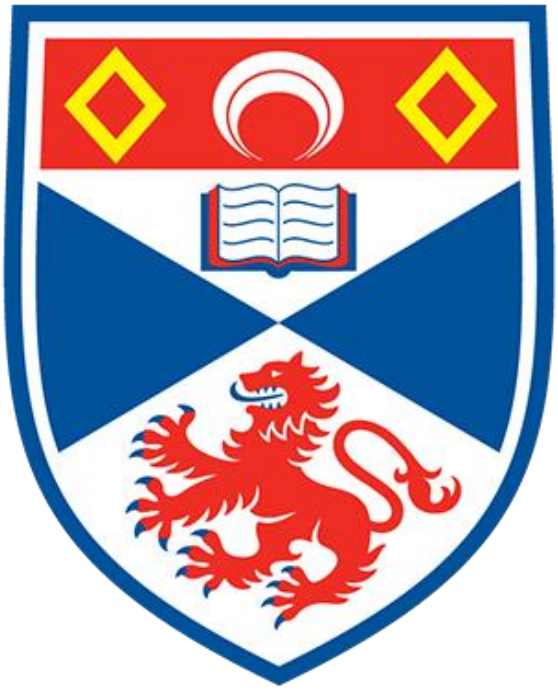
Exploring real time multi-object localisation and classification

Project in Mathematics and Statistics

By Callum McMahon



I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.

Abstract

Building and deploying a system that employs deep learning techniques for real time processing poses many technical challenges. These involve creating models that are lightweight enough to perform fast inference as well as parallelising processes to make most efficient use of the system hardware available. These ideas are explored using a real-world case study, with the aim of creating an accurate model which can compute results in real time.

The problem studied requires the use of convolutional neural networks to perform bounding box object localisation as well as multilabel classification on images coming from a video feed. The computation needs to be fast enough for results to be shown alongside the live feed whilst maintaining adequately high framerates. These results were achieved while maintaining suitable accuracy.

A copy of the code can be found at: https://github.com/CallumMcMahon/Dobble-Computer-Vision

Table of Contents

1 Introduction

The card game Dobble consists of a deck of 55 cards, each of which contains 8 symbols. The constraints on these symbols are such that any two cards contain only a single symbol in common.

The aim of the game involves every player holding a card and trying to find the matching symbol between their card and another placed centrally between everyone. They need to announce the matched symbol before moving on to the next card in their pile. Players win by making their way through the deck of cards finding the matching symbols faster than other players.

In the chaos of the game it is often difficult to verify that claimed matches were in fact correct. An extra person acting as a referee is not feasible as they would need to perform the job of all players at the same time to verify that any claimed match was in fact there. This is where a model could be used. If a computer were able to identify the matching symbols instantly as they appeared, a referee would know which symbols they should expect to hear from the players.

Convolutional neural networks will be trained with the goal of identifying the symbols contained on each card in real time from a video source such as a webcam or smart phone. From there it is trivial for a computer to find the common symbols between the cards by calculating the intersection of the two sets. The hope is that a sufficient level of accuracy can be achieved that the system can be utilised by a referee in real time.

2 Method

This section will start out with a discussion of the different tools available relevant to the task. It provides an overview of the inference they perform as well as their relative strengths and weaknesses.

2.1 What is a multilabel classifier

A multilabel classifier takes as input an image containing a variable number of objects and predicts which ones are present from a list of objects that the model was trained with. It does not distinguish scale, location or the number of occurrences of an object within the image.

2.2 What is a bounding box classifier

A bounding box classifier takes as input an image containing a variable number of objects and predicts rectangular bounding boxes which encase each object present in the image, as well as make predictions on the class of each object. The bounding boxes are constructed using only horizontal and vertical edges, and predictions become less accurate as the scale of objects decreases relative to the size of the image.

2.3 What is image segmentation

Image segmentation is the process of taking an input image and assigning a class to each pixel in the image. There is no constraint that pixels need to group together into coherent regions as each pixel is classified separately.

2.4 Pros and cons of each method

Due to the requirement of identifying the same symbol on multiple cards in the image, a multilabel classifier cannot be used on its own. The model would identify the presence of the symbol within the image but not distinguish if it appears in multiple locations.

The bounding box classifier would need to draw bounding boxes around every symbol in the image. The symbols would be quite small relative to the size of the input image thus making this a challenging task for a bounding box classifier. At such a scale, symbols are likely to not be detected.

The image segmentation classifier would produce an output which is a lot more detailed than we require. Processing would need to be performed to translate the noisy pixel classifications into distinct symbols. This would lead to computation taking longer per frame, potentially slowing down the framerate.

2.5 Overview of the chosen model pipeline

The solution proposed is to incorporate a bounding box classifier with a multilabel classifier.

The bounding boxes will identify the cards visible in the frame. This is a task that can be performed accurately as the cards will be of substantial size relative to the frame. The number of objects to localise will also be limited to 4-5 cards which simplifies the task for the model.

Once bounding box predictions have been made, those locations within the image will be cropped and passed to a second model, a multilabel classifier. This multilabel classifier will try to identify which symbols are on the individual card it is given. The hope with this approach is that it can utilise the high accuracy of multilabel classifiers while retaining knowledge about which symbol belongs to which card through separating then with the bounding box predictor.

The downside of this two-model approach is that it will increase inference times. It is hoped that this will not be too substantial.

2.6 Requirements for deep learning

Neural networks will be used to create the models. This is a field known as deep learning. There are three key aspects needed in creating such a model. Training data that the model will attempt to learn from, a model architecture which will specify the parameters that will be tuned to make predictions, and a loss function which will tell the model how good its predictions are.

2.7 Multilabel Classifier

2.7.1 Data

Data comes in the form of training examples. Since supervised learning will be used, data will be made up of input-output pairs. The data should represent every aspect of the cards, as this is what the model will use to try to understand the concepts underlying the data. For this reason, it is favourable to have as much data as possible. Having data from different scenarios will help the model to generalise its predictions to new unseen data.

Due to the nature of the data, image samples only contain pictures of one or more of the 55 cards in the deck. This leads itself well to the process of synthetic data generation [1]. This involves an initial effort followed by the ability to generate arbitrarily large datasets. First, high resolution pictures of each image in the deck need to be taken [2], then transparency is to be added to each image at pixel locations that don't form part of the card. This leads to images containing only pixels of cards and not of the background surrounding them, as seen in Figure 1. The symbols displayed on each card are also recorded.

Figure 1. A single dobble card with transparency added to parts containing background

Background images are now required to simulate different work surfaces that the cards might be placed on when the model is used. These images are acquired by collating a large background image dataset [3] and scraping images from google with various relevant search terms used.

The labour-intensive part of the process is now complete. New image data can be created through copying one or more of the 55 card images onto a random background image.

The process of data augmentation can be used to vary the image data further. Transformations are applied to the data at training time using a dataloader [4]. This ensures that each time the model is reshown the same image, it appears slightly differently. These transformations simulate different lighting conditions, image zoom, rotations, stretches, and viewing angles of the cards from the camera. Each transformation has a range dictating how much it is applied to the image. Ranges are chosen for each transformation based on how much variety can be expected from different imaging setups.

To produce the training data for the multilabel classifier, the quality of the input data must be considered. The images will be taken from the bounding box predictions of another model. This means that cards are not expected to be centered well within the image, and parts of the card may not be contained in the image at all. The model will need to familiarise itself with partial views of symbols on the card, as this may be the only information available to make predictions. The data is generated by copying a random card onto a random background image, resized to dimensions 200x200. A crop of size 150x150 is taken from a random position in the image, as shown in Figure 2. This ensures that the card is not always centered and some of the card may be removed in the cropping process. Symbols associated with the card are noted to be used as the data's expected output.

Figure 2. Example of an image with a random crop shown by the red square. A partial view of the dragon symbol is expected to still be identifiable.

The input of each datapoint will be images of size 150x150. Due to each pixel location having an associated red, green and blue intensity value, the image can be thought of as having a third dimension for the 3 channels of colour. This leads to an input of size 3x150x150.

The output will represent the 8 symbols present on the cards. It is a 55-long vector with a binary encoding, where there is a 1 in position i if class i is present on the image and a 0 otherwise.

2.7.2 Architecture

The architecture determines how the model represents its understanding of the data. It specifies how the input is processed to form a prediction. A better architecture for a task will be able to train quicker as the model's parameters will more easily be able to represent the structure in the data.

Neural network architectures that form the foundation for building the desired models are explained first, with a focus on the key aspects needed to understand the final models.

2.7.2.1 Architecture of a general neural network

The simplest neural network structure is called a multilayer perceptron. It is most easily explained using diagrams such as Figure 3.

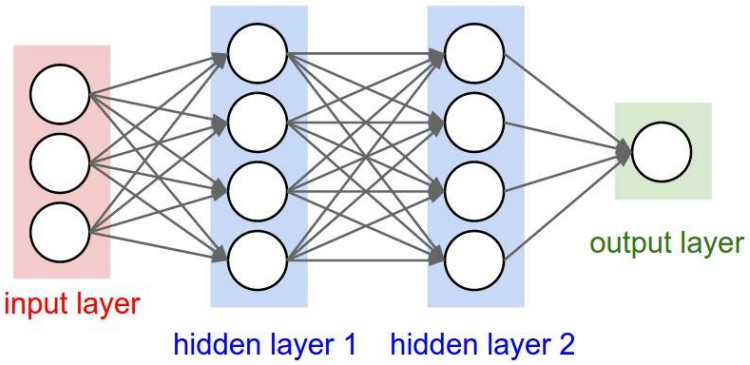


Figure 3. A visual representation of a multi-layer perceptron [5]

Data enters the model through the input vector. Then, a weighted sum of the input values is applied to a non-linear function to compute the value of a hidden node. In recent years the most common non-linear activation used is the rectified linear unit (ReLU) given by the following formula.

$$f(x) = \max(0, x)$$

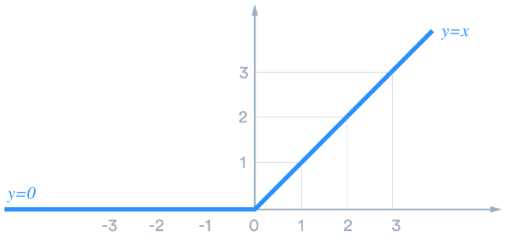


Figure 4. The ReLU function [6]

If the inputs are named $x_1, x_2, x_3, \ldots$ and the first set of weights are named $w_{1,1}, w_{1,2}, w_{1,3}, \ldots$ with bias parameter β_1 then the formula for the output of the first hidden node is given where $f(x)$ is defined above.

$$h_1 = f(x_1 w_{1,1} + x_2 w_{1,2} + x_3 w_{1,3} + \cdots + \beta_1)$$

This process is repeated using different weightings to compute the values of more hidden nodes. The process can be represented by vector-matrix multiplication where $\boldsymbol{X}$ is the 1xN input vector, $\boldsymbol{\beta}$ is the NxM weight matrix and $\boldsymbol{H}$ is the resultant 1xM hidden layer vector.

$$\boldsymbol{H} = f(\boldsymbol{X}\boldsymbol{\beta})$$

It can be thought that each node in the hidden layer represents a different feature of the input. The process of vector-matrix multiplication is repeated using the hidden layer to produce more hidden layers. Each hidden layer is expected to represent more complex features of the input data. After a chosen number of layers, the result of these vector-matrix multiplications is deemed the output of the model.

The parameters of the model are the weights in the weight matrices. These are the values that get adjusted as the model is trained to produce more accurate predictions.

The universal approximation theorem states that any function can be approximated arbitrarily accurately by a multilayer perceptron. This means that the function we wish to model which takes as input the image data and outputs a vector of probabilities of whether each symbol is present on the card can be approximated using neural networks. The problem in practice is that this simple architecture struggles to converge on useful parameters which produce the desired function. Convolutional neural networks are a more complex architecture design which handles image data particularly well.

2.7.2.2 Architecture of a convolutional neural network

The problem with trying to use multilayer perceptrons for image data is that the pixel values need to be turned into a vector to be input into the model. This removes all the positional structure in the data.

Intuitively it can be seen that for a given feature of an image such as a texture, object or pattern, the most natural way to observe it is to relate the pixels in a nearby area. Pixels far away from a region should bear little relevance to understanding the structure of the region, as is shown in Figure 5.

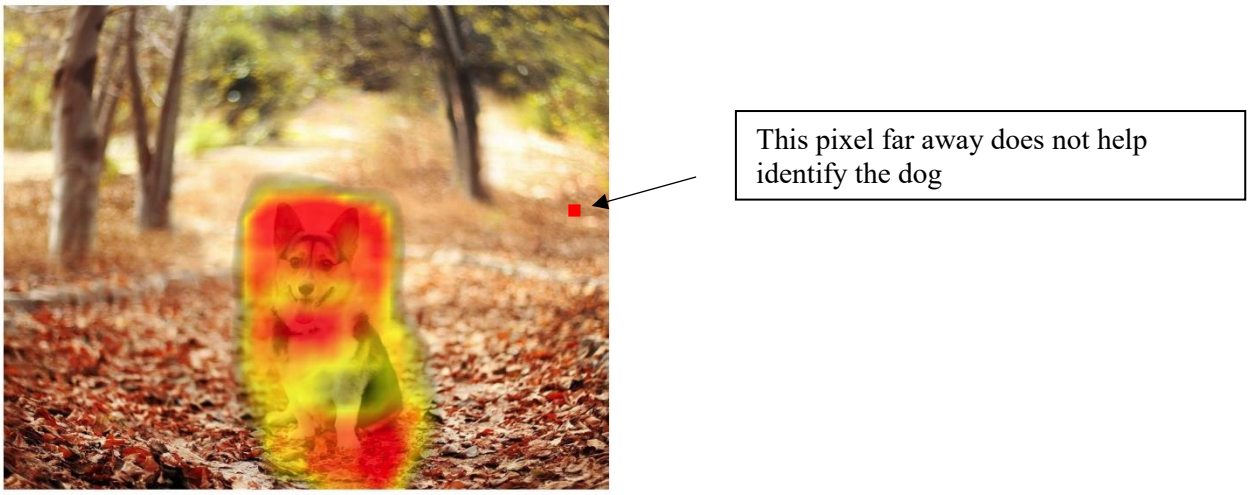


Figure 5. Heatmap of region useful to classifying the image as a dog [7]

It therefore makes sense to only use a weighted sum of the pixel values in a small region to understand that part of the image. A bias term is also added. This region is usually chosen to be a 3x3 square of pixels. Since images contain 3 channels for the red, green and blue pixel values, there needs to be 3x3x3 weights for each region. These weights are called a kernel, and are visualised in Figure 6.

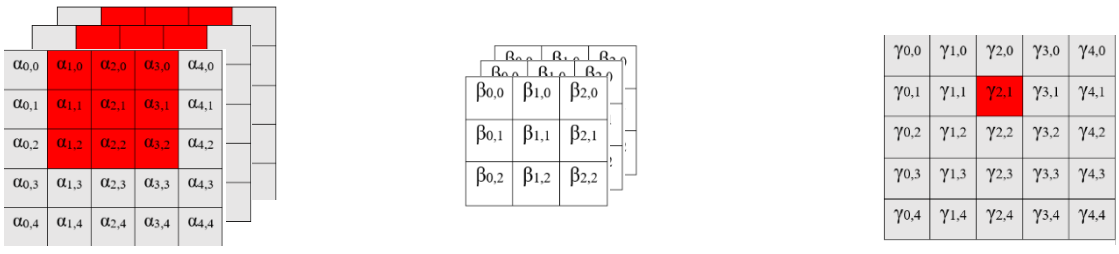


Figure 6. The three-dimensional image array and kernel used to produce a feature map. The area of the input that the kernel is being applied to as well as the corresponding output is shown in red.

$$\gamma_{2,1} = f(\alpha_{1,0}\beta_{0,0} + \alpha_{2,0}\beta_{1,0} + \alpha_{3,0}\beta_{2,0} + \alpha_{1,1}\beta_{0,1} + \cdots + \alpha_{3,2}\beta_{2,2} + \cdots + bias)$$

The kernel is passed over the image in a sliding window fashion to find relations amongst different regions of 3x3 pixels. The sliding window can have an offset from the previous window by 1 or more pixels. This is known as the stride of the kernel. Padding of zero values are added around the edge of the input so kernels can still be applied along the edge. These concepts are shown in Figure 7.

Stride 1

0	0	0	0	0	0	0
0	$\alpha_{0,0}$	$\alpha_{1,0}$	$\alpha_{2,0}$	$\alpha_{3,0}$	$\alpha_{4,0}$	0
0	$\alpha_{0,1}$	$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{3,1}$	$\alpha_{4,1}$	0
0	$\alpha_{0,2}$	$\alpha_{1,2}$	$\alpha_{2,2}$	$\alpha_{3,2}$	$\alpha_{4,2}$	0
0	$\alpha_{0,3}$	$\alpha_{1,3}$	$\alpha_{2,3}$	$\alpha_{3,3}$	$\alpha_{4,3}$	0
0	$\alpha_{0,4}$	$\alpha_{1,4}$	$\alpha_{2,4}$	$\alpha_{3,4}$	$\alpha_{4,4}$	0
0	0	0	0	0	0	0

Window position 1

0	0	0	0	0	0	0
0	$\alpha_{0,0}$	$\alpha_{1,0}$	$\alpha_{2,0}$	$\alpha_{3,0}$	$\alpha_{4,0}$	0
0	$\alpha_{0,1}$	$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{3,1}$	$\alpha_{4,1}$	0
0	$\alpha_{0,2}$	$\alpha_{1,2}$	$\alpha_{2,2}$	$\alpha_{3,2}$	$\alpha_{4,2}$	0
0	$\alpha_{0,3}$	$\alpha_{1,3}$	$\alpha_{2,3}$	$\alpha_{3,3}$	$\alpha_{4,3}$	0
0	$\alpha_{0,4}$	$\alpha_{1,4}$	$\alpha_{2,4}$	$\alpha_{3,4}$	$\alpha_{4,4}$	0
0	0	0	0	0	0	0

Window position 2

$\gamma_{0,0}$	$\gamma_{1,0}$	$\gamma_{2,0}$	$\gamma_{3,0}$	$\gamma_{4,0}$
$\gamma_{0,1}$	$\gamma_{1,1}$	$\gamma_{2,1}$	$\gamma_{3,1}$	$\gamma_{4,1}$
$\gamma_{0,2}$	$\gamma_{1,2}$	$\gamma_{2,2}$	$\gamma_{3,2}$	$\gamma_{4,2}$
$\gamma_{0,3}$	$\gamma_{1,3}$	$\gamma_{2,3}$	$\gamma_{3,3}$	$\gamma_{4,3}$
$\gamma_{0,4}$	$\gamma_{1,4}$	$\gamma_{2,4}$	$\gamma_{3,4}$	$\gamma_{4,4}$

Output

Stride 2

0	0	0	0	0	0	0
0	$\alpha_{0,0}$	$\alpha_{1,0}$	$\alpha_{2,0}$	$\alpha_{3,0}$	$\alpha_{4,0}$	0
0	$\alpha_{0,1}$	$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{3,1}$	$\alpha_{4,1}$	0
0	$\alpha_{0,2}$	$\alpha_{1,2}$	$\alpha_{2,2}$	$\alpha_{3,2}$	$\alpha_{4,2}$	0
0	$\alpha_{0,3}$	$\alpha_{1,3}$	$\alpha_{2,3}$	$\alpha_{3,3}$	$\alpha_{4,3}$	0
0	$\alpha_{0,4}$	$\alpha_{1,4}$	$\alpha_{2,4}$	$\alpha_{3,4}$	$\alpha_{4,4}$	0
0	0	0	0	0	0	0

Window position 1

0	0	0	0	0	0	0
0	$\alpha_{0,0}$	$\alpha_{1,0}$	$\alpha_{2,0}$	$\alpha_{3,0}$	$\alpha_{4,0}$	0
0	$\alpha_{0,1}$	$\alpha_{1,1}$	$\alpha_{2,1}$	$\alpha_{3,1}$	$\alpha_{4,1}$	0
0	$\alpha_{0,2}$	$\alpha_{1,2}$	$\alpha_{2,2}$	$\alpha_{3,2}$	$\alpha_{4,2}$	0
0	$\alpha_{0,3}$	$\alpha_{1,3}$	$\alpha_{2,3}$	$\alpha_{3,3}$	$\alpha_{4,3}$	0
0	$\alpha_{0,4}$	$\alpha_{1,4}$	$\alpha_{2,4}$	$\alpha_{3,4}$	$\alpha_{4,4}$	0
0	0	0	0	0	0	0

Window position 2

$\gamma_{0,0}$	$\gamma_{1,0}$	$\gamma_{2,0}$
$\gamma_{0,1}$	$\gamma_{1,1}$	$\gamma_{2,1}$
$\gamma_{0,2}$	$\gamma_{1,2}$	$\gamma_{2,2}$

Output

Figure 7. The 3x3 area highlighted in red shows the window that the kernel is applied to as it moves across the image. A stride of 2 moves the window more pixels at a time. The dimension of the corresponding outputs is visible.

The size of the output from applying a kernel depends on the stride. If a stride of 1 is used, the output will be of the same size as the input. If a stride of 2 is used, the output will have half the size of the input, with the ceiling function applied.

$$f(n) = n \qquad \text{for stride} = 1$$

$$f(n) = \lceil \tfrac{n}{2} \rceil \qquad \text{for stride} = 2$$

A kernel can be thought of as a tuned set of parameters which can extract a single feature from the image, which the model believes is useful for distinguishing between classification categories. Therefore, the output of applying a kernel is called a feature map. The number of kernels created each layer is specified by the user. More kernels require more model parameters to be optimised but can provide a more robust and detailed representation of classes, through the extraction of more features from the data.

Examples of features extracted by kernels at different depths in the model are shown as followed.

Early layers:	edges of different angles (horizontal/vertical/diagonal lines)
Middle layers:	textures like fur, and shapes like circles and corners
Final layers:	faces, hands, windows, wheels

Pooling layers are used to reduce the dimension of the feature maps to reduce the computational burden which allows for the use of more kernels for later layers. The most common form is max pooling which splits the feature map into 2x2 cells and outputs the maximum from each cell, as shown in Figure 8.

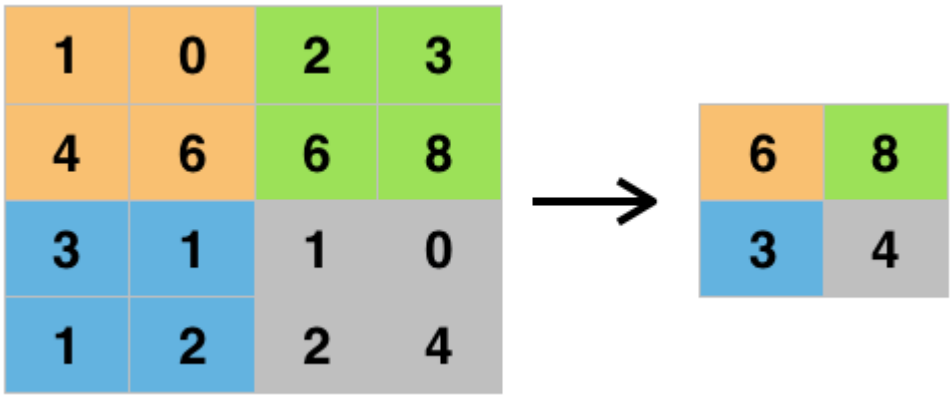


Figure 8. Max pooling is applied to a 4x4 grid. The relevant values compared are shown in the same colour as well as the location of the corresponding output [8].

Once the dimension of the feature maps has been reduced sufficiently and the number of feature maps is adequate for the number of classes to distinguish between, the activations of each feature map are averaged and passed through one or two fully connected layers which interpret the features. They decide which features combine to form the different classes. The output of these fully connected layers is a vector representing each of the classes. A softmax function is applied to the output to turn the predictions into probabilities. The formula for the softmax function is given:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

This covers all the basic operations occurring in a convolutional neural network. Figure 9 shown a diagram of a full model.

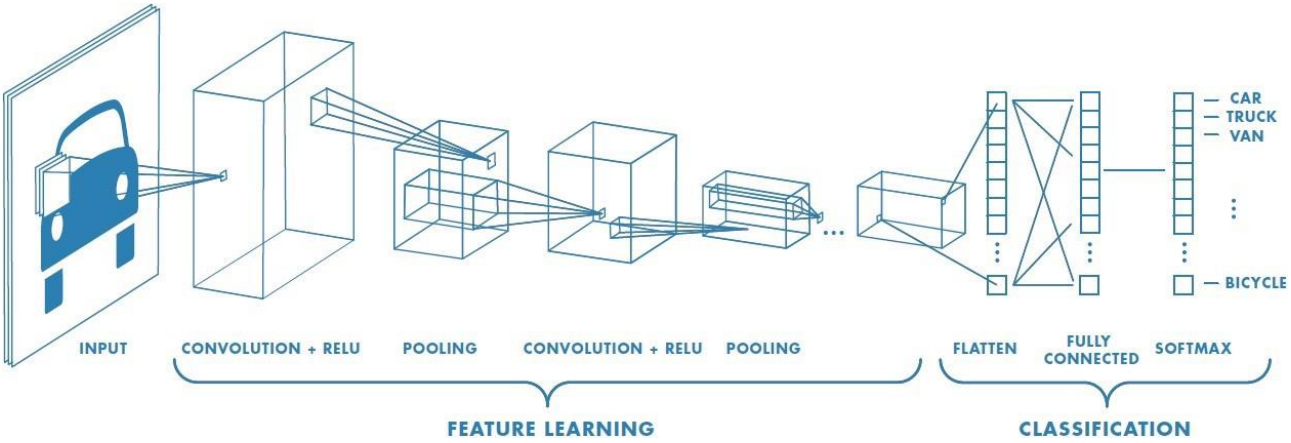


Figure 9. All layers of a convolutional neural network are shown. Starting with the input image, information is passed through layers of convolutions and pooling followed by a fully connected layer and the softmax function. [9]

2.7.2.3 Architecture of a Multilabel classifier

The difference between a standard image classifier and a multilabel classifier is the possibility of multiple outputs. This happens when more than one of the possible classes is present in the image at the same time.

A standard convolutional neural network will naturally identify all classes present in an image since the features will be identified irrelevant of whether there are other objects elsewhere. This will lead to large activations in the corresponding nodes of the layer prior to softmax being applied. Softmax is designed to single out one class as being much more likely than the rest. This makes sense for normal CNNs as the assumption when using the model is that only one object is present in the input image. For multilabel classification this is not the case. The softmax function can simply be removed leaving the outputs in the range $(-\infty, \infty)$.

At inference time, predictions can be made in one of 2 ways. Either through the use of a threshold value, usually set to 0, above which all classes are predicted as being present in the image. The alternative is to use a top-k function which identifies the k most likely predictions. In our case the number of symbols on each card is fixed at 8, so we can use this prior to select the 8 most likely predictions.

A standard CNN architecture called resnet-34 [10] is used as the base model that this modification is applied to. The architecture employs a few other techniques to improve performance and simplify training which are not covered here. A basic list of all layers is given in Table 1, with residual connections omitted.

Output size	Number of output feature maps	Layer description	Repeated?
224x224	3	Input	
112x112	64	7x7 conv, stride 2	
56x56	64	3x3 max pool, stride 2	
56x56	64	3x3 conv	6 times
28x28	128	3x3 conv, stride 2	
28x28	128	3x3 conv	7 times
14x14	256	3x3 conv, stride 2	
14x14	256	3x3 conv	11 times
7x7	512	3x3 conv, stride 2	
7x7	512	3x3 conv	5 times
1x1	512	7x7 average pool	
	1000	fully connected layer	
	55	fully connected layer	
	55	Sigmoid	

Table 1. Layers of the ResNet-34 architecture, with residual connections omitted. The last layer is altered to output 55 classes.

2.7.3 Loss function

The loss function will compare the model's predictions to the ground truth during training. As the predictions get closer to the correct output, the loss function will output a lower value. This will inform the parameters of the model how to change to decrease the loss function, bettering the accuracy of the model.

For multilabel classification the loss function used is sigmoid cross-entropy. First the outputs from the model are passed through a sigmoid function which changes the range to (0, 1) using the formula

$$S(x) = \frac{1}{1 + e^{-x}}$$

These values are then used in the binary cross-entropy formula where x_c is the predicted probability that a class is present in the picture and y_c is a binary indicating of value 0 or 1 depending on whether class c is actually in the image.

$$L(\boldsymbol{X} \mid \boldsymbol{Y}) = -\sum_{c=1}^{M} y_c \, log(x_c)$$

Once the loss function has been computed, the chain rule is applied to the prediction vector **X** to determine how the weights in the model need to be altered in order to lower the value of the loss function. This process is called backpropagation and is handled automatically by the deep learning library used to implement the

models. AdamW is the optimiser used for backpropagation as it is shown to be the fastest currently available, given good choices of parameters for the algorithm [11].

2.8 Bounding box detector

2.8.1 Data

When considering the placement of the cards within real world image inputs, a few scenarios are designed to simulate the relative positioning of cards. Examples are shown in Figure 10.

Figure 10. Synthetic images generated. Different backgrounds are chosen at random for each image. Arrangement and size of cards is specified with random variation applied.

The locations of the cards are randomised slightly with each new image generated to aid with generalisation of the network. Background images are also sampled randomly from the set previously mentioned. The hope is to have cards of different scales located in all parts of the image so that the model doesn't have unresponsive input regions. These would occur where no cards appeared in the training data therefore the model assumes that cards cannot appear in that location during normal use.

The input data will be 3 channels by 224 by 224 pixels as this is the image size that standard convolutional neural networks are trained on.

When generating the data, the positions of the cards are known. The pixel coordinates of the top left and bottom right corners of each card is recorded to be used as the target output of the corresponding input image. These values are combined into a 5 card by 4 position parameters array. If less than 5 cards are present, the remaining positions are padded with zeros. This is so that each training sample has the same size to be given to the model. The padding is removed inside the model when computing the loss during training.

2.8.2 Architecture of a bounding box detector

2.8.2.1 Historical approaches

Many different approaches have been tried over the years to localise the position and classification of objects within an image.

A naïve approach would be to pass a window over sections of the image, as shown in Figure 11, where the "visible" part is used as the input to a standard convolutional network. If the classifier makes a confident prediction believing that an object is visible in the window, then the window position used is claimed to be a bounding box for an object. This imposes an assumption on the bounding boxes that they are a certain aspect

ratio and size, determined by the window specified at initialisation of the algorithm. To alleviate this issue multiple passes of the image can be made using windows of different sizes and aspect ratios.

Figure 11. Sliding window passing over image. In the first view the model is confident that it sees a dog's face, in the second view it is less confident and in the third it is more confident. The first and third locations would be classified as bounding boxes

There are a few problems with this approach. Firstly, there is the issue that performing a forward pass through a convolutional neural network is computationally taxing. For example, inference using the standard ResNet-34 model takes around 70ms [12] using a computer harnessing the optimisations of a modern GPU. CPU inference takes even longer at around 140ms. To achieve modest accuracy using the sliding window algorithm, hundreds of passes would need to be performed through the network to make predictions on object locations. Though the task can be parallelised across more hardware simultaneously, it would still mean that most user hardware would not be able to make predictions at a speed fast enough to keep up with the framerate of a video feed.

The other problem with this approach is that the task of prediction and localisation are intimately related. The process of making a class prediction contains a lot of information about which regions of pixels caused the decision. Because of this, a lot of information is left unused if the tasks of prediction and localisation are kept as separate processes.

2.8.2.2 A modern approach using SSD

Looking at the design of modern convolutional neural network architectures, it is seen that before the final layers of the model which are fully connected, an average pooling layer is applied to the feature maps. In the case of Resnet-34, there are 512 feature maps each with dimensions 7x7. If we think of these feature maps as being able to identify cards within the image, then the average pooling layer collates this information into a single number which tells the model how card-like the whole image is.

Observing these feature maps in the later layers, it can be seen that cells which are most activated when an image is passed through the network correspond to similar locations of features in the image. Figure 12 shown such feature maps.

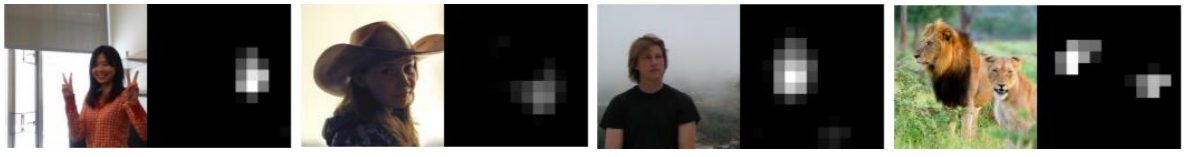


Figure12. Image and corresponding feature map activations which identifies faces within images [13]

These feature maps preserve the geometry of information due to a property called the receptive field [14]. It describes how the information from the input image passes through the network to different layers. Because convolutions only use a small kernel to produce activations in the next layer, the values of the second layer

can be shown to only be influenced by a small region of pixels, a 3x3 grid in our case. This means that if a feature is to be identified in the second layer, it is known exactly which pixels in the input image were responsible for it, as can be seen in Figure 13.

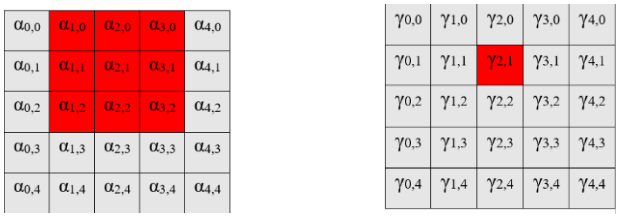


Figure 13. Layers 1 and 2 shown. $\gamma_{2,1}$ is only influenced by the 3x3 grid of input pixels

At the next layer this calculation can be repeated. A cell in layer 3 is computed using a 3x3 grid in layer 2. Each one of those 3x3 cells in layer 2 are computed using a 3x3 grid of input pixels. It is shown that the region taken up by all these overlapping 3x3 pixel regions form a 5x5 grid.

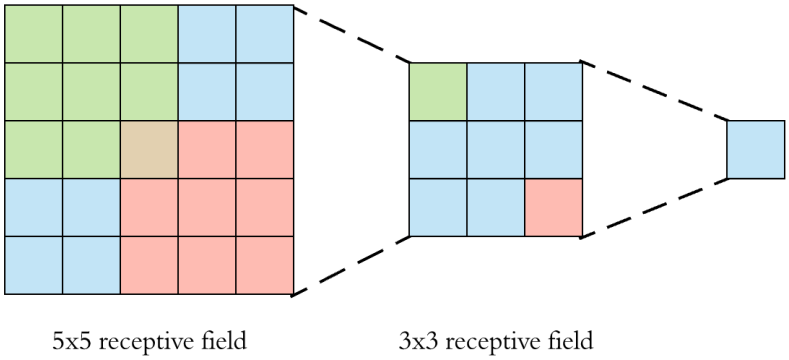


Figure 14. Cells from 3 layers are shown. The single cell on the right is influenced by a 3x3 region of cells in the previous layer which in turn is influenced by a 5x5 region of cells in the layer before it [15].

Therefore, given a feature map which identifies cards, the model has adequate information to make a prediction on the location of the card within the image. An architecture [16] will be designed to extract this information.

Cells in the feature map will be used to make predictions. A prior for the size and location of an object prediction is assumed to be of a similar size and location as the receptive field of a cell. The receptive field of a cell is approximated by the area that the cell takes in the original image if the feature map were to be stretched over the image, as visualised in Figure 15.

Figure 15. Feature maps of dimensions 4x4, 2x2 and 1x1 are shown in red. They are superimposed onto the input images to show which area of the image each cell is most influenced by. [17]

Predictions will be made using feature maps of dimension 4x4, 2x2 and 1x1 [18] as these are the scales that cards are expected to be encountered in during normal use.

A different feature map will be used for each piece of information we wish to extract. It is found that predicting the center and size of bounding boxes is an easier task for neural networks than predicting the top left and bottom right corners of the bounding boxes directly. Since we have an x, y, width and height prediction to make, each will be predicted on a separate feature map. A fifth feature map will also be used to predict how likely the cell is to contain an object. This is because each cell in the feature map will be making a prediction regardless of whether it makes sense to do so, for example when there is no object present. The "probability of a card" feature map will be used to filter out relevant predictions where the corresponding size and location outputs will have meaning.

Unlike a standard architecture where the prediction is output from the last layer of the model, predictions will be extracted from the feature maps of the last few layers before the end of the model. This modification will be done to a Resnet-34 architecture. As mentioned earlier the feature maps of a Resnet-34 model decrease in dimension to 7x7. The average pooling and fully connected layers following this will be removed and replaced with more convolutional layers, detailed in Table 2.

Output size	**Number of output feature maps**	**Layer description**
7x7	512	3x3 conv
4x4	5	3x3 conv, stride 2
2x2	5	3x3 conv, stride 2
1x1	5	3x3 conv, stride 2

Table 2. The last 7x7 convolutional layer of Resnet-34 is shown as well as the new convolutional layers added.

The 5 feature maps at sizes 4x4, 2x2 and 1x1 are all used to make predictions.

Restrictions need to be placed on the cell predictions as to ensure it is predicting an object centered in its cell, shown in Figure 16. This will encourage the cell to only make predictions on objects it thinks are nearby, as these are where the cell will make the most accurate predictions.

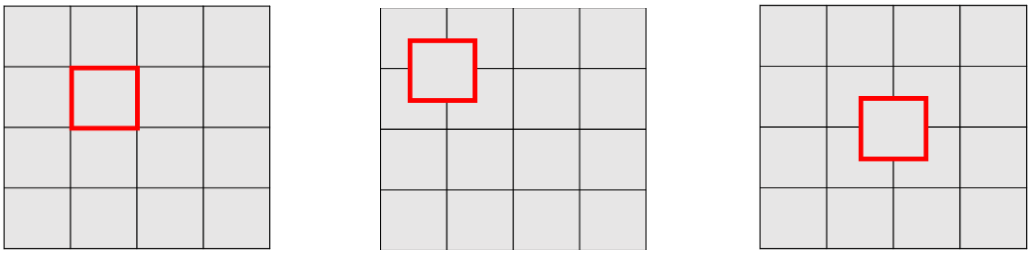


Figure 16. Prior position (left), minimum x and y predictions (center), maximum x and y predictions (right)

The other restriction imposed is that the size of the object prediction is close to the same scale as the cell in the feature map. This is because we can't expect a small cell to summarise the location of an object which spans many cells. We also can't expect a large cell to accurately locate a tiny object within it. The height and width predictions therefore only change the predicted dimensions of the bounding box from 50% to 150% of the size of the cell, shown in Figure 17.

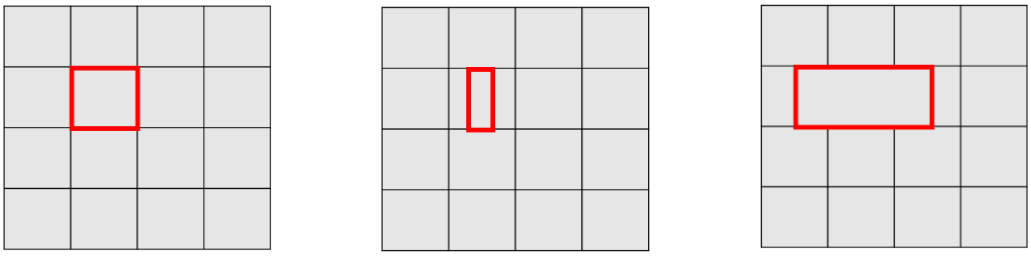


Figure 17. Prior position (left), minimum width predictions (center), maximum width predictions (right)

Probabilities that a card object is present are also scaled to the range (0, 1). The transformations applied to achieve these restrictions can be thought of as link functions. Given predictions $t_x, t_y, t_w, t_h, t_p \in \mathbb{R}$ and prior values for the positions of the center of the cell c_x, c_y as well as its dimensions c_w, c_h, the parameters of the bounding box prediction are given as follows where σ is the sigmoid function transforming values to the range (0, 1)

$$b_x = c_x + c_w(\sigma(t_x) - \frac{1}{2})$$

$$b_y = c_y + c_h(\sigma(t_y) - \frac{1}{2})$$

$$b_w = c_w(\sigma(t_w) + \frac{1}{2})$$

$$b_h = c_h(\sigma(t_h) + \frac{1}{2})$$

$$b_p = \sigma(t_p)$$

2.8.3 Loss function

Given an input image with bounding box positions, the model needs to know how close its predictions are so that it can know how to improve.

Currently there are bounding box predictions for every cell in the feature map and a list of a few target bounding boxes. As mentioned previously if the model doesn't think that a cell contains an object then the x, y, width and height predictions will be meaningless. Therefore, those predictions should not be penalised. The loss function needs a way of assigning the most relevant cells with the task of predicting bounding box positions which they will be penalised on for accuracy, as shown in Figure 18.

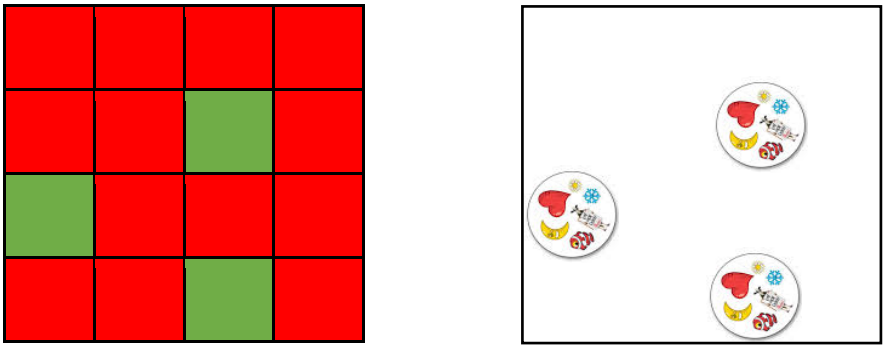


Figure 18. Green cells in feature map assigned responsibility to predict bounding box positions. Red cell bounding box predictions ignored

To make this assignment, a matching algorithm is used. It will assign an object bounding box to the cell which overlaps it the most. The metric used to determine object-cell overlap is called the Jaccard Index, also know as Intersection over Union (IoU).

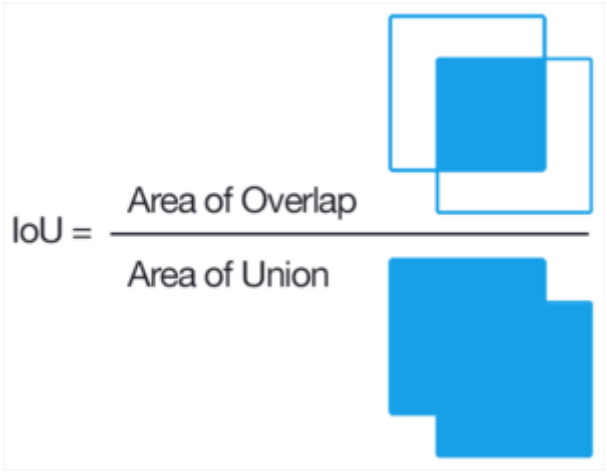


Figure 19. Visual representation of the Jaccard Index metric [19]

Given the top left and bottom right corners of a cell x_1, y_1, x_2, y_2 and object $\alpha_1, \beta_1, \alpha_2, \beta_2$, we compute the coordinates of the corners of the intersection area $min_x, min_y, max_x, max_y$

$$min_x = \max(x_1, \alpha_1)$$

$$min_y = \max(y_1, \beta_1)$$

$$max_x = \min(x_2, \alpha_2)$$

$$max_y = \min(y_2, \beta_2)$$

$$area_{intersection} = \max(0, max_x - min_x) * \max(0, max_y - min_y)$$

$$area_{cell} = (x_2 - x_1) * (y_2 - y_1)$$

$$area_{object} = (\alpha_2 - \alpha_1) * (\beta_2 - \beta_1)$$

$$area_{union} = area_{cell} + area_{object} - area_{intersection}$$

$$IoU = \frac{area_{intersection}}{area_{union}}$$

For each target object, the matching algorithm calculates the IoU metric of all cells and the object. The cell with the highest IoU score is deemed responsible for that object.

Binary cross-entropy loss is used to measure how accurate the "probability of a card" prediction is for each cell. For cells not responsible for a card, predictions are expected to be as close to zero as possible. For cells responsible for a card, predictions are expected to be as close to one as possible.

M is the number of cells making predictions. y_c is equal to 1 if cell c is responsible for an object and 0 otherwise. x_c is the predicted probability that an object is present in the cell.

$$BCE = -\sum_{c=1}^{M} y_c \, log(x_c)$$

L1 loss is used to determine how close relevant predictions for bounding boxes are to reality. For each cell responsible for a prediction, the absolute different between the top left and bottom right corners of the predicted and actual bounding boxes is used. Predictions are converted from center x, y, width and height to top left x_1, y_1 and bottom right x_2, y_2 pixel coordinates for this calculation. Target corner coordinates are given by $\alpha_1, \beta_1, \alpha_2, \beta_2$.

$$x_1 = b_x - \frac{b_w}{2}$$

$$y_1 = b_y - \frac{b_h}{2}$$

$$x_2 = b_x + \frac{b_w}{2}$$

$$y_2 = b_y + \frac{b_h}{2}$$

$$L1 = |x_1 - \alpha_1| + |y_1 - \beta_1| + |x_2 - \alpha_2| + |y_2 - \beta_2|$$

Mean squared error was also tried with worse results so L1 loss was kept. The full loss function is therefore given as follows.

$$Loss = BCE + L1$$

This concludes all necessary steps required in specifying an object detector. A working model is built by following this procedure [20].

3 Implementation details

Below is a list of optimisations which help train the models faster, as well as improve the performance and stability of the models in real use.

3.1 Using resnet-34 as a backbone

It is mentioned how the models constructed are modifications on standard CNN architecture designs with the last few layers repurposed for our requirements. Namely the Resnet-34 model was used as the original image classifier. When Resnet-34 was originally designed by Microsoft in 2015, they trained the model for many days across a large number of powerful computers to optimise the parameters of the model. Their training data consisted of the ImageNet dataset [21], containing over 14 million images labelled across 1000 non-overlapping classes. This led to very useful and generalisable feature maps, specifically at lower layers, which remain useful for identifying objects not part of the 1000 classes that the model was trained on.

Through the process of transfer learning, we can use the parameters that were optimised by Microsoft's research team. They serve as starting values for the parameters of our modified model.

3.2 Differential learning rates

When updating the parameters of the model through backpropagation, a learning rate needs to be specified. This is a scaler which affects the magnitude of a parameter update.

Due to how the models are created, the parameters for most layers start the training process already heavily optimised from transfer learning. The parameters in the last few layers however need to be initialised randomly. This means that when updating the parameter values, we need to alter the last few layers much more than the first few. Differential learning rates is the process of setting different learning rates to different layers so that they are affected by different amounts during the training process. The first few layers should have their learning rate set to an order of magnitude smaller than the last few layers. The layers in between should have learning rates which smoothly transition from the smaller learning rate to the larger one.

3.3 Learning rate finder

Setting a good value for the learning rate is crucial for training a model fast and with good generalisable capabilities. Setting the learning rate too low means that the model will need to perform more parameter updates to reach good parameter values. This means that the model needs to train for longer. There is also the problem that to perform more parameter updates, the model needs to see more training examples. Due to there being a limited supply of data, the model reuses the same data multiple times in what is known as epochs. The more times a model sees a training example, the more likely it is to overfit to this example. Therefore, unnecessary parameter updates should be avoided. Setting the learning rate too high will also lead to problems, as the model will struggle to converge on a solution. A trade-off is needed when deciding on a learning rate.

A learning rate finder function can be used to aid in this search. By only training the model for a single parameter update a lot can be learned about the effectiveness of the learning rate parameter. Comparing the loss function for different learning rate values, we can directly choose the optimal learning rate for the model.

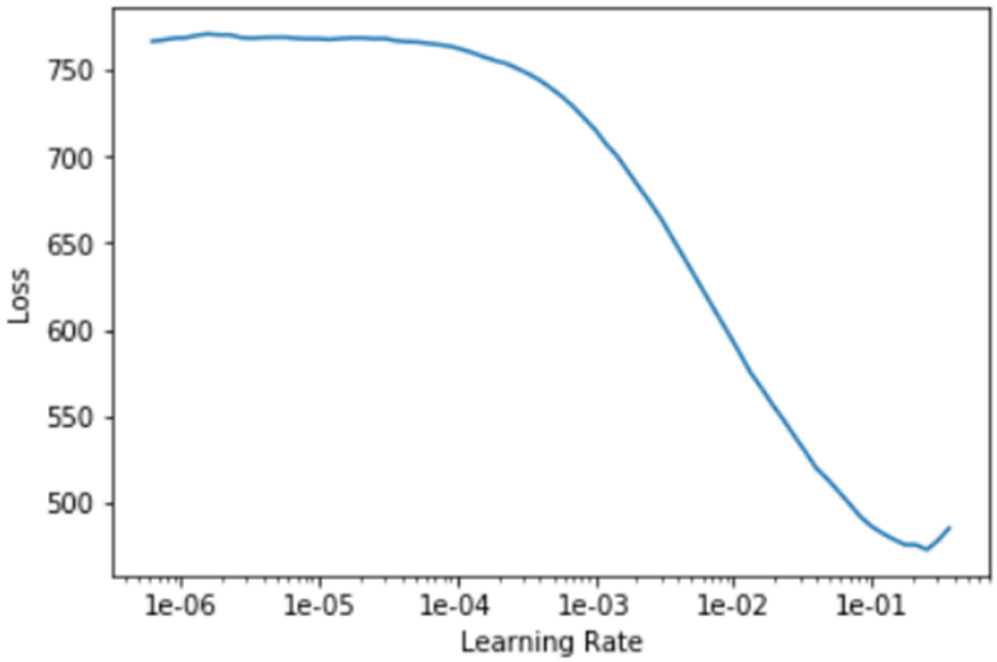


Figure 20. The loss of the object detector model after one parameter update is shown as a function of the learning rate parameter.

Optimal learning rates are in the region where the slope of the function is steepest. For this model a learning rate of 1e-2 is chosen, using Figure 20.

3.4 Test Time Augmentation

In theory if image data is reflected horizontally and vertically at random during training, the model shouldn't contain a bias for certain orientations of cards during inference. However, for the multilabel classifier it is found to help predictive accuracy if predictions are made of cards in all four different orientations [22]. The predictions of all four versions of the input are then averaged to create a final prediction. This ensures that regardless of whether the model has lower accuracy for certain symbols in certain orientations, these problems are smoothed out during averaging.

3.5 Parallelising inference through batch operations

For each frame there is likely more than one card that is identified, cropped and passed along to the multi-label classifier. For each of these cards there are 4 versions that inference needs to be performed on due to test time augmentation. This leads to as many as 20 images that need to be processed by the model during a single frame. If this action was to be done in series using a loop, computation times would slow down dramatically. This is where the parallelisation capabilities of modern GPUs are used. By combining all card images into a batch, they can all be processed by the model concurrently [23]. This provides the speedups required to perform inference with this model in real time.

3.6 Methods of dealing with fluctuating predictions

Due to each frame from the video feed being treated as a separate task to process, symbol predictions often vary between frames. To reduce the variety in the output seen by the end user, the model remembers the predictions from the 10 most recent frames and outputs the average prediction. This is a low enough number of frames that if new cards are shown then old predictions don't affect the average of 10 for very long. It also ensures that if bad lighting or hands partially obscure the cards for a few frames at a time then the output is not affected.

4 Results

After training both the multilabel classifier and the object detector for 20 epochs at differential learning rates of between 1E-5 and 1E-2, accuracy seemed high enough to be tested on a real video feed. A graphical user interface was made to display the results [24]. Bounding boxes are drawn around card location predictions, symbols predictions for each card are written and matching pairs of symbols are computed and listed on each video frame, as can be seen in Figure 21.

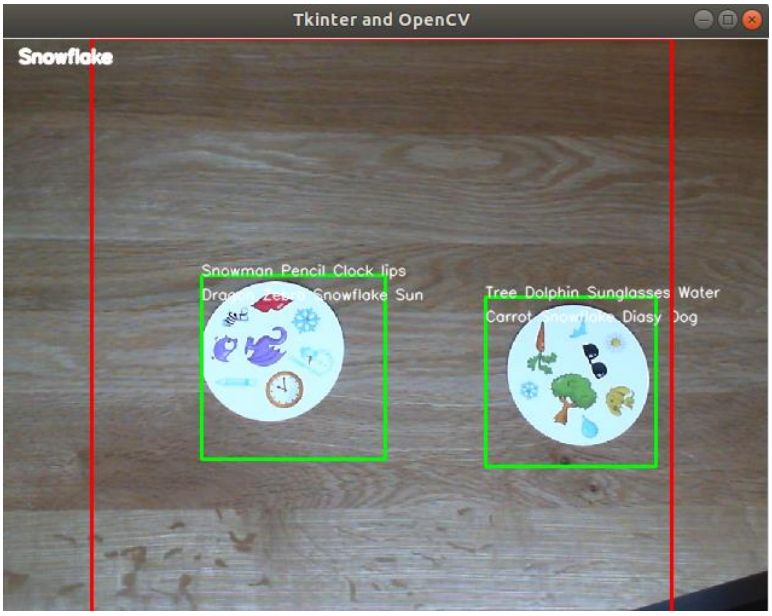


Figure 21. GUI of full model. The red box shows the area of the image passed to the models which needs to be square. The green boxes show bounding box predictions. The white text above the green boxes show the symbol predictions of the multilabel classifier. The text in the top left corner shows symbols in common between the cards.

4.1 Accuracy

Model accuracy was hard to measure since the model was trained on synthetic data. In theory the model could get very good at making predictions on the training dataset but still perform poorly on a video feed if the synthetic data doesn't adequately represent real data. Therefore, accuracy scores shown during training need to be taken with a level of scepticism.

A dataset of labelled real data was not created due to the labour-intensive nature of the task. Accuracy on real data was therefore determined by presenting the model with a small number of images where accuracy was determined visually.

4.1.1 Training accuracy

During training, accuracy on a validation dataset reaches 100% for the multilabel classifier. These results could be believable as the task is not believed be too challenging. There is still some concern that the model is overfitting the synthetic data and will struggle to generalise to the imperfections of real data.

For the object detector model, validation loss reached a value of 21.9 per batch of 16 images. Upon inspecting examples, all cards were identified. The loss was primarily caused by small deviations in the location of the bounding box. This accuracy would be deemed adequate as the bounding boxes encased all symbols on every card.

4.1.2 Real data accuracy

When applied to real cards, it is found that the object detector network works remarkably well. It keeps up with the video frame rate as well as consistently making predictions that very accurately encase the cards visible. The model deals with the cards moving fairly quickly within the image and can handle the 5-card setup expected of it identifying all cards correctly, as shown in Figure 22.

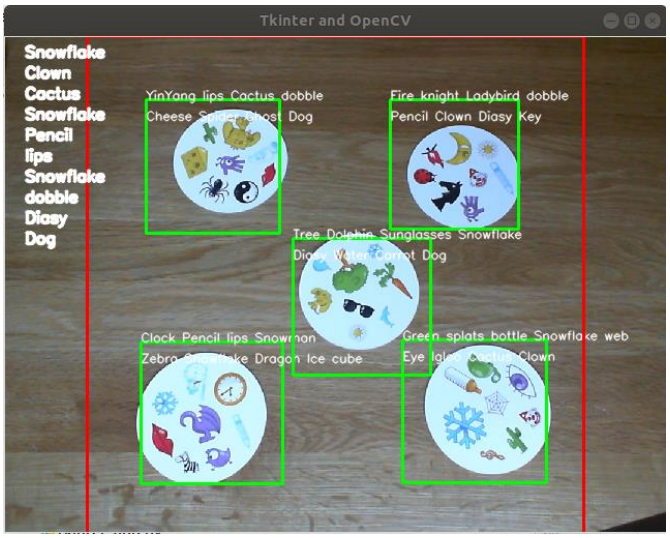


Figure 22. The model is shown to still work properly with 5 cards visible.

With the multilabel classifier however, there were issues with the model's accuracy now that the data isn't generated. Predictions are found to accurately identify 6 to 7 of the symbols on the cards, getting confused by symbols that resemble each other in their colour such as the light blue ice cube and igloo symbols.

4.2 Speed

Experiments were performed to establish the minimum acceptable framerate for the video feed. 15 frames per second was concluded to be the minimum acceptable value. The model was tested on a webcam which can output up to 30 frames per second.

As more cards are shown to the model, the multilabel classifier needs to perform inference on more cards. This is expected to impact the framerate somewhat. Recorded framerates with different numbers of cards in view are given in Table 3.

Number of cards used	**Framerate (fps)**
No model used	30
No cards shown	30
1 card	23
2 cards	23
3 cards	22
4 cards	21
5 cards	20

Table 3. Framerates of the video feed for different numbers of cards shown

These results show that the methods used are computationally efficient enough to be run in real time.

4.3 Generalisation capabilities

The models are tested in a variety of scenarios to assess their robustness. In conditions where performance is lacking, more data could be generated to let the model familiarise itself more with the conditions.

4.3.1 Camera angle

Both the object detector and multilabel classifier were trained on images of cards face on. Data augmentation simulated slight changes in viewing angles. Experiments show that cards can still be detected at angles of up to 45 degrees.

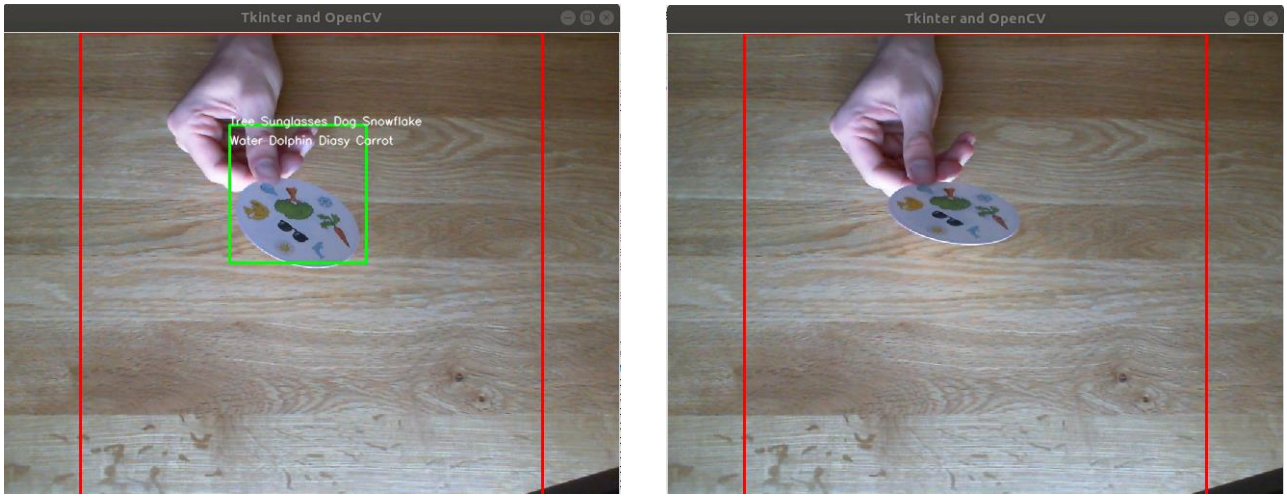


Figure 23. The card is shown to be detected at an angles of 45 degrees (left), past which the card stops being detected (right). All symbols are still successfully detected as long as a bounding box is found.

4.3.2 Partial view of the card

When the model is used in a game, the players' hands can be expected to conceal part of the card unintentionally. Experiments show that the model is robust to hands covering up to half of the card.

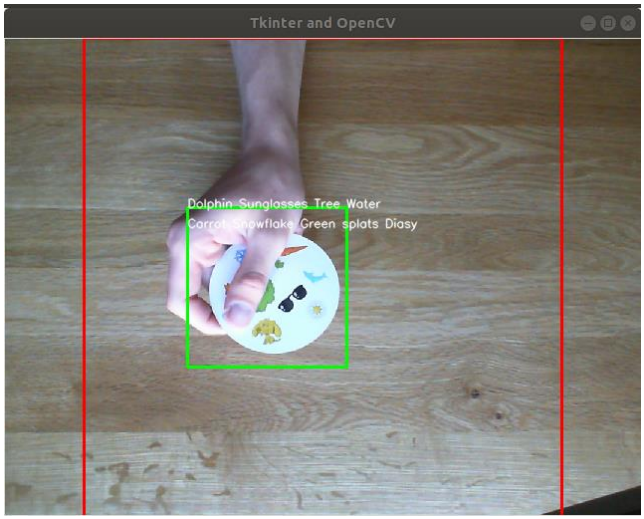


Figure 24. A hand partially blocks the view of the card. A bounding box is still correctly placed around the card.

4.3.3 Lighting

When experimenting with different lighting conditions, it is found that camera software is quite good at bringing out detail in the cards by adjusting the exposure of frames automatically. Given this, the models did not struggle detecting the cards and classifying the symbols any more than under normal lighting.

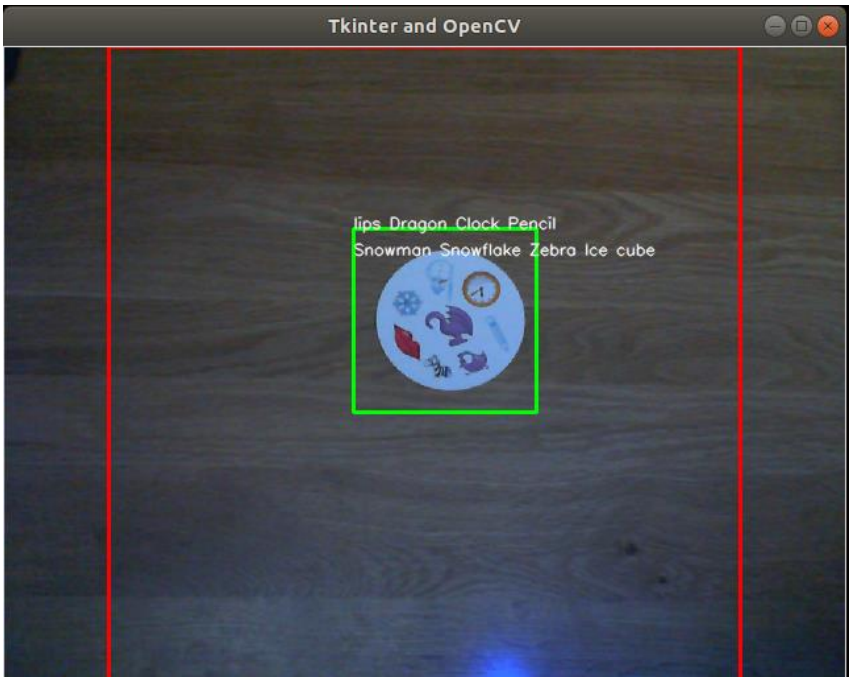


Figure 25. The central card is identified even in low lighting conditions

4.3.4 Camera quality

The object detector model was trained on images of size 224 by 224 pixels. The current implementation of the program scales the incoming feed down to this size before providing it to the model. Therefore, additional image quality will not improve the models' performance. This could be an area to improve upon if the project was studied further. Video feeds of sizes less than 224 by 224 would not work in the current implementation.

4.3.5 Backgrounds (white)

Peer reviewers questioned the model's performance if cards were placed on a white background. At attempt was made to test this although the difference in texture between the card and white background led to the card's outline being visible regardless.

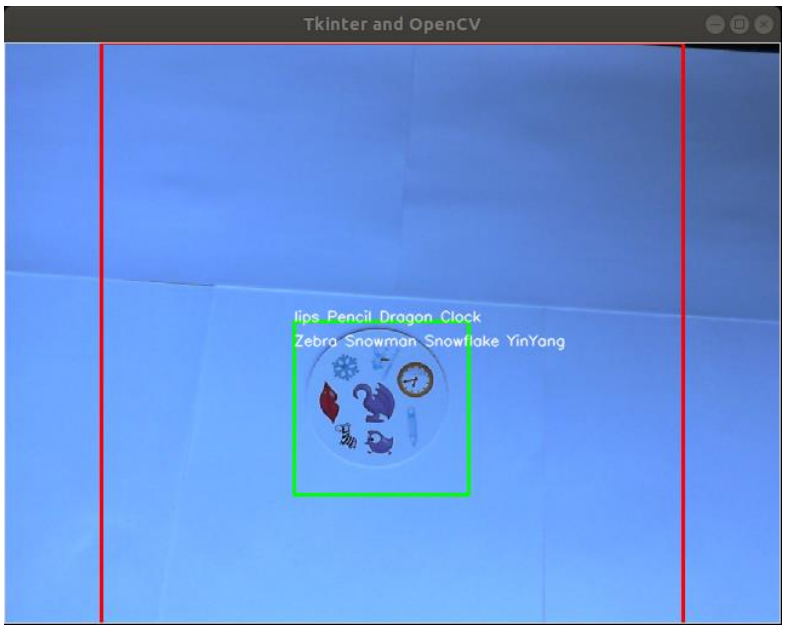


Figure 26. A card on a white background is still detected.

5 Conclusion

Due to the large size of deep neural networks, many technical challenges need to be considered when developing models for use in real time systems.

A case study is explored, to understand the real-world challenges of implementing a fast deep learning model. A hybrid solution to the case study was found which utilises the benefits of both object detection and multilabel classification. The 2-stage approach is shown to still be sufficiently fast.

The design of object detection networks as well as multilabel classifiers was explored, with a focus on choosing designs with fast inference capabilities. A modern approach to object detection was explained, achieving results orders of magnitude faster than methods from over 5 years ago. A modern model architecture was chosen for the multilabel classifier to ensure that inference times were on par with state-of-the-art solutions.

Experiments using the model showed that inference times were adequately fast for the purposes of the model. Video framerate maintained above an acceptable standard for inputs of varying levels of difficulty. The quality of the model was also adequate, performing accurately enough to fulfil its intended purpose.

6 Bibliography

[1] geaxgx, "playing-card-detection," GitHub, 20 June 2018. [Online]. Available: https://github.com/geaxgx/playing-card-detection. [Accessed 15 January 2019].

[2] L. Reynolds, "Dobble," Flickr, 4 October 2012. [Online]. Available: https://www.flickr.com/photos/lwr/sets/72157660922894042/. [Accessed 4 February 2019].

[3] M. Cimpoi, S. M. a. I. Kokkinos, S. Mohamed and A. Vedaldi, "Describing Textures in the Wild," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2014.

[4] J. a. o. Howard, *fastai,* GitHub, 2018.

[5] A. Karpathy, "Neural Networks Part 1: Setting up the Architecture," Stanford, 21 January 2015. [Online]. Available: http://cs231n.github.io/neural-networks-1/. [Accessed 27 April 2019].

[6] D. Liu, "Relu," TinyMind, 30 November 2017. [Online]. Available: https://www.tinymind.com/learn/terms/relu. [Accessed 20 April 2019].

[7] xslittlegrass, "Sliding FullyConvolutional net over larger images," Stack Exchange, 10 June 2017. [Online]. Available: https://mathematica.stackexchange.com/questions/144060/sliding-fullyconvolutional-net-over-larger-images/148033#148033. [Accessed 24 April 2019].

[8] Aphex34, "max_pooling with 2x2 filter and stride = 2," Wikipedia, 16 December 2015. [Online]. Available: https://commons.wikimedia.org/wiki/File:Max_pooling.png. [Accessed 24 April 2019].

[9] "Example of a network with many convolutional layers," MathWorks, [Online]. Available: https://uk.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. [Accessed 24 April 2019].

[10] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 770-778, 2016.

[11] S. Gugger and J. Howard, "AdamW and Super-convergence is now the fastest way to train neural nets," fastai, 2 July 2018. [Online]. Available: https://www.fast.ai/2018/07/02/adam-weight-decay/. [Accessed 27 April 2019].

[12] B. M. S. Teerapittayanon and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," *2016 23rd International Conference on Pattern Recognition (ICPR),* pp. 2464-2469, 2016.

[13] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs and H. Lipson, "Understanding Neural Networks Through Deep Visualization," *CoRR,* vol. 1506.06579, 2015.

[14] W. Luo, Y. Li, R. Urtasun and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 29,* pp. 4898-4906, 2016.

[15] G. Seif, "Illustration of how stacking two 3x3 Convolutions will get you a receptive field of 5x5," Towards Data Science, 12 December 2018. [Online]. Available: https://towardsdatascience.com/a-guide-for-building-convolutional-neural-networks-e4eefd17f4fd. [Accessed 20 April 2019].

[16] L. W. e. al., "SSD: Single Shot MultiBox Detector," *Computer Vision – ECCV 2016,* pp. 21-37, 2016.

[17] G. Ko, "American staffordshire terrier puppies sitting in a box," Shutterstock, [Online]. Available: https://www.shutterstock.com/image-photo/american-staffordshire-terrier-puppies-sitting-box-1048123303?src=vJpigZ_ZWo2wOF7Vnctvyg-1-0. [Accessed 25 April 2019].

[18] J. Howard, "Lesson 9 - Single shot multibox detector (SSD)," Fastai, 7 May 2018. [Online]. Available: http://course18.fast.ai/lessons/lesson9.html. [Accessed 5 February 2019].

[19] A. Rosebrock, "Intersection over Union (IoU) for object detection," 7 November 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Accessed 26 April 2019].

[20] R. Singh and D. Jha, "singleshotdetector," GitHub, 28 January 2019. [Online]. Available: https://github.com/rohitgeo/singleshotdetector. [Accessed 26 March 2019].

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *CVPR09,* 2009.

[22] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *3rd International Conference on Learning Representations*, San Diego, CA, USA, 2015.

[23] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo and A. Ylä-Jääski, “Latency and Throughput Characterization of Convolutional Neural Networks for Mobile Computer Vision,” *Proceedings of the 9th ACM Multimedia Systems Conference,* pp. 204-215, 2018.

[24] P. Silisteanu, “Python OpenCV - show a video in a Tkinter window,” Solarian Programmer, 21 April 2018. [Online]. Available: https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window/. [Accessed 2 April 2019].