Non-exam Assessment Computer Science

OCR for Natural Scene Text

Table of Contents

Analysis

Introduction

My overall aim for this program is to create a program for Natural Scene OCR (Optical Character Recognition), wherein the user can input a photo (.jpg or .png) of a natural scene that contains synthesised text (e.g. a photo of a sign) and the program will return a text output of what the text in the image reads. Natural scene text is simply a photo of synthesised text in a completely uncontrolled environment. The text could be at an angle, within a background or have other factors making it difficult to recognise (see the later section on natural scene text and the associated difficulties).

There are plenty of existing solutions to this, Natural Scene OCR is a very popular problem for neural networks, and is very useful for things like mapping programs (for reading street signs), self driving cars (reading road signs) and for things like speed cameras (reading numberplates).

Aims

- Ability to train the neural network, this will be hidden from the end user but is necessary
  - Take each pixel and get its brightness (I.e. how close the value is to white)
  - Assign each of these to a neuron
  - Send it through a certain number of hidden layers, each with another certain number of neurons
  - Once it's gone through these layers, it goes to the final layer, with a neuron for each character
- An easily understandable, clear and simple GUI containing:
  - An upload button that opens a file browser
  - Preview window allowing the user to crop and rotate their image
  - A button that allows the user to scan for text
- User can upload photos of natural scene text to be scanned (takes jpg or png)
- A crop and rotation function
- Find text, scan it and display it on-screen

Existing Applications

Optical character recognition technology has existed for over 100 years, predating neural networks and even computers. The first example of optical character recognition was in 1914 when Emanuel Goldberg created a machine

that read characters and converted them into telegraph code [1]. A notable motivation for a lot of OCR projects has been for blind and visually impaired users, so the text can be passed into a text-to-speech engine and read to the user. A notable example of this was in 1974, when Ray Kurzweil started Kurzweil Computer Products, Inc. aiming to create omni-font OCR to create text-to-speech tools for the blind. In January 1976, alongside his text-to-speech program, Kurzweil unveiled the CCD flatbed scanner, which was the first omni-font OCR method.
Since then, optical character recognition technology has been consistently moving forward, with modern examples being developed by companies like Google and Microsoft. For example, Google's C++ based open source 'Tesseract' command-line application is available on GitHub and has over 5.5k forks and over 29.5k stars and is used in many of their products like Google translate or Google Lens[3].

User Research

Identification of end users

The selection of scenarios where this could be used simply for the end outcome are limited as the model works best for scenarios with only some natural text. In the time it takes for one to input the data, crop the image if desired and then run the neural network, you could probably just read the image. However, this doesn’t mean the program is useless. The program would still be useful for demonstrating how you could approach natural scene text detection and how well it could work. It could also show which types of photos cause problems for natural scene text detection and what types of text cause problems.

In this case, the end users would likely be researchers on computer vision with AI or tech enthusiasts with an interest in computer vision. Another potential user group may be students who are studying computer vision or AI to show them what neural networks can do.

Potential User Interview

Q: What would you want in the interface?

A: I would want the interface to have basic image manipulation capabilities like cropping and rotating

Q: What would you be using the program for?

A: I would use it as demonstration tool to show the potential for natural scene text detection and show how it could potentially work.

Q: How would you want the program to output text?

A: I would want it to show text as a pop-up

Q: How would you want the interface to look?

A: Simple, this will be used to demonstrate to people with no or little to no technical knowledge, so the fewer complications, the better.

Q: Would you like to have a pre-trained network or be able to train the network yourself with custom hyper-parameters?

A: Definitely pre-trained, sourcing training data would be difficult and we would need someone with technical skills to train the network, we want this to be able to be used by specialists and non-specialists.

Current knowledge and what I need to learn

Currently have strong python knowledge for general tasks and GUIs. Basic knowledge of image handling in python. I will need to look into this in more detail. I also have a basic knowledge of NumPy and

I have already looked into the theory behind building neural networks and I have a limited idea of the calculus specific to neural networks. (as part of my presentations in computer science society) but my calculus knowledge generally is good (further maths). So I need to know more about the calculus of neural networks. As well as this, I will need to learn more about Tensorflow, which one of the most popular neural networking frameworks and is widely used for professional applications.

As for training data, I have already settled on the Chars74K dataset, which consists of thousands of 128×128 PNG images for each character in the Latin alphabet (i.e. 0-9 A-Z) in various fonts, which will provide ample training for the neural network. The Chars74K dataset has 1016 items for each character, so 36,576 total images. [4]

Natural scene text detection

Natural scene text detection is the process of locating where text is in an image. It is much more difficult when compared to detecting and identifying text in a controlled environment, wherein the details of the image are going to be the same and are known (e.g. focus, lighting etc.) Some things that may make text detection harder include:

- Sensor noise caused by interpolation or demosaicing techniques
- The angle or perspective of the scene text can be different
- Blur of any sort, commonly though focus errors or motion blur will be the reason behind the blur
- Lighting, shadowing and reflections can all drastically affect the image and the accuracy of text detection

- Resolution can vary, and may cause some methods of traditional text detection to fail if too low [5]

EAST: An Efficient and Accurate Scene Text Detector

EAST is a method of finding text in an image from a fully convolutional network model that when fed an image produces rotated rectangles (or quadrangles) where it finds text in a natural image. It is important to note that it does not identify the characters in text, it is not optical character recognition, it simply is used to identify and isolate where there is text to make it easier to feed into the neural network.

EAST is mostly used to find text in what's known as 'natural scenes' which are images where the text is not in a constrained or controlled environment (i.e. text that was designed with OCR in mind rather than humans, for example passports, which are scanned under specific lighting), which is exactly what I want to be able to do, as not all photos of text will be taken in the same way. [6]

OpenCV for EAST

OpenCV (Open Source Computer Vision Library) is an open source library that has a huge variety of computer vision tools and has integration with python. The library is used by companies like Honda, IBM and Google [7] and has 18 million+ downloads. It was released in August 1999 by developers working at Intel [8] and is being constantly updated and maintained.

OpenCV's computer vision tools makes implementing EAST much simpler, as the maths is incredibly complicated so I will be using it's libraries for that and only that. The actual character recognition will be done with a neural network built entirely from scratch. Additionally, OpenCV have already done an example of EAST in C++ [9], I will not be using this but instead attempting to adapt parts of it to python and also referencing a python-based guide (see reference [9])

Program Outline

GUI – Provides a user interface for the user, contains buttons with their respective call-backs and displays the user's documents. Also has the ability to upload, preview and crop images.

Segmentation – Uses OpenCV’s implementation of EAST to find text in the document. It then takes the isolated image of text and splits it into characters (segmentation). The image is then prepared for neural network processing, wherein the image is modified to be as close to having nothing but characters as possible, to do this, the algorithm I will use is going to be the watershed algorithm. For more detail on this please see the area where I have written about the watershed algorithm. I will then resize these different images to fit

the 30×30 size that the neural network takes in, and pass them into NeuralNetwork.

NeuralNetwork – Has the option to train the neural network or pass in data from the segmentation section to do OCR. Has all the different layers of the network with their weights, biases, links and the neurons, this information should be stored as arrays. Will have a way to train as well, then permanently store the weights and biases from training. Takes in inputs as 30×30 images of letters, converts each pixel to a brightness value (so 16384 different float values from 0-1) and then outputs what character they represent (classification).

Documented Design

Steps and objectives

1. UI
    - If an image hasn't been imported, a placeholder image is shown and none of the buttons are able to be clicked other than the one to import an image
    - The ability to import PNG, JPEG and JPG files
    - The ability to open a file dialogue to select an image file to import
    - The ability to preview images before they are put into the network
    - Error messages for any instances where the program can't handle one of the user's inputs
    - A simple clockwise and anticlockwise rotation feature
    - A cropping function:
        - A semi-transparent box to be dragged over a region of interest
        - When the mouse button is released, the user can either confirm or cancel the crop
        - The user's cursor should change in the crop
        - During cropping, the user can't start another crop
    - A button to recognise the text, when this has gone through, the results are displayed to the user
2. Text detection and character segmentation
    - The ability to find where text is and localise it to a region that contains text
        - Use the EAST model to get information on the word regions

        - Crop the image based on these regions
    - The ability to perform non-maximum suppression to get rid of weak or overlapping regions
    - A function to split these word regions into their constituent characters by:
        - Increasing contrast
        - Dilating
        - Thresholding
        - Finding contours
        - Drawing bounding boxes
    - A function to then fit these character images into a 30×30 image that the neural network can take in.
3. Creating training data
    - Get a set of different fonts and check them
    - Generate training and validation data based on these fonts
    - Augment the training data to add more variety using the following techniques:
        - Rotation
        - Offsetting
        - Different sized font
    - Randomly split the data into training and validation sets
    - Shuffle the data and labels before they go into the network
    - Put the data into a form the network can read
4. The neural network
    - Create a neural network that has sufficient complexity to be able to recognise the dataset to a high accuracy, using:
        - Dense layers
        - Convolutional layers
        - Max pooling layers
        - Dropout
    - Get the outputs from the network and formulate predictions for whole words

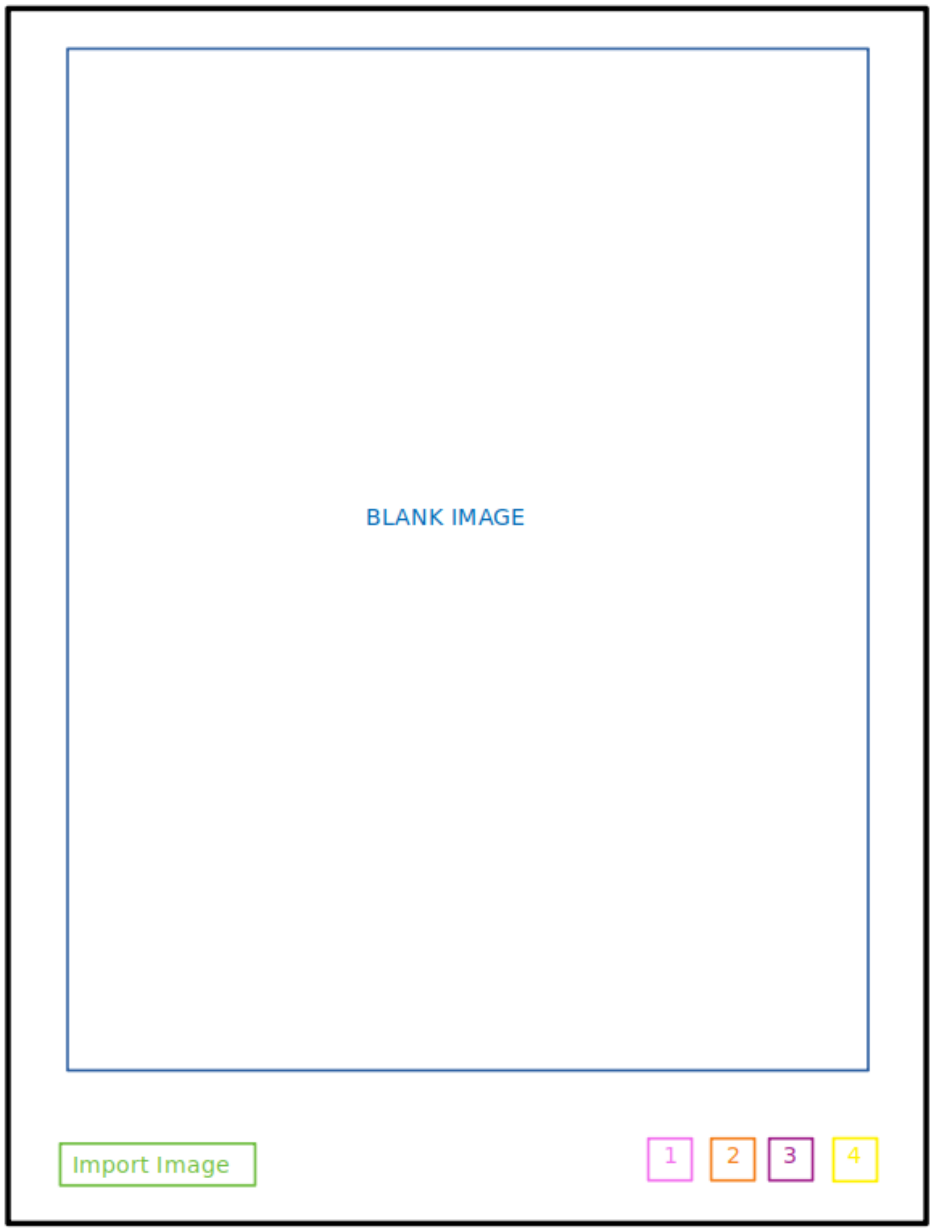
- After the data has come out of the neural network organise the words by line and position (i.e. left-to-right and top-to-bottom) and put them into one string

Main Program

UI overview

When opened, the UI should display a blank image (grey white checked pattern) in a A4 size. There will be padding on either side and top and bottom. Underneath there will be 2 buttons on the left and four on the right. The two buttons on left will be text ones, one will have import image and the other will have train network. The four buttons on the right will be a scan text button, an anticlockwise rotation button, a clockwise rotation button and a crop button. When the program is initially loaded, the four buttons will be disabled, as they can’t be used for the blank image.

Concept Illustration



1 is the anticlockwise button, 2 is the clockwise button button, 3 is the cropping button and 4 is the scanning button. When you've opened an image, it replaces the blank image with whatever you imported and resizes the window accordingly. The buttons are then no longer disabled. Please note that the colour coding is only present for illustration purposes.

Main Window

This subroutine creates all the buttons and adds the image to the canvas. It takes in the size of an image, the image itself and whether this is the initial call of the subroutine. If it is the initial call, the buttons on the right are disabled. If it is not, the buttons are normal and the canvas (for placing the image) is cleared and removed.

Pseudocode

```
IF init ← False THEN
     canvas.grid remove
     canvas.delete all
ENDIF
imgdisplay ← open(img)
canvas ← Canvas(root, width ← x, height ← y)
canvas.create_image(0,0, imgdisplay)
imageButton ← Button(text ← ”Import Image”,command ← imageDialog)
trainButton ← Button(text ← ”Train Network”,command ←
trainNetwork)
antiIco ← open(“Icons/anticlockwise.ico”)
antiRotate ← Button(root, image ← antiIco   command ←
imageRotate(False, img))
clockwiseIco ← open(“Icons/clockwise.ico”)
clockRotate ← Button(root, image ← clockIco      command ←
imageRotate(True, img))
cropIco ← open(“Icons/crop.ico”)
cropButton ← Button(root, image ← cropIco, command ←
crop(x,y,img))
IF init ← TRUE THEN
     antiRotate(state ← DISABLED)
     clockRotate(state ← DISABLED)
     cropButton(state ← DISABLED)
```

Add Image

Add image takes in the image. The image's size is taken, then resized by factors of 1.05 until its dimensions are below 1600×900 so it fits comfortably in the window. The main window subroutine is then called.

Pseudocode

```
imgsize ← list
imgsize ← (img.size)
WHILE imgsize[0]>1600 OR imgsize[1]>900 THEN
     imgsize ← [i DIV 1.05 FOR each in imgsize]
     img ← img.resize(imgsize, with LANCZOS)
ENDWHILE
```

```
mainWindow(*img.size,img,False)
```

Import Image

Uses tk.filedialog to create a file opening window like most common apps. It is set it to only show jpeg and png files. Opening a file calls the subroutine to reform the window with the new image. After that, the other 4 buttons for image manipulation are activated so they can be clicked.

Pseudocode

```
filename ← string
filename ← filedialog(initialdir ← '~/Pictures',title ← "Select
file",filetypes ← (("PNG Images","*.png"),("JPEG Images","*.jpg"),
("All files","*.*")))
userimg ← image
userimg ← Image.open(filename)
self.addImage(userimg,False)
```

Rotate Image

Rotates the image by 90º clockwise if the clockwise button is pressed (i.e. if the boolean passed is true) or rotates it 90º anticlockwise if the anticlockwise button is pressed (i.e. if the boolean passed in is false). Rotates the image using a PIL method without stretching then passes it to the Add Image subroutine. Takes arguments clockwise and img, where clockwise is a boolean for the direction and img is the image.

Pseudocode

```
IF clockwise TRUE THEN
    addImage(img.rotate(270))
ELSE THEN
    addImage(img.rotate(90))
END IF
```

Crop

Creates a partially transparent grey rectangle with a grey outline from where the user clicks to where they finish dragging. After they finish dragging, a conformation window is displayed where they can either crop the image or cancel the crop. This is done with 3 different bindings, where clicking the mouse button 1 is bound to Start Rectangle, moving the mouse with button 1 held down is bound to Grow Rectangle and releasing button 1 is bound to Crop End. The cursor is also changed to a cross.

Pseudocode

```
canvas.config(cursor ← ”cross”)
canvas.bind(“ButtonPress-1” to startRectangle)
canvas.bind(“Button1-Motion” to growRectangle)
canvas.bind(“ButtonRelease-1” to cropEnd(event, img))
shape ← canvas.create_rectangle
```

Start Rectangle

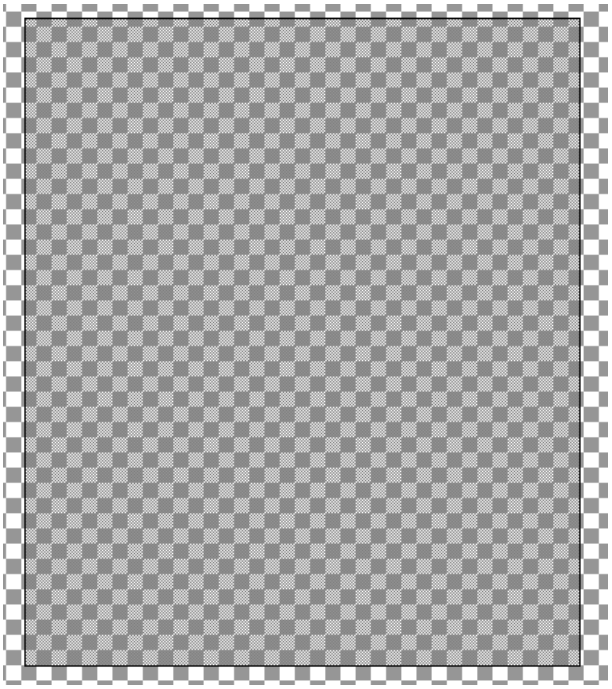
Essentially sets up the rectangle to start being drawn when this button is pressed, but nothing is held down. Also stores the initial event for drawing subsequent rectangles when it grows.

Pseudocode

```
start ← event
drawn ← None
```

Grow Rectangle

If this is not the first time calling this (i.e. if a rectangle has already been drawn), the previous rectangle is deleted, and a new one is created at the new co-ordinates of where the event has been activated. The rectangle is created as a rectangle going from start co-ordinates to the event's co-ordinates. The fill colour is set to the built in gray50 and a faux-transparent effect is created using a stipple.



A basic gray stipple

Pseudocode

```
canvas ← event.WIDGET
IF drawn THEN
    canvas.DELETE(drawn)
ENDIF
objectID ← SHAPE(start.x, start.y, event.x, event.y,   fill=”gray50”, stipple=”gray50”)
drawn ← objectID
```

Crop End

When the user releases the mouse button 1 after drawing their crop rectangle the crop, this event is activated with the event info of where the mouse button was released. The image is also passed in from the initial crop function. A window is presented to the user asking them whether they want to crop or cancel. If they want to confirm, the Do Crop subroutine is called with the event information and the image

Pseudocode

```
canvas.CONFIG(cursor=default”)
canvas.UNBIND (“ButtonPress-1”)
canvas.UNBIND (“Button1-Motion”)
canvas.UNBIND(“ButtonRelease-1”)
cropButton←BUTTON(cropConfirm, text=”Confirm”, command=doCrop)
cancelButton←BUTTON(cropConfirm, cancelCrop)
```

Cancel Crop

Destroys the crop window and removes the rectangle from cropping.

Pseudocode

```
cropConfirm.DESTROY()
canvas.DELETE(drawn)
```

Do Crop

Destroys the crop window, removes the rectangle form cropping then creates a cropped version of the image that contains only the area enclosed by the rectangle the user drew, then calls the add image subroutine with the new cropped version. The current image is passed in with the identifier img and the event is passed in with the identifier event.

Pseudocode

```
cropConfirm.DESTROY()
canvas.DELETE(drawn)
cropImg ← img.CROP(start.x, start.y, event.x, event.y)
```

Text Scan

The program saves the current image to a temporary file and then calls the imagePrep's textDetection subroutine with the path to te temporary file. This returns a list of coordinates with which the image is cropped to return a list of images of words.

For each image in the list of images of words the image is then saved to another threshed image and the imagePrep's thresholdImage is called with that path. The return of this function is converted to an array then inverted.

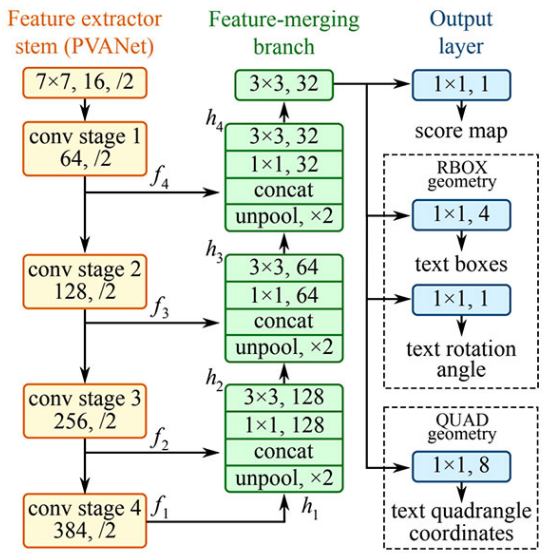
For each image in this list the image is then saved to the previously used temp and imagePrep's characterSegment is called with the path of the new temp. It returns as a list of characters, each character in the list is converted to an array and then this list is appended to another list.

For each image in this list, imagePrep's fitImage method is called with the image and the flag True. The return is added to a list called resizedList.

Isolating individual character images

OpenCV for EAST

EAST is a method of finding text in an image from a fully convolutional network model that when fed an image produces rotated rectangles (or quadrangles) where it finds text in a natural image. It is important to note that it does not identify the characters in text, it is not optical character recognition, it simply is used to identify and isolate where there is text to make it easier to feed into the neural network. East is end-to-end,



A diagram of the OpenCV FCN [6]

which is useful for me, as it means I don’t have to provide any further information to the neural network other than an input image, making it more like how a real OCR system would work and also it allows the overall system to be applied to a wider range of different images and scenarios.

EAST was proposed in a paper written in July 2017 by Xinyu Zhou et al. of Megvii Technology Inc. from Beijing. The paper aimed to provide a model for text detection that worked for challenging scenarios, while remaining simple and efficient, and the model has already found implementations in different programs.

EAST is mostly used to find text in what's known as 'natural scenes' which are images where the text is not in a constrained or controlled environment (i.e. text that was not designed for OCR in mind, for example a photo of a road sign), which is exactly what I want to be able to do, as not all photos of text will be scanned or taken in the same way.

The EAST model works by feeding the image input into a fully convolutional network (or FCN for short), which will then output a series of predictions for areas that it thinks contains words (see the convolutional neural networks section for how a convolutional network works). There is minimal pre-processing performed on the input, but there is some thresholding and NMS (non-maximum suppression). This is to keep the detector efficient. As well as this the model is end-to-end so it avoids computationally expensive sub-algorithms (e.g. candidate aggregation, word partitioning etc.).

The EAST pipeline takes in an image into the FCN and the output is multiple channels of pixel-level score maps for text and geometry for bounding boxes around text. A predicted channel is a score map, where there are pixel values between 0 and 1 and they represent how confident the network is that there is text in that pixel.

As the size of word regions can vary massively, finding where large words exist would require features from the late-stage of a neural network, but predicting the geometry for small words would need low level information from the early stages. Because of this, the EAST network uses features from different levels to be able to detect both large and small words.

As merging a large number of channels for large feature maps would increase computation massively, a 1×1 convolutional layer will bottleneck the channel number, cutting it down. Then a 3×3 convolutional layer will fuse together the remaining inputs. This is the feature-merging path.

The final layer is the output layer, it takes the information from the feature-merging branch and splits it into the score map, box geometry, quadrangle coordinates and rotation angle.[6]

OpenCV is probably the most popular tool for computer vision and as such they have fantastic integration for text detection models as well as many built in tools to make working with these models easy.

Text Detection

Takes in an image (variable name ‘image’), and returns a list of ‘bounding boxes’, which are areas where the model predicts there to be text. The version of the EAST model that I use is a pre-trained version supplied by the site pyimagesearch.com [9]. The image is read into a form that OpenCV can understand, then the pretrained network is loaded (east_text_detection.pb), a blob is created from the image which is then forwarded into the model. The model returns the geometry of where it thinks there are boxes and their respective scores.

The program then goes through the list of boxes and creates a list of the bounding boxes and their respective confidence scores. The confidence scores state how likely it is for the current bounding box to contain text. Bounding boxes with a confidence score below 0.2 are ignored After this, imutils’s implementation non max suppression (NMS) is applied to the bounding boxes (using the confidence scores), which removes bounding boxes that are weak and overlapping. Non-max suppression works through a fairly simple process:

A diagram to show the effects of non-max suppression [10]

> *Let the list of bounding boxes be B, the corresponding confidence scores be S and the filtered output be D.*
>
> 1. *Select the bounding box with the highest confidence score, remove it from B and add it to D.*
> 2. *Compare this bounding box with all the bounding boxes, calculate the IOU (Intersection over Union) of this proposal. If the IOU is greater than 0.3, remove this bounding box from B.*
> 3. *Take the next highest confidence bounding box, remove it from B and add it to D.*
> 4. *Calculate IOU again, remove boxes with higher IOU than 0.3*
> 5. *Repeat until there are no bounding boxes left in B [10]*

The remaining boxes should mark the co-ordinates which contain words, and they are returned to main.

Pseudocode for Text Detection

```
image←READ(image)
constant orig←COPY(image)
(H,W)←image.SHAPE[:2]
constant
layerNames←[“feature_fusion/Conv_7/Sigmoid”,”feature_fusion_concat
_3”]
constant net←READ_NETWORK(“east_text_detection.pb”)
constant blob←BLOB_FROM_IMAGE(image)
net.SET_INPUT(blob)
constant(scores, geometry)←net.FORWARD(layerNames)
constant(numRows,numCols)←scores.SHAPE[2:4]
boundBox←[]
confidences←[]
FOR y←0 TO numRows
     scoresData←scores[0,0,y]
     xData0←geometry[0,0,y]
     xData1←geometry[0,1,y]
     xData2←geometry[0,2,y]
     xData3←geometry[0,3,y]
     anglesData←geometry[0,4,y]
     FOR x←0 TO numCols
          IF scoresData[x]<0.2 THEN
               CONTINUE
          ENDIF
          offsetX,offsetY←(x*4.0,y*4.0)
          angle←anglesData[x]
          cos←COS(angle)
          sin←SIN(angle)
          h←xData0[x]+xData2[x]
          w←xData1[x]+xData3[x]
          endX←offsetX+(cos*xData1[x])+(sin*xData2[x])
          endY←offsetY-(sin*xData1[x])+(cos*xData2[x])
          startX←endX-w
          startY←endY-h
          boundBox.APPEND((startX,startY,endX,endY))
          confidences.APPEND(scoresData[x])
     ENDFOR
ENDFOR
finalBoxes←NON_MAX_SUPPRESSION(boundBox,probabilities=confidences)
REUTRN finalBoxes
```

Character Segmentation

The function takes in an image of a word and returns the individual characters. Once again reads the input image into a form OpenCV can read. The image is then adjusted and thresholded. From there, using OpenCV’s built in function of findContours, the program gets the contours. The RETR_EXTERNAL method is used to find the contours, which only gets the extreme outer contours. The CHAIN_APPROX_SIMPLE flag leaves only the endpoints of the contours rather than every bit of information about them. [11] The list is then sorted and boundaries are marked where there is a significant enough gap. This list of boundaries is then used to crop the image, giving us a list of images of individual characters, which is returned.

The following is an example of an image of the word ‘AIRCRAFT’ that has been localised using EAST going through the character segmentation pipeline.

The original Image

1) Convert the image to greyscale

2) Modify the image with alpha of 5 (multiply all pixel values in the image by 5) and beta of -220 (decrease all pixel values by 220)

3) Dilate the image with a kernel size of 2×1. A high level explanation is that dilating will increase the white space (see later for more detail)

4) Threshold the image (the image is also inverted if the thresholded image has majority white pixels in the background. This is because the contour detection only works for white on black)

5) Contours are detected in the image and bounding boxes are drawn based on the contours

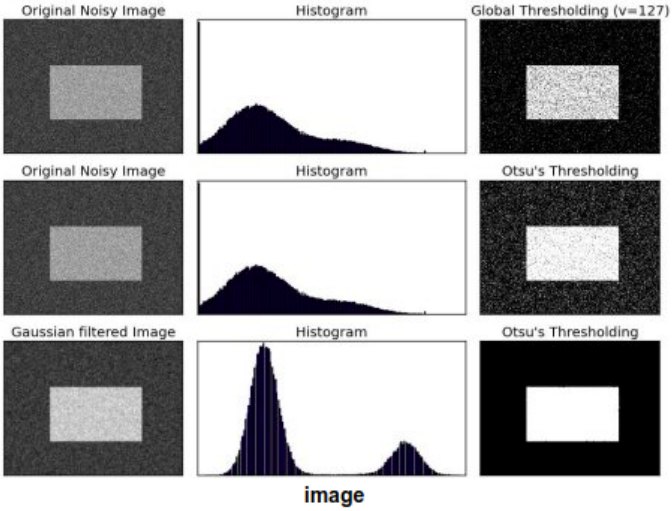
6) The bounding boxes are used to crop the original greyscaled image into its subsequent character images

The Gaussian blur simply blurs an image by a Gaussian function, where each pixel is transformed by putting it through a Gaussian function of the form:

$$G(x,y)=\frac{1}{2\pi\sigma^2}^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where sigma is a variable and will be adjusted to give different amounts of blur. An additional parameter is the kernel size. Where $G(x,y)=\omega\times f(x,y)$ ω is the kernel. The size of this kernel can massively effect how much an image is blurred by. In my examples, I use a 3×3 kernel as this blurs by a reasonable amount, maintaining detail and definition of lines, while eliminating noise.

Dilation is a basic method for greyscale images where at a high level, it increases the size of white objects. It takes the neighbourhood maximum when the kernel (a 2x1 matrix of 1s in this case) is passed over it. [32]



A comparison between Otsu's method with Gaussian blur, unblurred Otsu's method and global thresholding [11]

Thresholding is performed using Otsu's thresholding method. Otsu's method is named after it's creator, Noboyuki Otsu (大津展之) and is used to return a single intensity threshold, separating pixels into foreground and background, represented as black and white respectively. The method searches for a threshold that minimizes the weighted sum of variances between background and foreground. This allows the algorithm to adapt differently for any image it is tasked with, giving a threshold that is more balanced (i.e. doesn't threshold by too much or by too little). [10] The thresholded image is returned.

Pseudocode for character segmentation

(where image is the path to an image file)

```
image←READ(image)
thresholded=OTSU_THRESHOLD(image)
w,h=image.SHAPE[:2]
IF COUNT_NON_ZERO(thresholded) < ((w*h)//2) THEN
     image ← (255-image)
ENDIF

orig ← image
weighted ← ADD_WEIGHTED(image, alpha=5, beta=-220)
kernel ← ONES(2,1)
dilated ← DILATE(weighted,kernel)
thresholded ← OTSU_THRESHOLD(dilated)
```

```
thresholded ← (255-thresholded)
contours,hierarchy ← FIND_CONTOURS_EXTERNAL(thresholded)
sortedConts ← SORTED(contours)
charlist←[]
w,h ← image.SHAPE[:2]

FOR i,ctr IN ENUMERATE(sortedConts)
     x,y,w,h ← BOUNDING_RECTANGLE(ctr)
     IF w>=6 OR h>=10 THEN
          charlist.APPEND(orig[y:y+h,x:x+W])
     ENDIF
ENDFOR
RETURN charlist
```

Image fitting

Takes in an image of a character and returns the thresholded image fitted onto a 30×30 image, maximising the size of the original image without losing any of it.

First of all, the image size is taken and assigned to variables to x and y. If either of the x or y is above 40, x and y are divided by 1.05 until they are both below 40. If either of the x or y is below 40, they are multiplied by 1.05 until the next multiplication would put them above 40. The character image is resized with the dimensions as this new x and y. A new black image is created, and an offset is calculated for x and y, where the x offset and y offset equal (40-x)//2 and (40-y)//2 respectively. The resized character image is pasted onto the black image at the coordinates of the offset, which puts it directly in the middle of the black image. The new image is smoothed using PIL's built in SMOOTH_MORE, then returned.

Pseudocode for image fitting

Where image is the array of an image

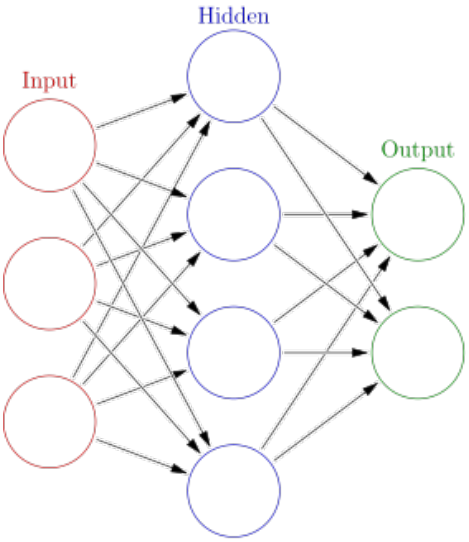
```
w,h ← image.SHAPE[:2]
threshtolded ← OTSU_THRESHOLD(image)
image ← SMOOTH(image)

x,y ← image.SIZE

WHILE x>30 OR y>30
     x,y←(i/1.05 FOR i IN (x,y))
 ENDWHILE
WHILE x<30 OR y<30
     IF ANY((i*1.05>30 FOR i IN (x,y)) THEN
          BREAK
     ELSE THEN
          x,y←(i*1.05 FOR i in (x,y)
     ENDIF
ENDWHILE
x,y=(FLOOR(i) FOR i IN (x,y)
image ← image.RESIZE(x,y)
img←NEW_IMAGE(mode="L",(30,30),color=255)
offset ← ((30-x) DIV 2,(30-y) DIV 2)
```

```
img ← img.PASTE(image,offset)
img ← SMOOTH(img)
return img
```

The OCR CNN

What is a neural network?

A neural network, at its most basic level, consists of a series of nodes (usually referred to as neurons) with weighted connections to other neurons (weights) on the next ‘layer’ of neurons. These types of layers with just linear neurons are referred to as dense layers.



A basic neural network [17]

Neurons are roughly analogous to real biological neurons in brains in the fact that they will cause other neurons to fire. Each neuron on one layer is uniquely linked to all the layers on the previous layer with weights, where the weights are an adjustable parameter. Each neuron will also have it’s own unique bias, which is another adjustable parameter. The value of a neuron is dependent on all the neurons on the previous layers multiplied by their respective weights for that particular neuron added to the bias for that neuron. The value from this is then put through an activation function to get the final value of the neuron.

There can be any number of layers in between the input and the output, these are called hidden layers. A neural network takes in two inputs, data and labels. Using the data with the labels, the network can ‘learn’ to recognize patterns in its input data by adjusting the weights between neurons, then using gradient descent to find the weights that minimize the loss function of the neural network.

A loss function is essentially a measure of how well a neural network has performed. It is based on the squared difference between the network’s output and the labels on the training data. The process of minimizing this loss function is known as backpropagation.[18]

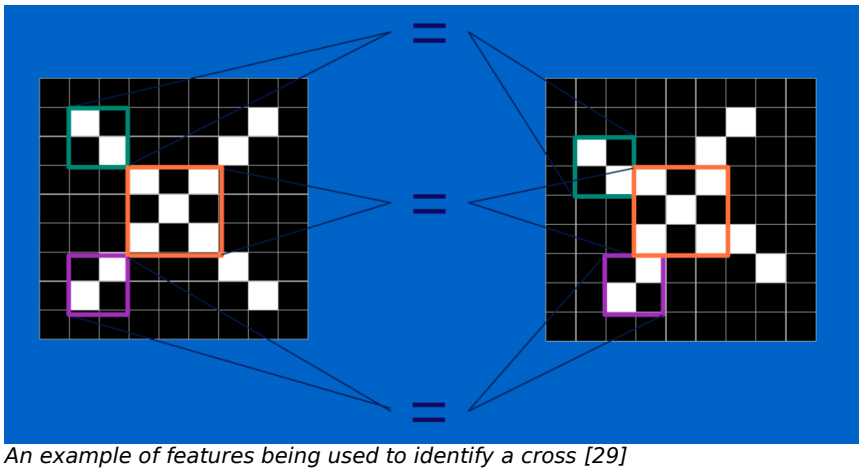
Backpropagation works by first propagating forward through the network and getting the output loss. Then, the rate of change in total loss is calculated with respect to the output layers (by differentiating). After that the rate of change for an output neuron will be calculated with respect to the net input to that neuron. Then the rate of change for the total net input will be calculated with respect to a parameter.

This process is repeated for all weights and biases, where the values for rate of change are used to calculate adjustments that should be made to the parameter. After a batch/mini-batch has gone through the network, the adjustments are all averaged out together and then each parameter is adjusted based on this value. A mini-batch is when the data is taken and split into smaller groups, this is simply to improve performance, as putting all the data into the network would take a massive toll on memory, putting the data in as a batch would cause a better overall accuracy but would be incredibly slow.

Neural networks can encounter a variety of problems such as overfitting, which is when a model learns the training dataset too well. It will perform well on the training dataset, but when set to other tasks, it won't be able to perform even slightly as well. This could be from the network learning irrelevant patterns in the training data, like the noise or even just memorising the dataset or for many other reasons. [24]

Convolutional Neural Networks (CNNs)

CNNs are specialized neural networks designed for extracting 'features' in a 2D image, they are not exclusively used for this, but it is their main use. A feature in an image is a part of the image, a small array 2D of values. CNNs will compare images piece by piece and see if there are features in common, this is how they see similarity.



An example of features being used to identify a cross [29]

By comparing small parts of the image, it doesn't matter where those parts are, the network will still be able to recognize them. This means that even if a character is rotated or distorted, the CNN can still recognize it.

When the CNN receives a new input, it tries to match features in every possible position. It calculates how the feature matches across the whole image and in doing so a filter is produced. This process is called convolution.

To calculate the match of the feature to a position in the image, each pixel in the feature is multiplied by each pixel in the area of the image that is being considered, then the answers are totalled and divided by the number of pixels in the feature. If there is a full match, there will be a 1 returned, if there is no match, there will be a 0 returned.

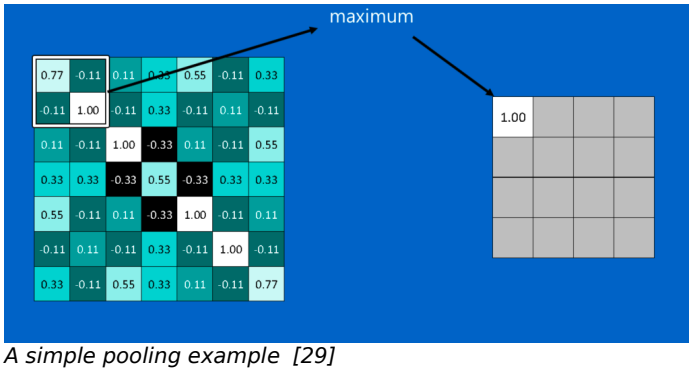
The process is repeated with all possible positions for the feature, the answer from each convolution is taken and a new 2D array is created from it. This 2D

array is like a filtered version of our original image, it shows where in the image you can find a feature.

After this, the convolution is repeated for all of the other features, giving us a set of filtered images. This is what happens on only one of the convolutional layers in the network.

Pooling

Pooling is when large images are taken and shrunken down to preserve their important details, a small window is stepped across an image, and the maximum value is taken, usually windows are 2x2 or 3x3 and take strides of 2 pixels.

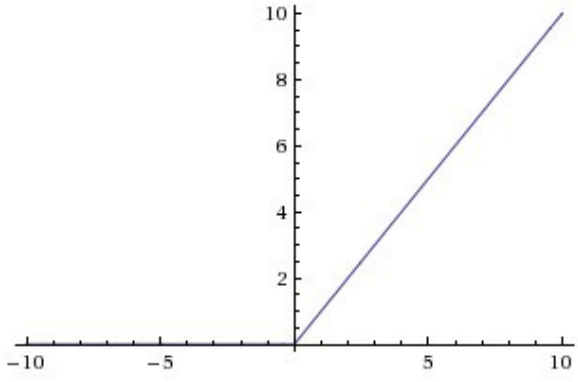


A simple pooling example [29]

This means that an image will have less pixels, and keeps the best fits of each feature in the window. The effect of this is that you fit features regardless of their position, meaning characters can be fitted even if they are in a different position. [29]

ReLU and ELU Activation

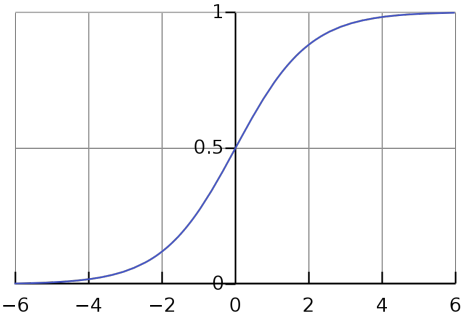
Both of these are examples of activation functions. An activation function is a function that converts an input signal of a node in a neural network to the output signal. As the sum of the weight*input+bias could be anywhere from -∞ to ∞, we need activation functions to put it between 0 and 1 [23]



An graph of a ReLU function [23]

ReLU stands for Rectified Linear Unit, at first glance it is a very simple function and seems much less effective than say a Sigmoid or tanh function, but ReLU has lots of advantages. Where A(x) is an activation function, ReLU is $A(x)=max(0,x)$, and has a graph that looks like the image on the right. The ReLU function gives an output x if x is positive and will otherwise give zero. ReLU is non linear in nature and combinations of ReLU are non linear. This means ReLU is a good approximator, so any function can be approximated by combinations of ReLU.

Consider a large neural network with sigmoid and tanh activations, almost all activations will be processed to describe the output of a network. This is computationally expensive



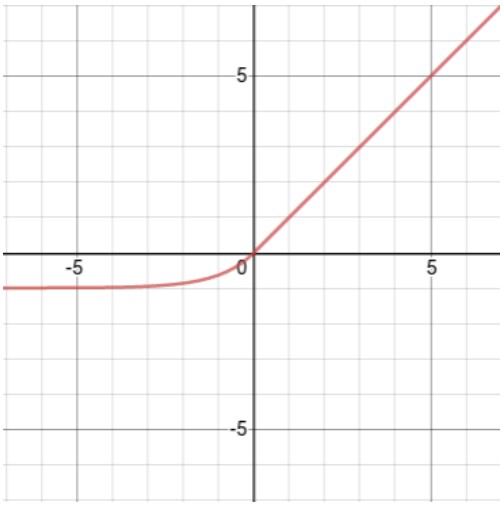
A graph of a basic sigmoid curve [25]

and could lead to overfitting. Furthermore, even if the value is massively low, the computational loss will not change while the activation may as well be zero. Additionally, sigmoid and tanh functions have the problem with very small gradients at high values, so if units are more ‘saturated’, they will have tiny activation values, basically ‘killing’ the neuron and preventing the neural network from being able to learn further. [26]

ReLU avoids this problem by having some neurons not activate (if they are below 0), hence making activations sparse and efficient. Imagine a network with random weights, probability dictates that about 50% of the network will yield 0 activation (due to ReLU only activating above 0), so fewer neurons are firing. This is a sparse activation and the network is ‘lighter’ as a result.

ReLU can encounter some problems despite all these benefits. Because of the horizontal line in ReLU for negative x, the gradient can go towards 0. For activations in that region, the gradient will be 0, this means the weights will not get adjusted during gradient descent. To put that into perspective, the neurons will simply not respond to any variations in error or input (as when gradient is 0, nothing can change). This makes whole areas of the network just passive to any changes in input and is referred to as dying ReLU.

ELU is another alternative to these activation methods and is fairly similar in terms of how it works. ELU is like ReLU or Leaky ReLU, but instead of having a linear gradient if the value is negative or just simply returning zero, it has an exponential. ELU’s value for a (or the alpha) can be adjusted also, but in my model I have just set it to 1, a good default.



An example ELU graph

The equation for ELU is as follows

$$A(x)=\begin{cases} x & \text{if } x>0, \\ \alpha(e^{x}-1) & otherwise \end{cases}$$, it produces higher classification accuracy with faster deep learning times. They also avoid the dying ReLU problem.

I initially chose to use ELU as a result of its usually high accuracy on CNNs [30], but after multiple attempts, I found ReLU to achieve much better results in the end. So my program uses ReLU activation for all of the layers.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is one of the oldest and still one of the most popular ways of optimizing neural networks. To explain SGD, its easier to first consider regular gradient descent, the basic aim is to ‘descend’ a gradient into the point where the loss is the lowest, this point is called a minima. Gradient descent as an algorithm is iterative, it starts from a random point and steps

down the slope until it reaches the lowest point of the function. But how does it move down in steps? Well it basically follows 6 steps:

1. Find the gradient of the function with respect to each parameter (weight, bias etc.)
2. Decide a random initial value for each parameter
3. Update the gradient function by substituting in these parameter values
4. Calculate the step sizes for each feature as: step size=gradient * learning rate
5. Calculate the new parameters with the formula: new parameters=old parameters – step size
6. Repeat steps 3-5 until gradient is close to 0 or equal to 0

The formula in the end looks something like this:

$Q(w)=\frac{1}{n}\sum_{i=1}^{n} Q_i(w)$ where w is the parameter that minimizes the function Q(w), the Q_i represents the i^{th} observation in the training set

$w=w-\eta\nabla Q(w)=w-\eta\sum_{i=1}^{n}\nabla Q_i\frac{(w)}{n}$ Here, η is the learning rate and ∇ represents the gradient. [27]

The learning rate is a parameter that the user can change to influence how the algorithm converges. A larger learning rate is fast, but it would mean the steps that the algorithm takes down the slope would be huge, hence meaning that it could miss the minima.

However, it would be insanely computationally expensive to compute this on a system of any substantial size as you would be computing derivatives with respect to each of the variables for every single item of training data, this would require you to do billions of calculations, even for a small network. This is where the stochastic part comes in. Stochastic literally means random, so stochastic gradient will randomly pick one data point from the whole data set at each iteration, which massively lessens the computational loss.

The formula for stochastic gradient descent is of the form:

$w=w-\eta\nabla Q_i(w)$ [27]

This also links into mini-batching, where the data points will be randomly sampled from the mini-batch. This strikes a balance between the goodness of gradient descent and the speed of SGD. [28]

Adam

Adam is currently one of the most popular optimization methods for neural networks. It was published in a paper in 2014 and was showcased to have incredible speed gain over SGD without sacrificing training results.

Adam combines two different gradient descent extensions:

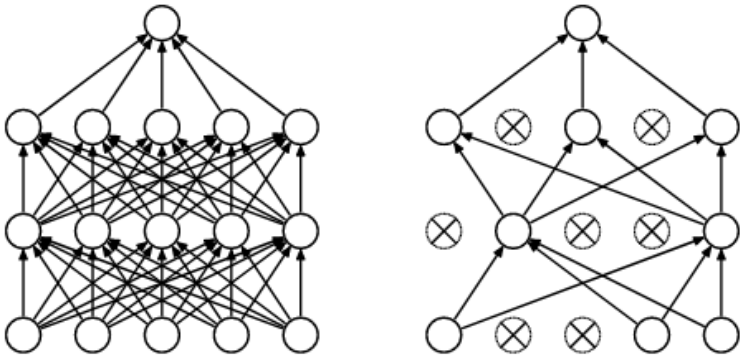
- Adaptive Gradient Algorithm (AdaGrad)– Maintains a per-parameter learning rate improving performance on problems with sparse gradients
- Root Mean Square Propagation (RMSProp) – Maintains per-parameter learning rates based on the average of recent gradient changes for a weight (i.e. how fast it is changing).

Adam will calculate the a moving average of the gradient and the squared gradient. The two parameters beta1 and beta2 control the decay rates of these averages. [30]

Adam has been found to have success on CNNs with non-linear activations, pooling and convolution and converge significantly faster than AdaGrad and other optimizers. [31]

Dropout

Dropout is a very common way to prevent overfitting with fully connected layers in neural networks. It was first proposed in a paper from 2014 by Nitish Sriva et al. wherein they suggested that by randomly dropping neurons and their respective connections temporarily, you can help reduce overfitting.



(a) Standard Neural Net (b) After applying dropout.

When dropout is used, a value is assigned to the dropout for a particular layer. This value dictates the probability a neuron and all its connections (in and out) will be dropped. For example, a common value is 0.5, so for this you would expect roughly half of the neurons to be dropped along with their connections. The original paper suggests that for hidden layers, 0.5 appears to be an optimal probability value for most tasks.

The reason behind the effectiveness of dropout can be explained using the biological analogy of sexual reproduction in evolution. In sexual reproduction, half of the genes are taken from the mother and the other half from the father with a very small amount of mutation. On the other hand there is asexual

reproduction, where all the genes from one parent are passed down to the next with slight mutations. While asexual reproduction seems like it would allow a good set of genes to be passed down through generations, sexual reproduction is how most organisms have evolved.

This is likely because when sexual reproduction occurs, sets of good genes are split up and will thus have to learn work independently (i.e. achieve something useful either alone or with a small group of others). As a result, this creates an organism which is on the whole more ‘well-rounded’, it will be adapted to do more different things and its overall fitness will be higher.

In neural networks, when using dropout each hidden unit will have to learn to work with a random sample of other units, leading it to create useful features on its own and not rely on other units to account for it. This means that hidden neurons won’t end up detecting useless features, and hence will overfit less. [33]

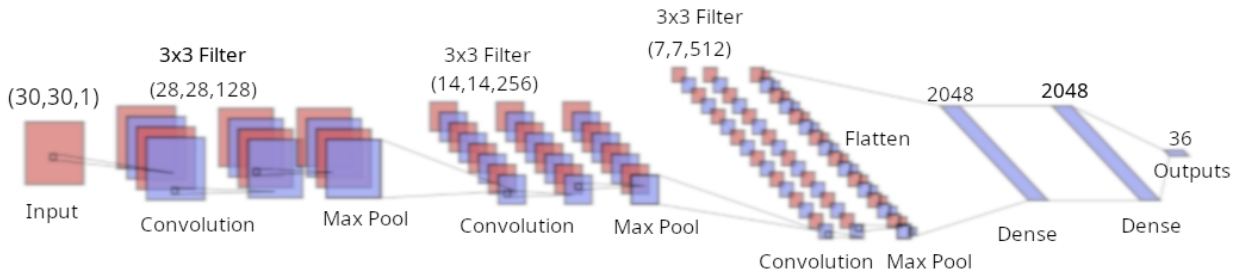
Neural Network Structure Overview

After much testing and deliberation I have settled on a network that has convolutional layers, pooling and dense layers. The structure is as follows:

Layer	No. of adjustable parameters	Activation	Details
Input layer (inputs)	0	None	30×30 image input with 1 colour channel (black and white)
2D Convolutional (conv1)	1280	ReLU	Depth of 128, filter size of 3×3. Outputs 28×28 with 128 channels.
2D Convolutional (conv2)	147584	ReLU	Depth of 128, filter size of 3×3. Outputs 28×28 with 128 channels.
Max Pooling 2D (pool2)	0	None	Pool size of 2×2. Outputs 14×14with 128 channels.
2D Convolutional (conv3)	295168	ReLU	Depth of 256, filter size of 3×3. Outputs 14×14 with 256 channels.
2D Convolutional (conv4)	590080	ReLU	Depth of 256, filter size of 3×3. Outputs 14×14 with 256 channels.
Max Pooling 2D	0	None	Pool size of 2×2. Outputs

(pool4)			7×7 with 128 channels.
2D Convolutional (conv5)	1180160	ReLU	Depth of 512, filter size of 3×3. Outputs 7×7 with 512 channels.
2D Convolutional (conv6)	2359808	ReLU	Depth of 512, filter size of 3×3. Outputs 7×7 with 512 channels.
Max Pooling 2D (pool6)	0	None	Pool size of 2×2. Outputs 7×7 with 128 channels.
Flatten (flatten)	0	None	Flattens the layers for dense layers, outputs 8192 neurons.
Dense (dense1)	16779264	ReLU	2048 neurons, trained with 0.5 dropout
Dense (dense2)	4196352	ReLU	2048 neurons, trained with 0.5 dropout
Outputs (outputs)	73764	Softmax	36 neurons, gets the final predictions. There is 1 neuron for each possible character

This neural network model has a total of 25,623,460 trainable parameters (weights and biases). The diagram below represents the structure:



When this training completes, the weights and biases are saved to the ‘default_weights.h5’ file, which can then be loaded by the program for getting predictions. I have trained the network with a batch size of 512 for 5 epochs on data that has been randomly shuffled on each epoch.

Tensorflow and Keras

I have chosen to create my neural network using Tensorflow’s high-level Keras API. Tensorflow is Google’s open-source software library for dataflow and

differential programming, but is commonly used for machine learning and neural networks. It was developed by the Google Brain Team for Google's own internal use but was released publicly on November 9 2015. [19] It is currently being used by a massive number of major companies, ranging from Intel to Twitter. [20] It is practically the industry standard for neural networks and is a common starting point for most new learners too.

Keras is an open-source Python neural-network library, that was initially released on the 27th of March 2015. It has 200,000 users (as of November 2017) [21] and is integrated into Tensorflow's core library. It's user friendly and modular, so adding new layers or modifying the training process is straightforward [22], which seemed good for me as this would be the first ever time I've worked with neural networks.

Data preparation

Data can't just be put straight into a network as an image, it needs to be put into a form that can be understood as neurons. Neurons have a value from 0-1, meaning that the easiest form for images to be in is an array, with each pixel being represented as a number from 0-1, where 0 is black and 1 is white.

As well as this, the labels must be as numbers, so I have created a list which assigns all 36 of the possible characters to a number between 0 and 36.

It's also important to have some validation data to get an idea of how well the network is performing. Validation data is data that the neural network hasn't seen before and is not trained on, but instead at the end of a training epoch (a full cycle through all the training data), the validation data is run through the network and the error is calculated (how far the validation predictions are from what they should be).In the next section I detail how I created the training and validation data.

The last step I took in preparing the data was shuffling both the test and training sets manually. Although Keras has a built in shuffling parameter, I chose to use it only to shuffle the mini-batches within the data, so I needed to shuffle the whole data beforehand.

Creating the training data

To create suitable training data, I decided to synthesize my own data by taking a large font pack and make images using it and PIL.

Modules summary

- Pillow (PIL) – For creating the images and drawing on them
- String (string) – For listing out the possible characters for the image
- OS (os) – For managing and listing directories

Google Fonts

Google has made all their font files from Google Fonts freely available via GitHub as TTF files. [31] Conveniently, this is also the font format that PIL uses, so it was easy to integrate, all I had to do was clone the repository then use a simple command to find and move all the .ttf files recursively.

```
FOR /R "Downloads\fonts-master\ofl\" %i IN (*.ttf) DO MOVE "%i"
"Documents\NEA\charImages\Fonts\"
```

Checking and filtering the fonts

An issue I found with the fonts is that when it came to using them for training data, they may display incorrectly or be otherwise unsuitable for creating reliable images. For this, I created a simple program that allowed me to view the fonts as PIL would display them and from there I could easily choose whether to keep or remove the fonts using key bindings. This allowed me to hand-pick from over 2000 fonts in under an hour, which meant I could have a massive training dataset without sacrificing quality by letting hundreds of unsuitable fonts slip through the cracks. Checking by traditional means would simply have taken too long.

The program I made for this was called fontChecker.py, it had GUI class and was built on a tkinter window using PIL for images (so as to show how my actual program would have displayed the fonts).

It creates a 1200 by 500 white image with PIL, draws on a sample of black text and the font name and displays it. Space is bound to keep the font, then move to the next preview and return is bound to delete the font then move to the next preview. The three modules used are OS, Tkinter and Pillow (PIL).

After filtering the fonts, I was left with a total of 2281 fonts with which to create my training data.

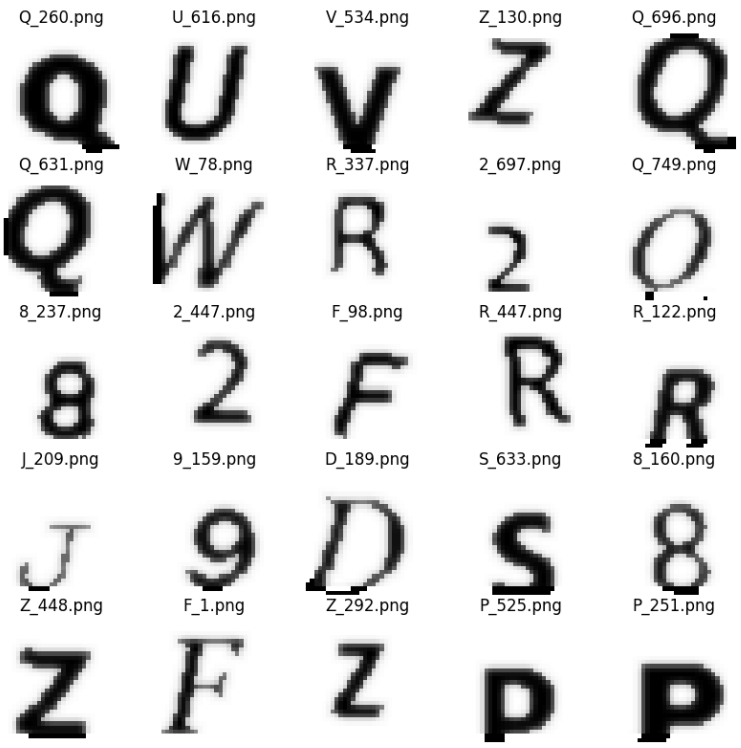
Pseudocode for checking fonts

```
CLASS GUI(Frame)
     INSTANTIATION METHOD (self,master)
          Frame.init(self,master)
          PRIVATE root ← master
          PRIVATE counter ← 0
          PRIVATE fontlist ← LISTDIR("charImages/Fonts/")
          PRIVATE x ← NEW_IMAGE(MODE="L",SIZE=(1200,500),color=255)
          updateLabel(self)
          root.BIND("<space>",pass1(self))
          root.BIND("<Return>",del1(self))
     PUBLIC METHOD updateLabel (self)
          x ← NEW_IMAGE(MODE="L",SIZE=(1200,500),color=255)
          fontchoice=fontlist[counter]
          font ← IMAGEFONT.TRUETYPE("charImages/Fonts/" +
fontchoice,SIZE=70)
```

```
          x.DRAWTEXT((0,0),TEXT="A B C D E F G H I J K L \n M N O P
Q R S T U V W X Y Z \n 0 1 2 3 4 5 6 7 8 9 \
n"+fontchoice,FILL=0,FONT=font)
          DISPLAY (x)

     PUBLIC METHOD pass1 (self)
          counter ← counter+1
          updateLabel(self)

     PUBLIC METHOD del1 (self)
          fontchoice ← fontlist[counter]
          REMOVEFILE(""charImages/Fonts/" + fontchoice)
          counter ← counter+1
          updateLabel(self)

instance ← GUI(Tk)
```

Training Data Generation algorithm

- The first character is selected
- The images are created as white 30×30 greyscale images by PIL
- The font is set as size 30 plus a random amount from -2 to 2 and is set to the first font in the directory
- The character is drawn in the centre of the image plus a random offset in x and y between -5 and 5
- The image is rotated by a random amount between -5 and 5 degrees
- The image is thresholded using Otsu's thresholding
- The image is smoothed with PIL's built in method
- The image is saved into the 'train' directory as PNG files, with filenames of their character and a number corresponding to whichever font they represent
- This process is repeated for each font, then for each character, it goes through the font directory 8 times
- A random sample of the fonts are taken and used as validation data, 500 images for each character



A sample of the training data created via this process

As well as this, I have also resized the Chars74K dataset to 30×30 and put it

into the same format as my training data, so it can contribute to expanding the dataset. This left me with a total of 693,216 training images and 17,992 validation images.

Pseudocode for resizing Chars74K

(before I ran this, I manually renamed all the folders to the character they contain)

```
constant CharsDir ← LISTDIR("Chars74K")
FOR i ← 0 TO LEN(CharsDir)
     subdir ← LISTDIR("Chars74K"+CharsDir)
     FOR j ← 0 TO LEN(subdir)
          im←IMAGEOPEN("Chars74K/"+CharsDir[i]+"/"+subdir[j])
          im←im.RESIZE((30,30)
          im←im.OTSUTHRESHHOLD
          im←SMOOTH(im)
          im.SAVE("charImages/train"+CharsDir[i]
+"_ch"+INT_TO_STRING(j))
     ENDFOR
ENDFOR
```

Pseudocode for generating training data

```
CHANGEDIR("charImages/")
constant charlist ← LIST(ASCII_UPPERCASE+ASCII_DIGITS)
constant fontlist ← LISTDIR("Fonts/")*8
FOR i←"" IN charlist
     FOR j←0 TO LEN(fontlist)+200
          x←NEW_IMAGE(mode="L",size=(30,30),color=255)
          IF j<=LEN(Fontlist)-1 THEN
               fontchoice←fontlist[j]
          ELSE THEN
               fontchoice←RANDOM_CHOICE(fontlist)
          ENDIF
          x←x.DRAWTEXT((30-w+RANDOM_INT(-10,10))DIV 2,(30-
h+RANDOM_INT(-10,0))DIV 2, i, fill=0, font=fontchoice)
          angle←RANDOM_INT(-15,15)
          IF angle<0 THEN
               angle←360+angle
          x←x.ROTATE(angle)
          x←x.OTSUTHRESHOLD
          x←SMOOTH(x)
          IF j<=LEN(Fontlist)-1 THEN
               x.SAVE("train/"+i+"_"+INT_TO_STRING(j)+".png")
          ELSE THEN
               x.SAVE("valid/"+i+"_"+INT_TO_STRING(j)+".png")
          ENDIF
     ENDFOR
ENDFOR
```

Implementation

Overview

The system is developed in Python 3.7.5. There are 3 python files for storing the program ‘mainProgram.py’, ‘imagePrep.py’ and ‘cnnModel.py’. There is one additional program that I used to create the training data and that is ‘trainDataGen.py’. ‘transparent.png’ is a placeholder file for when you open up the program. ‘default_weights.cpkt’ are saved weights for the CNN model. ‘tmp’ stores the temporary file temp.png. ‘Icons’ stores all the .ico files for the program. ‘charImages’ stores the ‘Fonts’ folder (which contains all the .ttf files for generating the training data), the ‘train’ folder (which stores generated training data) and the ‘valid’ folder (which stores generated validation data).

- **cnnModel.py** – Contains the subroutines for creating the CNN, for loading training data and for getting predictions for a given input image.
- **imagePrep.py –** Contains subroutines for text detection, character segmentation and fitting the image to the correct size (60×60).
- **mainProgram.py –** Contains the GUI which allows the user to access all the other functions for the importing, manipulating and scanning the images for text.
- **trainDataGen.py –** An isolated file that is used to create training data using the fonts in charImages.
- **fontChecker.py –** Another isolated file that is used to check fonts that will be used to create training data
- **chars74KResize.py –** An isolated file used to put chars74K data into the same form as the training data

Technical Skills

Building complex neural network models

Using the Tensorflow Keras functional API I have been able to create a complex convolutional neural network model. The model works by first defining an input layer, then chaining layer calls to specify how the model should propagate forwards, leading to a final output layer. The model is then defined by these inputs and outputs. This model is compiled and can then be used for training and predicting.

Exception handling

For generating training data, I have used exception handling in the case of an unreadable font. Rather than terminate the entire program, forcing me to check

which font produces an error and why. This allows me to import large font packs without worrying about checking every individual font

Image manipulation (in array and PIL form)

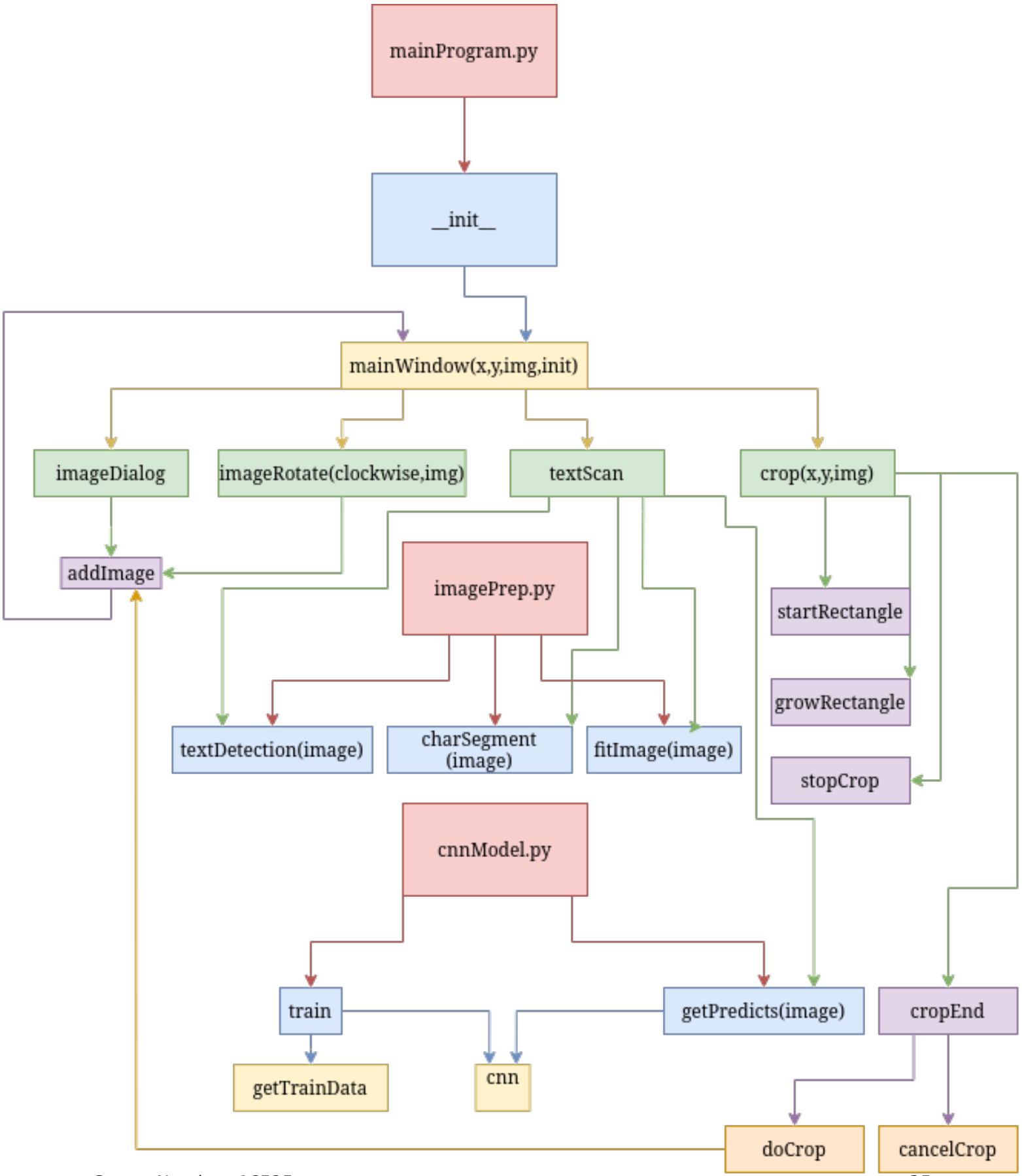
For the part of the program dedicated to preparing images so they are suitable for the neural network, I have utilised a wide range of image processing techniques.

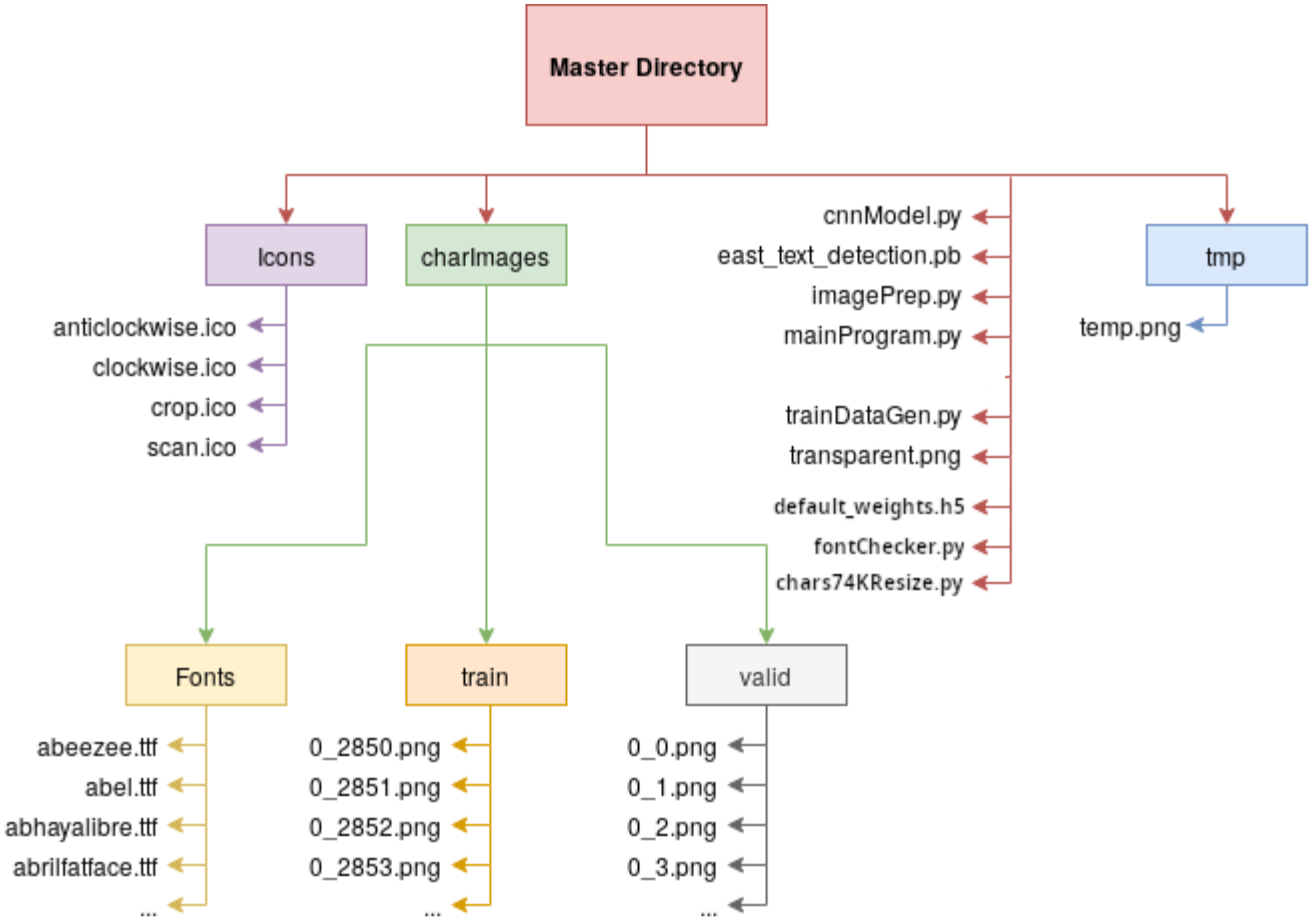
Some of the techniques involve direct array manipulation and others will involve using libraries like OpenCV or PIL to perform more calculation-heavy and complex manipulations and convolutions.

Array manipulations

For much of the program I have had to use various advanced techniques to manipulate arrays for things like sorting based on a variety of factors and for things like preparing data for the neural network.

Hierarchy Diagram





Directory Structure Diagram

Code listing

mainProgram.py

```
import os,tkinter.messagebox,numpy as np, imagePrep,cnnModel
from tkinter import filedialog, ttk
from tkinter import *
from PIL import Image, ImageTk
class GUI(Frame):
    def __init__(self,master):
        Frame.__init__(self, master)
        self.__root=master
        self.__root.title("OCR AI")
        self.grid()
        #Setup the main window with the title OCR AI and
        #grid file management
        os.chdir(os.path.dirname(os.path.realpath(__file__)))
        #Change to the current path
        blankImg=Image.open("transparent.png")
        self.mainWindow(595,842,blankImg,True)
        #Sets up the main window  with a placeholder image
        self.__root.mainloop()
    def imageDialog(self):

self.__filename=filedialog.askopenfilename(initialdir=os.path.expanduser('~/
Pictures'),title="Select file",filetypes=(("Images","*.png *.jpg *.jpeg"),("PNG
```

```
Images","*.png"),("JPG Images","*.jpg"),("JPEG Images","*.jpeg"),("All
files","*.*")))
        #Gives the user a filedialog to select from different
        #image files (JPG,PNG and JPEG)
        try:
            self.__userimg=Image.open(self.__filename)
            self.addImage(self.__userimg)
            #Adds the image to the main canvas
        except:
            tkinter.messagebox.showerror("Error","Opening image failed, not a
recognized format.")
            #If the format is unrecognised, show an error message
    def addImage(self,img):
        imgSize=img.size
        while imgSize[0]>1600 or imgSize[1]>900:
            imgSize=[int(i//1.05) for i in imgSize]
            img=img.resize(imgSize,Image.LANCZOS)
            #decreases the image size in incrememnts of 1.05 until it is
            #below 1600 in width and below 900 in height
        imgSize=[round(i/32)*32 for i in imgSize]
        img=img.resize(imgSize,Image.LANCZOS)
        #Resizes the image to the earlier calculated size
        self.mainWindow(*img.size,img,False)
        #Reforms the main window
    def imageRotate(self, clockwise, img):
        if clockwise:
            self.addImage(img.rotate(270,expand=1))
        else:
            self.addImage(img.rotate(90,expand=1))
            #Rotates the image then passes it to the addImage subroutine
    def cancelCrop(self):
        self.__cropConfirm.destroy()
        self.__canvas.delete(self.__drawn)
        self.__cropButton.config(state="normal")
        #Removes the rectangle and returns the crop button to its normal state
    def doCrop(self, event,img):
        self.__canvas.delete(self.__drawn)
        self.__cropConfirm.destroy()
        if event.x<self.__start.x and event.y>self.__start.y:
            self.__cropImg=img.crop((event.x, self.__start.y, self.__start.x,
event.y))
        elif event.x>self.__start.x and event.y<self.__start.y:
            self.__cropImg=img.crop((self.__start.x, event.y, event.x,
self.__start.y))
        elif event.x<self.__start.x and event.y<self.__start.y:
            self.__cropImg=img.crop((event.x, event.y,self.__start.x,
self.__start.y))
        else:
            self.__cropImg=img.crop((self.__start.x, self.__start.y, event.x,
event.y))
        #Accounts for cropping in any direction (e.g. bottom right to top left)
        #Without this, certain directions for cropping would give errors
        self.__cropButton.config(state="normal")
        #Returns the image back to normal
        self.addImage(self.__cropImg)
        #Crops the image based on the user's crop and then passes the cropped
image to addImage
    def cropEnd(self, event,img):
        self.__cropConfirm=Toplevel(self.__root)
        self.__cropConfirm.title("Crop")
        self.__canvas.config(cursor="")
```

```
        self.__canvas.unbind("<ButtonPress-1>")
        self.__canvas.unbind("<B1-Motion>")
        self.__canvas.unbind("<ButtonRelease-1>")

self.__cropButtonA=Button(self.__cropConfirm,text="Confirm",command=lambda:
self.doCrop(event,img))

self.__cancelButton=Button(self.__cropConfirm,text="Cancel",command=self.cancelC
rop)
        #Gives the user the option of cancelling their crop or continuing with
it
        self.__cropLabel=Label(self.__cropConfirm, text="Crop
selection?").grid(row=0, column=0, columnspan=2, padx=3, pady=3)
        self.__cropButtonA.grid(row=1, column=0, padx=3, pady=3)
        self.__cancelButton.grid(row=1, column=1, padx=3, pady=3)
        self.__cropConfirm.grid()
        #Creates the window with the crop selection label
    def growRectangle(self,event):
        self.__canvas = event.widget
        if self.__drawn: self.__canvas.delete(self.__drawn)
        objectId = self.__shape(self.__start.x, self.__start.y, event.x,
event.y,fill="gray50", stipple="gray50")
        #Creates a square with a grey faux-transparency
        self.__drawn = objectId
        #Draws the semi-transparent cropping box for every time it grows
    def startRectangle(self,event):
        self.__start=event
        self.__drawn=None
        #Event handler for starting a crop
    def stopCrop(self,event):
        self.__canvas.unbind("<ButtonPress-1>")
        self.__canvas.unbind("<B1-Motion>")
        self.__canvas.unbind("<ButtonRelease-1>")
        self.__cropButton.config(state="normal")
        #Event handling for cancelling the crop
    def crop(self,x,y,img):
        self.__cropButton.config(state=DISABLED)
        self.__canvas.config(cursor="cross")
        #Changes the cursor to a cross
        self.__canvas.bind("<ButtonPress-1>",self.startRectangle)
        self.__canvas.bind("<B1-Motion>", self.growRectangle)
        self.__canvas.bind("<ButtonRelease-1>",lambda event:
self.cropEnd(event,img))
        self.__root.bind("<Escape>",self.stopCrop)
        #Creates the binds for stopping the crop and growing the rectangle
        self.__shape=self.__canvas.create_rectangle
        #Starts the cropping process
    def textScan(self):
        spell=SpellChecker(distance=1)
        testList,testStr=[],[]
        self.__img.save("tmp/temp.png")
        coordslist=imagePrep.textDetection("tmp/temp.png")
        #Runs the image through the textDetection
        if len(coordslist)==0:
            tkinter.messagebox.showerror("Error","No text found")
            #If the text detection doesn't return anything, show an error
        else:

imagesList,threshedList,characterList,resizedList,centerlist,heightlist,predictl
ist=[],[],[],[],[],[],[]
            for i in coordslist:
```

```
                centerlist.append([i[0]+(i[2]-i[0])//2,i[1]+(i[3]-i[1])//2])
                #Get a list of the centers of all the words
                heightlist.append(i[3]-i[1])
                #Get a lsit of the heights of all the words
                imagesList.append(self.__img.crop(i))
                #Makes a list of each image of words as returned by
textDetection
            for x in imagesList:
                x.save("tmp/temp.png")
                characterList.append([imagePrep.fitImage(x) for x in
imagePrep.charSegment("tmp/temp.png")])
                #Segment the characters and fit them to 30x30, add the result
to the characterlist
            for x in characterList:

predictlist.append(cnnModel.getPredict("default_model",np.asarray(x,dtype=np.flo
at32)/255.0))
                 #Gets the predictions for every character in each word then
adds it to a list of words
            centerlist, heightlist,predictlist = zip(*sorted(zip(centerlist,
heightlist,predictlist), key=lambda x: x[0]))
            #Sort the 3 lists based on the distance from left to right
            highest,height=zip(*sorted(zip(centerlist, heightlist), key=lambda
x: x[1]))
            #Sort these two lists by the y-coordinate of the centerlist
            highest=highest[-1] #Set the heighest to the last value in the list
            height=height[-1]
            lines=[[highest]]
            #Create a new line with the highest y-coord
            lineHeight=[[height]]
            #Set the lineHeight initially to the first height
            found=False
            for i in range(len(centerlist)):
                found=False
                for j in range(len(lines)):
                    avgHeight=sum(lineHeight[j])/len(lineHeight[j])
                    #Get the average height of a line
                    if any(centerlist[i][1]<(x[1]-avgHeight/2) for x in
lines[j]) or any(centerlist[i][1]<(x[1]+avgHeight/2) for x in lines[j]):
                        lines[j].append(centerlist[i])
                        lineHeight[j].append(heightlist[i])
                        found=True
                        break
                        #If the word being checked is within half of the line's
average height of any
                        #word center in the line, then it is added to that line
and the loop breaks

                if not found:
                    lines.append([centerlist[i]])
                    lineHeight.append([heightlist[i]])
                #if the word doesn't fit within half of the average height
                #of any center in any line it is added to a new line.
            lines[0].pop(0)
            #Remove the first item of the first line as this would be a
duplicate
            predictPrint="".join([("".join([predictlist[centerlist.index(j)]+" "
for j in i])+"\n") for i in lines])
            #Join the predictions into a string with a newline for every line
            tkinter.messagebox.showinfo("Image Text",predictPrint)
            #Display the predictions in a message box
```

```
    def mainWindow(self,x,y,img,init):
        if init==False:
            self.__canvas.grid_forget()
            self.__canvas.delete("all")
            #Reset the canvas if this isn't the first time being called
        self.__img=img
        self.__imgdisplay=ImageTk.PhotoImage(img)
        self.__canvas=Canvas(self.__root,width=x,height=y,highlightthickness=1,
highlightbackground="black")
        self.__canvas.grid(row=0,padx=20,pady=10,columnspan=21)
        self.__canvas.create_image(0,0,image=self.__imgdisplay,anchor="nw")
        #Adds the image to the canvas as a preview
        self.__imageButton=Button(self.__root, text="Import Image",
command=self.imageDialog,anchor="e")
        self.__imageButton.grid(column=0,row=1,pady=10,padx=2)
        self.__antiIco=ImageTk.PhotoImage(file="Icons/anticlockwise.ico")

self.__antiRotate=Button(self.__root,image=self.__antiIco,command=lambda:
self.imageRotate(False,img),anchor="w")
        self.__antiRotate.grid(column=17,row=1,padx=2,pady=10)
        self.__clockwiseIco=ImageTk.PhotoImage(file="Icons/clockwise.ico")
        self.__clockRotate=Button(self.__root,
image=self.__clockwiseIco,command=lambda: self.imageRotate(True,img),anchor="w")
        self.__clockRotate.grid(column=18,row=1,padx=2,pady=10)
        self.__cropIco=ImageTk.PhotoImage(file="Icons/crop.ico")
        self.__cropButton=Button(self.__root,
image=self.__cropIco,command=lambda: self.crop(x,y,img),anchor="w")
        self.__cropButton.grid(column=19,row=1,padx=2,pady=10)

self.__scanIco=ImageTk.PhotoImage(Image.open("Icons/scan.ico").resize((32,32),Im
age.ANTIALIAS).convert("RGBA"))
        #Resizes the scan icon to fit in the canvas
        self.__scanButton=Button(self.__root, image=self.__scanIco,
command=self.textScan, anchor="w")
        self.__scanButton.grid(column=20, row=1, padx=2, pady=10)
        #Creates all the icons and their respective buttons on the UI
        if init:
            self.__antiRotate.config(state=DISABLED)
            self.__clockRotate.config(state=DISABLED)
            self.__cropButton.config(state=DISABLED)
            self.__scanButton.config(state=DISABLED)
        #Disables the buttons if the user hasn't loaded an iamge yet

instance=GUI(Tk())
```

imagePrep.py

```
from imutils.object_detection import non_max_suppression
import numpy as np, cv2, os,random
from math import floor
from PIL import Image, ImageFilter, ImageDraw
os.chdir(os.path.dirname(os.path.realpath(__file__)))
#Sets the path to be where the file is incase it isn't already there
def textDetection(image):
    image = cv2.imread(image)
    #load the input image and get the image dimensions
    orig = image.copy() #Copies the image
    (H, W) = image.shape[:2]

    layerNames = ["feature_fusion/Conv_7/Sigmoid","feature_fusion/concat_3"]
    #gets the two layers we need
```

```
    net = cv2.dnn.readNet("east_text_detection.pb")
    #Load the pre-trained EAST text detector
    blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),(123.68, 116.78, 103.94),
swapRB=True, crop=False)
    #Creates a blob (collection of binary data) from the image
    net.setInput(blob)
    (scores, geometry) = net.forward(layerNames)
    #Puts the blob into the network on the layers we want

    (numRows, numCols) = scores.shape[2:4]
    #gets the number of rows and columns
    boundBox=[]
    confidences=[]
    for y in range(0,numRows):
        scoresData=scores[0,0,y] #Gets tbe confidence scores
        xData0=geometry[0,0,y]
        xData1=geometry[0,1,y]
        xData2=geometry[0,2,y]
        xData3=geometry[0,3,y]
        anglesData=geometry[0,4,y]
        #Gets all geometry needed to make bounding boxes
        for x in range(0, numCols):
            if scoresData[x]<0.2:
                continue#If the score doesn't meet the minimum confidence,
ignore
            (offsetX,offsetY)=(x*4.0,y*4.0)#Gets the offset factor (to account
for size)
            angle=anglesData[x]
            cos=np.cos(angle)
            sin=np.sin(angle)
            #Calculate the rotation angles if any
            h=xData0[x]+xData2[x]
            #Get width and height of bounding box
            w=xData1[x]+xData3[x]
            endX=int(offsetX+(cos*xData1[x])+(sin*xData2[x]))
            endY=int(offsetY-(sin*xData1[x])+(cos*xData2[x]))
            startX=int(endX-w)
            startY=int(endY-h) #Get the start and end co-ords
            boundBox.append((startX,startY,endX,endY))
            confidences.append(scoresData[x])

    finalBoxes = non_max_suppression(np.array(boundBox), probs=confidences)
    #Finds bounding boxes that are weak and/or overlapping and removes them
using non max suppression
    return finalBoxes

def charSegment(image):
    image=cv2.imread(image,0)#Read image
    _,thresholded=cv2.threshold(image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    #Thresholds the image using Otsu's method and binary thresholding
    w,h=image.shape[:2]
    if cv2.countNonZero(thresholded) < ((w*h)//2):
        image=(255-image)

        #Checks if image is black on white, inverts if not
    orig=image

    weighted=cv2.addWeighted(image, 5,np.zeros(image.shape, image.dtype), 0, -
220)
    #Increases contrast and decreases brightness
```

```
    kernel=np.ones((2,1),np.uint8)
    dilated=cv2.dilate(weighted,kernel,iterations=1)

    #Dilates the image to increase distinction between pixels

    _,thresholded=cv2.threshold(dilated,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    thresholded=(255-thresholded)
    #Thresholds the image using Otsu's method and binary thresholding

contours,hierarchy=cv2.findContours(thresholded.copy(),cv2.RETR_EXTERNAL,cv2.CHA
IN_APPROX_SIMPLE)
    #Finds the countours in the image (RETR_EXTERNAL finds external contours
only)
    sortedConts=sorted(contours,key=lambda ctr: cv2.boundingRect(ctr)[0])
    #Draws bounding boxes based on the contours
    charlist=[]
    w,h=image.shape[:2]

    for i, ctr in enumerate(sortedConts):
        x,y,w,h=cv2.boundingRect(ctr)
        #Gets the bounding box dimensions
        if w>=6 or h>=10:
            charlist.append(orig[y:y+h,x:x+w]) #'crops' the image arrays
    return charlist #Adds the character images to a list

def fitImage(image):
    w,h=image.shape[:2]
    _,thresholded=cv2.threshold(image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    #Thresholds the image
    image=Image.fromarray(thresholded)
    image.filter(ImageFilter.SMOOTH_MORE)
    #Smooths the image

    x,y=image.size
    while x>30 or y>30:
        x,y=(i/1.05 for i in (x,y))
    while x<30 or y<30:
        if any((i*1.05>30 for i in (x,y))):
            break
        else:
            x,y=(i*1.05 for i in (x,y))
    #Checks the size of the image and either enlarges it or shrinks it to fit in
multiples of 1.05

    x,y=(int(floor(i)) for i in (x,y))
    image=image.resize((x,y),Image.LANCZOS).convert("L")
    #Resizes the image to fit the x,y determined earlier

    img=Image.new("L",(30,30),color=255)
    offset=((30-x)//2,(30-y)//2)
    img.paste(image,offset)
    #Puts the image in the centre of a 30x30 white canvas
    img.filter(ImageFilter.SMOOTH_MORE)#Smooths the image again
    return np.array(img)
```

cnnModel.py

```
import numpy as np, os,random,string
from tqdm import tqdm #used to track progress
from tensorflow.keras.layers import Conv2D, Dense, Input, MaxPooling2D,
Activation, Flatten, Dropout
```

```
from tensorflow.keras.models import Model,load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.activations import elu
from PIL import Image
os.chdir(os.path.dirname(os.path.realpath(__file__)))
def getTrainData():
    charlist=list(string.ascii_uppercase+string.digits)
    #Gets a list of all possible characters
    traindir=os.listdir("charImages/train")
    testdir=os.listdir("charImages/valid")
    #List all the training and validation data
    data,labels,validdata,validlabels=[],[],[],[]
    for i in tqdm(traindir,ascii=True,desc="Train: "):
        try:
            im=Image.open("charImages/train/"+i)
            data.append(np.asarray(im))
            #Opens each image and adds it to the data list
            labels.append(charlist.index(i[0]))
            #Get the first character of the filename and
            #uses the list to convert it to a number (as tensorflow
            #can't read strings) then adds it to the labels list
        except:
            pass
    for i in tqdm(testdir,ascii=True,desc="Validation: "):
        try:
            im=Image.open("charImages/valid/"+i)
            validdata.append(np.asarray(im))
            validlabels.append(charlist.index(i[0]))
        except:
            pass
        #Adds the validation images and labels to the validation lists
    c = list(zip(data, labels))
    random.shuffle(c)
    data, labels = zip(*c)
    #Shuffle the data and labels by the same amount
    data,labels=np.asarray(data,dtype=np.float32),np.asarray(labels)
    data=data/255.0
    #Convert the image data to the correct form for the neural network
    c = list(zip(validdata, validlabels))
    random.shuffle(c)
    validdata, validlabels = zip(*c)
    #Shuffle the data and labels by the same amount

validdata,validlabels=np.asarray(validdata,dtype=np.float32),np.asarray(validlab
els)
    validdata=validdata/255.0
    #Convert the image data to the correct form for the neural network
    data = data.reshape((data.shape[0], data.shape[1], data.shape[2], 1))
    validdata = validdata.reshape((validdata.shape[0], validdata.shape[1],
validdata.shape[2], 1))
    #Rehsape the data to be acceptable for the network
    return data,labels,validdata,validlabels
def trainModel():
    earlystop = EarlyStopping(monitor='val_loss', patience=3,verbose=1,
mode='auto')
    #Callback EarlyStopping will stop the training if the validation loss
doesn't appear
    #to be changing, this is to prevent overfitting
    model=cnn(True)
    #Gets the model with the flag training as True
    data,labels,validdata,validlabels=getTrainData()
```

```
    #Gets all the training data
    model.fit(data, labels,
    epochs=5,batch_size=256,
    validation_data=(validdata,validlabels),
    shuffle="batch",
    steps_per_epoch=None,
    verbose=1,use_multiprocessing=True,callbacks=[earlystop]
    )
    #Train the model with 5 epochs, batch size of 256, data shuffling
    #multiprocessing and the early stopping callback
    model.save_weights("default_weights.h5")
    #Save the model's weights to the default_weights.h5 file
def cnn(training):
    inputs=Input(shape=(30,30,1))
    #Create a layer for the inputs

    conv1=Conv2D(128, (3,3), padding="valid",strides=1, activation="relu",
kernel_initializer="he_uniform")(inputs)
    #Create a 2D convolutional layer with a depth of 128 and a 3x3 kernel,
activation is relu

    conv2=Conv2D(128, (3,3), padding="same",strides=1, activation="relu",
kernel_initializer="he_uniform")(conv1)
    pool2=MaxPooling2D(pool_size=(2, 2),strides=2,padding="same")(conv2)
    #Add a pooling layer with pool size 2x2 and 2 strides

    conv3=Conv2D(256, (3,3), padding="same",strides=1, activation="relu",
kernel_initializer="he_uniform")(pool2)

    conv4=Conv2D(256, (3,3), padding="same",strides=1, activation="relu",
kernel_initializer="he_uniform")(conv3)
    pool4=MaxPooling2D(pool_size=(2, 2),strides=2,padding="same")(conv4)

    conv5=Conv2D(512, (3,3), padding="same",strides=1, activation="relu",
kernel_initializer="he_uniform")(pool4)

    conv6=Conv2D(512, (3,3), padding="same",strides=1, activation="relu",
kernel_initializer="he_uniform")(conv5)
    pool6=MaxPooling2D(pool_size=(2, 2),strides=2,padding="same")(conv6)

    flatten=Flatten()(pool6)
    #Flatten the convolutional layers

    dense1=Dense(2048,kernel_initializer="he_uniform", activation="relu")
(flatten)
    #Add a dense layer with 2048 neurons and relu activation

    if training:
        dropout1=Dropout(0.5)(dense1)

        dense2=Dense(2048,kernel_initializer="he_uniform", activation="relu")
(dropout1)
        dropout2=Dropout(0.5)(dense2)
        #If the model is training, use 0.5 dropout

        outputs=Dense(36,activation="softmax")(dropout2)
    else:
        dense2=Dense(2048,kernel_initializer="he_uniform", activation="relu")
(dense1)
        #If it is not training, do not use dropout
```

```
        outputs=Dense(36,activation="softmax")(dense2)
        #Create an output layer with 36 neurons, one for each character
        #This uses softmax, which is useful for getting a maximum

    model=Model(inputs=inputs,outputs=outputs)
    #Set the model as end-to-end with the inputs and outputs

model.compile(optimizer="Adam",loss='sparse_categorical_crossentropy',metrics=['
accuracy'])
    #Compile the model with the adam optimiser and
sparse_categorical_crossentropy loss
    return model

def getPredict(filename,image):
    image=image.reshape((image.shape[0], image.shape[1], image.shape[2], 1))
    #Reshape the images to be suitable for the network
    charlist=list(string.ascii_uppercase+string.digits)
    model=cnn(False)
    #Get the model with the training flag as false
    model.load_weights("default_weights.h5")
    #Load the weights
    predicts=model.predict(image,verbose=0)
    results=np.argmax(predicts,axis=1)
    endstr="".join([charlist[i] for i in results])
    #Get the predictions and join them into a string for the word
    return endstr
```

trainDataGen.py

```
from PIL import Image,ImageFont,ImageDraw,ImageFilter
import string,os,random,cv2,numpy as np
from tqdm import tqdm #Used for measuring progress
charlist=list(string.ascii_uppercase+string.digits)
#List all uppercase characters and all digits
os.chdir("charImages")
fontlist=(os.listdir("Fonts/")*8)
#List the fonts directory 8 times
for i in charlist:
    for j in tqdm(range(len(fontlist)+500),ascii=True, desc=i+": "):
        #For each character and for each font
        try:
            x=Image.new(mode="L",size=(30,30),color=255)
            #Create a new 30x30 image with the black and white mode
            draw=ImageDraw.Draw(x)
            #Create a drawing object for the image
            if j<=len(fontlist)-1:
                fontchoice=fontlist[j]
            #Set the font to the one being cycled through in the list
            else:
                fontchoice=random.choice(fontlist)
            #Choose a random font for the last 500 items (for validation)
            font=ImageFont.truetype("Fonts/"+fontchoice,size=28+
(random.randint(-4,2)))
            #Choose a random font size between 24 and 30
            w,h=font.getsize(i)
            draw.text(((30-w+random.randint(-10,10))//2,(30+random.randint(-
10,0)-h)//2),i,fill=0,font=font)
            #Add the text to the centre of the image, with an x offset between -
10 and 10
            #and a y offset between -10 and 0
            angle=random.randint(-15,15)
```

```
            if angle<0:
                angle=360+angle
            x=x.rotate(angle,fillcolor=255)
            #Rotate by a random angle between -15 and 15
            x=np.array(x)

_,thresholded=cv2.threshold(x,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
            x=Image.fromarray(thresholded)
            #Threshold the image using Otsu's method
            x=x.filter(ImageFilter.SMOOTH_MORE)
            #Smooth the image
            if j<=len(fontlist)-1:
                x.save("train/"+i+"_"+str(j)+".png")
            else:
                x.save("valid/"+i+"_"+str(j)+".png")
            #Save the last 500 items to the valid data folder
            #and all the others go to the train data folder
        except:
            pass
        #Try except loop in the case of errors with reading fonts
        #prevents program from stopping if one font is broken
```

fontChecker.py

```
import os
from tkinter import *
from PIL import Image,ImageDraw,ImageFont,ImageTk
os.chdir(os.path.dirname(os.path.realpath(__file__)))
class GUI(Frame):
    def __init__(self,master):
        Frame.__init__(self, master)
        self.__root=master
        self.__root.title("Font Checker")
        self.grid()
        #Setup a new tkinter window with title Font Checker
        self.__counter=0
        #Create a counter variable iwth value 0
        self.__fontlist=(os.listdir("charImages/Fonts/"))
        #List all downloaded fonts
        self.updateLabel()
        self.__root.bind("<space>",self.pass1)
        self.__root.bind("<Return>",self.del1)
        #Bind space to pass1 and Return to del1
        self.__root.mainloop()
    def updateLabel(self):
        self.__x=Image.new(mode="L",size=(1200,500),color=255)
        draw=ImageDraw.Draw(self.__x)
        #Create a new 1200,500 white image in B&W mide
        fontchoice=self.__fontlist[self.__counter]
        #Select the font based on what the counter is at
        try:
            font=ImageFont.truetype("charImages/Fonts/"+fontchoice,size=70)
            draw.text((0,0),"A B C D E F G H I J K L \n M N O P Q R S T U V W X
Y Z \n 0 1 2 3 4 5 6 7 8 9 \n"+fontchoice,fill=0,font=font)
            #Create an image with this sample text and the fontname at size 70
of the chosen font
            self.__imgdisplay=ImageTk.PhotoImage(self.__x)
            self.__label=Label(self.__root,image=self.__imgdisplay)
            self.__label.grid(row=0,column=0)
            #Display the previously created image
        except:
```

```
            os.remove("charImages/Fonts/"+fontchoice)
            #If the font couldn't be read, remove it from the font directory
    def pass1(self,event):
        self.__counter+=1
        self.__label.grid_forget()
        self.updateLabel()
        #Add one to the counter then show the new image
    def del1(self,event):
        fontchoice=self.__fontlist[self.__counter]
        os.remove("charImages/Fonts/"+fontchoice)
        #Remove the font if del is chosen
        self.__counter+=1
        self.__label.grid_forget()
        self.updateLabel()
        #Add one to the counter then show the new image
instance=GUI(Tk())
```

Testing

Test Plan

Test No.	Screen	Test Purpose	Test Description	Expected Outcome	Type	Result
1	Main (no image)	Test if image buttons are disabled as expected when program is initialised and an image has not yet been loaded, as this would cause errors otherwise.	Click buttons without loading an image	Buttons do nothing	Erroneous	Pass
2	Import Image	To ensure users can correctly upload JPG images into the main GUI	Click the import image button and open an example 500×500 JPG file	The image appears in the main window	Typical	Pass
3	Import Image	To ensure users can correctly upload PNG images into the main GUI	Click the import image button and open an example 500×500 PNG file	The image appears in the main window	Typical	Pass
4	Import Image	To test if users can upload oversized	Click the import image	The image appears in	Typical	Pass

		images and that they get resized to fit	button and open an example 4000×4000 JPEG image	the main window but smaller		
5	Import Image	Ensure non-image files are handled correctly	Click the import image button and open an example .odt file	An error message appears telling the user this is an erroneous format	Erroneous	Pass
6	Main (image loaded)	Test if image buttons click when there is an image loaded	Click buttons after loading image	Buttons click and their functions activate	Typical	Pass
7	Main (image loaded)	Test if images crop correctly	Click the crop button, draw a rectangle from a point in the top left to a point in the bottom right. Confirm the crop on the following windowDuring the crop the cursor changes and the button is disabled. The window refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again	During the crop the cursor changes and the button is disabled. The window refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again	Typical	Pass
8	Main (image loaded)	Test if images crop correctly for negative y coordinates	Click the crop button, draw a rectangle from a point in the bottom left to a point in the top right.	During the crop the cursor changes and the button is disabled. The window	Typical	Pass

			Confirm the crop on the following window	refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again. During the crop the cursor changes and the button is disabled. The window refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again		
9	Main (image loaded)	Test if images crop correctly for negative x coordinates	Click the crop button, draw a rectangle from a point in the top right to a point in the bottom left, Confirm the crop on the following window	During the crop the cursor changes and the button is disabled. The window refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again	Typical	Pass
10	Main (image	Test if images crop correctly for	Click the crop button, draw a	During the crop the	Typical	Pass

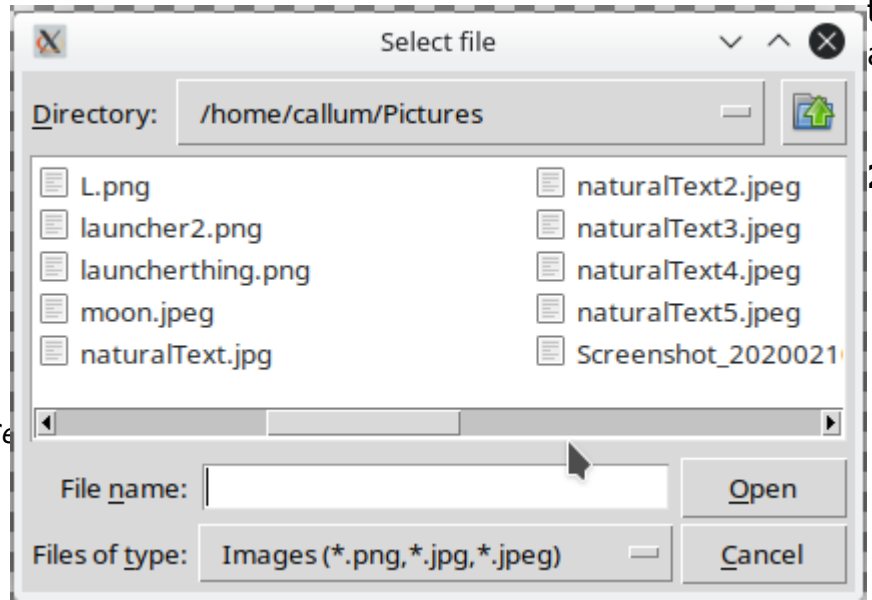
	loaded)	negative x and y coordinates	rectangle from a point in the bottom right to a point in the top left, Confirm the crop on the following window	cursor changes and the button is disabled. The window refreshes and shows the new cropped image, the cursor goes back to normal and the button is available to press again		
11	Crop confirm	To test if crops can cancel	Click the crop button, draw a rectangle then when asked to confirm click cancel	The button becomes enabled again and the cursor goes back to normal, the image is left unchanged	Typical	Pass
12	Main (image loaded)	To test if images rotate clockwise correctly	Click the clockwise rotation button	The image rotates clockwise by 90º	Typical	Pass
13	Main (image loaded)	To test if images rotate anti-clockwise correctly	Click the anti-clockwise rotation button	The image rotates anti-clockwise by 90º	Typical	Pass
14	Main (image loaded)	To test if images with no text are handled correctly	Click the text scan button with an image containing no text	An error message is displayed to tell the user the image has no text	Erroneous	Pass
15	Main (image loaded)	To test if text can be detected correctly	Click the text scan button with an image containing text on one line	A message box appears showing the contents of the text	Typical	Pass
16	Main (image loaded)	To test if text on multiple can be detected correctly	Click the text scan button with an image	A message box appears showing the	Typical	Pass

			containing text on two lines	contents of the text on different lines		
17	Main (image loaded)	To test if text in a non-natural scene can be detected	Click the text scan button with an image containing non-natural scene text	A message box appears showing the contents of the text	Typical	Pass
18	Main (image loaded)	To test if text in a natural scene can be detected	Click the text scan button with an image containing natural text	A message box appears showing the contents of the text	Typical	Pass
19	Main (image loaded)	To test if text in a more challenging natural scene can be detected	Click the text scan button with an image containing natural text	A message box appears showing the contents of the text	Typical	Partially passed

Evidence of tests

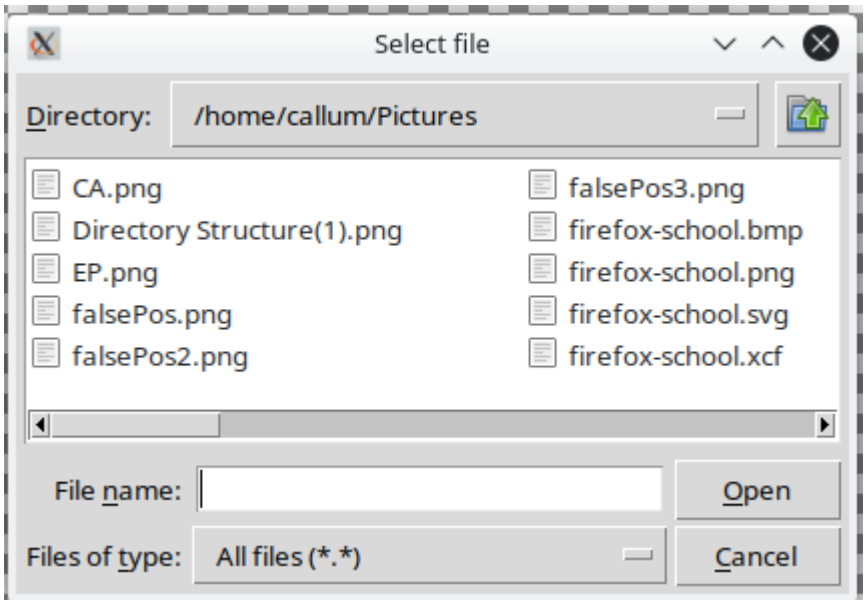
1: Disabled buttons

The above image is a cropped screenshot of the main program window with no images loaded. As you can see, there is the initial placeholder background and the buttons are disabled and unusable.

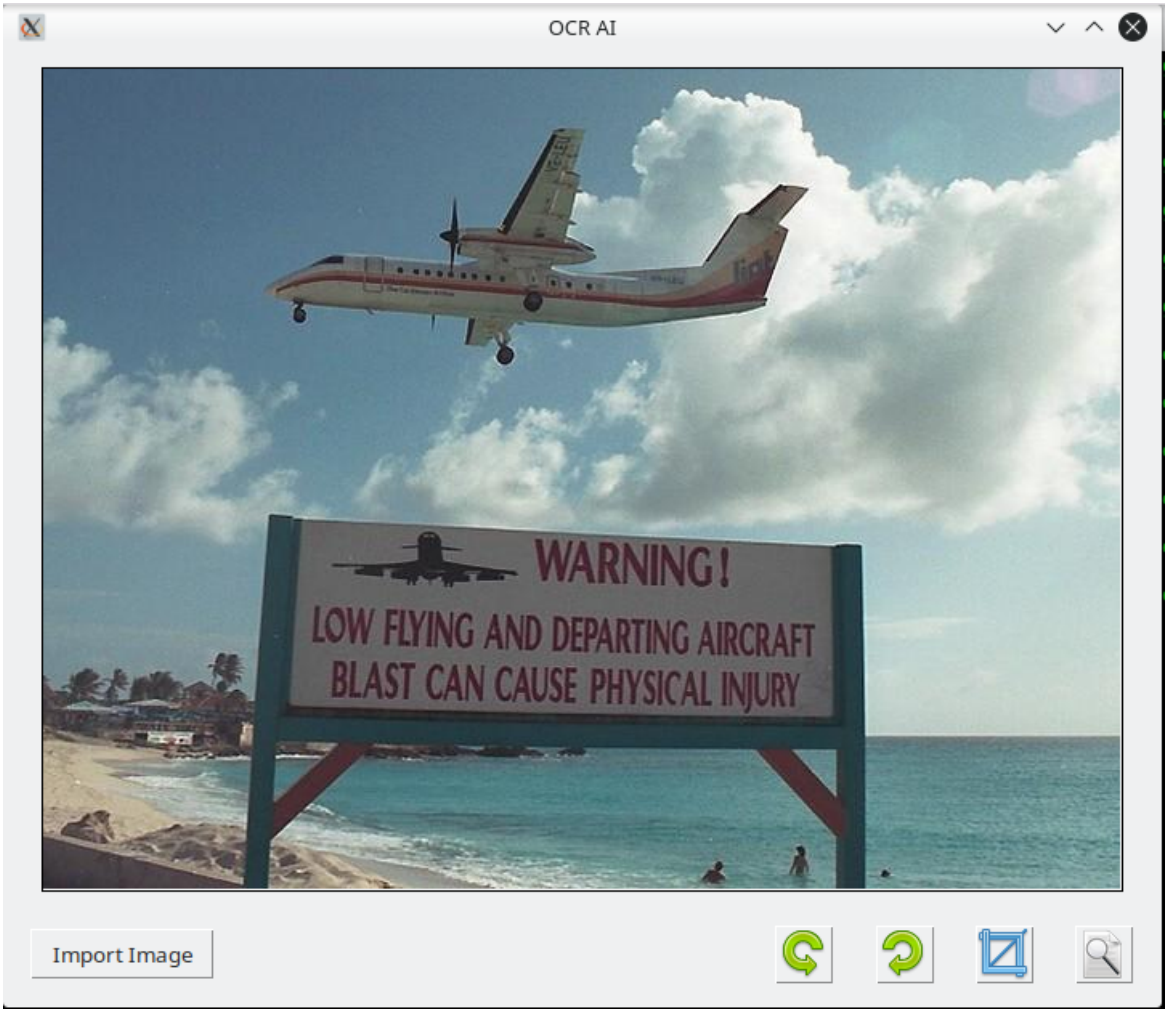


2: Importing an image

This is an image of the file dialogue for when the import image button is pressed, as you can see, it has opened into my pictures directory and has is showing jpg, png and jpeg image formats by default.



We can also choose to see all files, while some of these may cause errors, the errors can be handled and it is convention to have the option there.

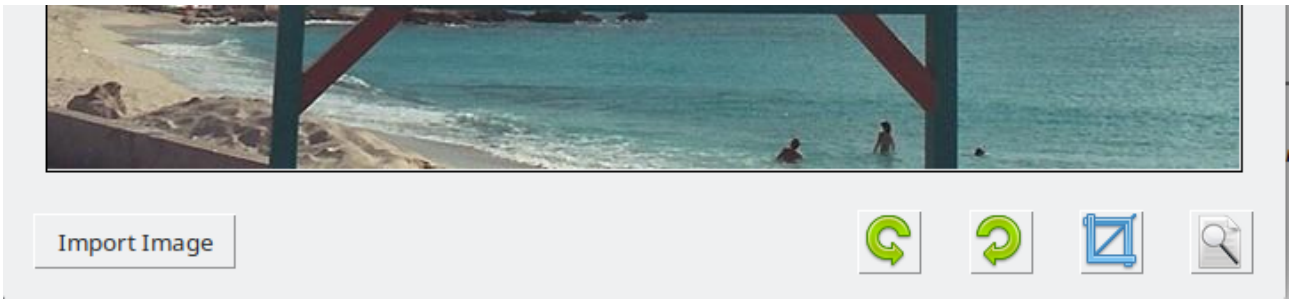


This is what an image looks like in the main UI after it has been imported via the file dialogue

5: Files that can’t be read as an image

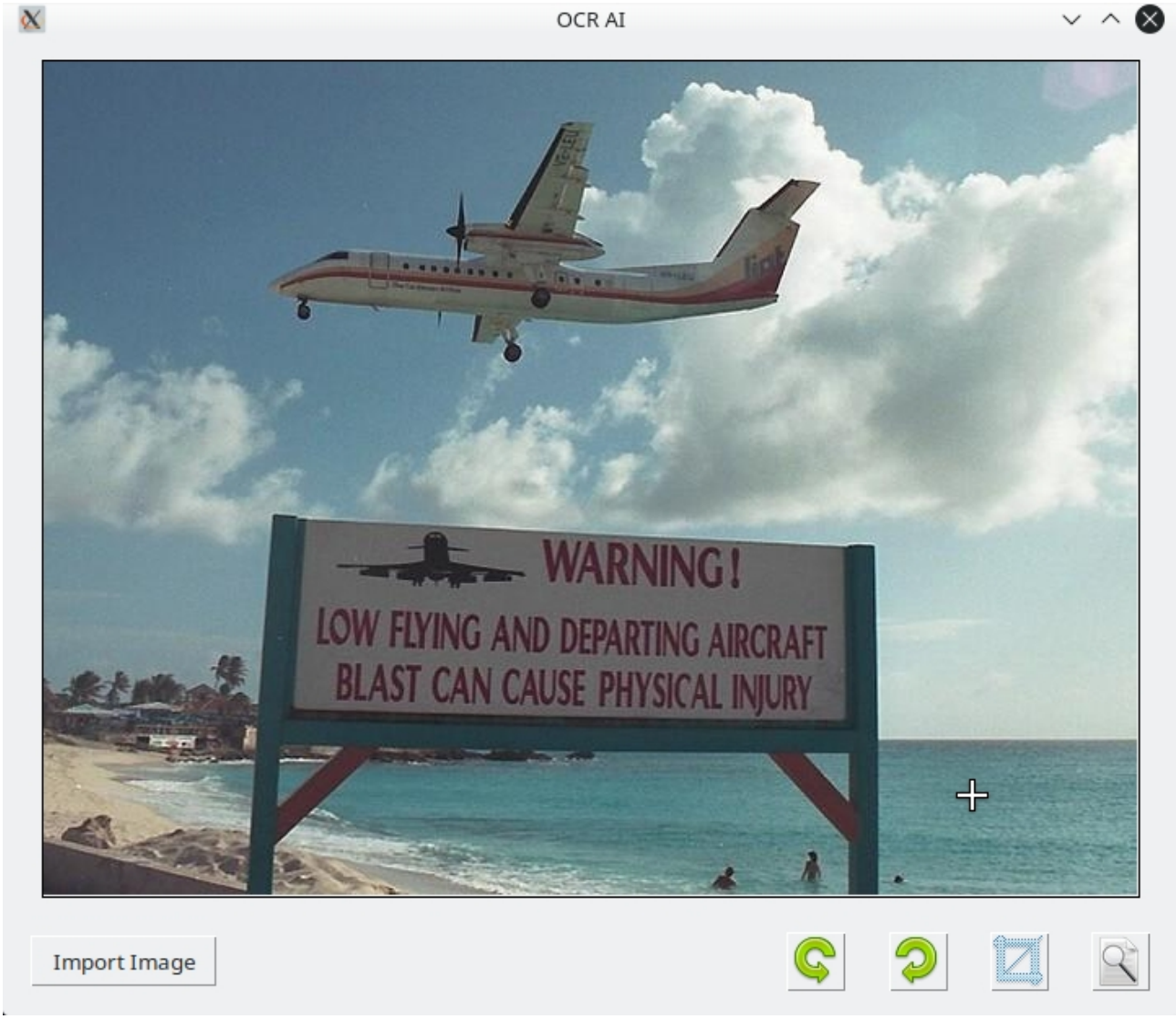
After opening an odt file (which obviously cannot be interpreted as an image) this error message is returned and the user is returned to the main screen

6: Loaded image

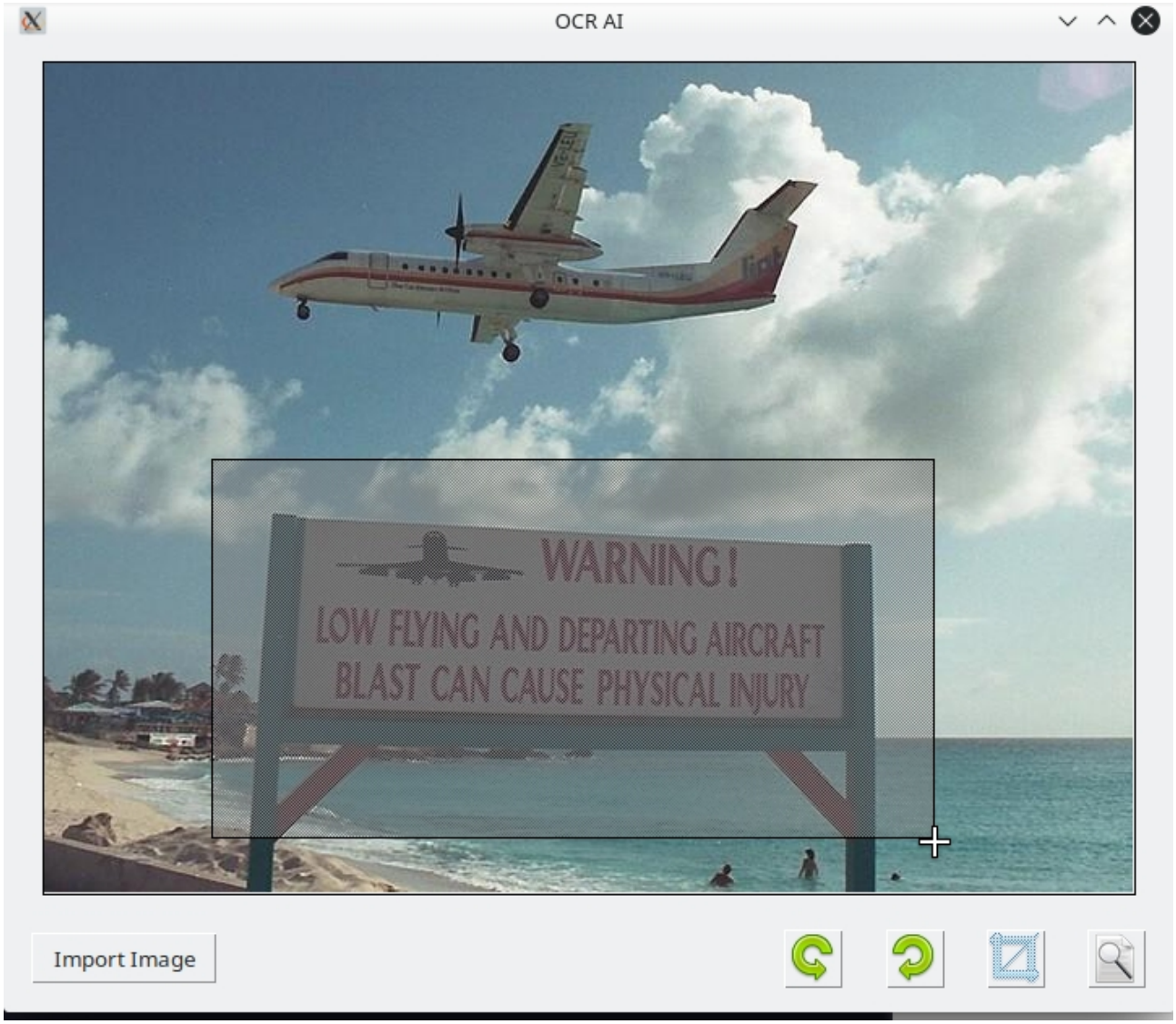


In this screenshot, an image has been loaded, and the buttons are now re-enabled and clickable.

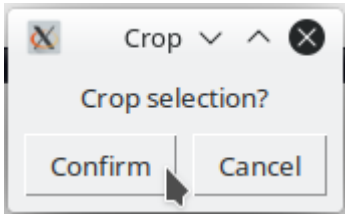
7: Cropping



This screenshot is right after the crop button has been pressed. The cursor has become a cross-hair and the crop button is disabled and unable to be clicked



At this point, the user has drawn a rectangle around the area of interest and when they release the mouse, they will be met with the crop confirm window



The user can choose to crop the image based on this rectangle, if they do the cropped image will now replace the image on the menu and the crop button will go back to being enabled and the cursor will return to normal. If they do not the crop button will be re-enable and the cursor returned to normal but nothing else will change.

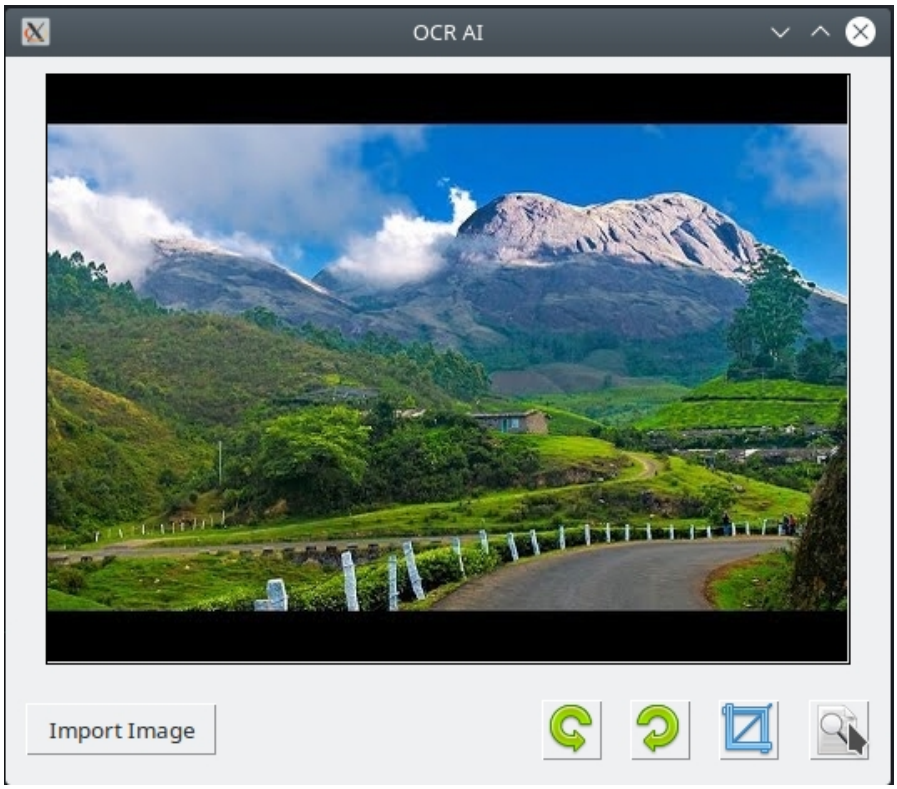
The above screenshot is after the user has cropped based on the earlier rectangle.

12: Rotating

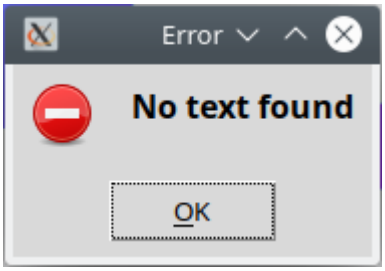
After the clockwise rotation button has been pressed, the image now displays as clockwise rotated

Similarly, this is the image after the anti-clockwise rotation button has been pressed.

14: Images with no text

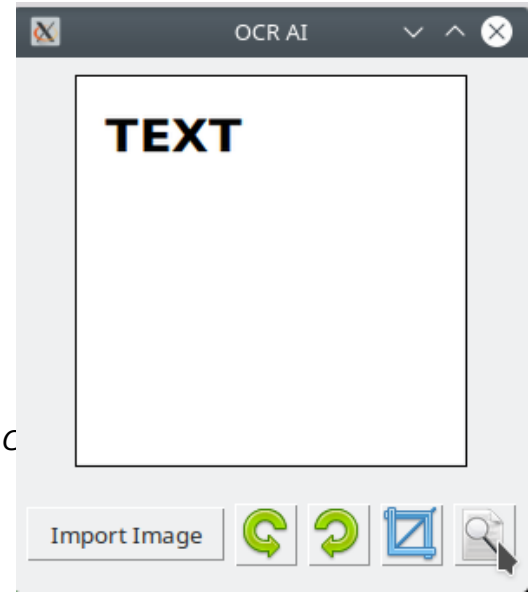


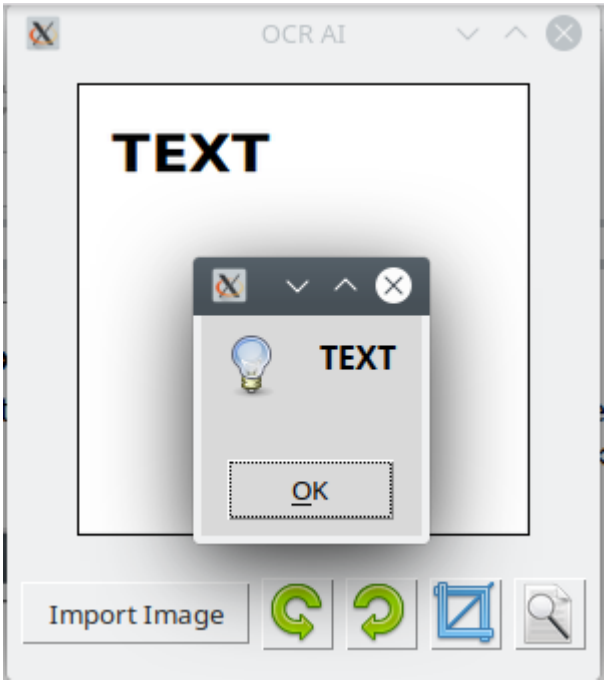
In this example, we have an image that clearly has no text but isn't completely blank. When we click the text scan button, the user will be presented with an error



15: Recognising text

In this first example, there is one word on one line that the network has to recognize. This is not a natural scene at all, so the network should have no trouble with it.





As expected, the network recognises it perfectly

16: Recognising text across multiple lines

Now we have a similarly simple example, but this time there are more words and they are spread across multiple lines.

Once again the network has recognised it perfectly.

17: Recognising text in a non-natural scene

In this example the text is now in a scene, but the scene is clearly not natural.

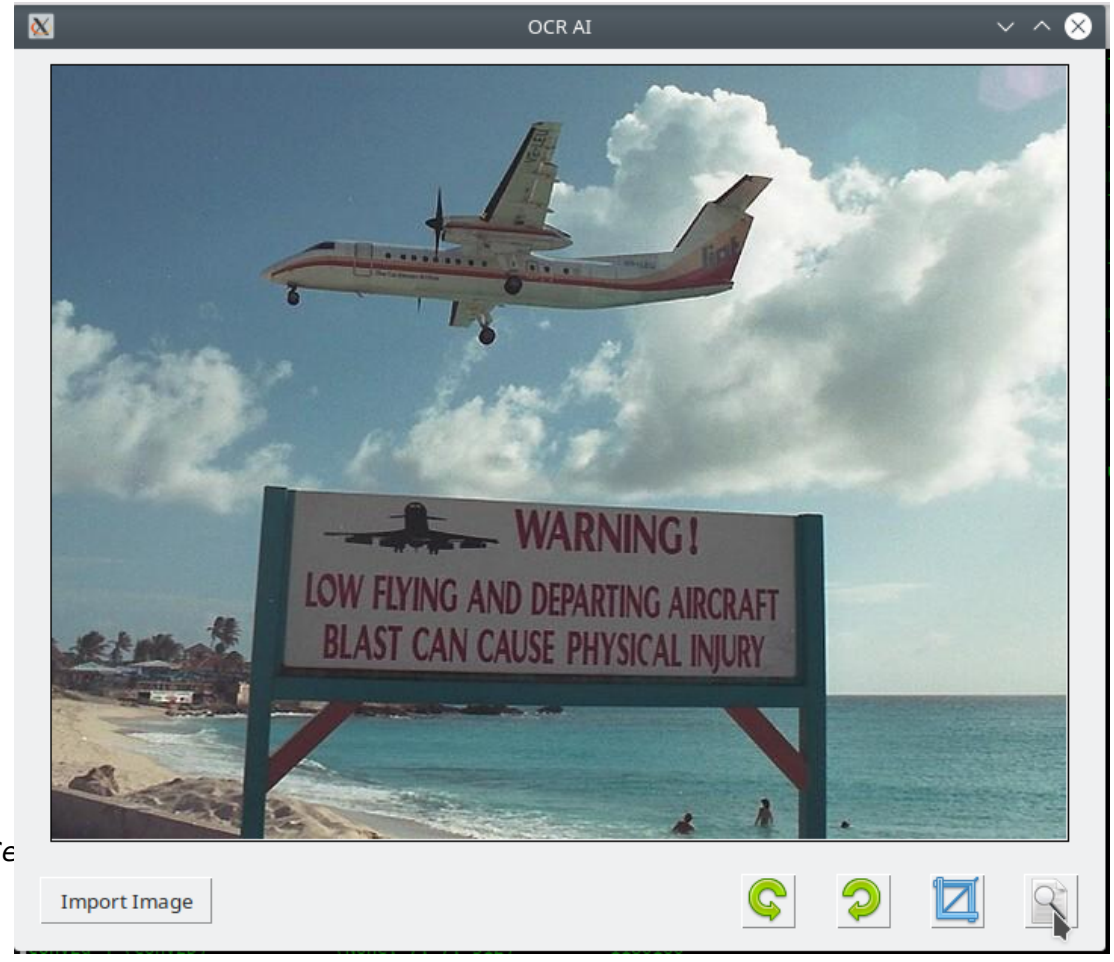
This is still a fairly easy task, so the neural network has recognised the text with full accuracy again.

18: Recognising text in a natural scene

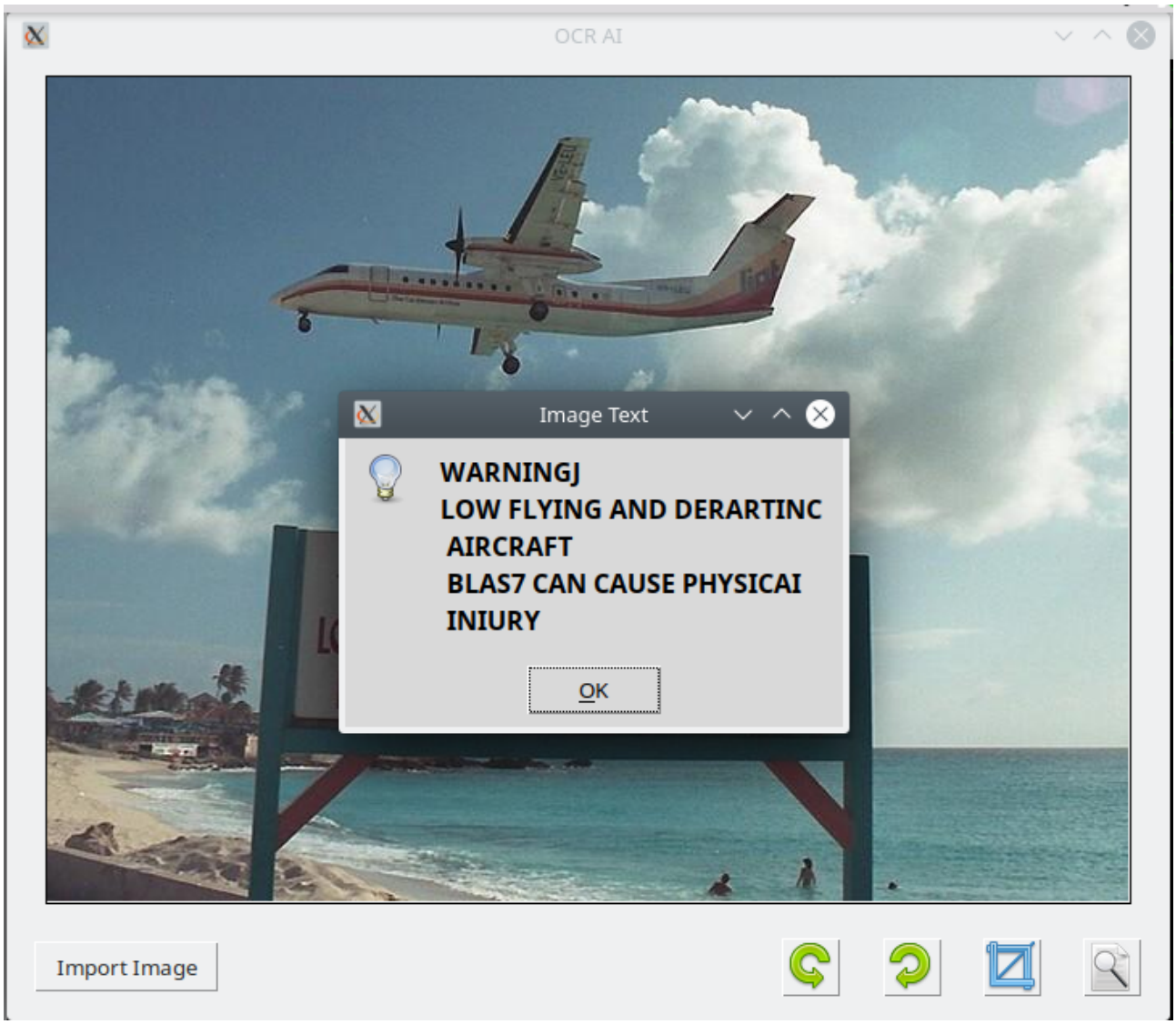
This example is now a 100% natural scene. Another challenge that is put forth by this is slightly offset text (rotated) and also the fact that this example is white on a dark background rather than black on white. It is also on multiple lines and contains another symbol that may accidentally be interpreted as a character.

The network has managed to recognise this almost perfectly (the only small error being the 0 instead of the letter O). It has also managed to split the lines perfectly and order the words perfectly.

19: Recognising text in a more challenging natural scene



In this scene there are plenty of challenges presented. The text is fairly low resolution, it is dark and low contrast, the text is rotated and the background is very busy with plenty of other features. As well as this, the letters are fairly close together, which would make character segmentation difficult.



Despite this, the network has managed to return a fairly good attempt. It couldn't deal with the exclamation mark as it was not trained to recognise it, and there are a few issues with some characters. But overall, it gets the majority of the characters correct.

Evaluation

Objectives Met

1. I have met my UI objectives fully. All of the functions I detailed in the documented design have been implemented without any complications or shortcomings.

2. For text detection and character segmentation I have met the objectives almost completely, the only issue being that sometimes the text detection may cut off words due to my inability to account for rotation of the text boxes.
3. As for the train data generation I met all of my objectives, which gave me a large and comprehensive training set.
4. The neural network has been implemented successfully, however I would say that the accuracy and speed aren't as good as I would like them, so it was not a perfect implementation.

User Feedback

I interviewed the client I had previously interviewed after I had showed them the program

Q: Were you happy with the performance?

A: Somewhat, I think that on a large scale it could begin to get slow, the performance was just about good enough for what I wanted.

Q: What were your thoughts on the user interface?

A: It was simple and got the job done so it exactly met my specifications. I liked also how it resized to fit pictures to a reasonable size, I could run it comfortably even when I imported high resolution images.

Q: Do you overall think the problem met your expectations?

A: On the whole, yes. The text recognition worked perfectly for lots of examples I tested it on. But there are some inaccuracies with the text detection. For example, when there are punctuation marks and when there are lowercase letters, the program doesn't know how to respond and will return a similar looking character.

Q: What would you want changed?

A: I would want capabilities for lowercase letters and punctuation, as they are still common and present in many examples of natural scene text. While most natural scene text will have just capitals, I still think lowercase characters and punctuation are important.

Q: Were there any parts of the program you would like to make a further comment on?

A: I thought the fact that the program showed user-friendly error messages was good, it is much better that just crashing. I also liked the cropping function function, it felt robust and smooth.

Possible Improvements

Using a CRNN (Convolutional Recurrent Neural Network) and the MJSynth dataset (a dataset of images of words), it is possible to achieve a much higher accuracy for text recognition of whole words. This is because CRNNs are trained on whole words and have a set lexicon, meaning they will only ever return English words. As this project's general application is for tasks like recognising English text in natural scenes rather than recognising groups of characters that may not necessarily be words (e.g. captcha tests), a CRNN would definitely be better in that regard. The only reason I didn't implement one was because of my lack of experience with neural networks. Earlier on in development, I attempted to implement a CRNN but it turned out to be much too difficult.

Another possible improvement would be adding lower case characters and punctuation. While most signs are fully in uppercase, some are in both upper and lower case. I chose not to implement different cases as there are so many similarities between certain uppercase characters and their lowercase counterparts. This can make achieving a higher accuracy much harder, so I chose to focus in on simply uppercase, but nevertheless it is possible. I also chose to not add punctuation due to the lack of it in the Chars74K dataset and also due to how similar it can look to other characters (e.g. an exclamation mark can look very similar to the characters i ,j, 1 and l).

I think that I could have possibly added a spell-checking/autocorrect algorithm to fix small errors in the text recognition. For example, sometimes the neural network will recognise capital O letters as a 0. A spell-checking algorithm would have been able to fix this and similar errors, making the program overall more robust.

A weakness of the program would also be the bounding boxes on the EAST text detection model. The model itself returns rotation angles for the bounding boxes, but when I tried to use this to rotate the words in the bounding boxes the results were consistently inaccurate and erroneous. If I had made this rotation compensation work, the results may would have been more accurate.

Conclusion

I think that the project has been a success on the whole, I met all my objectives and I am happy with the solution. I think the testing and feedback was thorough and closely related to my initial client interview. I would like the program to be more robust, which I could have done with some of the approaches I outlined in the possible improvements section. I think that for my current skill level I was able to achieve the best I could, while also furthering my knowledge and interest in neural networking.

References

[1] - ‘History of Computers and Computing, Internet, Dreamers, Emanuel Goldberg’, [online] Last updated n 26/03/2019, accessed on 05/09/2019 and found at: https://history-computer.com/Internet/Dreamers/Goldberg.html
[2] - ‘The Age of Spiritual Machines’ by Ray Kurzweil, published by Viking Press on 01/01/1999
[3] - ‘Tesseract Open Source OCR Engine’ (main repository) [online], developed by Google since 2006 and open sourced by HP in 2005 , found at: https://github.com/tesseract-ocr/tesseract
[4] - ‘Character Recognition in Natural Images’ by Teófilo E. de Campos, Bodla Rakesh Babu and Manik Varma from Proceedings of the International Conference on Computer Vision Theiroy and Applications (VISAPP) in February 2009. (the dataset is found at: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/)
[5] - ‘Natural Scene Text Understanding’ by Céline Mancas-Thillou, Bernard Gosselin, published by Faculté Polytechnique de Mons Belgium in June 2007
[6] - ‘EAST: An Efficient and Accurate Scene Text Detector’ by Xinyu Zhou et al., submitted to arXiv on 11/04/17 and revised on 10/07/17, found at arXiv:1704.03155v2
[7] - ‘OpenCV - About’ [online] by the OpenCV team, updated 2019, found at https://opencv.org/about/
[8] - ‘Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library’ by Adrian Kaehler and Gary Bradski, published by O'Reilly Media on 24/09/08
[9] ‘OpenCV Text Detection (EAST text detector)’ [online] by Adrian Rosebrock on 20/08/18 [online], accessed on 05/09/19, found at: https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
[10] ‘Non-maximum Suppression (NMS)’ [online] by Sambasivarao. K on 1/10/2019, accessed on 28/12/2019. Found at: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c
[11] ‘OpenCV: Image Thresholding’ [online] by the OpenCV Dev Team on 18/05/2018, accessed on 28/12/18. Found at: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html
[12]‘A threshold selection method from gray-level histograms’ by Noboyuki Otsu (大津展之), published by IEEE in January 1979
[13] ‘OpenCV Documentation: Structural Analysis and Shape Descriptors’ [online] by the OpenCV Dev Team on 21/01/2013, accessed on 28/12/2019, found at: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours

[14] ‘Understanding deep learning requires rethinking generalization’ by Chiyuan Zhang et al, submitted to arXiv on 10/10/2016, published in ICLR 2017, found at arXiv:1611.03530 [cs.LG]
[15] ‘Tensorflow: Open source machine learning’ [online] by Google’s official YouTube channel on 9/10/2015, accessed on 08/01/2020. Found at https://www.youtube.com/watch?v=oZikw5k_2FM
[16] ‘But what is a Neural Network? | Deep learning, chapter 1’ [online] by 3Blue1Brown(https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw, aka Grant Sanderson) on 5/10/2017, accessed on 08/01/2020. Found at https://www.youtube.com/watch?v=aircAruvnKk
[17] ‘Why Tensorflow’ [online] by Google on 11/10/2015, accessed on https://www.youtube.com/watch?v=aircAruvnKk 08/01/2020. Found at https://www.Tensorflow.org/about
[18] ‘Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis’ by Gregory Platetsky on 14/06/2018, accessed on 08/01/2020. Found at https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html/2
[19] ‘Why use Keras?’ [online] by Keras Development Team on 16/02/2018, accessed on 08/01/2020. Found at https://keras.io/why-use-keras/
[20] ‘Keras Documentation’ [online] by Keras Development Team on 27/03/2015, accessed on 08/01/2020. Found at https://keras.io/
[21] ‘Understanding Activation Functions in Neural Networks’ [online] by Avinash Sharma on 30/03/2017, accessed on 09/01/2020. Found at https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
[22] - ‘How to Avoid Overfitting in Deep Learning Neural Networks’ [online] by Jason Brownlee on 17/12/2018, accessed on 09/01/2020. Found at https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/
[23] ‘Logistic-curve.svg’ [online] by Qef (https://en.wikipedia.org/wiki/User:Qef aka Geoff Richards) on 02/07/2008, accessed on 09/01/2020. Found at https://commons.wikimedia.org/wiki/File:Logistic-curve.svg
[24] ‘neuralnetworks.thought-experiments/Activations’ by shekkizh (https://github.com/shekkizh aka Sarath Shekkizhar) on 21/08/2016, accessed on 09/01/2020. Found at https://github.com/shekkizh/neuralnetworks.thought-experiments/blob/master/Activations/Readme.md
[25] ‘The Tradeoffs of Large Scale Learning’ (pages 351-368) by Léon Bottou and Olivier Bousquet on 30/08/2011, published by MIT Press in 2012.
[26] ‘Stochastic Gradient Descent — Clearly Explained’ [online] by Aishwarya V Srinivasan on 07/09/2019, accessed on 10/01/2020. Found at https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

[27] ‘How do Convolutional Neural networks work?’ [online] by Brandon Rohrer on 18/08/2016, accessed on 01/02/2020. Found at https://brohrer.github.io/how_convolutional_neural_networks_work.html
[28] ‘Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)’ [online] by Djork-Arné Clevert et al., submitted on 23/11/2015, accessed on 02/02/2020. Found at arXiv:1511.07289
[29] ‘Google Fonts Files’ [online] by the Google Fonts team, submitted on 18/06/2015, accessed on 02/02/2020. Found at https://github.com/google/fonts
[30] ‘Gentle Introduction to the Adam Optimization Algorithm for Deep Learning’ [online] by Jason Brownlee on 03/07/2017, accessed on 04/02/2020. Foheightlistund at https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[31] ‘Adam: A Method for Stochastic Optimization’ by Dlederik P. Kingma and Jimmy Ba. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
[32] ‘Eroding and Dilating Image Objects’ [online] by IDL Online Help on 16/06/2005, accessed on 17/02/2020. Found at https://northstar-www.dartmouth.edu/doc/idl/html_6.2/Eroding_and_Dilating_Image_Objects.html
[33] ‘Dropout: A Simple way to Prevent Neural Networks from Overfitting’ by Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. Published in the Journal of Machine Learning Research 15 in 2014.