Tabulator Technical Manual

CA400

Authors:
Andrew Cullen - 17378471
Eoin McKeever - 17436202

Supervisor:
Alan Smeaton

29th March 2021

Abstract:

Guitar tabs are a key part in a guitar player's learning experience.They are also a useful tool in transcribing one's own play for future reference. Digital versions of guitar tabs are now more popular than hand sketched as they can be viewed on a phone, laptop or printed out to be shared with ease.

The project is to create a web app that will take a video of guitar play as input and create guitar tabs from the audio and visuals of said video. The user will then be able to download a version of the evaluated tabs to their local machine. The plan is to use machine learning to implement this project. Three different neural networks are used a CNN based network to identify the region of the image that is the guitar's fretboard. Openposes caffe hand detection model to identify the location of the player's hand and doremir to recognise the notes being played. With this information we will be able to create guitar tabs with exact fingering.

0. Table of contents

1. Introduction

1.1. Overview

1.2. Glossary

1.3 Starting design vs Finished design

2. Research

3. System Architecture

3.1 System architecture diagram

4. High Level Design

4.1 Data Flow Diagrams

4.2 User flow diagram

4.3 Class Diagrams

5. Implementation

5.1 Audio Component

5.2 Yolov4 Neural Network

5.3 Contour Detection

5.4 HandPose

5.5 Choosing Hand Position

5.6 Writing Tabs

5.7 Deployment

6. Problems and Resolutions

6.1 Length Calculation

6.2 Interacting With Black Box Elements

6.3 Hand Detection Inaccuracies

6.4 Removing Hand Detection Outliers

6.5 Horizontal Noise

6.6 Integration

7. Results

8. Future Work

9. Installation Guide

10. Appendices

1.1 Overview

Tabulator is a web application that allows guitar students and teachers alike to create digital copies of guitar tabs whenever they wish. These guitar tabs will be displayed to the user and be available to download for the user to keep. Allowing them to learn or remember the song that was played. The system will solve the problems of creating tabs from the users own guitar play, it allows the user to easily reproduce the music they played earlier so it can be reproduced easily or for creating easy to learn tabs from videos of guitar covers and tutorials available on the internet. It will also be a useful tool for guitar teachers (both online and personal) as one can simply record themselves playing instead of manually writing up a tab. The rest of this document illustrates the development process of the project as a whole. It explains the design and implementation of the various parts of the system along with explanations of the thought process used when implementing these parts.

```
e|-------------|-------------|-------------|-2--------|----------|----------
B|-------------|-3-----0-----|-------------|-------3--|-3--------|-------8--
G|-------------|-------------|-------------|----------|----------|----7-----
D|-------4-----|-------------|-------0-----|----------|----------|-9--------
A|-2-----------|-------------|-5-----------|----------|----7-----|----------
E|-------------|-------------|-------------|----------|----------|----------
```

Here is an example of tab notation. This kind of music notation portrays the notes played on each string in order. Each horizontal line represents a string (from high to low E strings). The number on each string is the fret position at which the left hand must press down on the string. Each vertical line splits the tabs into bars which gives an indication of the timing of when the notes are played. The closer together each number is in each bar the quicker one note is played after the other.

1.2 Glossary

- **Tabs -** A form of guitar music notation similar to sheet music that provides additional information of where each note is played on the guitar's fretboard.
- **Fret -** A rung of metal behind which the finger is placed on the guitar's neck which causes the desired note to be played.
- **Neck/Fretboard -** The part of a guitar with which the player's left hand is used to manipulate the strings of the instrument to create different notes.
- **Cnn -** Convolutional Neural Network, a class of deep neural networks, most commonly applied to analyzing visual imagery.
- **API -** a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
- **Morphological operations** - Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image e.g Erode,Dilate methods
- **MP4** - MP4s are one of the more common video file formats used for downloading and streaming videos from the internet.
- **Neural Network** - a computer system modelled on the human brain and nervous system.

- **YOLO -** is a convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.
- **RCNN** - Region Based Convolutional Neural Networks (R-CNN) are a family of machine learning models for computer vision and specifically object detection. In short a convolutional neural network which outputs a region i.e bounding box
- **Contour/Edge detection -** Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges.
- **Frames -** In video and animation, frames are individual pictures in a sequence of images.
- **Bounding box -** bounding boxes are the 4 (x, y) -coordinates of the object in the image.
- **ROI -** Region of interest
- **preprocessing -** subject (data) to preliminary processing.
- **Hough lines Transform -** The Hough Line Transform is a transform used to detect straight lines.
- **Binary image -** A binary image is one that consists of pixels that can have one of exactly two colors, usually black and white.
- **Noise -** Image noise is random variation of brightness or color information in images
- **Mask R-CNN -** Mask RCNN is an instance segmentation model that can identify pixel by pixel location of any object.
- **Aspect ratio -** The aspect ratio of a geometric shape is the ratio of its sizes in different dimensions. For example, the aspect ratio of a rectangle is the ratio of its longer side to its shorter side
- **Probability map -** A surface that gives the probability that the variable of interest is above (or below) some threshold value that the user specifies.
- **Machine learning -** Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so.
- **Agglomerative Hierarchical Clustering -** used to group objects in clusters based on their similarity.
- **Anaconda -** Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

1.3 Starting design vs Finished design

The project that we developed contains the core functionality that was specified in the functional spec. Users can upload a mp4 video file of guitar play to the web app for it to be broken down into digital guitars tabs that can be downloaded to the users local machine. Done through an audio and visual component. The main core functionality was implemented in full and a few added features were implemented. The systems functionality is shown below with added features shown.

Below is a list of system functions compared from start design to finished.

	Degree of compliance
Video Upload	Done
Video Conversion	Done
Tab Viewing	Done
Tab Download	Done
Video Viewing	Added
Tab Editing	Added

2. Research

The initial stage of this project consisted of an indepth research period where possible solutions were investigated. The bulk of the research was in the area of image processing and neural network implementation since no previous work had been done on these areas prior.

The research started with breaking down the detection problem into two. Detection of the hand and detection of the guitar neck. When investigating solutions to hand detection a research papers called Hand Keypoint Detection in Single Images using Multiview Bootstrapping written by Tomas Simon Hanbyul Joo Iain Matthews Yaser Sheikh in Carnegie Mellon University was read, it supplied the required understanding to help implement a version suitable for use in the project, using one of OpenPoses pretrained models.

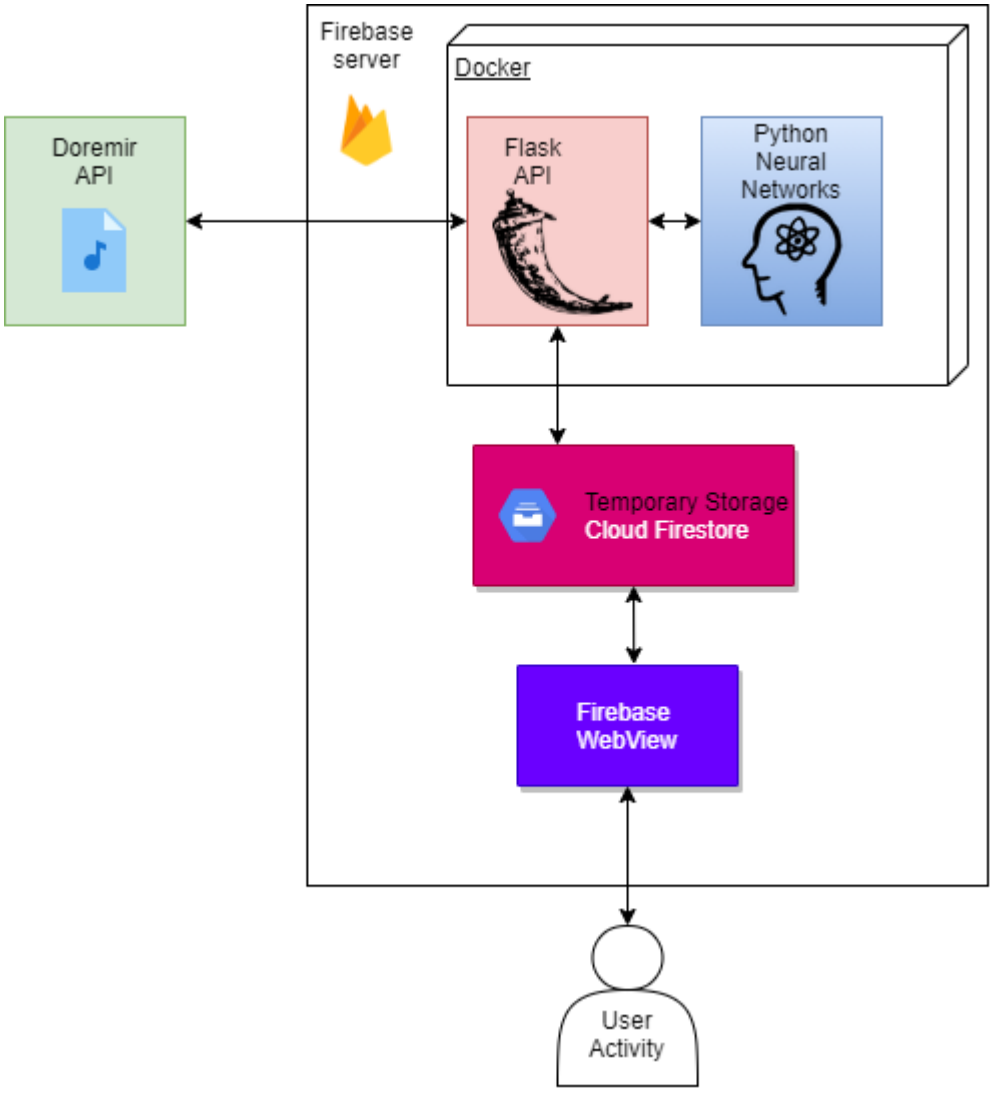
Next the research turned towards the guitar neck object detection problem. Locating and processing a suitable dataset to train the model on was the first order of business, unfortunately suitable images were few and far between. Images were pulled from google when they could be found, guitar video tutorials were also used to pull some images and the remainder of the dataset was filled with images taken using our own guitars. These images contained 5 variants of guitars in total. The dataset in total consisted of 745 clear images of guitars with necks annotated. This dataset was shared to Figshare and can be publicly accessed. The yolov4 object detection framework was used to train the neural network, the network locates regions of interest and classifies objects in one step

rather than two, vastly increasing its speed. Yolov4 was used specifically and the reason for this was it was computationally lighter then a lot of the other options.

Fortunately this year, in the second half of the semester a module involving image processing was sat, it sparked the idea of extracting horizontal and vertical lines respectively from the images to further narrow down the location of the neck of the guitar. OpenCV's open source python library was used to carry out this task. Further discussion of the implementation of the described research will be explained later in this document.

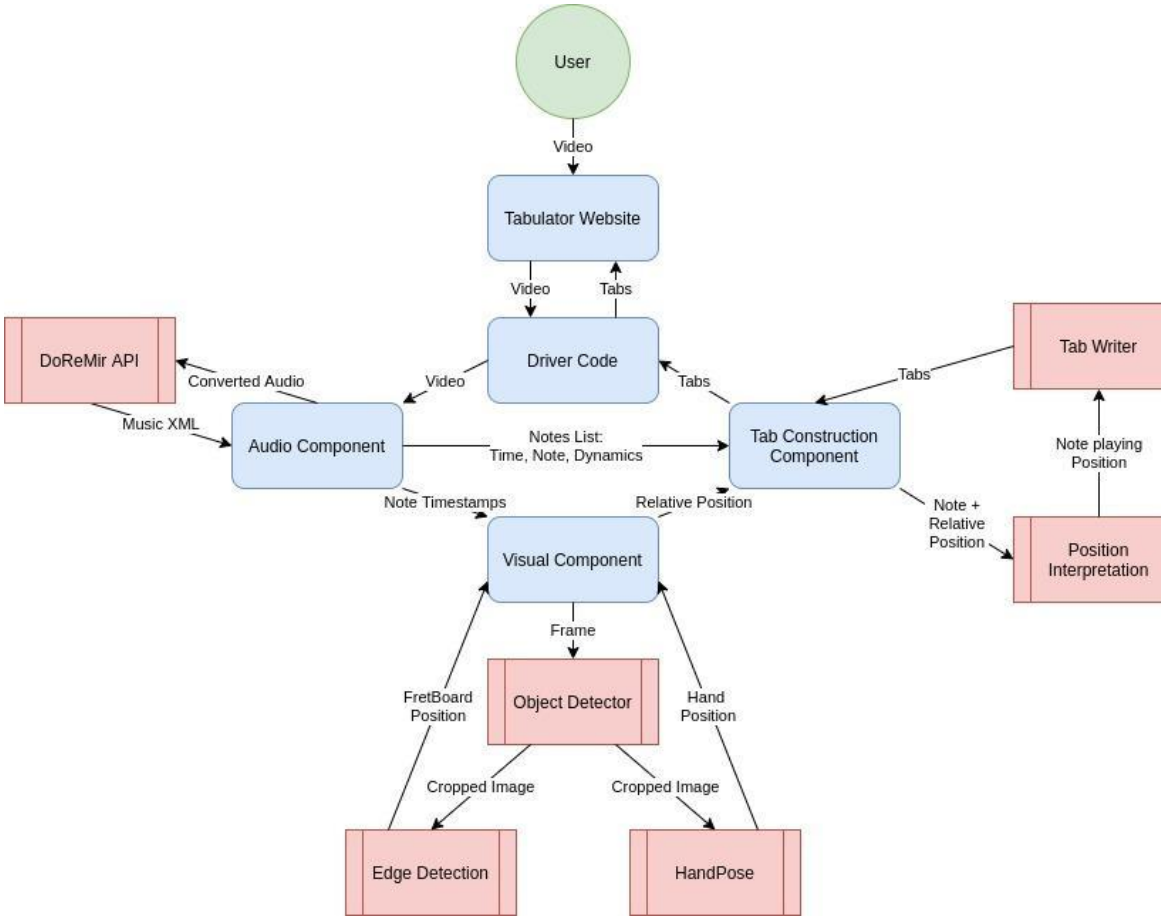
3. System Architecture

3.1 System architecture diagram

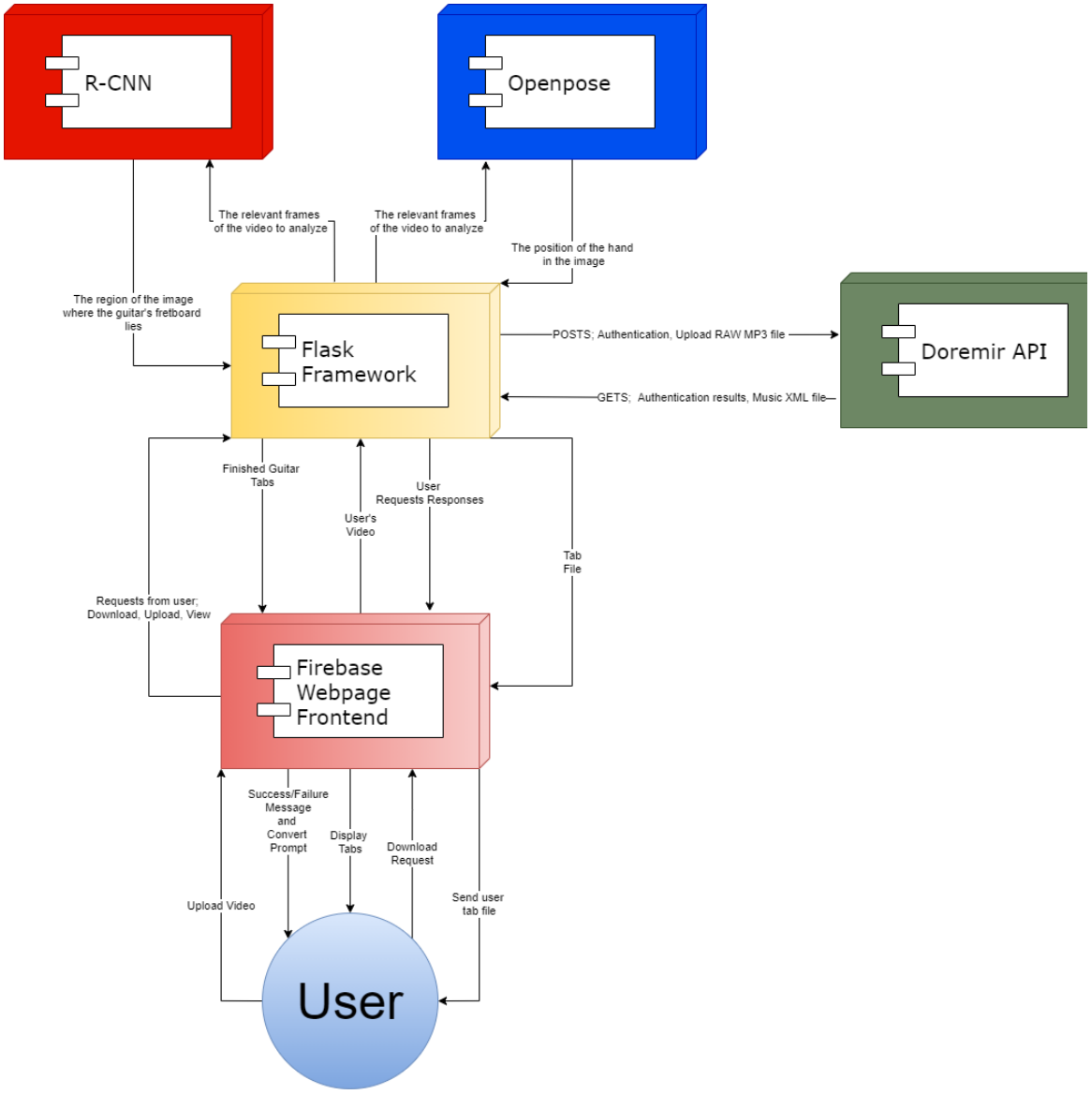


4. High-Level Design

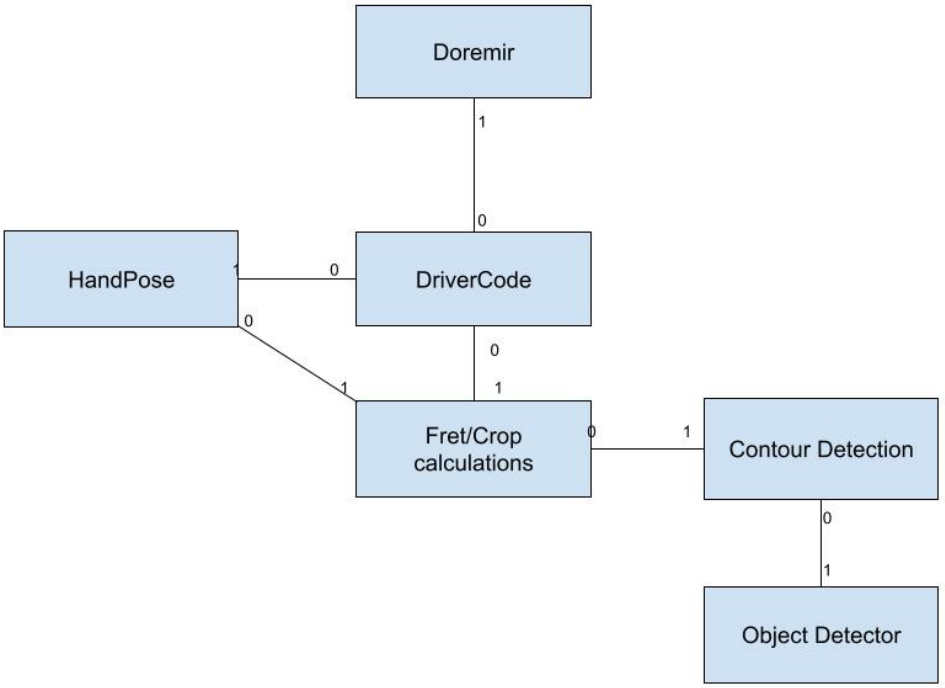
4.1 Data Flow Diagrams



4.2 High-Level Context Diagram



4.3 Class Diagrams



5. Implementation

5.1 Audio Component

The audio component takes the given mp4 file and converts it to mp3 for analysis. The DoReMir API then is called and returns a music.xml file. A list of dictionaries each a note is then created. These dictionaries are made, accounting for some errors the API makes in processing the audio (Ghost notes and tied notes are removed). These dictionaries contain timing information for each note as well as rests, which are interpreted producing timestamps for when each note is played for the visual component.

5.2 Yolov4 Neural Network

As was mentioned earlier the yolo framework was used to create the object detector that was tasked with locating the region of the image where the guitar neck is located. The OpenCV python library was used to deploy the model. The network was trained on grayscale images of dimension 608x608 and a batch size of 64 was used with subdivisions of 16. The network's first layer accepts a grayscale image in the form of a blob of dimensions size 608x608.

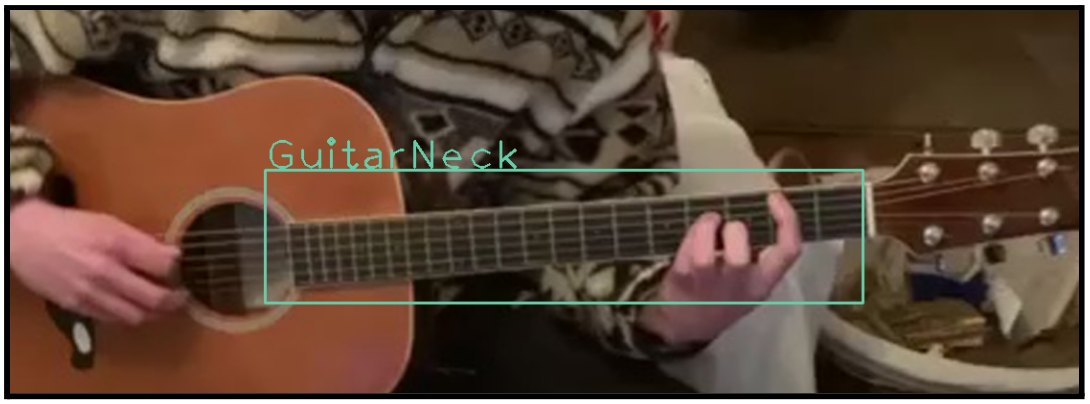
```
blob = cv2.dnn.blobFromImage(img, 0.00392, (608, 608), (0, 0, 0), True, crop=False)
net.setInput(blob)
```

Yolo is a convolutional neural network that divides an image into subcomponents, and conducts convolutions on each of those subcomponents before pooling back to create a prediction. The network consists of three detection layers. Each detection layer is fed by a convolutional layer with filter size 18 and a linear activation function. The number of object classes and which anchors are used is also specified here. Models were trained using transfer learning. Transfer learning's main advantage is improved results when using a smaller dataset, based on the dataset in question it was the logical choice.

```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 6,7,8
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=1
```

The object detector runs over the set of frames pulled from the audio component. Each frame is analysed and a bounding box is formed over the region where the object is most likely located. Here is a visual representation of the output.



In the code all bounding box coordinates are appended to a list called outputBox. The process of extracting the pixel coordinates can be seen below. Each frame is analysed and a bounding box is specified and appended to a list by the detector. This occurs only when a certain degree of confidence is met.

```
        for out in outs:

            for detection in out:

                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.5:
                    # Object detected
                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)

                    # Rectangle coordinates
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)


                    boxes.append([ x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

        if len(boxes) != 0:
            boxesPrior = boxes
            indexesPrior = indexes
            for i in range(len(boxes)):
                if i in indexes:
                    x, y, w, h = boxes[i]
                    q = [(x, y),(x, y + h), (x + w, y), (x + w, y + h)]

                    outputBox.append(q)
```

5.3 Contour Detection

Once the object detector has located and recorded the ROI's associated with the guitar neck of the frame. We used these ROI coordinates to narrow down the search area in question. OpenCV's python library is used again here to extract the horizontal and vertical lines located in each ROI. These horizontal and vertical lines are extracted using various morphological operations.The four points read in from each pointList are used to crop the image.

```
    pointList,images,errorString = NeckLengthCal()
    if len(errorString) > 0:
        return neckLeftX,neckRightX, pointList, errorString
    else:
        for i in range(len(images)):
            img = cv.imread(images[i])

            bottomLeft,topLeft,bottomRight,topRight = pointList[i]
```

Once an image is read and cropped appropriately it is passed to be preprocessed. The preprocessing essentially converts the image to a binary image by first blurring the image using a GaussianBlur this causes the background to blur reducing noise and it also sharpens any edges located in the foreground of the image It was done twice as under testing it seemed to improve the results. After this the image is converted to a binary image which highlights all edges of the image.

```
def preprocessing(img):
    img = cv.GaussianBlur(img,(9,9), 1)
    img = cv.GaussianBlur(img,(9,9), 1)
    img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    img = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, \
                               cv.THRESH_BINARY, 15, -2)
    return img
```

From this binary image the horizontal and vertical edge lines are extracted using the OpenCV methods getStructuringElement paired with erodes and dilates. Below shows the method of extracting horizontal lines. The horizontal line we are attempting to identify is the neck of the guitar.

```
def horizontalStructure(horizontal,thresh):
    cols = horizontal.shape[1]
    horizontal_size = cols // thresh
    horizontalStructure = cv.getStructuringElement(cv.MORPH_RECT, (horizontal_size, 1))
    horizontal = cv.erode(horizontal, horizontalStructure)
    horizontal = cv.erode(horizontal, horizontalStructure)
    horizontal = cv.dilate(horizontal, horizontalStructure)
    return horizontal
```

Above we start the thresh variable value low and slower increment until certain characteristics are met. These characteristics are defined using a method called the Hough line Transform. The thresh value is increased until a horizontal line exists that is of adequate length. We chose a scale factor of the horizontal pixel length of the image. We set the max gap between pixel to 50 to allow the horizontal line to bridge small gaps in places where noise may become an issue.An example of this is the tops of the guitar players finger may cut the top neck line in two, keeping the maxLineGap and minLineLength high remedies this issue.

```
def HorizontalHough(HLines):
    blank = np.zeros(HLines.shape[:2], dtype='uint8')
    lines = cv.HoughLinesP(HLines, 1, np.pi/180,100,minLineLength=int(HLines.shape[1]/1.25),maxLineGap=50)
    x = type(lines)
    if x == type(None):
        #can't draw horizontal hough lines, threshhold increased
        return False , blank

    else:
        for line in lines:
            x1,y1,x2,y2 = line[0]
            cv.line(blank,(x1,y1), (x2,y2), (255,255,255),5)
    return True, blank
```

The line we attempt to identify is the top line of the neck, It was found to be more difficult to use the bottom line of the neck because of the angle the guitar player's wrist mounts to the neck. This logic limits the system from dealing with video frames from an angle where the recorder is looking up at the guitar. But in the vast majority of cases the top line of the neck is visible. Using this line the image is cropped further. The furthest x value, closest to the headstock is used to crop the image. This line alone is not enough to classify the length of the neck because of the design of certain brands of guitars. The line is extended to the border of the image and another identical line is drawn below and parallel to the original line. These lines are used to narrow the search area further.

```
def FinalCals(finalContours,img):
    blank = np.zeros(img.shape[:2], dtype='uint8')
    finalContours = sorted(finalContours,key = lambda x:x[1], reverse = True)
    for con in finalContours:
        cv.drawContours(blank,con[4],-1,(255,255,255),1)

    if len(finalContours) == 0:
        errorString +=("No neck found in this video, please provide a clearer video!")
    else:
        line = finalContours[0][2]
        line1 = line
        cv.line(blank,(line[0][0][0],line[0][0][1]),(line[1][0][0],line[1][0][1]), (255,255,255),1)
        cv.line(blank,(line[0][0][0],line[0][0][1] + 70),(line[1][0][0],line[1][0][1] + 70), (255,255,255),1)
        line1[1][0][0] = line1[1][0][0] + 70
        line1[1][0][1] = line1[1][0][1] + 70
        square = np.concatenate((line, line1))

    return square
```

Pixel information regarding the two lines is sent to a function called CropCals which outputs pixel coordinates which are used below to crop the image further.

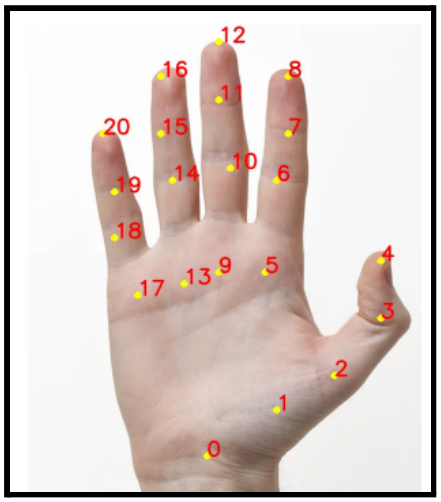
```
finalCal = FinalCals(longestContours,binaryHorizontalHoughLines)
width,top,bottom = CropCals(finalCal)
blank = binaryVerticalHoughLines[top:bottom,0:width]
y = np.where((blank == 255))
if len(y[1]) != 0:
    minX = min(y[1])

blank = binaryVerticalHoughLines[top:bottom,minX:width]
length = blank.shape[1]
```

Now with the search area narrowed further the vertical lines are investigated. Of these parallel vertical lines the one furthest left along the x axis is taken for the next crop. Vertical lines like the horizontal lines are chosen only when they satisfy certain characteristics, since the only vertical lines we're really interested in are located away from the guitar players hand there should be no reason for there to be gaps so a low value of 5 is set for MaxLineGap and a minLineLength of 20 to filter out any very small lines that result from noise in the image. The pixel length of this cropped image now represents the guitar neck

5.4 HandPose

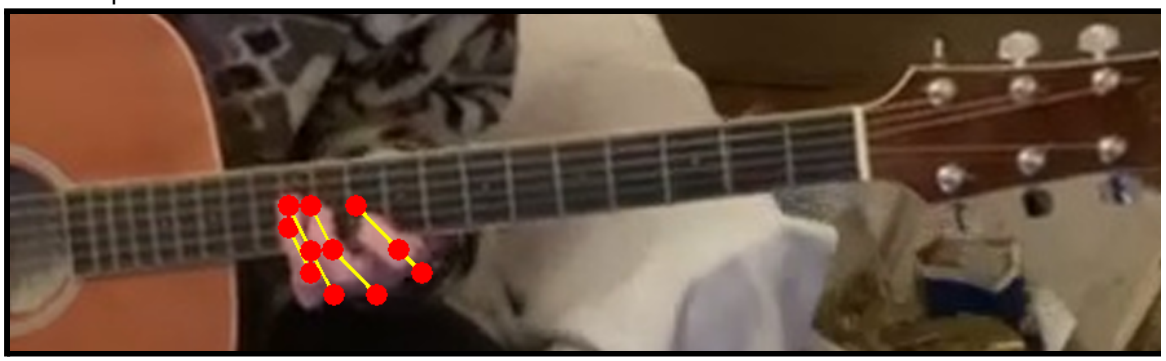
Now the system knows the notes and the location of the neck in pixel coordinates. Now all that is needed to evaluate the final result is the location of the hand in pixel coordinates. The HandPose model attempts to detect 21 detects key points on the hand. They are shown below. Key Points 6,7,8,10,11,12,14,15,16,18,19 and 20 are used when applicable and the rest are filtered out.



The images operated on are pulled from the audio components output frames. But before they are fed to the model they are cropped so only the side containing the neck of the guitar is included this is done to eliminate the inclusion of the player's strumming hand. Allowing the model to focus on only one hand. The model takes as input a RGB image of height 512 and a width based on this height of 512 multiplied by an aspect ratio of the original image so as to keep the image and the hand in question from becoming disproportionate. The threshold is kept low. The reason why is explained later in this document.

```
frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
aspect_ratio = frameWidth/frameHeight
threshold = 0.2
# input image dimensions for the network
inHeight = 368
inWidth = int(((aspect_ratio*inHeight)*8)//8)
```

Once the model is done analysing the image a output is returned, here is a visual representation of the output.



In the python code points are pulled from their maximum value in the probability map and these probabilities compared to the threshold value. If greater than the threshold value the point is added to the point list if not a None value is appended.

```
    for i in range(nPoints):
        # confidence map of corresponding body's part.
        probMap = output[0, i, :, :]
        probMap = cv2.resize(probMap, (frameWidth, frameHeight))
        # Find global maxima of the probMap.
        minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)
        if prob > threshold :
            # Add the point to the list if the probability is greater than the threshold
            cropPoints.append((int(point[0]) + int(cropX[j]), int(point[1])))
        else :
            cropPoints.append(None)
    allImagesPointList.append(cropPoints)
```

5.5 Choosing Hand Position

Choosing the hand position of each note was achieved by comparing the hand and fretboard positions together to get a relative position value giving the percentage value of the hand's position on the fretboard of the guitar while taking into account any cropping done to the image.

```
relativepos = 1 - (HandCoordinates[i][0] - LeftX[i] + poslist[i] * .39) / (poslist[i]*1.39)
```

This value was compared to each of the positions of each playable position for the note given from audio analysis. This was done through a lookup table for each note containing every string and fret the note can be played on. A formula is applied to each note to get its relative position.

```
#### Formula for fret postion percentag
def fretPos(n):
    return (1-(1/(1.059463**n)))
```

This formula is derived from guitar manufacturer's formulas for fret placement when they construct guitars. The closest possible playable position to the hand is used for our tabs with some bias towards open strings i.e. if the note can be played without putting the left hand down onto a fret it is accounted for. This is done by slightly biasing open strings and choosing the open string option when the hand is far away from any playable position.

5.6 Writing Tabs

The tab writing is done through a script which takes the list of positions the notes are interpreted to have been played on and iterating through it for each string while applying time seen as "-" characters for any rests as well as applying appropriate time with the same character for how long the note is played (¼ notes ⅛ notes etc.). Any appropriate warning messages appropriate to the user is also displayed if it is seen that the video may have been taken in an ill advised manner (The guitar and left hand must be centre of frame and entirely visible in each image). This gives an explanation as to why there may be inaccuracies for each individual note in the song.

5.7 Deployment

The web app is made using flask, dockerized using docker and deployed on firebase hosting using 'google cloud services'/'cloud run' to run the python scripts the app requires. The dockerfile and requirements.txt file contain all of the relevant packages required to run the flask app. The app can also be deployed locally in an environment defined by the environment.yml file. It can also be dockerized locally using docker and the container produced can be hosted locally for testing purposes.

6. Problems and Resolution

6.1. Length Calculation

In the initial stages of this project it was the plan to design a machine learning tool that could identify the guitar neck and output an accurate length and relevant pixel coordinates and to use that for the tab evaluation. Investigation was done into implementation of a mask R-CNN and other image segmentation models because it quickly became apparent that the object detectors bounding box was not enough to accurately predict the length. The object detector was trained 735 images in total all found and annotated manually but in relation to typical machine learning dataset it is not extremely large. Training using this data led to an object detector that produced decent results, enough to predict the general area of the guitar neck but not enough to take it's bounding box as the length. Another layer of accuracy was needed. If projects were held on campus this year the ideal solution to this problem would be to implement a model that makes use of image segmentation. But it was not an option at home due to the computational expenses they require. A basic version of mask R-CNN was implemented in the process of this project but unfortunately it was not used as it slowed the speed of the application down far too much to manage at home. A solution was needed to refine the bounding box further. The solution was implemented through a form of contour detection. The horizontal and vertical lines that occur on every guitar neck were analysed to further refine the boundingbox.

6.2 Interacting With Black Box Elements

When developing Tabulator, we had to use the DoReMir API whose inner working could not be known. Error handling when dealing with unexpected output therefore required much testing. This lack box nature of the API also caused some issues when integrating as different environments caused unexpected output when performing API calls. This was solved by configuring the environments of both partners to have the same requests, base64 and json module versions.

When dockerizing and uploading the project to google cloud run the app behaved differently than when ran locally..The gunicorn and gevent modules required patching of the requests module which was a difficult fix as no helpful error messages could be seen during the running of the python scripts when they were deployed on google cloud hosting. Only a "503 service unavailable" message was seen as the workers were in a hung state due to an incompatibility.

6.3 Hand Detection Inaccuracies

An issue that arose during the project was the hand detection model being unable to identify key points on the guitar player's hand. At first this issue was baffling as the images to our eyes seem to have relatively clear images of the hand. But what was quickly realised was due to the frames being pulled from videos taking on a phone camera. The frames were of quite low resolution. The combination of the slightly blurry environment and certain situations where the guitar players fingers are closed tightly together and positioned in unique angles that are needed to play the various guitar chords. It gave rise to situations where the hand detection model could not identify any points on the hand. This was a critical issue for our project. As has been said prior The input frames received are by nature lower resolution because they are pulled from a video in most situations a mobile phone camera. Due to this it was important to attempt to increase the resolution. A method called Super Resolution was implemented though the OpenCV extension library opencv-contrib-python. A model called FSRCNN_x4 was implemented. It was chosen because it gave a good trade off between performance and efficiency. The increased resolution improved the capabilities of the hand detection model. It was implemented through the python model OpenCV_contrib_python. The next page shows the implementation.

```
    sr = cv2.dnn_superres.DnnSuperResImpl_create()
    sr.readModel(model_path)
    sr.setModel(modelName, modelScale)

    for i in images_path:
        image = cv2.imread(i)
        upscaled = sr.upsample(image)
        upscaledImages.append(upscaled)
    return upscaled
```

6.4 Removing Hand Detection Outliers

A problem that arose from the hand detection section of the code was outliers were found when the threshold value was set to low meaning points which were not associated with the hand were identified as part of the hand.The model also seemed to struggle when the threshold value was set to high and would occasionally struggle to recognise any points at all. From various tests of the model it became noticeable that while decreasing the threshold gave rise to the occasional outlier, it also gave rise to more points being identified accurately in the correct hand locations. So when the threshold was low it gave more points to work with and made it far easier to remove any outliers as it was possible to use the mean of the cluster of points around the hand to identify any points that were plotted a certain percentage over the average distance from the mean, far enough away from the cluster to be considered outliers.

6.5 Horizontal Noise

In the contour detection model managing noise was a critical issue for the success of the detection. The solution to managing noise came down to implementing the correct blurring types, applying suitable amounts of dilation and eroding. This came mostly from manual tweaking to see what worked bested. But what was the main fix for the issue was managing the threshold value that controlled the edges that were let through. If this value was set to high, things like the strings of the guitar would start to come through, which would overshadow the neck of the guitar for horizontal length. But when the threshold was started low and slowly incremented up it firstly reduced any surrounding noise and secondly ensured the strings did not show up as horizontal lines before the neck was extracted.

6.6 Integration

In integrating the work of both partners in the project there was a degree of difficulty due to working from home off of separate environments due to covid 19 and not having the finances to afford two computers that could run ubuntu specific python version 3.6.9 which the system was built upon in the initial phases of development (one partner only had a laptop with windows 7 installed with little hard drive space for ubuntu to be installed). There were issues with our different versions of python on Windows and Linux systems behaving differently during integration. This made gitlab integration impractical for the majority of the project as our systems could only be integrated correctly on one of our systems until the solution was arrived at. During this time the image analysis side of the project unfortunately had to be developed and tested in a different manner to the system as a whole due to unsymmetrical bugs and behaviour on the two environments. This caused some integration problems before the solution was arrived at. This was overcome by using Anaconda with environments set up from the same environment.yml file.

7. Results

As a whole the project can be considered a success. All the fundamental elements of the system were implemented and integrated successfully. Although it is without question that further improvements can be made to the systems some of which are discussed later. With regards to the accuracy of the system the visual component of the system was tested on 100 frames of guitar play the results are as follows: The Object Detector once padding was applied produced 92 acceptable results, the hand detection model post outlier removal produced 97 acceptable results, edge detection produced 87 acceptable results. As has been mentioned prior a large portion of the dataset used to train the object detector consisted of images taken by the developers manually. It is to be expected that the object detector has a higher detect likelihood when shown these specific guitar types.The types consist of Stratocaster, Les Paul, Gretsch as well as cutaway and non-cutaway acoustic guitars.It should be noted that the testing data consists of these variants of guitars.

In terms of the system as a whole, the results in terms of testing were a great success. Every component of the system was integrated correctly and errors that were encountered in analysis were all handled in a successful manner. Minor errors were accounted for and adjustments were made using information from other notes or images so the output was mostly unaffected. Major errors in audio or visual were handled and reasons for them were stated to the user (Very poor image or poor audio quality).

8. Future Work

By the deadline of the project the main functional requirements were fulfilled that were specified at the beginning of the project. Although many possible improvements were still left out due to time constraints. Given more time a more sophisticated approach to cluster removal would have been implemented, perhaps a form of Agglomerative Hierarchical Clustering could have been implemented. Also If time constraints were not an issue a larger dataset would ideally have been made to further increase the accuracy of the object detector.

The contour detector runs into an error when the camera is below and angled up at the neck of the guitar if more time was available a solution would have been investigated further. Using the horizontal line information and the known position of the hand it could be possible to define a solution that identifies the horizontal line has been broken by the player's hand and bridges the gap accordingly . But unfortunately not enough time was available.

In future we would like to add our own audio analysis to the app using quantized midi files converted from the mp3 files that we currently use. The most improvement to any area of the app at the current moment is in the audio analysis as seen by our test results with nearly a third of the videos tested being rejected by DoReMir. If we were to create our own audio analysis we could incorporate polyphonic audio which would allow any type of song to be interpreted.

9. Installation Guide

The system can be used on any device with access to the web at the following url:

Welcome to Tabulator (tabulator4-2ta6xoy2zq-uc.a.run.app)

The app can also be deployed locally if downloaded from the project github (first move the node_modules and .firebase folders from 'res' to the 'tabulator' directory) and run on a python environment defined by the environment.yml file found in the tabulator/server/src path. (This methods works best on linux ubuntu systems)

Step 1: Download the project from
https://gitlab.computing.dcu.ie/mckeeve2/2021-ca400-mckeeve2-cullea37

Step 2: Move contents of the res directory to the tabulator directory.

Step 3: Create an environment as defined by the environment.yml file in the tabulator/server/src directory. Best done by creating an anaconda environment with the "conda env create -f environment.yml." command

Step 4: Use the "python app.py" command while in the "tabulator/server/src" directory to spin up the app on localhost 0.0.0.0:8080.

The app can also be dockerized locally and deployed in the same manner. Using step 1 and 2 above followed by (This method requires docker to be installed on the local machine):

Step 3a: Dockerize the app with the "sudo docker build server" command while in the tabulator directory in a linux system.

Step 4a: Run the docker container using the "sudo docker run $container name$" command. This will run the container on localhost 0.0.0.0:8080 also.

10. Appendices

1. OpenCV_contrib_python: opencv-contrib-python · PyPI
2. OpenPose Github repository: GitHub - CMU-Perceptual-Computing-Lab/openpose: OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation
3. Hand Keypoint Detection in Single Images using Multiview Bootstrapping written by Tomas Simon Hanbyul Joo Iain Matthews Yaser Sheikh in Carnegie Mellon University: arXiv:1704.07809v1
4. YOLOv4: Optimal Speed and Accuracy of Object Detection: arXiv:2004.10934v1
5. YOLO github repository - GitHub - AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)
6. Dataset - Guitars with guitar neck annotated. YOLO bounding box format. (figshare.com)
7. Fast and Accurate Image Super-Resolution with Deep Laplacian Pyramid Networks: arXiv:1710.01992v3
8. SuperResolution FSRCNN_x4 Github repository - GitHub - Saafke/FSRCNN_Tensorflow: Tensorflow implementation of 'Accelerating the Super-Resolution Convolutional Neural Network'.

9. DoReMir API: https://doremir.com/