81.5%
UEC-100	76.3%	94.6%
Food-101	77.4%	93.7%
UEC-256 With Bounding Box	63.8%	87.2%
UEC-100 With Bounding Box	77.2%	94.8%

[12]. They used the Inception module for their CNN. After the inception module was complete, they made the GoogleNet by combining modules. In total, the network had 22 layers.

They achieved the results shown in 2.1.

Other Methods

A Food Image Recognition System with Multiple Kernel Learning

In this paper, Joutou and Yanai [10], a practical use for food image recognition in the form of a mobile phone application was proposed. In order to classify the images, multiple kernel learning(MKL) was used. MKL is similar to am SVM expect that instead of a single kernel during training, MKL "treats with a combined kernel which is a weighted linear combination of several single kernels" Joutou and Yanai [10]. The idea behind this is that different food types are distinguishable by different factors and using this method, the best of these factors can be used for classification of that food type. In the experiment carried out by Joutou and Yanai [10], three different factors were used for learning:

- Color Histograms
- Gabor Texture Features
- Bag-of-Features using Scale Invariant Feature Transformation (SIFT)

50 different classifiers were created in a SVM using MKL with "one category as a positive set and other 49 categories as a negative set" Joutou and Yanai [10]. For each of these categories, a web scrape was carried out and then the best 100 images for each scrape was manually selected. Five-fold cross validation was utilised in the paper. MKL proceeded to yield results of 61.34% on the 50 food types and a Top-3 accuracy of 80.05% Joutou and Yanai [10]. The prototype mobile phone application resulted in a 37.55% user accuracy.

A Novel SVM Based Food Recognition Method for Calorie Measurement Applications

Another, quite successful, study was carried out using a SVM. Pouladzadeh et al. [18] had established that both colour and texture are very important but they also decided that shape and size are vital features to analyse. The proposed system has two main parts, segmentation followed by classification. In order to create a 'robust' system, a 'Robust Handling of Different Lighting Conditions' module is added to the system Pouladzadeh et al. [18]. This is so that various lighting conditions don't cause color data to be distorted.

Since this paper calls for calorie estimation, the first step of the system calculates the size of the food portion. In order to do this, a coin or the users thumb is included in the image taken so that the pixel count of the thumb and the food can be compared to estimate the size. Following this the image is segmented into various portions. The following step classifies each segment of the image by extracting color, texture and shape features and inputing these into a SVM Pouladzadeh et al. [18].

12 different food types were trained for this SVM with an average classification accuracy of 92.6%.

Measuring Calorie and Nutrition From Food Image

Another study that employed both a SVM and an emphasis on colour, texture and shape features, was carried out by Pouladzadeh, Shirmohammadi, and

Al-Maghrabi [16]. Size was also a factor in the calorie measurement module of the system. It was found that using all four of these features increases the overall accuracy.

In order to segment the image successfully, Gabor filters were applied to separate texture features while color was also utilised. For each segment established, size, shape, color and texture features were extracted and using a SVM, a classification was made. The SVM used the radial basis function kernel Pouladzadeh, Shirmohammadi, and Al-Maghrabi [16].

Calorie estimation was also a large part of this paper, and the users thumb was taken with the food in order to calculate food size.

In the prototype application, once the classification had been made, the user can confirm or change the prediction. Another feature of the application was in regards to "Partially Eaten Food" Pouladzadeh, Shirmohammadi, and Al-Maghrabi [16]. This was done by taking a picture before and after consumption and therefore only calculated the size of the food eaten and therefore more accurate calorie counts can be produced.

15 food types were trained using the SVM with 3000 images. The accuracy for the classifier averaged at 90.41% using 10 fold cross-validation Pouladzadeh, Shirmohammadi, and Al-Maghrabi [16]. There was also a calorie count accuracy of 86%. The best classification results were on single foods followed by non mixed and finally mixed foods produced the worst results.

Segmentation Assisted Food Classification for Dietary Assessment

Zhu et al. [25] had a strong focus on the segmentation aspect of a dietary assessment system. The segmentation of the food images was achieved "using Normalized Cuts based on intensity and colour" Zhu et al. [25]. Normalized Cuts is a graph based segmentation method. To aid the segmentation aspect of this study, a common background colour was introduced to the images. Segmentation refinement was also an important module in the experiment. This is the process by which neighbouring segments with the same classification

label are merged together. This also helps calculate a more accurate size estimation.

The classification of the segmented image was processed by using a SVM calculating colour and texture features. Gabor filters were used for the texture feature extraction Zhu et al. [25].

In the experimental results for this study, it was found that segmentation was not always successful "when the region of interest is camouflaged by making its boundary faint" Zhu et al. [25]. In their case, it was a can of coke that wasn't segmented correctly. The classification accuracy was of 56.2% and 95.5% with ground truth segmentation data Zhu et al. [25].

Possible Applications in Region Based CNNs

Ross Girshik and other contributers had some very positive results in the area of object detection using region based convolutional neural networks. There were four iterations of papers based on this work by Ross and groups in UC Berkley, Mircosoft and Facebook. A PHD student at the time of Ross's first paper also completed his dissertation on the subject. I will analyse this papers, their results (2.2) and the changes made through each iteration.

RCNN

In the first paper written by Ross Girshik, while researching at UC Berkeley, focused on two main insights. These were that "one can apply high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to localize and segment objects" and that "when training data is scarce, supervised pre-training for n auxilary task, followed by domain-specific fine-tuning, yields a significant performance boost" Girshick et al. [5].

The system that they developed followed these steps:

- Take image as input
- Extract approximately 2000 region proposals from the image

- Compute fixed length vectors of features for the regions using a convolutional neural network
- Use a Support Vector Machine (SVM) to classify these regions
- Bounding box regression for final region proposals

This system utilised selective search to gather these region proposals but they mention that a sliding-window detector is also an option. Ross Girshik and his team used the open source Caffe CNN library for this system. The system is quite efficient and scalable. It is scalable because of the fixed length vector of features which will remain constant regardless of inputs and additional outputs.

The team evaluated their results on a few metrics and test sets as seen in 2.2.

Fast RCNN

Ross Girshik's next iteration of work on region based convolution neural networks took place in Microsoft Research. This paper was titled "Fast R-CNN" as it's aim was to decrease training and testing time "while also increasing detection accuracy" Girshick [4].

This paper analyses why RCNN Girshick et al. [5] was slow and therefore how it could be improved. RCNN was classified to be slow because of three main factors:

- There are multiple stages to training as both a CNN and a SVM need to be trained.
- In training of the SVM, each region proposal must be written to disk and is therefore expensive.
- Object detection takes 47s per image Girshick [4].

Due to these problems with RCNN, a new algorithm, titled Fast RCNN was proposed. The architecture is as follows. An image is taken as input along

with a proposals for regions. The image is pushed through convolutional and pooling layers (using max pooling). A fixed-length vector of features is then extracted from each region proposal. These vectors are inputted to fully connected layers for bounding box location prediction Girshick [4].

At detection time, a pass through of the net is all that is needed so this runtime is significantly less than RCNN.

Faster RCNN

Due to the success of RCNN and Fast RCNN, Faster RCNN was introduced to combat the problem of region proposal computation Ren et al. [19].

The architecture for this system comprises of two modules. These consist of a convolutional neural network for region proposals (RPN) which the feeds into a Fast RCNN detector. These combine to produce a single neural network for object detection.

Instead of training these networks seperately, the team had to look at how to share layers between the two networks. There were three option available:

- Alternating training whereby RPN is trained, and then used to train Fast RCNN. The Fast RCNN network is then used to initialise RPN and the process is iterated Ren et al. [19]. This paper follows this approach.
- Approximate joint training.
- Non- approximate joint training.

Mask RCNN

The most recent paper on this topic was also written by Ross Girshik while working with Facebook AI Research He et al. [6]. Mask RCNN "extends Faster RCNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box regression" He et al. [6].

Table 2.2: Results from Region Based CNN Research

	VOC07	VOC10	VOC11	VOC12	COCO15	COCO16
RCNN	58.5%	53.7%	47.9%	N/A	N/A	N/A
Fast RCNN	70.0%	68.8%	N/A	68.4%	N/A	N/A
Faster RCNN	78.8%	N/A	N/A	75.9%	42.7%	N/A
Mask RCNN	N/A	N/A	N/A	N/A	N/A	63.1%

Mask RCNN has two modules, similar to Faster RCNN, where the first module is the Region Proposal Network. In the second module, in parallel to classification, a binary mask is outputted for each region. Bounding box regression and classification are done in parallel.

2.7 CNN APIs and Libraries

Tensorflow

Tensorflow is a deep learning software library for various machine learning paradigms. I will be using tensorflow to create neural networks. Tensorflow has two utilisations, through a Graphics Processing Unit (GPU) and also through a Central Processing Unit (CPU). GPU computation is recommended for CNN training.

Central Processing Unit Computation

It is quite easy to get tensorflow up and running if you are only using a CPU to train. I have successfully installed tensorflow cpu on both Windows and Ubuntu. For Windows you can download and install using the tensorflow website and on ubuntu you can using apt-get. Once installed, tensorflow can be imported into any python shell or script for use. Tensorflow can also be used in C. There will be various python implementations of neural networks in Chapter 3.

Graphics Processing Unit Computation

For use with a GPU, the set up for tensorflow is a bit more complicated. Firtsly you use check that the GPU in your machine is compatible for CUDA 8.0 using the NVIDIA website. If your GPU is compatible, you must install CUDA after signing up as an NVIDIA developer. CUDA 8.0 is compatible with tensorflow. You alos need to install cudnn6. The NVIDIA website contains tutorials to install these. Once these are installed, download and install tensorflow-gpu. This can imported into python similar to CPU computation.

OpenCV

Numpy

2.8 Evaluating the Output

There are various different metrics that can be used for evaluating the output of a classifier or segmentation algorithm, many of which we have seen in previous sections. I was analyse a few of these evaluations of results so that I can use them as a reference to apply to my own experiments. I will also look into some problems associated with evaluating models.

Research into Diagnosing Errors in Object Detectors

There has been some research into the question of how to evaluate object detectors, one of which I will discuss in detail Hoiem, Chodpathumwan, and Dai [8]. This paper in question "analyzes the influences of object characteristics on detection performance and the frequency and impact of different types of false positives" Hoiem, Chodpathumwan, and Dai [8]. They found that there were many effects that had influence on detectors as follows:

- occlusion

- size
- aspect ratio
- visibility of parts
- viewpoint
- localization error
- confusion with semantically similar objects
- confusion with other labeled objects
- confusion with background

The research team goes on to analyse false positives in object detectors. Localization errors were a large factor. This is were bounding boxes overlap to other objects in the image. Confusion with similar objects had a large influence on false positives also by which, for example, a dog detector had a high score for a cat Hoiem, Chodpathumwan, and Dai [8]. Confusion with dissimilar objects and confusion with background are the categories of the rest of the false positives they measured.

In conclusion the team would that "Most false positives are due to misaligned detection windows or confusion with similar objects" Hoiem, Chodpathumwan, and Dai [8]. They had some recommendations towards improves detectors as follows:

- Smaller objects are less likely to be detected
- Localization could be improved
- Reduce confusion with similar categories
- Robustness to object variation
- More detailed analysis

Detection Average Precision

The average detection accuracy of a system. See A.1.

Mean Average Precision

The mean accuracy of a system across all results. See A.2.

Distribution of top-ranked false positives

This metric is used to tell where most errors occur when false positives are evident. These errors are broken down into four categories of localization, similar objects, background confusion and others. See A.3.

Segmentation Mean Accuracy

This is the mean accuracy of segmentation. See A.4.

Per-category segmentation accuracy

This metric measures the accuracy of segmentation at a category level. See A.5.

Per-class segmentation accuracy

The accuracy of segmentation at a class level is analysed. See A.6.

Top-1 and Top-5 Accuracy

When a classifier is given an image it normally responds with a list of predictions along with a decimal representation of the likelihood that it is of that class. The Top-1 accuracy is the top valued prediction and Top-5 accuracy is the top 5 predictions.

Chapter 3

Experiments

3.1 Experiment 1

Overview

Network Architecture

Dataset

API's

Script

```
In [1]: import tensorflow as tf
        import input_data

In [2]: mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

Helper Functions

```
In [3]: #INIT WEIGHTS
        def init_weights(shape):
            init_random_dist = tf.truncated_normal(shape, stddev = 0.1)
            return tf.Variable(init_random_dist)

In [4]: #INIT BIAS
        def init_bias(shape):
            init_bias_vals = tf.constant(0.1, shape = shape)
            return tf.Variable(init_bias_vals)

In [5]: #CONV2D
        def conv2d(x, W):
            #x -> [batch, H, W, Channels]
            #W -> [filterH, filterW, ChannelsIn, ChannelsOut]
            return tf.nn.conv2d(x, W, strides = [1,1,1,1], padding = 'SAME')

In [6]: #POOLING
        def max_pool_2by2(x):
            #x -> [batch, H, W, Channels]
            return tf.nn.max_pool(x, ksize = [1,2,2,1], strides = [1,2,2,1], padding = 'SAME')

In [7]: #CONVOLUTIONAL LAYER
        def convolutional_layer(input_x, shape):
            W = init_weights(shape)
            b = init_bias([shape[3]])
            return tf.nn.relu(conv2d(input_x, W) + b)

In [8]: #NORMAL (FULLY CONNECTED)
        def normal_full_layer(input_layer, size):
            input_size = int(input_layer.get_shape()[1])
            W = init_weights([input_size, size])
            b = init_bias([size])
            return tf.matmul(input_layer, W) + b
```

Placeholders

```
In [9]: x = tf.placeholder(tf.float32, shape = [None, 784])
```

```
In [10]: y_true = tf.placeholder(tf.float32, shape = [None, 10])
```

Create Layers

```
In [11]: x_image = tf.reshape(x, [-1,28,28,1])

In [12]: #32 features for every 5 x 5 patch with 1(grayscale)
         convo_1 = convolutional_layer(x_image, shape = [5,5,1,32])
         convo_1_pooling = max_pool_2by2(convo_1)

In [13]: convo_2 = convolutional_layer(convo_1_pooling, shape = [5,5,32,64])
         convo_2_pooling = max_pool_2by2(convo_2)

In [14]: convo_2_flat = tf.reshape(convo_2_pooling, [-1,7*7*64])
         full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat, 1024))

In [15]: #DROPOUT
         hold_prob = tf.placeholder(tf.float32)
         full_one_dropout = tf.nn.dropout(full_layer_one, keep_prob= hold_prob)

In [16]: y_pred = normal_full_layer(full_one_dropout, 10)

In [17]: #LOSS FUNCTION
         cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_true,

In [18]: #OPTIMIZER
         optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
         train = optimizer.minimize(cross_entropy)

In [19]: init = tf.global_variables_initializer()

In [20]: steps = 1000

         with tf.Session() as sess:
             sess.run(init)

             for i in range(steps):
                 batch_x, batch_y = mnist.train.next_batch(50)
                 sess.run(train, feed_dict = {x:batch_x, y_true:batch_y, hold_prob:0.5})
                 if i%100 == 0:
                     print("ON STEP: {}".format(i))
                     print("ACCURACY: ")
                     match = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
                     acc = tf.reduce_mean(tf.cast(match, tf.float32))
                     print(sess.run(acc, feed_dict = {x:mnist.test.images, y_true:mnist.test.lab
                     print('\n')
```

```
ON STEP: 0
ACCURACY:
0.101
```

```
ON STEP: 100
ACCURACY:
0.94


ON STEP: 200
ACCURACY:
0.9603


ON STEP: 300
ACCURACY:
0.971


ON STEP: 400
ACCURACY:
0.9717


ON STEP: 500
ACCURACY:
0.9759


ON STEP: 600
ACCURACY:
0.9772


ON STEP: 700
ACCURACY:
0.9811


ON STEP: 800
ACCURACY:
0.9793


ON STEP: 900
ACCURACY:
0.973
```

Results

Analysis

3.2 Experiment 2

Overview

Network Architecture

Dataset

API's

Script

In [1]:
```
from __future__ import division, print_function, absolute_import

# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)

import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import PIL
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

In [2]:
```
# Training Parameters
learning_rate = 0.001
num_steps = 5000
batch_size = 128

# Network Parameters
num_input = 784 # MNIST data input (img shape: 28*28)
num_classes = 10 # MNIST total classes (0-9 digits)
dropout = 0.25 # Dropout, probability to drop a unit
```

In [3]:
```
# Create the neural network
def conv_net(x_dict, n_classes, dropout, reuse, is_training):

    # Define a scope for reusing the variables
    with tf.variable_scope('ConvNet', reuse=reuse):
        # TF Estimator input is a dict, in case of multiple inputs
        x = x_dict['images']

        # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
        # Reshape to match picture format [Height x Width x Channel]
        # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
        x = tf.reshape(x, shape=[-1, 28, 28, 1])

        # Convolution Layer with 32 filters and a kernel size of 5
```

```
            conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
            # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
            conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

            # Convolution Layer with 64 filters and a kernel size of 3
            conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
            # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
            conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

            # Flatten the data to a 1-D vector for the fully connected layer
            fc1 = tf.contrib.layers.flatten(conv2)

            # Fully connected layer (in tf contrib folder for now)
            fc1 = tf.layers.dense(fc1, 1024)
            # Apply Dropout (if is_training is False, dropout is not applied)
            fc1 = tf.layers.dropout(fc1, rate=dropout, training=is_training)

            # Output layer, class prediction
            out = tf.layers.dense(fc1, n_classes)

        return out
```

In [4]:
```
# Define the model function (following TF Estimator Template)
def model_fn(features, labels, mode):

    # Build the neural network
    # Because Dropout have different behavior at training and prediction time, we
    # need to create 2 distinct computation graphs that still share the same weights.
    logits_train = conv_net(features, num_classes, dropout, reuse=False, is_training=Tru
    logits_test = conv_net(features, num_classes, dropout, reuse=True, is_training=False

    # Predictions
    pred_classes = tf.argmax(logits_test, axis=1)
    pred_probas = tf.nn.softmax(logits_test)

    # If prediction mode, early return
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode, predictions=pred_classes)

    # Define loss and optimizer
    loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
        logits=logits_train, labels=tf.cast(labels, dtype=tf.int32)))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    train_op = optimizer.minimize(loss_op, global_step=tf.train.get_global_step())

    # Evaluate the accuracy of the model
    acc_op = tf.metrics.accuracy(labels=labels, predictions=pred_classes)
```

```
            # TF Estimators requires to return a EstimatorSpec, that specify
            # the different ops for training, evaluating, ...
            estim_specs = tf.estimator.EstimatorSpec(
              mode=mode,
              predictions=pred_classes,
              loss=loss_op,
              train_op=train_op,
              eval_metric_ops={'accuracy': acc_op})

            return estim_specs
```

```
In [5]: # Build the Estimator
        model = tf.estimator.Estimator(model_fn)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\tom13\AppData\Local\Temp\
INFO:tensorflow:Using config: {'_keep_checkpoint_every_n_hours': 10000, '_task_type': 'worker',
```

```
In [6]: # Define the input function for training
        input_fn = tf.estimator.inputs.numpy_input_fn(
            x={'images': mnist.train.images}, y=mnist.train.labels,
            batch_size=batch_size, num_epochs=None, shuffle=True)
        # Train the Model
        model.train(input_fn, steps=num_steps)
```

```
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\tom13\AppData\Local\Temp\tmp97az0nju\mode
INFO:tensorflow:step = 1, loss = 2.31252
INFO:tensorflow:global_step/sec: 62.4813
INFO:tensorflow:step = 101, loss = 0.165548 (1.602 sec)
INFO:tensorflow:global_step/sec: 67.4517
INFO:tensorflow:step = 201, loss = 0.149662 (1.485 sec)
INFO:tensorflow:global_step/sec: 67.1394
INFO:tensorflow:step = 301, loss = 0.208731 (1.487 sec)
INFO:tensorflow:global_step/sec: 67.1479
INFO:tensorflow:step = 401, loss = 0.0409326 (1.490 sec)
INFO:tensorflow:global_step/sec: 68.0719
INFO:tensorflow:step = 501, loss = 0.0213496 (1.470 sec)
INFO:tensorflow:global_step/sec: 68.771
INFO:tensorflow:step = 601, loss = 0.037192 (1.452 sec)
INFO:tensorflow:global_step/sec: 67.3723
INFO:tensorflow:step = 701, loss = 0.0822325 (1.485 sec)
INFO:tensorflow:global_step/sec: 68.6787
INFO:tensorflow:step = 801, loss = 0.0128403 (1.455 sec)
INFO:tensorflow:global_step/sec: 68.8452
INFO:tensorflow:step = 901, loss = 0.0358137 (1.455 sec)
INFO:tensorflow:global_step/sec: 68.75
INFO:tensorflow:step = 1001, loss = 0.0192664 (1.453 sec)
```

```
INFO:tensorflow:global_step/sec: 68.5087
INFO:tensorflow:step = 1101, loss = 0.00774879 (1.459 sec)
INFO:tensorflow:global_step/sec: 67.7028
INFO:tensorflow:step = 1201, loss = 0.0851327 (1.477 sec)
INFO:tensorflow:global_step/sec: 67.7039
INFO:tensorflow:step = 1301, loss = 0.0553374 (1.479 sec)
INFO:tensorflow:global_step/sec: 64.6995
INFO:tensorflow:step = 1401, loss = 0.0355084 (1.547 sec)
INFO:tensorflow:global_step/sec: 67.4277
INFO:tensorflow:step = 1501, loss = 0.0163228 (1.481 sec)
INFO:tensorflow:global_step/sec: 65.4895
INFO:tensorflow:step = 1601, loss = 0.00959715 (1.528 sec)
INFO:tensorflow:global_step/sec: 67.7365
INFO:tensorflow:step = 1701, loss = 0.00657101 (1.477 sec)
INFO:tensorflow:global_step/sec: 68.2635
INFO:tensorflow:step = 1801, loss = 0.00808421 (1.464 sec)
INFO:tensorflow:global_step/sec: 68.3004
INFO:tensorflow:step = 1901, loss = 0.00447089 (1.465 sec)
INFO:tensorflow:global_step/sec: 66.2228
INFO:tensorflow:step = 2001, loss = 0.00395705 (1.510 sec)
INFO:tensorflow:global_step/sec: 67.0613
INFO:tensorflow:step = 2101, loss = 0.0143454 (1.492 sec)
INFO:tensorflow:global_step/sec: 66.8144
INFO:tensorflow:step = 2201, loss = 0.00251255 (1.496 sec)
INFO:tensorflow:global_step/sec: 67.2051
INFO:tensorflow:step = 2301, loss = 0.00249033 (1.487 sec)
INFO:tensorflow:global_step/sec: 67.9835
INFO:tensorflow:step = 2401, loss = 0.0498065 (1.470 sec)
INFO:tensorflow:global_step/sec: 68.2512
INFO:tensorflow:step = 2501, loss = 0.00226328 (1.467 sec)
INFO:tensorflow:global_step/sec: 67.8173
INFO:tensorflow:step = 2601, loss = 0.000694454 (1.473 sec)
INFO:tensorflow:global_step/sec: 67.6461
INFO:tensorflow:step = 2701, loss = 0.0344697 (1.480 sec)
INFO:tensorflow:global_step/sec: 67.8946
INFO:tensorflow:step = 2801, loss = 0.00430426 (1.472 sec)
INFO:tensorflow:global_step/sec: 69.0382
INFO:tensorflow:step = 2901, loss = 0.033511 (1.447 sec)
INFO:tensorflow:global_step/sec: 67.9999
INFO:tensorflow:step = 3001, loss = 0.00375906 (1.470 sec)
INFO:tensorflow:global_step/sec: 68.4807
INFO:tensorflow:step = 3101, loss = 0.00979484 (1.462 sec)
INFO:tensorflow:global_step/sec: 67.3055
INFO:tensorflow:step = 3201, loss = 0.0192151 (1.488 sec)
INFO:tensorflow:global_step/sec: 68.755
INFO:tensorflow:step = 3301, loss = 0.00091159 (1.452 sec)
INFO:tensorflow:global_step/sec: 68.2027
INFO:tensorflow:step = 3401, loss = 0.00265687 (1.466 sec)
```

```
INFO:tensorflow:global_step/sec: 66.981
INFO:tensorflow:step = 3501, loss = 0.00322506 (1.493 sec)
INFO:tensorflow:global_step/sec: 67.9271
INFO:tensorflow:step = 3601, loss = 0.00257395 (1.475 sec)
INFO:tensorflow:global_step/sec: 68.4105
INFO:tensorflow:step = 3701, loss = 0.0113146 (1.459 sec)
INFO:tensorflow:global_step/sec: 68.9221
INFO:tensorflow:step = 3801, loss = 0.0119776 (1.453 sec)
INFO:tensorflow:global_step/sec: 68.3297
INFO:tensorflow:step = 3901, loss = 0.000739706 (1.464 sec)
INFO:tensorflow:global_step/sec: 67.7096
INFO:tensorflow:step = 4001, loss = 0.00202849 (1.474 sec)
INFO:tensorflow:global_step/sec: 68.4309
INFO:tensorflow:step = 4101, loss = 0.00123262 (1.462 sec)
INFO:tensorflow:global_step/sec: 68.589
INFO:tensorflow:step = 4201, loss = 0.0213804 (1.460 sec)
INFO:tensorflow:global_step/sec: 68.3938
INFO:tensorflow:step = 4301, loss = 0.00818979 (1.463 sec)
INFO:tensorflow:global_step/sec: 68.1871
INFO:tensorflow:step = 4401, loss = 0.0192472 (1.465 sec)
INFO:tensorflow:global_step/sec: 67.4095
INFO:tensorflow:step = 4501, loss = 0.009068 (1.483 sec)
INFO:tensorflow:global_step/sec: 68.0209
INFO:tensorflow:step = 4601, loss = 0.00785339 (1.470 sec)
INFO:tensorflow:global_step/sec: 68.466
INFO:tensorflow:step = 4701, loss = 0.00145256 (1.461 sec)
INFO:tensorflow:global_step/sec: 68.8636
INFO:tensorflow:step = 4801, loss = 0.0157578 (1.452 sec)
INFO:tensorflow:global_step/sec: 68.8196
INFO:tensorflow:step = 4901, loss = 0.0353847 (1.454 sec)
INFO:tensorflow:Saving checkpoints for 5000 into C:\Users\tom13\AppData\Local\Temp\tmp97az0nju\m
INFO:tensorflow:Loss for final step: 0.024679.
```

```
Out[6]: <tensorflow.python.estimator.estimator.Estimator at 0x226833dcd30>
```

```
In [7]: # Evaluate the Model
        # Define the input function for evaluating
        input_fn = tf.estimator.inputs.numpy_input_fn(
            x={'images': mnist.test.images}, y=mnist.test.labels,
            batch_size=batch_size, shuffle=False)
        # Use the Estimator 'evaluate' method
        model.evaluate(input_fn)
```

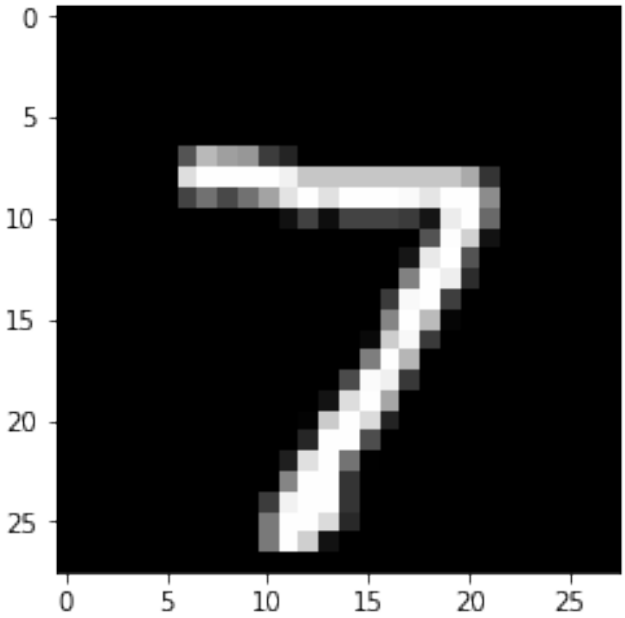
```
INFO:tensorflow:Starting evaluation at 2018-01-21-16:56:42
INFO:tensorflow:Restoring parameters from C:\Users\tom13\AppData\Local\Temp\tmp97az0nju\model.ck
INFO:tensorflow:Finished evaluation at 2018-01-21-16:56:43
INFO:tensorflow:Saving dict for global step 5000: accuracy = 0.989, global_step = 5000, loss = 0
```

```
Out[7]: {'accuracy': 0.98900002, 'global_step': 5000, 'loss': 0.052190915}
```

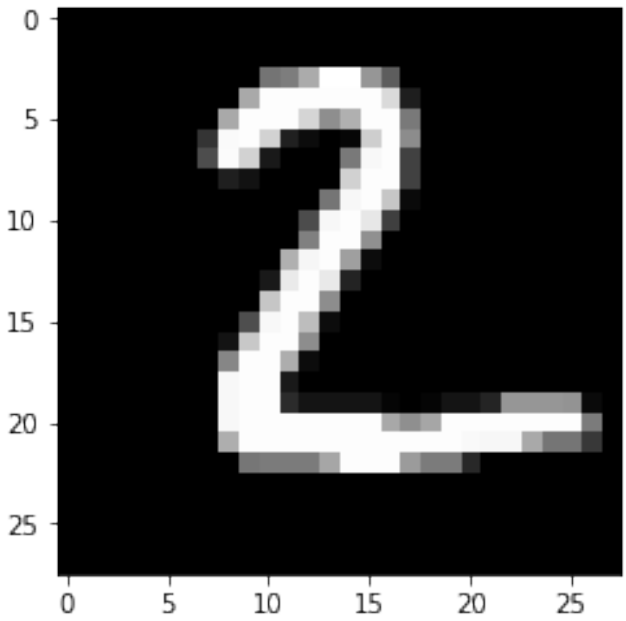
```
In [8]: # Predict single images
        n_images = 5
        # Get images from test set
        test_images = mnist.test.images[:n_images]
        # Prepare the input data
        input_fn = tf.estimator.inputs.numpy_input_fn(
            x={'images': test_images}, shuffle=False)
        # Use the model to predict the images class
        preds = list(model.predict(input_fn))

        # Display
        for i in range(n_images):
            plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray')
            plt.show()
            print("Model prediction:", preds[i])
```

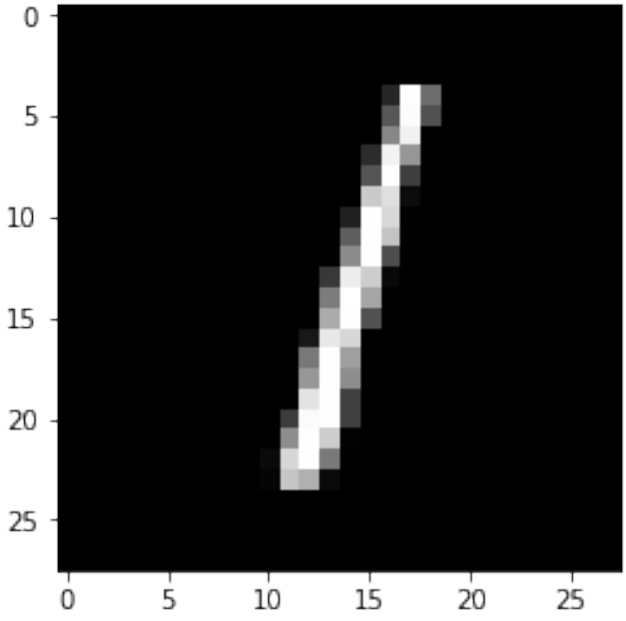
```
INFO:tensorflow:Restoring parameters from C:\Users\tom13\AppData\Local\Temp\tmp97az0nju\model.ck
```



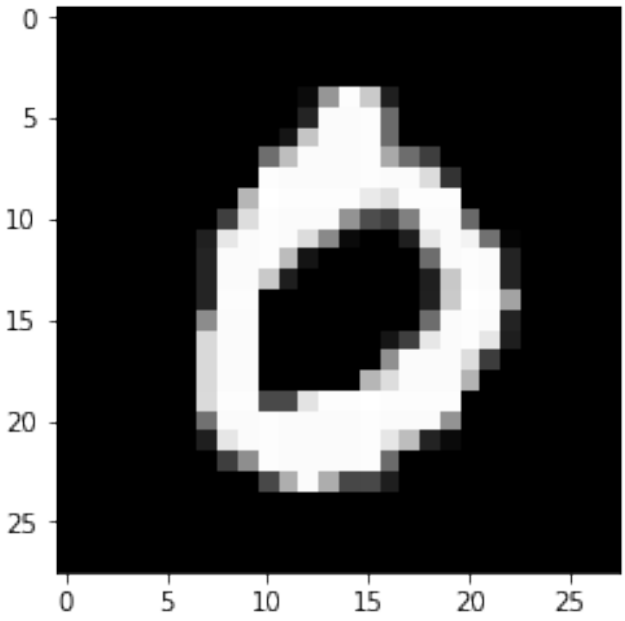

```
Model prediction: 7
```



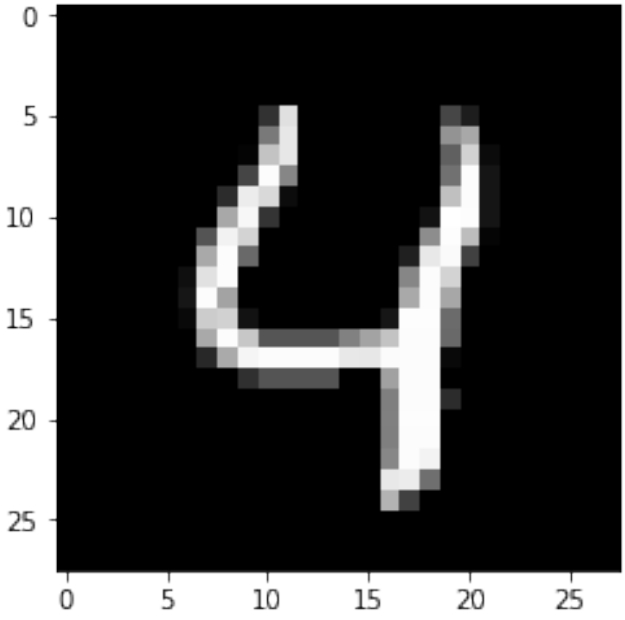

```
Model prediction: 2
```



```
Model prediction: 1
```




```
Model prediction: 0
```



```
Model prediction: 4
```

Results

Analysis

3.3 Experiment 3

Overview

Network Architecture

Dataset

API's

Script

0.1 CNN-Project-Exercise

We'll be using the CIFAR-10 dataset, which is very famous dataset for image recognition!

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

0.1.1 Follow the Instructions in Bold, if you get stuck somewhere, view the solutions video! Most of the challenge with this project is actually dealing with the data and its dimensions, not from setting up the CNN itself!

0.2 Step 0: Get the Data

** *Note: If you have trouble with this just watch the solutions video. This doesn't really have anything to do with the exercise, its more about setting up your data. Please make sure to watch the solutions video before posting any QA questions.* **

** Download the data for CIFAR from here: https://www.cs.toronto.edu/~kriz/cifar.html **

Specifically the CIFAR-10 python version link: https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

** Remember the directory you save the file in! **

```
In [1]: # Put file path as a string here
        CIFAR_DIR = 'C:/Users/tom13/jupyter/FYP/CIFAR-10/'
```

The archive contains the files data_batch_1, data_batch_2, ..., data_batch_5, as well as test_batch. Each of these files is a Python "pickled" object produced with cPickle.

** Load the Data. Use the Code Below to load the data: **

```
In [2]: def unpickle(file):
            import pickle
            with open(file, 'rb') as fo:
                cifar_dict = pickle.load(fo, encoding='bytes')
            return cifar_dict

In [3]: dirs = ['batches.meta','data_batch_1','data_batch_2','data_batch_3','data_batch_4','data

In [4]: all_data = [0,1,2,3,4,5,6]

In [5]: for i,direc in zip(all_data,dirs):
            all_data[i] = unpickle(CIFAR_DIR+direc)
```

```
In [6]: batch_meta = all_data[0]
        data_batch1 = all_data[1]
        data_batch2 = all_data[2]
        data_batch3 = all_data[3]
        data_batch4 = all_data[4]
        data_batch5 = all_data[5]
        test_batch = all_data[6]

In [7]: batch_meta

Out[7]: {b'label_names': [b'airplane',
          b'automobile',
          b'bird',
          b'cat',
          b'deer',
          b'dog',
          b'frog',
          b'horse',
          b'ship',
          b'truck'],
         b'num_cases_per_batch': 10000,
         b'num_vis': 3072}
```

** Why the 'b's in front of the string? ** Bytes literals are always prefixed with 'b' or 'B'; they produce an instance of the bytes type instead of the str type. They may only contain ASCII characters; bytes with a numeric value of 128 or greater must be expressed with escapes.

https://stackoverflow.com/questions/6269765/what-does-the-b-character-do-in-front-of-a-string-literal

```
In [8]: data_batch1.keys()

Out[8]: dict_keys([b'data', b'batch_label', b'labels', b'filenames'])
```

Loaded in this way, each of the batch files contains a dictionary with the following elements: * data -- a 10000x3072 numpy array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image. * labels -- a list of 10000 numbers in the range 0-9. The number at index i indicates the label of the ith image in the array data.

The dataset contains another file, called batches.meta. It too contains a Python dictionary object. It has the following entries:

- label_names -- a 10-element list which gives meaningful names to the numeric labels in the labels array described above. For example, label_names[0] == "airplane", label_names[1] == "automobile", etc.

0.2.1 Display a single image using matplotlib.

** Grab a single image from data_batch1 and display it with plt.imshow(). You'll need to reshape and transpose the numpy array inside the X = data_batch[b'data'] dictionary entry.**

** It should end up looking like this: **

```
# Array of all images reshaped and formatted for viewing
X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("uint8")
```

```
In [9]: import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
```

```
In [10]: X = data_batch1[b"data"]
```

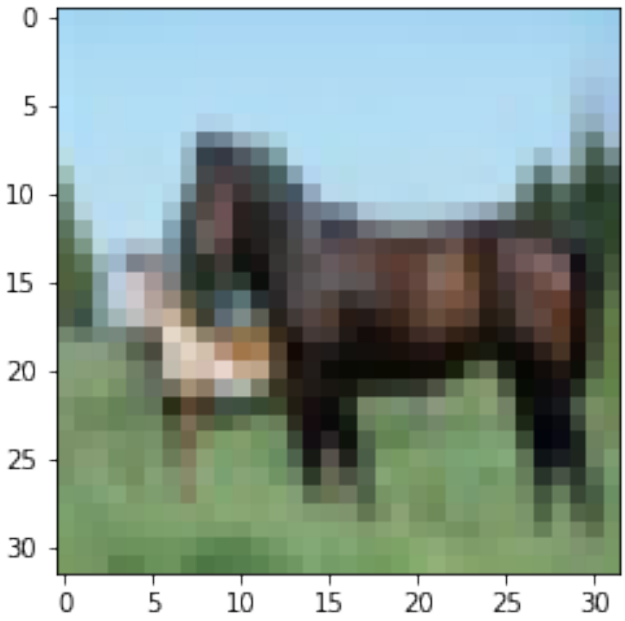
```
In [11]: X.shape
```

```
Out[11]: (10000, 3072)
```

```
In [12]: X = X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("uint8")
```

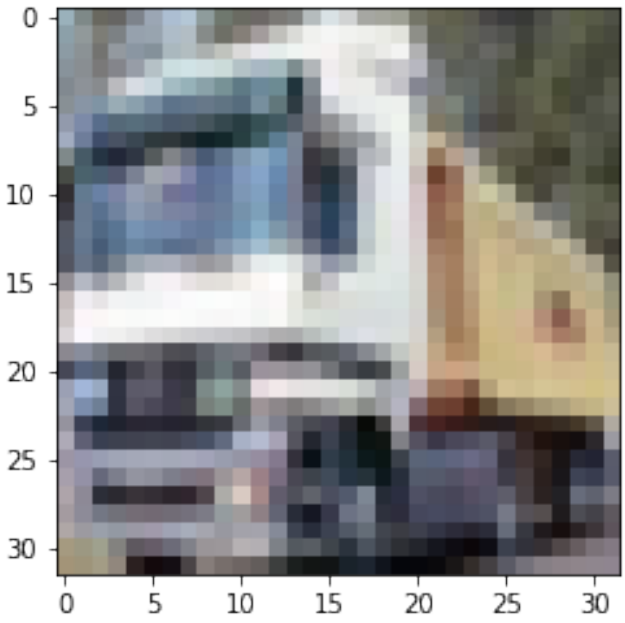
```
In [13]: plt.imshow(X[12])
```

```
Out[13]: <matplotlib.image.AxesImage at 0x287f0aa3780>
```



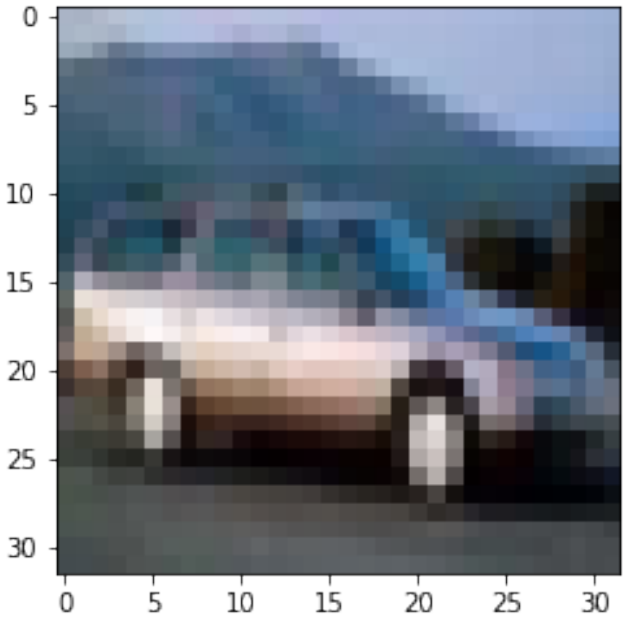

```
In [14]: plt.imshow(X[1])
```

```
Out[14]: <matplotlib.image.AxesImage at 0x287f0b0e390>
```



In [15]: plt.imshow(X[4])

Out[15]: <matplotlib.image.AxesImage at 0x287f0b53390>



1 Helper Functions for Dealing With Data.

** Use the provided code below to help with dealing with grabbing the next batch once you've gotten ready to create the Graph Session. Can you break down how it works? **

```
In [55]: def one_hot_encode(vec, vals=10):
             '''
             For use to one-hot encode the 10- possible labels
             '''
             n = len(vec)
             out = np.zeros((n, vals))
             out[range(n), vec] = 1
             return out

In [56]: class CifarHelper():

             def __init__(self):
                 self.i = 0

                 # Grabs a list of all the data batches for training
                 self.all_train_batches = [data_batch1,data_batch2,data_batch3,data_batch4,data_
                 # Grabs a list of all the test batches (really just one batch)
                 self.test_batch = [test_batch]

                 # Intialize some empty variables for later on
                 self.training_images = None
                 self.training_labels = None

                 self.test_images = None
                 self.test_labels = None

             def set_up_images(self):

                 print("Setting Up Training Images and Labels")

                 # Vertically stacks the training images
                 self.training_images = np.vstack([d[b"data"] for d in self.all_train_batches])
                 train_len = len(self.training_images)

                 # Reshapes and normalizes training images
                 self.training_images = self.training_images.reshape(train_len,3,32,32).transpos
                 # One hot Encodes the training labels (e.g. [0,0,0,1,0,0,0,0,0,0])
                 self.training_labels = one_hot_encode(np.hstack([d[b"labels"] for d in self.all

                 print("Setting Up Test Images and Labels")

                 # Vertically stacks the test images
                 self.test_images = np.vstack([d[b"data"] for d in self.test_batch])
```

```
        test_len = len(self.test_images)

        # Reshapes and normalizes test images
        self.test_images = self.test_images.reshape(test_len,3,32,32).transpose(0,2,3,1
        # One hot Encodes the test labels (e.g. [0,0,0,1,0,0,0,0,0,0])
        self.test_labels = one_hot_encode(np.hstack([d[b"labels"] for d in self.test_ba


    def next_batch(self, batch_size):
        # Note that the 100 dimension in the reshape call is set by an assumed batch si
        x = self.training_images[self.i:self.i+batch_size].reshape(100,32,32,3)
        y = self.training_labels[self.i:self.i+batch_size]
        self.i = (self.i + batch_size) % len(self.training_images)
        return x, y

    def next_test_batch(self, batch_size):
        # Note that the 100 dimension in the reshape call is set by an assumed batch si
        x = self.test_images[self.i:self.i+batch_size].reshape(100,32,32,3)
        y = self.test_labels[self.i:self.i+batch_size]
        self.i = (self.i + batch_size) % len(self.test_images)
        return x, y
```

** How to use the above code: **

```
In [57]: # Before Your tf.Session run these two lines
         ch = CifarHelper()
         ch.set_up_images()

         # During your session to grab the next batch use this line
         #batch = ch.next_batch(100)
```

```
Setting Up Training Images and Labels
Setting Up Test Images and Labels
```

1.1 Creating the Model

** Import tensorflow **

```
In [58]: import tensorflow as tf
```

```
In [59]: x = tf.placeholder(tf.float32, shape = [None, 32, 32, 3])
         y_true = tf.placeholder(tf.float32, shape = [None, 10])
         hold_prob = tf.placeholder(tf.float32)
```

1.1.1 Helper Functions

** Grab the helper functions from MNIST with CNN (or recreate them here yourself for a hard challenge!). You'll need: **

- init_weights
- init_bias
- conv2d
- max_pool_2by2
- convolutional_layer
- normal_full_layer

```
In [60]: def init_weights(shape):
             init_random_dist = tf.truncated_normal(shape, stddev=0.1)
             return tf.Variable(init_random_dist)

         def init_bias(shape):
             init_bias_vals = tf.constant(0.1, shape=shape)
             return tf.Variable(init_bias_vals)

         def conv2d(x, W):
             return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

         def max_pool_2by2(x):
             return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                                   strides=[1, 2, 2, 1], padding='SAME')

         def convolutional_layer(input_x, shape):
             W = init_weights(shape)
             b = init_bias([shape[3]])
             return tf.nn.relu(conv2d(input_x, W) + b)

         def normal_full_layer(input_layer, size):
             input_size = int(input_layer.get_shape()[1])
             W = init_weights([input_size, size])
             b = init_bias([size])
             return tf.matmul(input_layer, W) + b
```

1.1.2 Create the Layers

** Create a convolutional layer and a pooling layer as we did for MNIST. Its up to you what the 2d size of the convolution should be, but the last two digits need to be 3 and 32 because of the 3 color channels and 32 pixels. So for example you could use:**

```
convo_1 = convolutional_layer(x,shape=[4,4,3,32])
```

```
In [61]: convo_1 = convolutional_layer(x, shape = [4, 4, 3, 32])
         convo_1_pooling = max_pool_2by2(convo_1)
```

** Create the next convolutional and pooling layers. The last two dimensions of the convo_2 layer should be 32,64 **

```
In [62]: convo_2 = convolutional_layer(convo_1_pooling, shape = [4, 4, 32, 64])
         convo_2_pooling = max_pool_2by2(convo_2)
```

** Now create a flattened layer by reshaping the pooling layer into [-1,8 * 8 * 64] or [-1,4096] **

```
In [63]: convo_2_flat = tf.reshape(convo_2_pooling, [-1, 4096])
```

** Create a new full layer using the normal_full_layer function and passing in your flattend convolutional 2 layer with size=1024. (You could also choose to reduce this to something like 512)**

```
In [64]: full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat, 1024))
```

** Now create the dropout layer with tf.nn.dropout, remember to pass in your hold_prob placeholder. **

```
In [65]: full_one_dropout = tf.nn.dropout(full_layer_one, keep_prob = hold_prob)
```

** Finally set the output to y_pred by passing in the dropout layer into the normal_full_layer function. The size should be 10 because of the 10 possible labels**

```
In [66]: y_pred = normal_full_layer(full_one_dropout, 10)
```

1.1.3 Loss Function

** Create a cross_entropy loss function **

```
In [67]: cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_true,
```

1.1.4 Optimizer

** Create the optimizer using an Adam Optimizer. **

```
In [68]: optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
         train = optimizer.minimize(cross_entropy)
```

** Create a variable to intialize all the global tf variables. **

```
In [69]: init = tf.global_variables_initializer()
```

1.2 Graph Session

** Perform the training and test print outs in a Tf session and run your model! **

```
In [73]: with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())

             for i in range(10000):
                 batch = ch.next_batch(100)
                 sess.run(train, feed_dict={x: batch[0], y_true: batch[1], hold_prob: 0.5})

                 # PRINT OUT A MESSAGE EVERY 100 STEPS
                 if i%1000 == 0:
```

```
                    # Test the Train Model
                    matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))

                    acc = tf.reduce_mean(tf.cast(matches,tf.float32))
                    testSet = ch.next_test_batch(100)
                    print('Accuracy: ')
                    print(sess.run(acc,feed_dict={x:testSet[0] ,y_true:testSet[1],hold_prob:1.0
                    print('\n')
```

```
Accuracy:
0.08


Accuracy:
0.58


Accuracy:
0.66


Accuracy:
0.65


Accuracy:
0.69


Accuracy:
0.7


Accuracy:
0.7


Accuracy:
0.63


Accuracy:
0.67


Accuracy:
0.71
```

Results

Analysis

3.4 Experiment 4

Overview

Network Architecture

Dataset

API's

Script

In [1]:
```
from __future__ import print_function

import tensorflow as tf
import os
```

In [2]:
```
# Dataset Parameters - CHANGE HERE
MODE = 'folder' # or 'file', if you choose a plain text file (see above).
DATASET_PATH = 'C:/Users/tom13/jupyter/FYP/food-101/train' # the dataset file or root fo

# Image Parameters
N_CLASSES = 101 # CHANGE HERE, total number of classes
IMG_HEIGHT = 128 # CHANGE HERE, the image height to be resized to
IMG_WIDTH = 128 # CHANGE HERE, the image width to be resized to
CHANNELS = 3 # The 3 color channels, change to 1 if grayscale
```

In [3]:
```
# Reading the dataset
# 2 modes: 'file' or 'folder'
def read_images(dataset_path, mode, batch_size):
    imagepaths, labels = list(), list()
    if mode == 'file':
        # Read dataset file
        data = open(dataset_path, 'r').read().splitlines()
        for d in data:
            imagepaths.append(d.split(' ')[0])
            labels.append(int(d.split(' ')[1]))
    elif mode == 'folder':
        # An ID will be affected to each sub-folders by alphabetical order
        label = 0
        # List the directory
        try:  # Python 2
            classes = sorted(os.walk(dataset_path).next()[1])
        except Exception:  # Python 3
            classes = sorted(os.walk(dataset_path).__next__()[1])
        # List each sub-directory (the classes)
        for c in classes:
            c_dir = os.path.join(dataset_path, c)
            try:  # Python 2
                walk = os.walk(c_dir).next()
            except Exception:  # Python 3
                walk = os.walk(c_dir).__next__()
            # Add each image to the training set
```

```
                for sample in walk[2]:
                    # Only keeps jpeg images
                    if sample.endswith('.jpg') or sample.endswith('.jpeg'):
                        imagepaths.append(os.path.join(c_dir, sample))
                        labels.append(label)
                label += 1
        else:
            raise Exception("Unknown mode.")

        # Convert to Tensor
        imagepaths = tf.convert_to_tensor(imagepaths, dtype=tf.string)
        labels = tf.convert_to_tensor(labels, dtype=tf.int32)
        # Build a TF Queue, shuffle data
        image, label = tf.train.slice_input_producer([imagepaths, labels],
                                                     shuffle=True)

        # Read images from disk
        image = tf.read_file(image)
        image = tf.image.decode_jpeg(image, channels=CHANNELS)

        # Resize images to a common size
        image = tf.image.resize_images(image, [IMG_HEIGHT, IMG_WIDTH])

        # Normalize
        image = image * 1.0/127.5 - 1.0

        # Create batches
        X, Y = tf.train.batch([image, label], batch_size=batch_size,
                              capacity=batch_size * 8,
                              num_threads=4)

        return X, Y
```

In [4]:
```
# Parameters
learning_rate = 0.001
num_steps = 5000
batch_size = 32
display_step = 500

# Network Parameters
dropout = 0.75 # Dropout, probability to keep units

# Build the data input
X, Y = read_images(DATASET_PATH, MODE, batch_size)
```

In [5]:
```
# Create model
def conv_net(x, n_classes, dropout, reuse, is_training):
    # Define a scope for reusing the variables
```

```
        with tf.variable_scope('ConvNet', reuse=reuse):

            # Convolution Layer with 32 filters and a kernel size of 5
            conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
            # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
            conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

            # Convolution Layer with 32 filters and a kernel size of 5
            conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
            # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
            conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

            # Convolution Layer with 32 filters and a kernel size of 5
            conv3 = tf.layers.conv2d(conv2, 64, 3, activation=tf.nn.relu)
            # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
            conv3 = tf.layers.max_pooling2d(conv3, 2, 2)

            # Flatten the data to a 1-D vector for the fully connected layer
            fc1 = tf.contrib.layers.flatten(conv2)

            # Fully connected layer (in contrib folder for now)
            fc1 = tf.layers.dense(fc1, 1024)
            # Apply Dropout (if is_training is False, dropout is not applied)
            fc1 = tf.layers.dropout(fc1, rate=dropout, training=is_training)

            # Output layer, class prediction
            out = tf.layers.dense(fc1, n_classes)
            # Because 'softmax_cross_entropy_with_logits' already apply softmax,
            # we only apply softmax to testing network
            out = tf.nn.softmax(out) if not is_training else out

        return out
```

In [6]:
```
# Create a graph for training
logits_train = conv_net(X, N_CLASSES, dropout, reuse=False, is_training=True)
# Create another graph for testing that reuse the same weights
logits_test = conv_net(X, N_CLASSES, dropout, reuse=True, is_training=False)
```

In [7]:
```
# Define loss and optimizer (with train logits, for dropout to take effect)
loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits_train, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
```

In [8]:
```
# Evaluate model (with test logits, for dropout to be disabled)
correct_pred = tf.equal(tf.argmax(logits_test, 1), tf.cast(Y, tf.int64))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

In [9]:
```
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

```
In [10]: # Saver object
         saver = tf.train.Saver()

In [11]: # Start training
         with tf.Session() as sess:

             # Run the initializer
             sess.run(init)

             # Start the data queue
             tf.train.start_queue_runners()

             # Training cycle
             for step in range(1, num_steps+1):

                 if step % display_step == 0:
                     # Run optimization and calculate batch loss and accuracy
                     _, loss, acc = sess.run([train_op, loss_op, accuracy])
                     print('Step')
                     print("Step " + str(step) + ", Minibatch Loss= " + \
                           "{:.4f}".format(loss) + ", Training Accuracy= " + \
                           "{:.3f}".format(acc))
                 else:
                     # Only run the optimization op (backprop)
                     sess.run(train_op)

             print("Optimization Finished!")

             # Save your model
             #saver.save(sess, 'my_tf_model')
```

```
Step
Step 500, Minibatch Loss= 4.6192, Training Accuracy= 0.031
Step
Step 1000, Minibatch Loss= 4.6240, Training Accuracy= 0.031
Step
Step 1500, Minibatch Loss= 4.5927, Training Accuracy= 0.031
Step
Step 2000, Minibatch Loss= 4.5935, Training Accuracy= 0.000
Step
Step 2500, Minibatch Loss= 4.6218, Training Accuracy= 0.000
Step
Step 3000, Minibatch Loss= 4.6296, Training Accuracy= 0.000
Step
Step 3500, Minibatch Loss= 4.4787, Training Accuracy= 0.031
Step
Step 4000, Minibatch Loss= 4.3978, Training Accuracy= 0.000
Step
```

```
Step 4500, Minibatch Loss= 4.3245, Training Accuracy= 0.062
Step
Step 5000, Minibatch Loss= 3.8302, Training Accuracy= 0.188
Optimization Finished!
ERROR:tensorflow:Exception in QueueRunner: Run call was cancelled
ERROR:tensorflow:Exception in QueueRunner: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim
ERROR:tensorflow:Exception in QueueRunner: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim
ERROR:tensorflow:Exception in QueueRunner: Enqueue operation was cancelled
         [[Node: input_producer/input_producer/input_producer_EnqueueMany = QueueEnqueueManyV2[T
ERROR:tensorflow:Exception in QueueRunner: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim


Exception in thread QueueRunnerThread-batch/fifo_queue-batch/fifo_queue_enqueue:
Traceback (most recent call last):
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 914
    self.run()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 862
    self._target(*self._args, **self._kwargs)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    enqueue_callable()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    target_list_as_strings, status, None)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.CancelledError: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim

Exception in thread QueueRunnerThread-batch/fifo_queue-batch/fifo_queue_enqueue:
Traceback (most recent call last):
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 914
    self.run()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 862
    self._target(*self._args, **self._kwargs)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    enqueue_callable()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    target_list_as_strings, status, None)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.CancelledError: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim

Exception in thread QueueRunnerThread-batch/fifo_queue-batch/fifo_queue_enqueue:
Traceback (most recent call last):
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 914
```

```
    self.run()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 862
    self._target(*self._args, **self._kwargs)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    enqueue_callable()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    target_list_as_strings, status, None)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.CancelledError: Run call was cancelled

Exception in thread QueueRunnerThread-input_producer/input_producer-input_producer/input_produce
Traceback (most recent call last):
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 914
    self.run()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 862
    self._target(*self._args, **self._kwargs)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    enqueue_callable()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    target_list_as_strings, status, None)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.CancelledError: Enqueue operation was cancelled
         [[Node: input_producer/input_producer/input_producer_EnqueueMany = QueueEnqueueManyV2[T

Exception in thread QueueRunnerThread-batch/fifo_queue-batch/fifo_queue_enqueue:
Traceback (most recent call last):
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 914
    self.run()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\threading.py", line 862
    self._target(*self._args, **self._kwargs)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    enqueue_callable()
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    target_list_as_strings, status, None)
  File "c:\users\tom13\appdata\local\conda\conda\envs\tensorflow-gpu\lib\site-packages\tensorflo
    c_api.TF_GetCode(self.status.status))
tensorflow.python.framework.errors_impl.CancelledError: Enqueue operation was cancelled
         [[Node: batch/fifo_queue_enqueue = QueueEnqueueV2[Tcomponents=[DT_FLOAT, DT_INT32], tim
```

Results

Analysis

3.5 Experiment 5 - Retrain ImageNet Inception V3 Model

Overview

For my fifth experiment, I decided to take inspiration from Yanai and Kawano [24], where pre-training was used training a model for food classification. In order to achieve this, I retrained the final layer of the Inception V3 model which was trained on the ImageNet dataset. This is called Transfer Learning. I followed the tutorial by Google on the tensorflow website for direction on this process *How to Retrain Inception's Final Layer for New Categories* [9].

Firstly, in order to retrain the final layer of a model, a dataset must be prepared in the correct way. I used the Food-101 dataset Bossard, Guillaumin, and Van Gool [1] which I will analyse in a later section. The dataset must be structured so that there is a separated directory for each class with the directory name as the class name. These directories should contain all the images for this class.

Once this dataset has been set up correctly, a directory can be found on github which contains the necessary files for this tutorial. When the directory has been downloaded, the following command can be ran:

```
python tensorflow/examples/image_retraining/retrain.py \ --image_dir
~/dataset_directory
```

The first thing that the script will do is create bottleneck files for the images. A bottleneck is a term used to define the final layer before the output layer. This is so that for each image, we do not have to push it through the entire network during training *How to Retrain Inception's Final Layer for New Categories* [9].

After, the bottlenecks are created, the training can be completed. The images

are split into three sub directories of training, testing and validation. By default, these images are split into percentages of 80%, 10% and 10% respectively. The model is trained at a default of 4000 steps.

At the final stage of the script, the model is run on a batch of test images not yet seen and a final test accuracy is displayed. This can be seen in the Script section below.

The command used for using this model once it is trained is:

```
python tensorflow/examples/label_image.py
    --graph=/tmp/output_graph.pb
--labels=/tmp/output_labels.txt --input_layer=Mul
    --output_layer=final_result
--input_mean=128 --input_std=128 --image=~/image_directory
```

Network Architecture

The Inception V3 model network architecture was used for this experiment. The Inception V3 architecture was created by building on the existing Inception model aimed at efficient image classification Szegedy et al. [23].

Dataset

The dataset used for this experiment is the Food-101 dataset **Food 101**. This dataset has 101 classes with 1000 images for each class.

Libraries

Tensorflow and Numpy were used to run this scipt.

Script

The following snippets of code are from the retrain.py script.

```
# Add the new layer that we'll be training.
(train_step, cross_entropy, bottleneck_input, ground_truth_input,
final_tensor) = add_final_training_ops(
          len(image_lists.keys()), FLAGS.final_tensor_name,
          bottleneck_tensor,
          model_info['bottleneck_tensor_size'],
          model_info['quantize_layer'])

# Create the operations we need to evaluate the accuracy of our new
    layer.
evaluation_step, prediction = add_evaluation_step(
final_tensor, ground_truth_input)

# Merge all the summaries and write them out to the summaries_dir
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
                                     sess.graph)

validation_writer = tf.summary.FileWriter(
    FLAGS.summaries_dir + '/validation')

# Set up all our weights to their initial default values.
init = tf.global_variables_initializer()
sess.run(init)
```

```
# We've completed all our training, so run a final test evaluation on
# some new images we haven't used before.
test_bottlenecks, test_ground_truth, test_filenames = (
    get_random_cached_bottlenecks(
        sess, image_lists, FLAGS.test_batch_size, 'testing',
        FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
        decoded_image_tensor, resized_image_tensor,
            bottleneck_tensor,
        FLAGS.architecture))
```

Figure 3.1: Pizza

```
test_accuracy, predictions = sess.run(
   [evaluation_step, prediction],
   feed_dict={bottleneck_input: test_bottlenecks,
        ground_truth_input: test_ground_truth})
tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
               (test_accuracy * 100, len(test_bottlenecks)))
```

Results

The final test accuracy for this retrained model was 54.8%.

For example, an image of pizza, see 3.1 was fed into the model with the followng results:

- pizza 0.925
- pancakes 0.008
- nachos 0.007
- beef carpaccio 0.006
- tiramisu 0.004

Analysis

3.6 Experiment 6 - Retrain with Extended Dataset

Overview

As the Food-101 dataset mostly consisted of meals Bossard, Guillaumin, and Van Gool [1], I decided to extend the dataset slightly be including some single foods such as:

- cheese
- grapes
- banana
- apple
- orange
- spaghetti
- roll

In order to collect these images, I used the ImageNet repository to search for these foods individually and then downloaded the subset of images to be included with Food 101 Deng et al. [2]. I ran the retrain.py script on the extended dataset as in Experiment 5.

Network Architecture

The Inception V3 Model was used for this experiment.

Dataset

In this experiment I extended the Food-101 dataset.

Libraries

Tensorflow and numpy are used in the retrain.py script.

Script

As seen in Experiment 5.

Results

For this model, I achieved an accuracy of 55.3%.

For example, an image of a banana, see 3.2 was fed into the model with the followng results:

- banana 0.9962
- orange 0.0009
- cheese 0.0003
- frozen yoghurt carpaccio 0.0002
- churros 0.0001

Analysis

The slight increase in accuracy in Experiment 5, from 54.8% to 55.3%, makes sense. Since the model was pre-trained using the ImageNet dataset and all the new images I used were from ImageNet, we would expect a higher classification accuracy on the new additions to the dataset. This would overall increase the average classification accuracy.

Figure 3.2: Banana

3.7 Experiment 7 - Retrain with Parameter Tuning

Overview

Within the retrain.py script *How to Retrain Inception's Final Layer for New Categories* [9], as mentioned in previous experiments, there are various parameters that can be set and changed. I tested out a few combinations of the parameters to see if I could increase the test accuracy of the model.

Network Architecture

The Inception V3 architecture was used for this model.

Dataset

The Food-101 dataset Bossard, Guillaumin, and Van Gool [1] with additional classes, as per experiment 6, was used for this model.

Figure 3.3: Graph of accuracy of the test dataset during training

Libraries

The libraries in use for this experiment are tensorflow and numpy.

Script

Script as seen in experiment 5 but with some additions to the command as seen below:

```
python tensorflow/examples/image_retraining/retrain.py \ --image_dir
~/dataset_directory \ --how_many_training_steps 4000 \
    --learning_rate 0.01 \
--testing_percentage 10 \ --validation_percentage 10
```

Some further parameters could be set such as:

- –flip_left_right
- –random_crop
- –random_scale
- –random_brightness

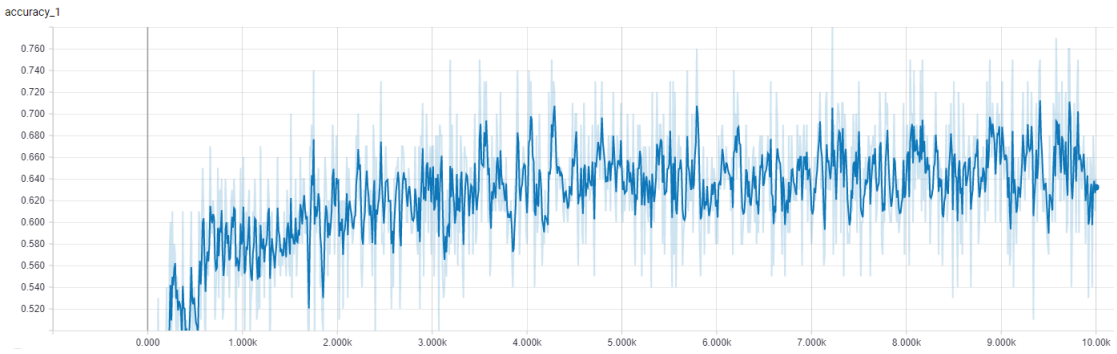


Figure 3.4: Graph of accuracy of the validation dataset during training

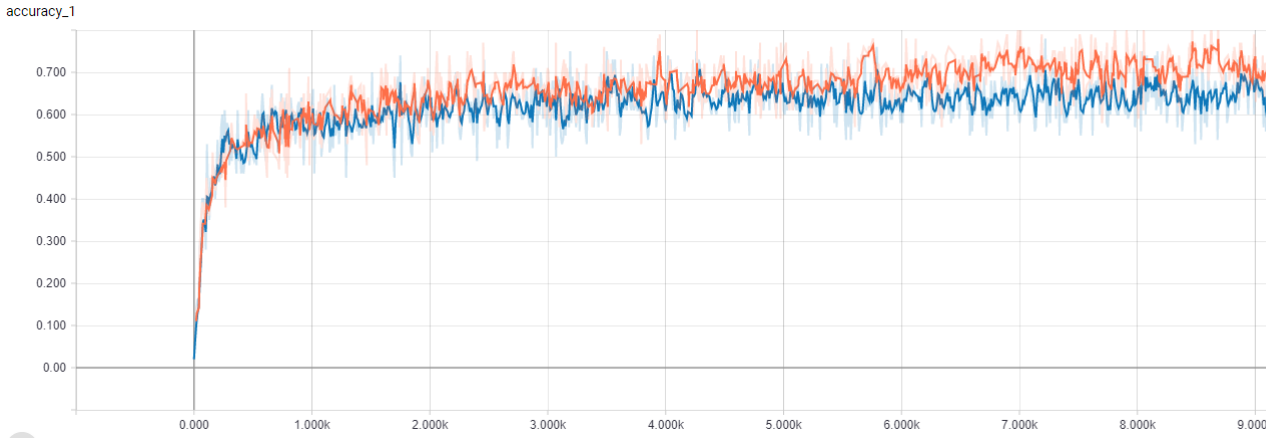


Figure 3.5: Comparison of accuracy

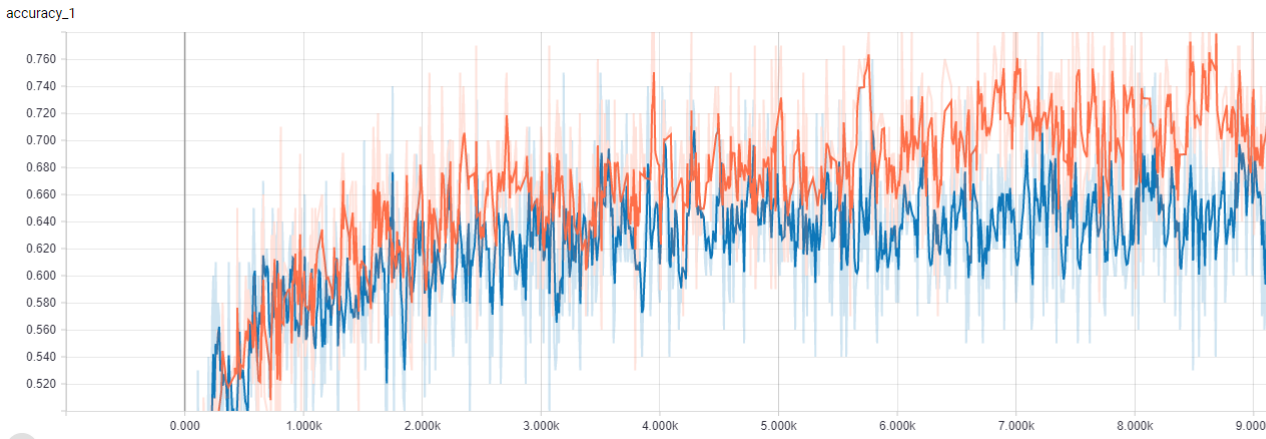


Figure 3.6: Comparison of accuracy

Table 3.1: Comparison of parameters

Parameter Tuning	Steps	Learning Rate	Test %	Validation %	Results
Configuration 1	8000	default	default	default	59.1%
Configuration 2	8000	0.1	default	default	65.8%
Configuration 3	10000	0.1	default	default	66.3%
Configuration 4	12000	0.1	default	default	66.6%
Configuration 5	10000	0.2	default	default	66.0%
Configuration 6	10000	0.1	15	15	66.3%

Results

The results of each set of parameters can be seen in Table 3.1. The set of parameters that seem to be the most effective are 10000 steps with a 0.1 learning rate. Graphs of this model can be see in Figures 3.4 and 3.3. These are based on the validation set and then the test set respectively. A side by side comparision can also be seen in Figures 3.5 and 3.6 where orange is for during training and blue for the validation set.

Analysis

There were two separate factors that each increased classification accuracy of about 5% each. These were training steps and learning rate.

Training steps are related to the number of images so before, when the training steps were at 4000, not all of our training images were being used. As the steps were increased twofold we saw a 3.8% increase in accuracy.

Another parameter that increased accuracy significantly was learning rate. The default learning rate is 0.01 which I increased to 0.1. This resulted in an increase of 6.7%. This is most likely due to the fact that since we are only looking at the last layer, we can afford to change the weights more significantly.

3.8 Experiment 8 - Sliding Window

Overview

In the preious experiments, I have looked at the one-shot approach to food image classification. That is, the model will give a prediction of the most likely food item in that image. This is a problem when there are multiple food in an image, see 3.7. There are a few options to combat this problem. Firstly, I could detect objects in the image, segment the image according to these objects and then run each segment through the model. A simple approach to this would be to segment the image into a number of sections and then run each section through the model. In order to follow the latter approach, I used a sliding window approach. This sliding window would move across the image and classify the segment of the image in the window. I had three options for window sliding shape as defined by a command line argument.

```
python sliding_window.py --image=~/image_dir --window_shape grid
```

There are three options for window shape:

- Grid based window as per 3.8
- Row based window as per 3.9
- Column based window as per 3.10

Libraries

For this experiment Tensorflow provided the classification of each segment while also helping with resizing along with Numpy. OpenCv was used to implement the sliding window as per *Sliding Windows for Object Detection with Python and OpenCV* [21]

Figure 3.7: Bowl of fruit

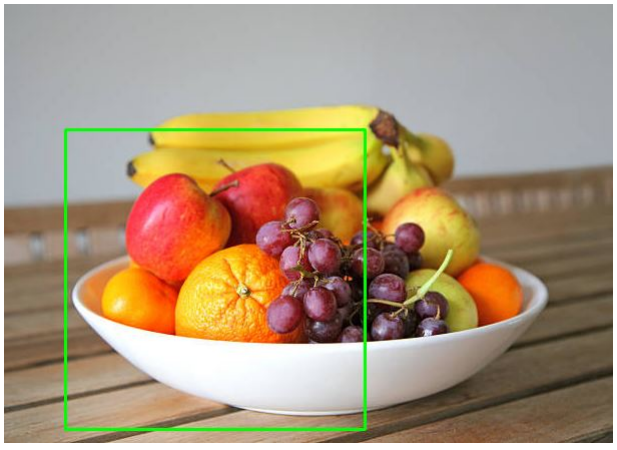


Figure 3.8: Grid based window

Figure 3.9: Row based window

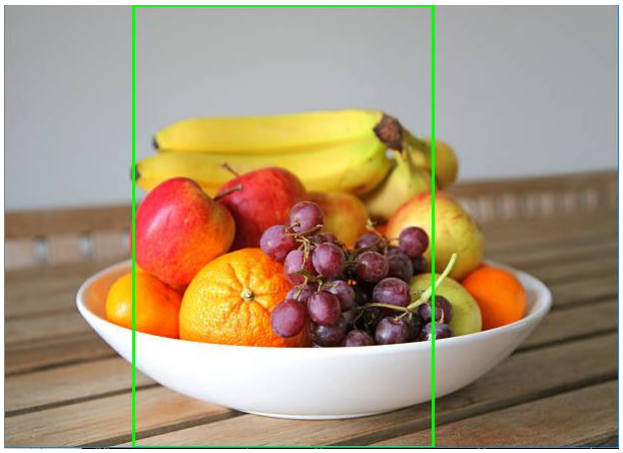


Figure 3.10: Column Based Window

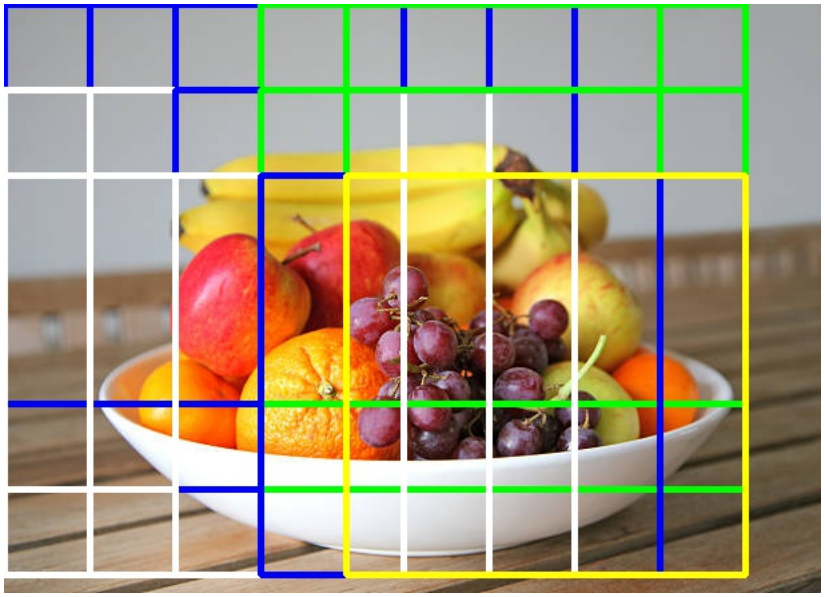


Figure 3.11: Fruit with Color Overlay

Script

There were four main elements to the script. Firstly, extracting a window to be classified. Secondly, resizing the image to be compatible with the Tensorflow model. Thirdly, running the window through the Tensorflow model and finally saving a new image with a coloured overlay of classifications.

Extracting the window from the image

```
# loop over the image pyramid
for resized in pyramid(image, scale=1.5):
      # loop over the sliding window for each layer of the pyramid
      for (x, y, window) in sliding_window(resized, stepSize=32,
          windowSize=(winW, winH)):
         # if the window does not meet our desired window size,
             ignore it
           if window.shape[0] != winH or window.shape[1] !=winW:
              continue
```

```
def sliding_window(image, stepSize, windowSize):
   # slide a window across the image
   for y in xrange(0, image.shape[0], stepSize):
      for x in xrange(0, image.shape[1], stepSize):
         # yield the current window
         yield (x, y, image[y:y + windowSize[1], x:x + windowSize[0]])
```

Resizing the window

```
window = cv2.resize(window, (299, 299))
```

```
resized_image = tf.reshape(image, [1, input_height, input_width, 3])
resized = tf.image.resize_area(resized_image, [input_height,
    input_width])
normalized = tf.divide(tf.subtract(resized, [input_mean]),
    [input_std])
```

Running the Tensorflow model

```
with tf.Session() as sess:
   numpy_image = sess.run(normalized)

with tf.Session(graph=graph) as sess:
    results = sess.run(output_operation.outputs[0],
                       {input_operation.outputs[0]: numpy_image})
   probabilities = np.squeeze(results)
```

Saving the image with colour overlay

As seen in Figure 3.11, each square represents a window and each colour is for a different classification. Blue is for an apple, yellow for banana, green for grape, white for orange and black if an unexpected prediction is made.

Table 3.2: My caption

Food type	No. of Top-1 Classifications
Apple	5
Banana	1
Grape	4
Orange	5

Table 3.3: My caption

Food type	No. of Top-1 Classifications
Apple	1
Banana	0
Grape	0
Orange	0
Other	3

```
cv2.rectangle(display_image, (x, y), (x + winW, y + winH),
   colour_dict.get(top1,
(0,0,0)), 4)
```

Results

Grid based window

The grid based window resulted in fifteen separate classification. As seen in 3.7, there are multiple fruits in the image. Of these fruits, our model is trained on four, apple, banana, orange and grapes. This method classified all four to Top-1 accuracy at least once each. This method took 42.8 seconds to run.

Table 3.4: My caption

Food type	No. of Top-1 Classifications
Apple	3
Banana	1
Grape	0
Orange	0
Other	1

Row based window

The row based method resulted in four predictions as follows in 3.3. Out of these four predictions, only one classified a known fruit at Top-1 accuracy, an apple. An apple was also predicted to Top-5 accuracy in another instance. The runtime of this method was 16.1 seconds.

Column based window

The column based window approach had five total predictions and ran for a total of 13 seconds. As seen in 3.4, two out of four known fruits were classified to a Top-1 accuracy with all other fruits predicted to Top-5 accuracy. Only one Top-1 prediction did not contain a correct fruit.

Analysis

These results are very interesting because while a banana was only predicted to Top-1 accuracy once in grid based, once in column based and zero times in row based, if the whole image is ran through the model, a banana is at the Top-1 accuracy.

3.9 Experiment 9 - MobileNet

Overview

Due to the fact that the end goal for this project is to have a smartphone application that a user can use to keep track of their calorie measurement, there are a couple of options in how to achieve this. Firstly, an image can be taken on the phone and sent to a server to run a classification algorithm. Secondly, a model can be stored on the phone for computation. I decided to train the model, using transfer learning as before, but on a different architecture, MobileNet.

Network Architecture

The network architecture used for this experiment is MobileNet *How to Retrain Inception's Final Layer for New Categories* [9]. This architecture is designed to be smaller so that it can be used on smartphones which have less powerful resources available.

Dataset

The Food 101 dataset Bossard, Guillaumin, and Van Gool [1] with added classes was used for this experiment.

Libraries

Tensorflow and numpy.

Script

The retrain.py script *How to Retrain Inception's Final Layer for New Categories* [9] was used, with a different command paramater.

```
python tensorflow/examples/image_retraining/retrain.py \ --image_dir
~/dataset_directory \ --architecture mobilenet_1.0_224 \
--how_many_training_steps 10000 \ --learning_rate 0.1
```

Results

The final test accuracy of this model came to 50.2%.

Analysis

There was a decrease of 16.1% in this model to the highest accuracy from experiment 7. This is due to the smaller architecture which is aimed to be faster and smaller with the expected decrease in accuracy.

3.10 Experiment 10 - Recursive Refinement

Overview

After the sliding window code was run on Figure 3.7 in experiment 8, it was observed that a sliding window was predicting grapes correctly in regions that contained a bunch of grapes. Since it would make sense that the model would be able detect an individual grape, it was decided that I would run recursive refinement on a window that contained a grape. Due to the model requiring a 299 x 299 image size, the window could only be refined once as very small segments could not be resized up to 299 x 299. I decided to use a window of 70 x 70.

Script

As you would think with recursive refinement, a recursive function would used, but I found this unnecessary due to image size restrictions. Instead, a condi-

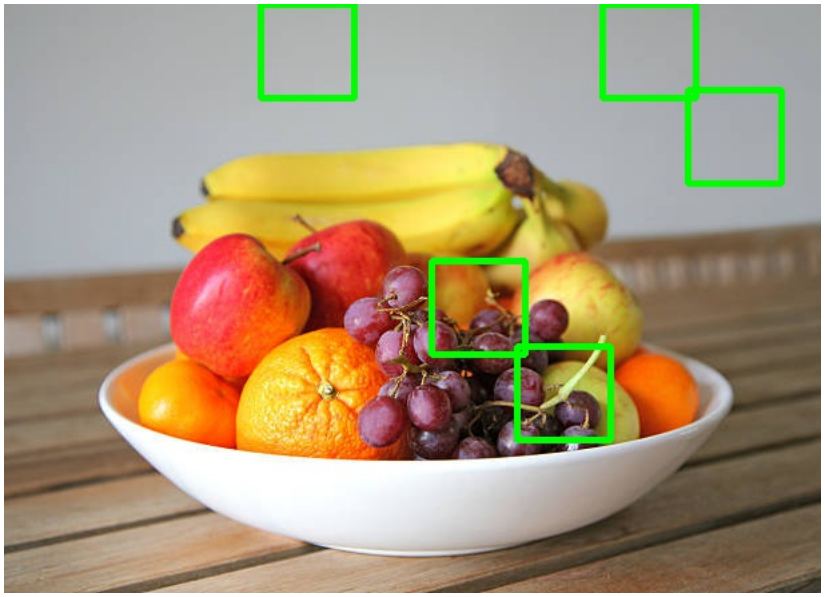


Figure 3.12: Recursive refinement 1

tional for loop was added to the existing code.

```
if top1 == "grape" and window_shape == "grid" and rr_grape:
        for (x_grape, y_grape, grape_window) in
            sliding_window(window_resized, stepSize=64,
            windowSize=(70, 70)):
           #reshape to square
           grape_window_resized = cv2.resize(grape_window, (299,
               299))
           top1_grape = subSample.classify(grape_window_resized,
               window_shape)
           if top1_grape == "grape":
              cv2.rectangle(display_image, (x_grape + x, y_grape +
                  y), (x_grape + x + 70, y_grape + y + 70),
                  colour_dict.get(top1, (0,0,0)), 4)
              #cv2.imshow("Window", grape_window_resized)
              cv2.waitKey(1)
time.sleep(0.025)
```

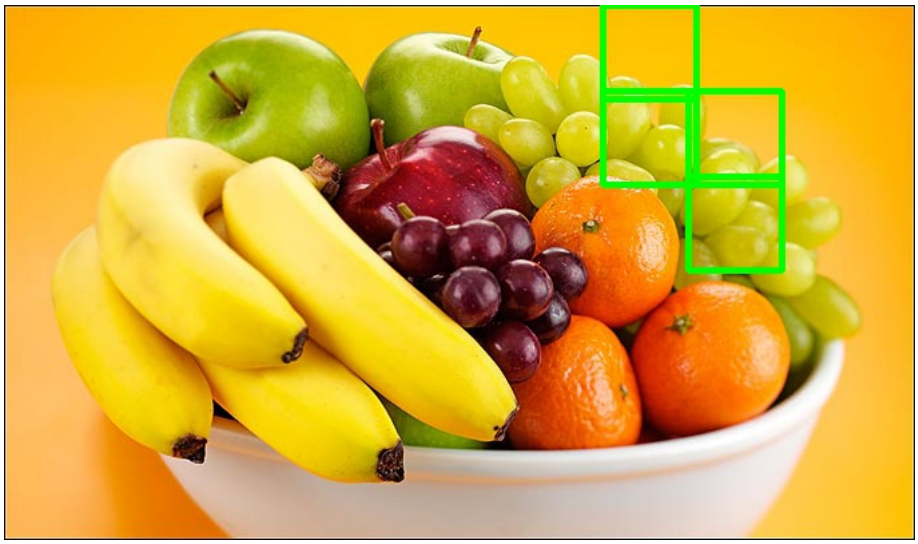


Figure 3.13: Recursive refinement 2

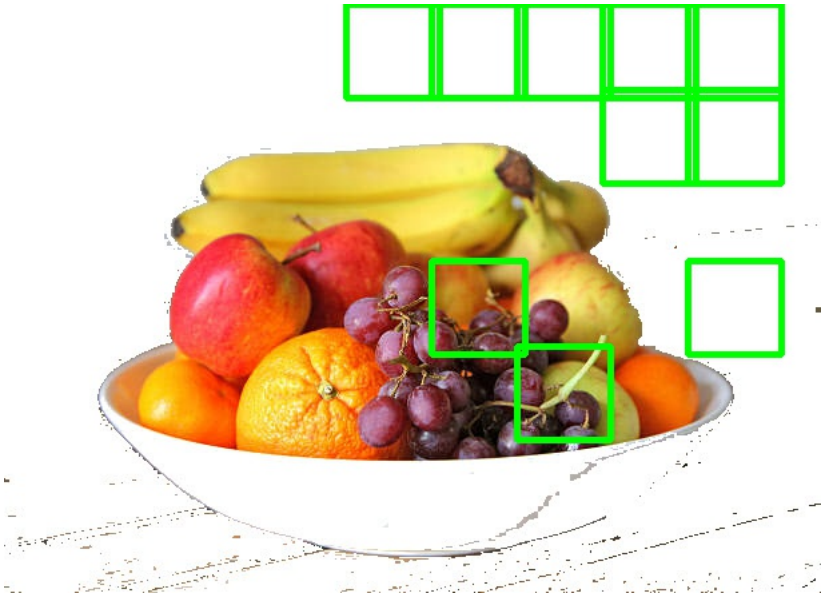


Figure 3.14: Recursive refinement 3

Figure 3.15: Bowl of fruit with background removed

Results

Some very interesting results on three separate images. Two new images are seen here which we will explore in future experiments. In all three images, while we are getting some expected predictions, grapes are being classified in locations that have nothing resembling a grape. These can be viewed in Figures 3.12, 3.13 and 3.14.

Analysis

3.11 Experiment 11 - Impact of Backround

Overview

As we can see in Experiment 9, using sliding windows to classify many sections of an image, there were some cases where some unexpected predictions were

Table 3.5: Comparison of fruit image sliding window results with and without bakground

Food type	Grid	Row	Column	Filled Grid	Filled Row	Filled Column
Apple	5	1	3	10	0	4
Banana	1	0	1	0	0	0
Grape	4	0	0	1	0	1
Orange	5	0	0	3	0	0
Other	0	3	1	1	4	0

made. Due to this, the decision was made to analyse the effect the background of the image has on its classification. The sliding window code was then ran on a new image. This new image was the same fruit bowl as used previously but the background was filled in as white as per Figure 3.15.

Results

Grid

For grid based sliding window approach, the results turned out to be less sucessful than with the background. In this experiment, fourteen out of fifteen of top-1 classification were of an expected food type rather than fifteen out of fifteen with the background present. We expected the food types of apple, orange, grape and banana to appear in this image but while a banana was detected to a top-5 accuracy on a few occasions it was never predicted to a top-1 accuracy. The contrast between the image results can be seen in Table 3.5.

Row

The row based sliding window again had worse result than its counterpart, with zero out of four correct classifications as opposed to one. In this case, an orange appeared at top-5 accuracy once. The most common prediction was ice-cream which appeared at top-1 accuracy in three out of four instancs.

Figure 3.16: Alternative Bowl of fruit

Column

In contrast to our previous two methods of sliding window, this method outperformed its counterpart with correct predictions of all five windows while before we only had four out of five. In this experiment, a aple was predicted four times and a grape once, with all correct fruits appearing to top-5 accuracy.

Analysis

It is quite interesting that removing the background to the image reduced our accuracy overall. Many white foods were classified instead which makes sense due the impact of colour expected.

3.12 Experiment 12 - Alternative Test Image

Overview

In our three previous sliding window oriented experiments, we had only used a single image. In order to see whether this image had biases unknown to us,

Table 3.6: Comparison of fruit bowl images

Food type	Grid	Row	Column	New Grid	New Row	New Column
Apple	5	1	3	4	1	0
Banana	1	0	1	5	0	5
Grape	4	0	0	2	1	0
Orange	5	0	0	0	0	0
Other	0	3	1	1	2	1

I decided to use another fruit bowl image. This image was selected as fruit took up a larger portion of the image as seen in Figure 3.16

Results

Grid

The performance of this experiment was slightly worse than with the previously used image. When I ran the grid based sliding window on Figure 3.16, fourteen out of fifteen predictions had an expected value. Out of the fourteen predictions orange was not predicted to top-1 accuracy at all. This can be seen, in comparision to previously used image, in Table 3.6.

Row

In the column based window for the new fruit image, the results were not very successful as has been the trend for most row based classification. Two out of four predictions had an expected value at top-1 accuracy.

Column

The column based approach had a similar result to its counterpart in that only one of its predictions was unexpected. Although, due to the size of the new image, another column was created and thus has a better overall accuracy.

Analysis

A possible reason that an orange was not classified in any of these images is because in Figure 3.16, a more mandarin food is displayed.

3.13 Experiment 13 - Scale?

Overview

Network Architecture

Dataset

API's

Script

Results

Analysis

Chapter 4

Empirical Studies with Food Images

Chapter 5

Discussion and Conclusion

Bibliography

[1] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. "Food-101 – Mining Discriminative Components with Random Forests". In: *European Conference on Computer Vision*. 2014.

[2] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[3] Jeffrey Donahue. "Transferrable Representations for Visual Recognition". PhD thesis. EECS Department, University of California, Berkeley, May 2017.

[4] Ross Girshick. "Fast R-CNN". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2015.

[5] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[6] Kaiming He et al. "Mask R-CNN". In: *arXiv preprint arXiv:1703.06870* (2017).

[7] Ye He et al. "Food image analysis: Segmentation, identification and weight estimation". In: *Multimedia and Expo (ICME), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1–6.

[8] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. "Diagnosing Error in Object Detectors". In: *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 340–353.

[9] *How to Retrain Inception's Final Layer for New Categories*. `https://www.tensorflow.org/tutorials/image_retraining`. Accessed: 2018-01-24.

[10] Taichi Joutou and Keiji Yanai. "A food image recognition system with multiple kernel learning". In: *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE. 2009, pp. 285–288.

[11] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. "Food detection and recognition using convolutional neural network". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 1085–1088.

[12] Chang Liu et al. "Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment". In: *International Conference on Smart Homes and Health Telematics*. Springer. 2016, pp. 37–48.

[13] George F. Luger. *Artificial Intelligence. Structures and Strategies for Complex Problem Solving/Luger GF*. Pearson Education Limited, 2005.

[14] Dongyuan Mao, Qian Yu, and Jingfan Wang. "Deep Learning Based Food Recognition". In: ().

[15] Tom M. Mitchell. *Machine Learning*. International Edition 1997. New York: The McGraw-Hill Companies, Inc., 1997, pp. 81–126.

[16] Parisa Pouladzadeh, Shervin Shirmohammadi, and Rana Al-Maghrabi. "Measuring calorie and nutrition from food image". In: *IEEE Transactions on Instrumentation and Measurement* 63.8 (2014), pp. 1947–1956.

[17] Parisa Pouladzadeh, Shervin Shirmohammadi, and Abdulsalam Yassine. "Using graph cut segmentation for food calorie measurement". In: *Medical Measurements and Applications (MeMeA), 2014 IEEE International Symposium on*. IEEE. 2014, pp. 1–6.

[18] Parisa Pouladzadeh et al. "A novel SVM based food recognition method for calorie measurement applications". In: *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*. IEEE. 2012, pp. 495–498.

[19] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Neural Information Processing Systems (NIPS)*. 2015.

[20] Evan Shelhamer, Jonathan Long, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 39.4 (Apr. 2017), pp. 640–651.

[21] *Sliding Windows for Object Detection with Python and OpenCV*. `https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/`. Accessed: 2018-01-30.

[22] *Support Vector Machines for Machine Learning*. `https://machinelearningmastery.com/support-vector-machines-for-machine-learning/`. Accessed: 2017-12-20.

[23] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[24] Keiji Yanai and Yoshiyuki Kawano. “Food image recognition using deep convolutional network with pre-training and fine-tuning”. In: *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[25] Fengqing Zhu et al. “Segmentation assisted food classification for dietary assessment”. In: *Computational Imaging IX*. Vol. 7873. International Society for Optics and Photonics. 2011, 78730B.

Appendix A

Appendix

A.1 Image Segmentation

Fully Convolutional Neural Networks for Semantic Segmentation

There has been a very interesting paper from UC Berkeley focused on using Full Convolutional Networks for semantic segmentation Shelhamer, Long, and Darrell [20]. Fully Convolutional Networks (FCN) do not have any fully connected layers. They are replaced with more filtering layers. Nvidia Digits have a semantic segmentation implementation based off the work of this paper.

They took this approach because "feedforward computation and backpropogation are much more efficient when computer layer-by-layer over an entire image instead of independently patch-by-patch" Shelhamer, Long, and Darrell [20]. This was also because they were focused on object detection. Normal classifiers do not work very well when they are to classify more than one subject in an image and image segmentation was a way to solve this.

There are a set of steps you can follow to turn a CNN into a FCN for semantic segmentation as follows ie. change to a convolutional layer from a fully connected one:

- The size of the filters must be set to the size of the input layers.
- For every neuron in the fully connected layer, have a filter.

Segmentation

Graph Based Segmentation

Graph cut segmentation has been used extensively in image segmentation. OpenCV has an implementation of a graph cut algorithm called grabcut which has been used to segment food on occasion Pouladzadeh, Shirmohammadi, and Yassine [17].

Table A.1: FCN Resulys Shelhamer, Long, and Darrell [20]

	FCN
VOC11 mean IU	62.7
VOC12 mean IU	62.2
PASCAL VOC10 pixel acc.	67.0
PASCAL VOC10 mean acc.	50.7
PASCAL VOC10 mean IU	37.8
PASCAL VOC10 f.w. IU	52.5

Table A.2: Results

	Single Food Portion	Non-mixed Food	Mixed Food
Color and Texture	92.21	N/A	N/A
Graph Based	95 (3% increase)	5% increase	15% increase

According to Pouladzadeh, Shirmohammadi, and Yassine [17], "Graph cut based method is well-known to be efficient, robust, and capable of finding the best contour of objects in n image, suggesting it to be a good method for separating food portions in a food image for calorie measurement". Along with the graph cut segmentation algorithm, this research team also used color and texture segmentation. Gabor filters were used to measure texture features Pouladzadeh, Shirmohammadi, and Yassine [17]. When color and texture segmentation was applied, the method came into difficulty with mixed foods but by applying graph cut segmentation, clearer object boundaries were shown.

In conclusion, the accuracy of the classification increased when using graph based segmentation rather than color and texture as seen in A.2.

Local Variation Framework

Another paper was published in which the research team attempted to create a food calorie extimation system He et al. [7]. This system would compromise of three steps, image segmentation, image classification and weight estimation. For the segmentation module, a local variation approach to segmentation was

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 (Girshick et al.)	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA (Uijlings et al. 2013)	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets (Wang et al. 2013a)	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM (Fidler et al. 2013)	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

Figure A.1: Detection Average Precision Donahue [3]

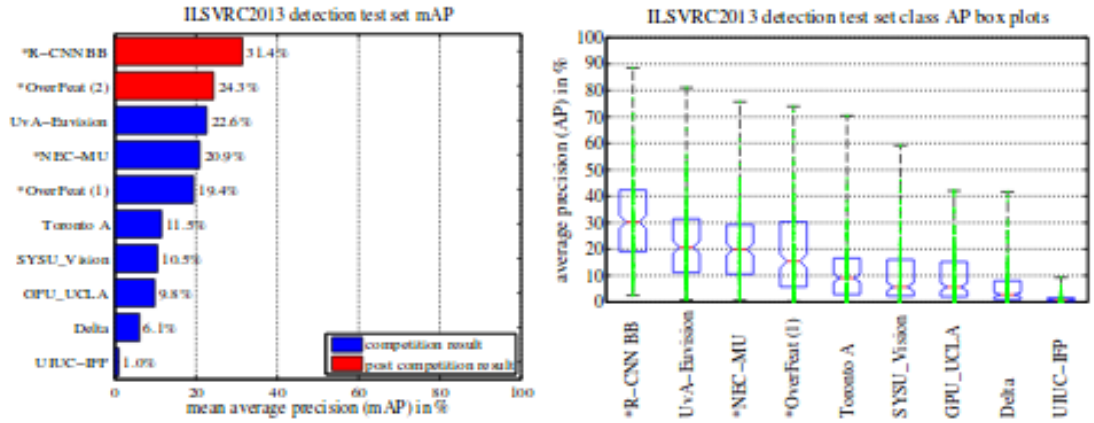


Figure A.2: Mean Average Precision Donahue [3]

performed. Local variation is by which intensity differences between neighbouring pixels is measured. This is a type of graph based segmentation.

The team also carried out some segmentation refinement when the segmentation algorithm had been performed. This consisted of removed small segments (defined as less than 50 pixels) and trying to prevent over and under segmentation. After classification was performed on each segment, segments with low confidence values were removed He et al. [7].

Conclusion

Both of the above papers of Pouladzadeh, Shirmohammadi, and Yassine [17] and He et al. [7] used a graph based segmentation. The first paper used a more generic implementation while the second used a local variation framework. Both methods provided successful results in the image segmentation process.

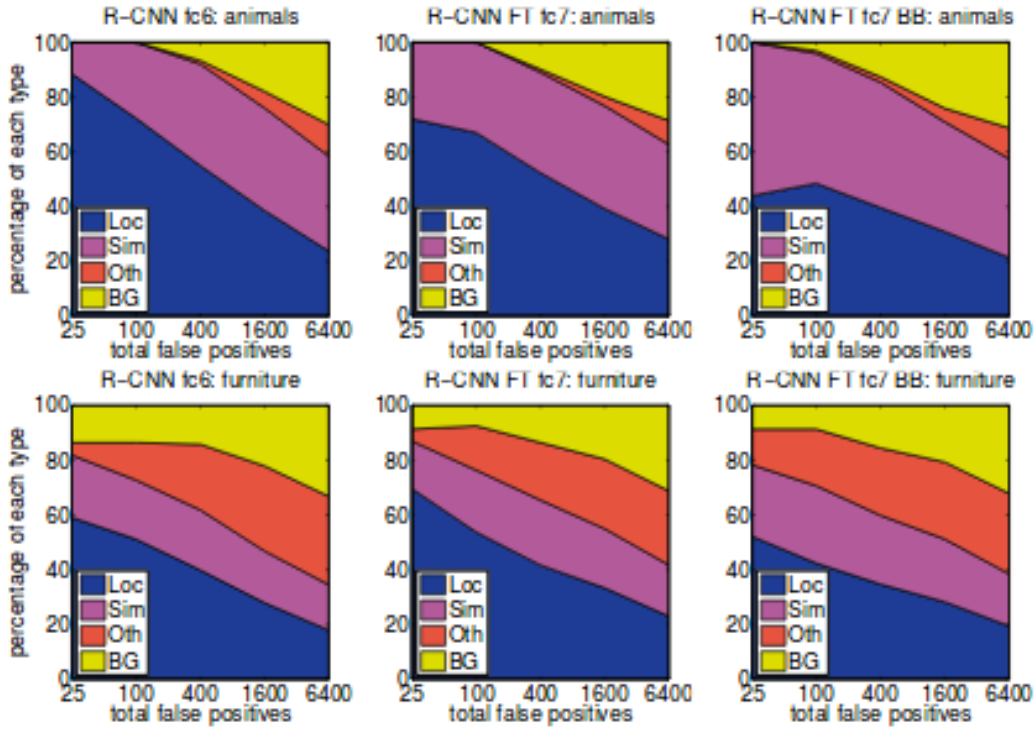


Figure A.3: Distribution of top-ranked false positives Donahue [3]

	full R-CNN		*fg* R-CNN		*full+fg* R-CNN	
O_2P (Carreira et al., 2012)	fc_6	fc_7	fc_6	fc_7	fc_6	fc_7
46.4	43.0	42.5	43.7	42.1	**47.9**	45.8

Figure A.4: Segmentation Mean Accuracy Donahue [3]

VOC 2011 val	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
O_2P Carreira et al. 2012	**84.0**	**69.0**	21.7	47.7	42.2	42.4	**64.7**	**65.8**	57.4	**12.9**	37.4	20.5	43.7	35.7	52.7	51.0	**35.8**	**51.0**	28.4	59.8	49.7	46.4
full R-CNN fc_6	81.3	59.2	23.9	42.9	40.7	38.8	59.2	56.5	53.2	11.4	34.6	16.7	48.1	37.0	51.4	46.0	31.5	44.0	24.3	53.7	51.1	43.0
full R-CNN fc_7	81.0	52.8	**25.1**	43.8	40.5	42.7	55.4	57.7	51.3	8.7	32.5	11.5	48.1	37.0	50.5	46.4	30.2	42.1	21.2	57.7	**56.0**	42.5
fg R-CNN fc_6	81.4	54.1	21.1	40.6	38.7	**53.6**	59.9	57.2	52.5	9.1	36.5	**23.6**	46.4	38.1	53.2	51.3	32.2	38.7	**29.0**	53.0	47.5	43.7
fg R-CNN fc_7	80.9	50.1	20.0	40.2	34.1	40.9	59.7	59.8	52.7	7.3	32.1	14.3	48.8	42.9	54.0	48.6	28.9	42.6	24.9	52.2	48.8	42.1
full+fg R-CNN fc_6	81.1	60.4	23.2	48.4	**47.3**	52.6	61.6	60.6	**59.1**	10.8	**45.8**	20.9	**57.7**	43.3	**57.4**	**52.9**	34.7	48.7	28.1	60.0	48.6	**47.9**
full+fg R-CNN fc_7	82.3	56.7	20.6	**49.9**	44.2	43.6	59.3	61.3	57.8	7.7	38.4	15.1	53.4	**43.7**	50.8	52.0	34.1	47.8	24.7	**60.1**	55.2	45.7

Figure A.5: Per-category segmentation accuracy Donahue [3]

class	AP	class	AP	class	AP	class	AP	class	AP
accordion	50.8	centipede	30.4	hair spray	13.8	pencil box	11.4	snowplow	69.2
airplane	50.0	chain saw	14.1	hamburger	34.2	pencil sharpener	9.0	soap dispenser	16.8
ant	31.8	chair	19.5	hammer	9.9	perfume	32.8	soccer ball	43.7
antelope	53.8	chime	24.6	hamster	46.0	person	41.7	sofa	16.3
apple	30.9	cocktail shaker	46.2	harmonica	12.6	piano	20.5	spatula	6.8
armadillo	54.0	coffee maker	21.5	harp	50.4	pineapple	22.6	squirrel	31.3
artichoke	45.0	computer keyboard	39.6	hat with a wide brim	40.5	ping-pong ball	21.0	starfish	45.1
axe	11.8	computer mouse	21.2	head cabbage	17.4	pitcher	19.2	stethoscope	18.3
baby bed	42.0	corkscrew	24.2	helmet	33.4	pizza	43.7	stove	8.1
backpack	2.8	cream	29.9	hippopotamus	38.0	plastic bag	6.4	strainer	9.9
bagel	37.5	croquet ball	30.0	horizontal bar	7.0	plate rack	15.2	strawberry	26.8
balance beam	32.6	crutch	23.7	horse	41.7	pomegranate	32.0	stretcher	13.2
banana	21.9	cucumber	22.8	hotdog	28.7	popsicle	21.2	sunglasses	18.8

Figure A.6: Per-class segmentation accuracy Donahue [3]

Table A.3: Project Plan

Task/Deliverable	Deadline
Interim Report	21/12/17
Select method and paper to replicate	10/01/18
Iteration 1	???
Iteration 2	???
Iteration 3	???
Draft Final Report	08/03/18
FYP Product	10/04/18
FYP Report	17/04/18