Computer science coursework: Handwritten Alfa-numeric character recognition

Table of Contents

[Analysis: 2](#_Toc101891841)

[Problem Identification: 2](#_Toc101891842)

[My clients: 2](#_Toc101891843)

[Features: 3](#_Toc101891844)

[Current solutions: 3](#_Toc101891845)

[Limitations: 4](#_Toc101891846)

[Data Inputs and Outputs: 5](#_Toc101891847)

[Success Criteria: 5](#_Toc101891848)

[Diagram of current systems: 6](#_Toc101891849)

[Character detection: 6](#_Toc101891850)

[Neural Network: 6](#_Toc101891851)

[Design: 7](#_Toc101891852)

[Key points of the solution: 7](#_Toc101891853)

[Neural Network: 7](#_Toc101891854)

[Character extraction: 9](#_Toc101891855)

[Structures: 11](#_Toc101891856)

[File GUI: 11](#_Toc101891857)

[File Model: 12](#_Toc101891858)

[File Dataset: 12](#_Toc101891859)

[File Training: 12](#_Toc101891860)

[File Character extraction: 12](#_Toc101891861)

[Pseudocode 13](#_Toc101891862)

[File GUI: 13](#_Toc101891863)

[File Model: 14](#_Toc101891864)

[File Dataset: 15](#_Toc101891865)

[File Train: 15](#_Toc101891866)

[File letterFinder: 15](#_Toc101891867)

[Developing the coded solution: 16](#_Toc101891868)

[Neural Network: 16](#_Toc101891869)

[Dataset: 16](#_Toc101891870)

[Model design: 19](#_Toc101891871)

[Training Process: 24](#_Toc101891872)

[Character detection: 28](#_Toc101891873)

[Pre-Processing: 28](#_Toc101891874)

[User Interface: 30](#_Toc101891875)

[Evaluation: 30](#_Toc101891876)

[Success Criteria that have met: 31](#_Toc101891877)

# Analysis:

## Problem Identification:

Currently there is still a large amount of data stored in hard copies, like forms, charts, etc., this data is not future proof and can easily be destroyed or lost. Hard copy data is also difficult to use as the data cannot be directly used by computers for processing. A software that recognizes handwritten characters from a form or document will greatly reduce the time and effort required to process information gathered in physical formats, the data can directly be stored in a database, or it can be processed to yield desired results.

The user would require an image of the document they wish to convert into a digital format.

The solution will recognize and extract handwritten alfa-numeric characters from an Image and parse then into a neural network that will recognize the character. The prediction from the neural network will then be output and saved into a file or database.

## My clients:

Users can be of any age, the category of users is very broad as the idea of the program can be applied in many situations, an average user may want to google a word or phrase, a company might want to scan their documents, or a scientist may want to analyse their data.

Businesses can employ OCR capabilities to convert images and PDFs (typically originating as scanned paper documents) to save time and resources that would otherwise be necessary to manage unsearchable data. Once transferred, OCR-processed information can quickly be used by businesses for better data management.

The benefits of OCR technology to businesses include:

* Elimination of manual data entry
* Resource savings due to the ability to process more data faster and with fewer resources
* Error reductions
* Reallocation of physical storage space
* Improved productivity

The OCR utility can also be applied to any administrative, data management workload where physical data is abundant and difficult to manage but commonly accessed.

In this fast-paced world, the bank is one of those institutions that use OCR the most. Many banks use OCR technology to achieve better transaction security and risk management.

The use of OCR software in banks can also scan many customers’ important handwritten documents like their loan documents and more. Additionally, adding facial recognition software with OCR increases security.

Increasingly, banking is done online, therefor having easy access to you forms and papers via a digital format is an important feature.

## Features:

The program will scan handwritten text from an image and convert it into a digital format, like a text file, this is called an Optical Character Recognition program (OCR).

There are 3 key parts of the program.

Machine learning model, this will predict the letter in the image that has been passed through by the user. It is done by training the underlying machine learning model, this is done by parsing though large amounts of training data. This process requires a training model and a training loop which are key features of the program.

Character extraction, the program will extract the text that the user wants to be converted from the image format to a text format.

GUI, this will allow the user to interact with the program.

## Current solutions:

There is code developed for the entire program and parts of the program, libraries like PyTesseract make it very easy as the entire development of the machine learning and neural network model is skipped, this allows focus to be shifted to the UI development.

**Machine learning:**

There are also many solutions that detail how to design your own neural network, like the TensorFlow tutorial for recognizing handwritten numbers using the MNIST dataset. From this tutorial you can easily re-purpose what you have learnt to recognize both hand-written numbers and letters, taking letter data from the A-Z Kaggle dataset. There are also many people that post their own solutions creating models with different combinations of layers and techniques to speed up the process of identifying the characters.

The simplest libraries for creating, and designing neural networks are PyTorch and TensorFlow. They both offer different benefits.

TensorFlow:

* Easy to write, it is quite beginner friendly as it uses the Keras API which has very ‘obvious’ implementations, the different layers have very clear-cut application and the method to define it is quite straight forward. The API also allows for higher level methods and practices, where they allow the creating of custom-made layers if necessary. Classes and functional methods can also be used to implement models.
* There is a large amount of functionality present in the library, it has pre-installed datasets for testing and experimentation, it has a large variety of optimizers and activation functions that can readily be used by the developer.
* Extensive documentation
* It is developed by Google
* Only CUDA support

<https://www.tensorflow.org/about>

PyTorch:

* Classes are heavily used, where the programmer defines the model in a more direct manner compared to TensorFlow. Here the programmer has more control over the fine details of the neural network.
* Extensive documentation
* Open Source
* Allows AMD GPUs to be used on Linux

<https://pytorch.org/features/>

**Using TensorFlow**

I decided to use TensorFlow(TF) as the library has more than enough customization for my project. The Keras library ,which comes with TF, includes different layer types and pre-built neural networks. A variety of layers were used, such as Convolutional2D, Dense, MaxPooling, Dropout, and Flatten. The standard training model was used to

**Character detection:**

The main library to use for character detection is Open-CV, there are probably alternatives however the library is used as standard across computer vision, e.g., it is utilized in PyTesseract which is developed by google.

The documentation and the resources for the library are very extensive as they detail how to use each part of the library using though examples.

<https://opencv.org/about/>

**UI:**

The UI is being developed using the Tkinter library, this library is very suited for simple window-based applications.

## Limitations:

There are many limitations for this project. Firstly, the neural network may make the wrong prediction, this is a worry as the dataset that I am using may not be very comprehensive in the different variations of handwriting styles. Secondly, the neural network cannot recognize any symbols other than the 26 letters of the English alphabet and the digits from 0-9, this means the recognition is very limited and cannot be used for mathematical problems or languages that use a different lettering system, like Mandarin or Hindi.

**Hardware Limitations:**

The current system is not usable on other devices such as mobile, its is also not usable on Linux and MacOS as the libraries have issues running on them.

The Tensorflow library allows for GPU acceleration for Nvidia GPUs, however this has not been accommodated in the program. Therefore the program can run on both AMD and Nvidia GPUs.

## Data Inputs and Outputs:

The files will be in an image format and will be selected by the user in the interface, the image will then be passed through the machine learning algorithm and the result will be outputted to a text file.

## Success Criteria:

Model

* Dataset
  + - The dataset needs to be loaded and pre-processed to be used later in the program
* Model config
  + - The neural network model will need to be defined, what layers, layer activation, optimizers, loss metrics etc.
* Training
  + - The training loop where the model is trained to recognize characters
* Storing model weights
  + - Path where the model weights are stored so they can be retrieved when required
* Re-trainable
  + - Allows the developer to change the model and allows for weights and parameters to be adjusted quickly without hassle.
    - Means develop a CLI style interface

Character extraction (Letter Finder)

* Filters
  + - Gray scale
    - Binarize
    - Prepares the image for contour detection
* Find contours
  + - Finds the contours of the image
* Extract required contours
  + - Retrieves contours that correspond to the given specification

GUI

* Files can be selected for OCR
  + - Filedialog required, interaction with os module.
* Files are displayed
  + - The given image will need to be displayed in the main window
* OCR is applied to file
  + - Files are parsed to the character extraction file
    - Parse extracted contours into ML model
    - Store predictions in a new text file
    - The image specified is sent through the prior files and a prediction is generated, it is then stored into a text file as the output.
* Display text file
* Exit

## Diagram of current systems:

### Character detection:

Diagram

Description automatically generated

### Neural Network:

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# Design:

## Key points of the solution:

### Neural Network:

**Dataset:**

MNIST, KAGGLE A-Z datasets are the datasets I decided to use. They are datasets that are derived from the NIST special database 19 which is officially backed by the US government. They have variations of all 36 characters, 26 English alphabet and 10 digits, they are in a 28x28 pixel format and have already had the canny filter applied, colour values are 1 or 0, reducing the need for postprocessing. The MNIST dataset comes packaged in with the TensorFlow library and the Kaggle A-Z dataset needs to be installed from the Kaggle website.

**MNIST:** 60 000 images

<http://yann.lecun.com/exdb/mnist/>

**Kaggle:** 372 451 images

<https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format>

**NIST 19:** <https://www.nist.gov/srd/nist-special-database-19>

**Layer composition:**

There are many different designs/selections of layers for a neural network, the 2 simplest implementations for this project are, a mix of Convolutional layers and Dense layers or just Dense layers.

The most basic, and my original, implementation for this project is using purely Dense layers. Here the neural network will take the colour values of each of the pixels in a 28x28 pixel image after it has been pre-processed, I.e., 784 inputs being either 1 or 0. These inputs will then be transformed, they will be multiplied by a weight and summed with a bias, this is the Dense layer. The Dense layers weight and bias are determined by an activation function, this function usually dictates the method in which each value is used. The initial model I started with consisted of a ‘Flatten’ layer, which takes the 28x28 image and makes it into an array of dimensions 784x1, and 2 Dense layers, the activation for this was a Sigmoid function. The example below shows the bare bones model with a ReLu activation function

**Basic Dense NN in tensorflow:**

Source Code Example:

import tensorflow as tf

model = tf.keras.Sequential([

    tf.keras.layers.Flatten(input\_shape=(28, 28)),

    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(10)

])

Output:

Text

Description automatically generated

This method is very inefficient as the neural network activates each pixel leading to increased computing times. Dense layers are also not specifically designed to deal with images, rather they are more general purpose and tend to be used at the end to make the prediction while specialized layers condense and refine the data to reduce processing time and computer resources. Therefor the Convolutional layer is used.

Convolutional Layers are designed to be used on images and computer vision, the number of weights per layer is a lot smaller, which helps a lot with high-dimensional inputs such as image data.

**Basic Convolutional NN:**

Source code:

CNN = model = tf.keras.Sequential([

        # Convolutional layers

        tf.keras.layers.Conv2D(32, (5, 5), input\_shape=(28, 28, 1), padding='same', activation='relu'),

        tf.keras.layers.Conv2D(32, (5, 5), input\_shape=(28, 28, 1), padding='same', activation='relu'),

        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Conv2D(64, (3, 3), input\_shape=(28, 28, 1), padding='same', activation='relu'),

        tf.keras.layers.Conv2D(64, (3, 3), input\_shape=(28, 28, 1), padding='same', activation='relu'),

        tf.keras.layers.MaxPooling2D((2, 2)),

        # flattening the array

        tf.keras.layers.Flatten(),

        # applying the FC layers

        tf.keras.layers.Dense(64, activation='relu'),

        tf.keras.layers.Dense(26, activation='softmax')

    ])

Output: Text

Description automatically generated

<https://towardsdatascience.com/ultimate-guide-to-input-shape-and-model-complexity-in-neural-networks-ae665c728f4b>

<https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b>

### Character extraction:

**Pre-Processing:**

The images need to be pre-processed before they can be fed into the Neural Network, this means applying filters.

Gray Scale – the colours are from 1 to 0 decimals included

Binarization - Changing all colour values in the image from 3 channel RGB to 1 or 0 where 1 is black and 0 is white, this is done using the threshold function from OpenCV

Canny - this filter is applied for edge detection and is the main way to extract the edges from the image to use in the Neural network

“Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various [computer vision](https://en.wikipedia.org/wiki/Computer_vision) systems. Canny has found that the requirements for the application of [edge detection](https://en.wikipedia.org/wiki/Edge_detection) on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the centre of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.” *- Wikipedia*

<https://en.wikipedia.org/wiki/Canny_edge_detector>

**Finding Contours:**

OpenCV “findContours” is used to locate the contours from the Canny image, The function retrieves contours from the binary image using the algorithm the user specifies, e.g., RETR\_EXTERNAL

**Sorting and resizing contours:**

Out of all the contours that are found I need to sort the contours out, I need to select the contours that I can pass through my neural network, this is done by restricting the types of bounding boxes that can be created from all the contours found.

Lastly, I will resize the contours so that they conform to the dimensions that my neural network can handle, 28x28, using some image manipulation.

## Structures:

### File GUI:

Class OCR

* Initialization
  + Main loop where the order of operations is created
* File selection
  + Allows users to select the files they wish to OCR
  + Add File
    - * Does the file handling
* File Viewer
  + Allow the selected file to be view in the main window
  + Allows the final text file to be viewed in the main window
* Character extraction
  + Parses the selected file though the Character extraction function defined in a separate file.
* Predictions
  + Creates and instance of the ML model
  + Loads the Weights
  + Parses the ROIs from the Character extraction into the model
  + The results stored in a separate .txt file
* Quit
  + Removes temporary directories
  + Destroys all windows

Structure:

New: Old:

A screenshot of a phone

Description automatically generated with medium confidenceA picture containing graphical user interface

Description automatically generated

### File Model:

* Layers
  + Convolutional
  + MaxPooling
  + Flatten
  + Dense
* Model compilation

Text

Description automatically generated

### File Dataset:

* creating dataset
  + Load datasets from disk and library (TensorFlow)
  + Combine into data and labels
  + Insert into array
  + Save into .npy file format
* Loading dataset
  + Randomize the images and labels with same permutation
  + Split dataset into test and train data

A picture containing text

Description automatically generated

### File Training:

* Parser
  + Arg: Train, Boolean
  + Arg: Load weights, Boolean
  + Arg: Evaluate, Boolean
  + Arg: Epoch, int
  + Arg: Test, Boolean
  + Return: args
* Main
  + Checkpoint for weights (where they are stored)
  + Load dataset from the dataset program
  + Class names (a-z and 0-9)
  + Initiating the model
  + Arg functionality

Text

Description automatically generated

### File Character extraction(LetterFinder):

* Directory creation
  + Check if pre-existing directories exist
  + If not, create temporary directories to store ROI and final images
* Filters
  + Binarizes image
  + Applies canny function to image
  + Finds the contours of the image (edge detector)
  + Store contours into and array
* Display characters selected on the image
  + Min and max contour width and height
  + Iterate over the contours in the contour array
  + Create bounding boxes for each of the contours that fall within the min max contour width and height
  + Store each bounding box in a list
  + Save image as final image
* Regions of interest
  + From bounding boxes array iterate over each one and cut that portion of the image out
  + Store in an array
  + Create images for each ROI in the array
  + Store in the ROI directory
* Outputs
  + roiPath and finalPath (path to directories for the final image and roi images)

Text

Description automatically generated Text

Description automatically generated with low confidence Text

Description automatically generated with low confidence

## Pseudocode

### File GUI:

**Class** Main:

**Function** Initialize:

* resultPath, the path where the text file is stored
* window, the main window of the GUI
* Frame, the main frame of the GUI
* Call function buttons() defined below

**Function** Buttons(self):

* Define width, height for all buttons
* Create the frame for the buttons
* **Function** fileView():
  + Retrieve the file from the specified by the user using the filedialog and storing it in self.window.filename
  + Set self.filename = self.window.filename
  + Open the file with the partial library function
  + Define fileViewButton with parameters - *(buttonFrame, text= filename), width=width, height=height, anchor=CENTER, command=openFile)*
  + Place button onto the grid
* addFile button with parameters - *(buttonFrame, text='Add File', width=width, height=height, anchor=CENTER, command=fileView)*
* charExtr button with parametes - *(buttonFrame, text='OCR', width=width, height=height, anchor=CENTER, command=self.characterExtraction)*
* quit button with parameters – (buttonFrame, text='Quit', width=width, height=height, anchor=CENTER, command=self.quit)
* place all the buttons onto the button grid
* place button grid onto the main grid

**Function** fileViewer(self, path, row, column):

* create the frame for the image, frameImg
* open the image specified
* create label for the image
* place image on grid in frameImg
* place frame in main grid

**Function** characterExtaction(self):

* set self.roiPath, self.finalPath to result from character\_extraction(self.filename) function found in letter\_finder.py
* use the self.fileViewer() function to display the image
* call the self.modelApplication()

**Function** modelApplication(self):

* set directory as self.roiPath
* open a new file
* loop through each image in the self.roiPath directory
* apply the model to each image using applyModel() function from model.py file
* store the result in a text file
* after loop has finished close text file

**Function** quit(self):

* try:
  + removing self.roiPath, self.finalPath directories
  + destroy all windows
* except AttributeError:
  + destroy all windows

**IF** \_\_name\_\_ == ‘\_\_main\_\_’:

* program = Main()
* program.window.mainloop()

### File Model:

**Function** create\_model(numClasses):

* model definition, layer composition
* compile the model with model.compile, parameters - (optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
* return the model

**Function** applyModel(imagePath):

* define the location of the model weights
* define class\_names
* instantiate the model
* load the weights
* model.predict(imagePath)
* return the class\_names[np.argmax(predictions)]

### File Dataset:

**Function** loading\_data\_set():

* load the numbers dataset from the tensorflow library
* load the alphabet characters from the character set .csv file
* store the dataset into arrays split into labels and images
* combine each of the arrays
* save the arrays in a .npy file

**Function** dataset():

* load the images and labels from the .npy file
* set shuffle the images and labels, ensure that the pairing of the image and label stays constant
* split the dataset into training and testing data using the train\_test\_split() from the sklearn library.
* Return the images and labels

### File Train:

**Function** Parser():

* Create variable to store the argument parser
* Add arguments
  + Training – whether the model should be trained
  + Load weights – whether the pre-existing values should be loaded from the checkpoint directory
  + Evaluate – whether the model should be tested against the test data, previously split from the dataset.
  + Epochs – how many re-runs of the training data should be made during the training process.
* Execute the parser
* Return parsed args

**Function** main(args):

* Store model weight path in variable
* Instance the dataset
* Create the class names, different possible predictions the program can make
* Create an instance of the machine learning model
* Check args and execute correspondingly

### File character extraction(LetterFinder):

**Function** boundingBox(imgContours (1st copy of the original image), contours):

* Create an array for storing all the dimensions of the bounding boxes, ‘rect\_d’
* Define the minimum and maximum contour height and width
* Iterate over ‘contours’ and create a bounding box for each one
* If the contour lines up with the min max width and height add it to the rect\_d array
* Else move on to the next
* Return the rect\_d array

**Function** ROI():

* Create an array to store roi bounding box, ‘roi\_n’
* Iterate over each of the bounding boxes in the rect\_d array
* Find the width, height, x-coordinate, y-coordinate for each contour
* Let store that part of the image in temp var ‘roi’
* Reshape the image slice
* Pad the image slice
* Append the image slice to roi\_n
* Create an image with all the contours bounding boxes displayed and save to finalPath as ‘final image.jpg’
* Return roi\_n

**Function** characterExtraction(imagePath):

* Specify ROI directory, final directory, and parent directory for the images
* If the ROI directory and final directory are not defined (they are temporary directories) then create them
* Read the image in from the imagePath specified
* Create 2 copies of the original
* Apply filters to original image
  + RGB to grayscale
  + Threshold the image (binarize)
  + Apply canny filter to the image
* Find the contours of the image

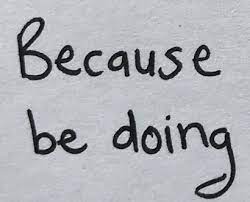
## Test Data:

Dataset tests:

Text

Description automatically generated

Character recognition test data:



Model test data:

Mnist and KAGGLE A-Z dataset split into test and training data

# Developing the coded solution:

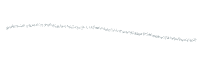
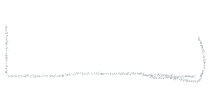
## Neural Network:

### Dataset:

The dataset needed to be retrieved, this is the MNIST dataset and the KAGGLE A-Z dataset, the MNIST dataset is already prebuilt into TensorFlow and the KAGGLE dataset can easily be downloaded from the KAGGLE website.

**MNIST dataset:** mnist = tf.keras.datasets.mnist

**KAGGLE dataset** – This dataset is not in the right/most efficient format, it can be installed as a .csv file form the Kaggle site, this however then needs to be turned into a NumPy array as the tensors in TensorFlow are not compatible with .csv formats. Doing this every time we need to retrain the model is very inefficient, therefor I found a small program that converts .csv into an array and edited it to be able to store these arrays as .npy files. I decided to include the MNIST dataset into the .npy files needed for the Kaggle dataset as that made everything easier and more manageable.

Text

Description automatically generated



**RED** – Loading dataset**|CURLY** – Resizing or combining the datasets**|GRAY** – Creating/Saving .npy file

The last thing I did for Datasets was creating a function that can create a dataset split between training data and test data.

Text

Description automatically generated

Before splitting the data, I am shuffling the data as it is in order, where all the A’s are at the front followed by the B’s, etc, this is done with the permutation function from NumPy.

Here I am using the train\_test\_split from the sk.learn library, another machine learning library, with the default split ratio being 3:1 for training data

There were multiple iterations of this, and was quite messy and inefficient at first, this is because I had not split the overarching problem of pre-processing the dataset into smaller parts making me try to complete all the data manipulation at once making it especially hard to follow and fix. Text

Description automatically generated

Text

Description automatically generated

I also made a Unit test for the dataset, this allowed me to check if the dataset was properly split into the test and train components

Graphical user interface, text

Description automatically generated

### Model design:

The model is the machine learning framework, it is the method in which the program will recognize what character is displayed, it is the core component of the project.

When stating with the project I had no experience in machine learning and as such there are multiple models of varying effectiveness employing different techniques.

**Model1 (Linear style):**

Text

Description automatically generated

This is the same as the example from the design section of this document, this is the most basic model, and it only predicts for numbers ranging from 0 to 9. It is comprised of a Flatten layer, which transforms a 2-dimensional array into a single dimension array, and 2 Dense, fully connected, Neural layers.

Image e.g. (with one output)



This model gave me fairly accurate predictions on the character being predicted. Results being: #insert results

After applying this model to the numeric dataset, I decided to implement the same model for the Alphabetic characters, this also yielded similar results in terms of accuracy

#Insert results

Text

Description automatically generated

Next I started to look at machine learning algorithms and techniques that are specifically used for image processing. Convolutional layers are not densely connected, not all input nodes affect all output nodes. This gives convolutional layers more flexibility in learning. Moreover, the number of weights per layer is a lot smaller, which helps a lot with high-dimensional inputs such as image data.

When a model uses Convolutional layers, it requires some specific supporting layers and features.

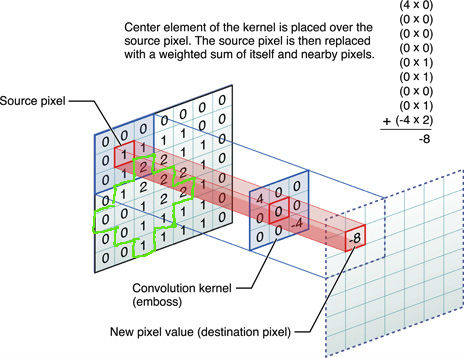
**Model 2 (Convolutional)(current model):**

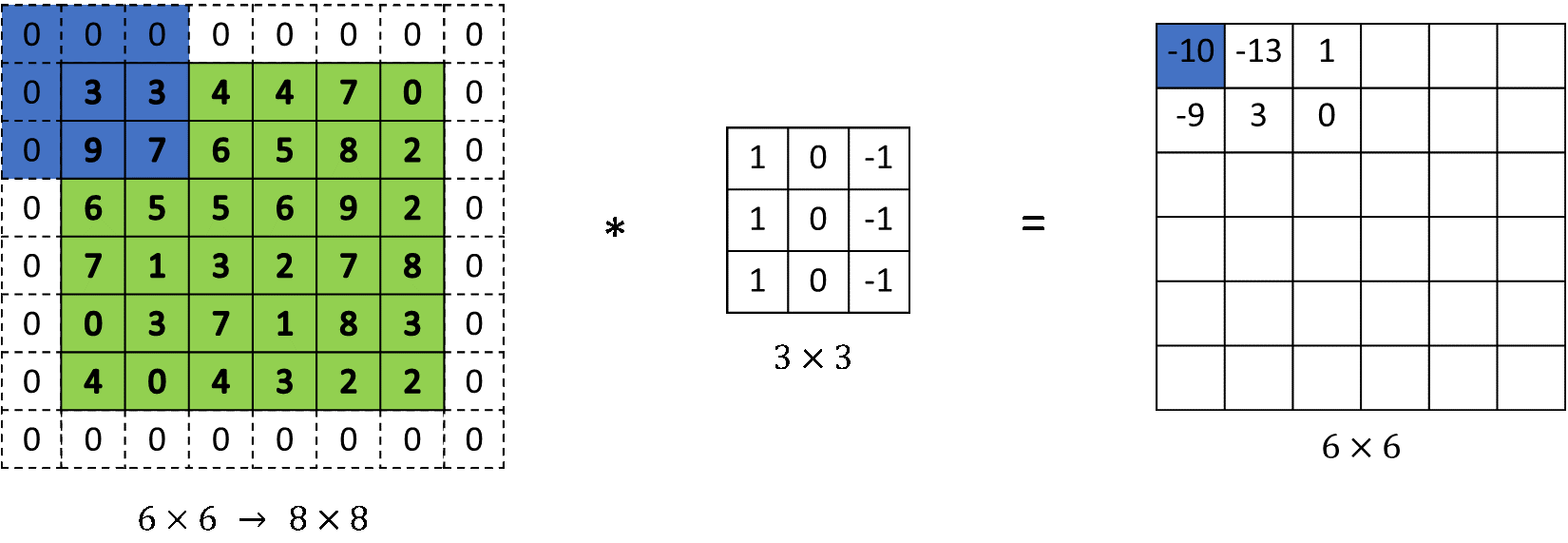
The kernel, window, iterates over the image until each pixel has been covered. A dot product between the pixels included in the window and the window are taken, the result of this dot product is known as a ‘feature’. A ‘feature map’ is a collection of ‘features’. Different types of kernels have different values in the matrix.

Convolutional layers also have a certain padding added to the edges of the image passed through, this is because when the kernel is centred on a corner pixel 1 side or more of the kernel is going to be out of the image, this can cause errors.

First – CNN

Second – Padding





Max-Pooling:

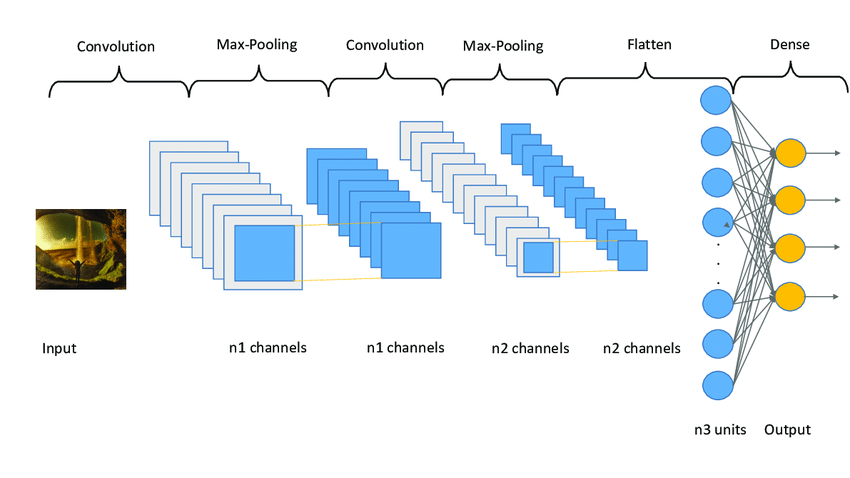
After a convolution has been applied to the data and a feature map has been created the feature map is passed through a Max-Pooling layer.

The Max-Pooling layer takes the largest value in each patch of the feature map; this is to reduce the total number of samples but have the most comprehensive overview of all the important features in the image.



Dense layers:

After the convolutions have been applied the final feature map after Max-Pooling is sent through the Dense, fully connected, layers so that the predictions can be made. Here I used the same framework as before just changing the number of resulting predictions from 10 to 36(0-9, A-Z).



Text

Description automatically generated

Text

Description automatically generated

<https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b>

### Training Process:

**Previous method:**

**Text

Description automatically generated**

**Text

Description automatically generated**

Nr of lines: 108

**Parser:**

I decided to use a CLI style input method for training my model. I did this as hard coding any small changes like number of epochs, if new weights or previously existing weights should be used was very tedious, it allowed for a better experience testing different models.

The parser takes in 4 possible arguments, training the model, loading previous weights, evaluating the model, and specifying the number of epochs.

Training – whether the model should be trained

Load weights – whether the pre-existing values should be loaded from the checkpoint directory

Evaluate – whether the model should be tested against the test data, previously split from the dataset.

Epochs – how many re-runs of the training data should be made during the training process.

A screenshot of a computer

Description automatically generated with medium confidence

**Storing weights and biases:**

Text

Description automatically generated

**Instantiating model, dataset, and class names:**

Text

Description automatically generated

**Training loop:**

Text

Description automatically generated

Nr of lines: 70

When run:





## Character detection:

The program needs to be able to find the characters it will convert from text format into string data. For this I need to create or find a sub-program that will efficiently detect all the handwritten characters in an image.

### Pre-Processing:

Before the characters can be extracted from the image, I first need to apply filters to make the contours and shapes easier to detect. Specifically, I need to apply filters on the image so that it can match up to the images used for training the neural network.



First, I Gray scale the image this is to make it so that the pixel values are in a range between 1 and 0, meaning they can be float values, this will also help with thresholding the image. After I threshold the image, this means that the pixel values above a threshold will be set to 255, black, and if they are below, set to 0 ,white. Next, I apply a gaussian blur this will help the program detect the edges of the characters in the image. Lastly, I apply a Canny filter to clearly outline any edges clearly.

Original: Canny:

A picture containing text, building, window

Description automatically generatedGraphical user interface

Description automatically generated with low confidence

Feature extraction:

Now that the image has been prepared the I use the findContours function built into the OpenCV library, this will find all the connected contours, the white lines, in the canny image. It will find them and store them in an array.

Then these contours will be filtered based on contour width and height as the letters will be relatively smaller contour size, this is done by creating “Bounding boxes” around the contours.

A for loop iterates through the elements in the array called ‘cnts’. Each element in the array is a contour, and then the contour has a bounding box created for it, this will specify the width, height, x, and y coordinates. After the bounding boxes are separated based on their width and height. If they are within a certain threshold they will deemed as a ‘character’. Then the character will be extracted from the image

Text

Description automatically generated

Text

Description automatically generated

## User Interface:

Graphical user interface, text

Description automatically generated

The user interface will allow the user to interact with the application. The main functionality of the GUI is the selection and processing of the users image of choice. The ‘Find image’ button will prompt the user to select an image from their computer, this will add the file under the ‘files to OCR’. After the user can click the filename in the ‘files to OCR’, this will display the image under the image label. The user can then click the ‘OCR’ button to apply the machine learning algorithm, this will cause a text file to be generated which will be shown under the ‘OCR text file’ label. The user can then select the file and it will be displayed in the main window

# Evaluation:

The requirements of the success criteria have been partially met. The machine learning component of the program has been fully developed and should be fully functional, it allows for the program to take arguments from the command line and then execute the functions properly. It loads the models previous weights, and it can also override the previous training weights if the user wishes (by default it loads previous weights).

Character detection is different, the character detection was the most difficult. Finding and extraction characters in the image requires that the program be able to distinguish between what a character is and what is not a character, this is the problem, my program is able to find all the edges and extract all the edges however it is very difficult for it to then lift only the character out of the image. This issue is far less prominent in font-based text, where the program is easily able to extract letters that are written in a digital font, however when this font character is passed though the neural network it causes major errors, the output is often that all characters are F or Y.

## Success Criteria that have been met:

Model

* Dataset
* Model config
* Training
* Storing model weights
* Re-trainable

Character extraction

* Filters
  + Gray scale
  + Binarize
* Find contours
* Extract required contours
  + Contours can be extracted just not the right ones

GUI

* Files can be selected for OCR
* Files are displayed
* OCR is applied to file
  + Files are parsed to the character extraction file
  + Parse extracted contours into ML model
    - The images can be parsed to the model
  + Store predictions in a new text file
* Display text file
* Exit

## Fulfilled Success criteria:

Machine learning:

The Machine learning model had met all the criteria, the code runs well both with and without command line Interface. All the intended features are functional, having different model types, training epochs can be customized directly from the CLI, model can be trained on previous weights or new ones, the model can easily be evaluated with or without training the model previously. The data for the model is cleaned up and handled efficiently and is done in a separate file. The unit tests designed for the program has no issues.

Character extraction:

The main program has no issues, the filters for the images are applied correctly, they are creating the right contours as show below. The program can find contours and shapes from the image using the findContours method and extract them using the bounding box functions and creating separate images to feed into the ML model.

*Right: Canny image, Left: Testing image of the bounding boxes*

A picture containing text, blackboard

Description automatically generatedText

Description automatically generated

Graphics User Interface:

The GUI is functional, it has buttons and working functions, the image being OCRd is displayed, the image file can be switched once loaded, the exit button is functional, the image is parsed through to the character extraction program.

## Partially fulfilled success criteria:

The combination of all the elements of the program is an issue, the modules work separately but the piping of data from one component to another needs work to be usable. The main issue with the compatibility between modules is the data transformation required to fit a contour into the specified array size given by the machine learning model.

After fitting the image into the required size, the chain needs to be optimized, this is because the predictions generated by the program will be inaccurate or not reach the desired level of accuracy.

The GUI has a very basic user interface, al the key operations can be used, however the number of features, user settings, and layout are not up to expectation. The UI is very barebones with the layout, all the buttons are crammed together and provide little help for the user to interact with the application. The features I wish to have implemented are, allowing the user to highlight areas to extract text from, allow the user to open multiple image files, preview the text file in the application and allow the user to switch detection models. The user should be able to change core settings of the program, like being able to run the underlying model with a GPU.

## Un-fulfilled success criteria:

The character extraction has issues in finding the correct contours to extract. The program will regularly box incorrect regions and multiple letters together. There are a few solutions to this problem. Use a standardized data input format, letters are in separate boxes and data is clearly outlined. The other solution is to apply machine learning to train the program to be able to identify the desired characters more effectively. Each solution has its own problems, standardizing data formats will decrease the applicability and usage of the program and training the program to be able to pick characters out more effectively will require a new training loop and new methods as such as un-supervised training increasing the scope of the project massively.

## Future Development:

There are many areas of the project that can be improved and adjusted to improve the overall user experience. Increasing accuracy, adding an editing feature for images, allowing for users to switch prediction model depending on the language, applying unsupervised machine learning to the character detection component where the program will learn through trial and error what characters generally look like, applying natural language processing where the program can guess what the text is based on a dictionary (feature can be turned on or off).

Accuracy:

Increasing accuracy is a very general and complicated problem, this has many fixes.

Having a different machine learning model, like the RESNET50 or VGG-16, will ensure that the underlying model is primed for success. It would also reduce the amount of time spent developing and researching the machine learning algorithms.

Adding the unsupervised ML algorithm would increase the efficiency of finding the correct contours thereby ensuring a higher number of useful predictions for these characters. This feature was not implemented as the development time, research required and increase in complexity was too much, otherwise this would be very good edition to the application.

Having more training and test data would help in optimizing the model.

Different Language Support:

Adding support for other languages is the next step, this will allow the applicability of the application to increase. It can be done by either allowing other people to select their own model and weights, or the developer can slowly add different model weights and models.

Dictionary and NLP:

Adding an NLP system that will auto complete the words seen and output them as the result will be able to give more legible results.

Final Testing:

Model:





Character Recognition:

Text

Description automatically generated

UI:

Graphical user interface, table

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

Qr code

Description automatically generated