Logic & AI

Handwritten Equations to LaTeX Conversion

Reading handwritten equations and converting them to latex

Ashraf Adel 196280
Farah Aymen 194233
Jacinta Samir 206562

Table of Contents

1. Project Summary 3

PAGE 3

PEAS 3

2. Problem description 4

3. Data Set Used 5

4. Proposed Approach 6

4.1 Preparing The dataset For Training: 9

4.2 Training The Model 10

4.3 Developing The GUI 11

“App” Class 11

“SnipWidget” Class 16

5. Experiments 19

5.1 First Attempt 19

5.2 Second Attempt 20

5.3 Third Attempt 21

5.4 Fourth Attempt 22

5.5 Fifth Attempt 23

5.6 Sixth Attempt 24

6. Results 25

7. Tools Used 28

8. How to Use 28

1. Project Summary

This project takes an image of a handwritten equation, segments it into symbols, evaluates these symbols on a neural network, then it gets its LaTeX and displays it to the user on the GUI. The user in general snips the equation he wrote and gives it to the system, then the system displays the latex for him. It first takes the image and segments it by first detecting each connected region together and crops according to it. It takes the approach from top to bottom and from right to left in detecting where the symbols at so it segments them in order. Then, using an artificial neural network, the network takes those symbols one by one and evaluates it with the trained model. Next, the model converts the detected symbol to what refers to the LaTeX of it.

PAGE

Preceptors	Actions	Goals	Environment
- Snipped image of the equation	- Predict the handwritten equation - Display the LaTeX equivalent	- Minimize the incorrect translated symbols	- The GUI the user uses - The equation image.

PEAS

Performance Measure	Environment	Actuators	Sensors
- Measure the accuracy of the trained neural network model using evaluation metrics (accuracy metric)	- Predict the handwritten equation. - Display the LaTeX equivalent.	- Minimize the incorrect translated symbols	- The GUI the user uses - The equation image.

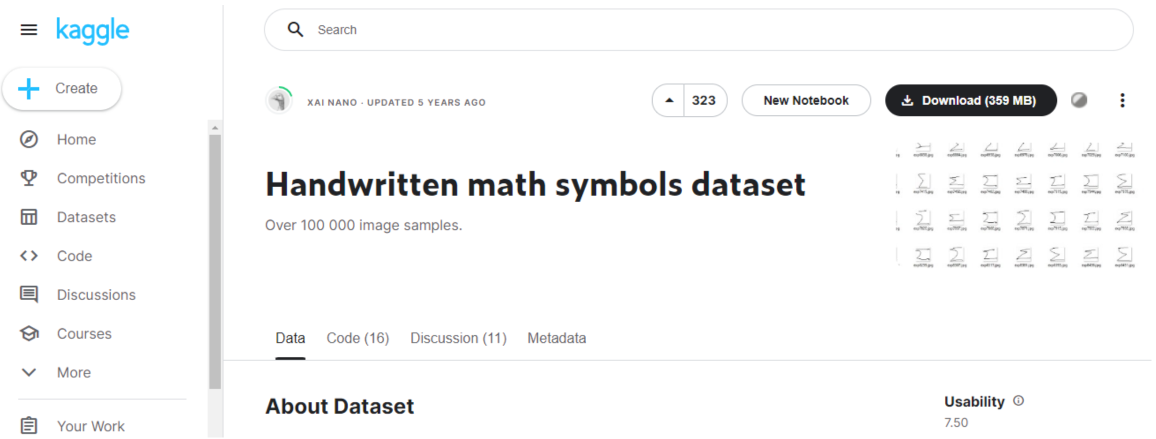
- The arguments in the compile function of the NN model is considered the evaluation function

2. Problem description

It has been a problem for many users to find a picture of an equation online but can't take the text itself or writing an equation on paper but can't just plop it as a text. Writing equations symbol by symbol can be difficult, exhausting and time consuming. This project aims to make the process easier by just inserting an image of the equation and getting its LaTeX. This is considered as a real-life problem where this project will ease the users from this burden. It will help researchers, students, etc. to easily insert whatever equation regardless on how complex it is and it will be automatically translated to them immediately.

3. Data Set Used

The dataset used in this project is found on Kaggle it is called *Handwritten Math Symbols Dataset*



It offers around 79 different classes and over 100,000 image samples. Since the dataset was huge and had a drawback that it was imbalanced, we used only part of it. We used the necessary symbols of total 32 classes. The dataset is loaded in pickle files for further use.

4. Proposed Approach

The approach of this project is to first read the image and perform some pre-processing on it so it can segment it properly.
First through the following function, it converts the image to grayscale and performs a median blur on it if “med Filter” is True. Second, it detects the edges on the filtered image with canny. Third, it dilates the image and find the contours of it. Fourth, it puts bounding boxes on the

```python
def extractSymbols(imgOrig: Union[Image.Image, np.ndarray],
                    showSteps = False,
                    returnSteps = False,
                    medFilter = False,
                    verticalSymbols = False): # The ": Union[]" syntax is t
    debugImgSteps = []
    if isinstance(imgOrig, Image.Image): # if true, then "imgOrig" is a PIL
        imgOrig = np.array(imgOrig)[:, :, ::-1] # the "-1" is to get the nd
    imgGray = cv2.cvtColor(imgOrig,cv2.COLOR_BGR2GRAY)
    if medFilter:
        imgGray = cv2.medianBlur(imgGray, 5)
        debugImgSteps.append(imgGray)

    imgCanny = cv2.Canny(imgGray, 50,180)
    debugImgSteps.append(imgCanny)

    kernel = np.ones((5,5), np.uint8)
    imgDilated = cv2.dilate(imgCanny, kernel, iterations=5)
    debugImgSteps.append(imgDilated)

    contours, _= cv2.findContours(imgDilated, cv2.RETR_EXTERNAL
                                    , cv2.CHAIN_APPROX_NONE)

    boundingBoxes = []
    for contour in contours:
        x,y,w,h = cv2.boundingRect(contour)
        boundingBoxes.append((x,y,w,h))

    global rowsG # Global because it will be used in leftRightTopBottom()
    rowsG, _, _ = imgOrig.shape
    key_leftRightTopBottom = cmp_to_key(leftRightTopBottom) # Wrapper to al

    if (verticalSymbols):
        boundingBoxes = sorted(boundingBoxes,
                                key=key_leftRightTopBottom)
    else:
        boundingBoxes = sorted(boundingBoxes, key=lambda x : x[0])

    symbols = []
    for box in boundingBoxes:
        x,y,w,h = box
        mathSymbol = imgOrig[y:y+h, x:x+w]
        mathSymbol = cv2.cvtColor(mathSymbol, cv2.COLOR_BGR2GRAY) #converti
        mathSymbol = cv2.resize(mathSymbol, (45,45),
                                interpolation=cv2.INTER_AREA) #to have the same
        debugImgSteps.append(mathSymbol)
        mathSymbolF = mathSymbol.astype('float32') #optional: tensorflows d
        symbols.append(mathSymbolF.reshape(1, 45, 45)) # reshaped to be com

    if showSteps:
        dispImages(debugImgSteps)

    if returnSteps:
        return symbols, debugImgSteps

    return symbols
```

symbols with a defined function called leftRightTopBottom that makes it scan the image from left to right and top to bottom so it can read the image sequentially.

Regarding the leftRightTopBottom() function, it gives priority to the y-coordinate of the math symbol in the image; if the symbol is at the top, then it is read first, then the symbol at the middle, and so forth. Note that this function is the key function used in sorting only if “verticalSymbols” parameter is “True” in extractSymbols()

```
def leftRightTopBottom(tup1, tup2):
    x1, y1, _, _ = tup1
    x2, y2, _, _ = tup2
    rows = rowsG
    yRegion1, yRegion2 = -1, -1

    for i in range(4):
        if y1 < rows/4 + rows*(i/4.0):
            yRegion1 = i
            break
    else:
        if yRegion1 == -1:
            yRegion1 = 4

    for i in range(4):
        if y2 < rows/4 + rows*(i/4.0):
            yRegion2 = i
            break
    else:
        if yRegion2 == -1:
            yRegion2 = 4

    if yRegion1 < yRegion2:
        return -1
    elif yRegion2 < yRegion1:
        return 1
    elif x1 <= x2:
        return -1
    else:
        return 1
```

the dispImages() is used to display the image processing steps that extractSymbols() have done to crop the symbols from the input images, and this function will only run if the parameter “showSteps” is “True”

```
def dispImages(imgs):
    for img in imgs:
        cv2.imshow('Image', img)
        cv2.waitKey(0)
    else:
        cv2.destroyAllWindows()
```

Then, we created a dictionary to hold the LaTeX equivalent for each class so after the model detects it, it matches with it.

```python
dic = {
    "-": r"-",
    "(": r"(",
    ")": r")",
    "+": r"+",
    "=": r"=",
    "0": r"0",
    "1": r"1",
    "2": r"2",
    "3": r"3",
    "4": r"4",
    "5": r"5",
    "6": r"6",
    "7": r"7",
    "8": r"8",
    "9": r"9",
    "geq": r"\geq",
    "gt": r">",
    "i": r"i",
    "in": r"\in",
    "int": r"\int",
    "j": r"j",
    "leq": r"\le",
    "lt": r"<",
    "neq": r"\neq",
    "pi": r"\Pi",
    "sum": r"\sum",
    "theta": r"\theta",
    "times": r"\times",
    "w": r"w",
    "X": r"\X",
```

First, we loaded the dataset into a pickle file so it would be easiest to dump and load from through a function:

```python
def loadData(dataDir):
    imgs = []
    labels = []
    for key, value in dic.items():
        path = os.path.join(dataDir, key)
        for imgName in os.listdir(path):
            try:
                img = cv2.imread(os.path.join(path, imgName), cv2.COLOR_BGR2GRAY)
                imgs.append(img)
                labels.append(value)
            except Exception as e:
                print(e)
    return (imgs, labels)
```

```python
#imgs, labels = loadData('mathSymbolsDataset/')
#with open("x_symbols.pickle", 'wb') as f:
#    pickle.dump(imgs, f)
#with open("y_latex.pickle", 'wb') as f:
#    pickle.dump(labels, f)
```

```python
with open("x_symbols_reduced.pickle", 'rb') as f:
    imgs = pickle.load(f)
with open("y_latex_reduced.pickle", 'rb') as f:
    labels = pickle.load(f)
```

Next, we created another dictionary to revert the numeric code that the model predicted to its original class.

```
latexToNums = {k: v for v, k in enumerate(np.unique(labels))}
#this dictionary is to revert the predicted numeric code back to latex:
numsToLatex = {v: k for v, k in enumerate(np.unique(labels))}
latexToNums
```

```
Output exceeds the size limit. Open the full output data in a text editor
{'(': 0,
 ')': 1,
 '+': 2,
 '-': 3,
 '0': 4,
 '1': 5,
 '2': 6,
 '3': 7,
 '4': 8,
 '5': 9,
 '6': 10,
 '7': 11,
 '8': 12,
 '9': 13,
 '<': 14,
 '=': 15,
 '>': 16,
 '\\Pi': 17,
 '\\X': 18,
```

4.1 Preparing The dataset For Training:

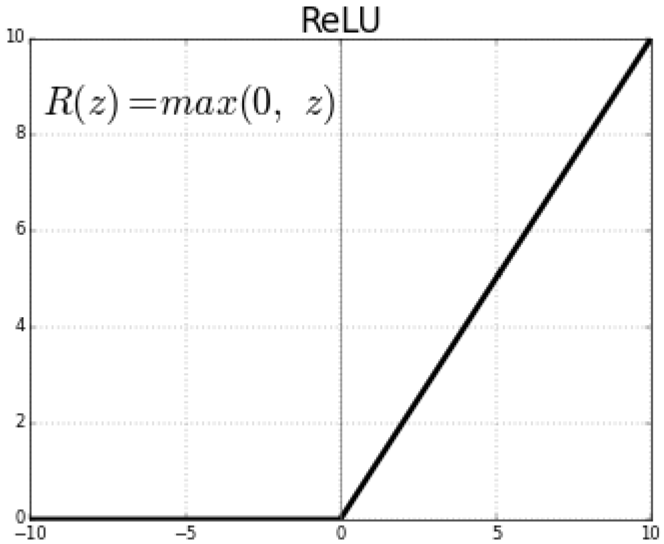
We split the data set from trainging and testing then normalize the images so it would be more efficient during training.

```
x_train, x_test, y_train, y_test = train_test_split(imgs, labels, test_size=0.33, stratify=labels, random_state=42)
```

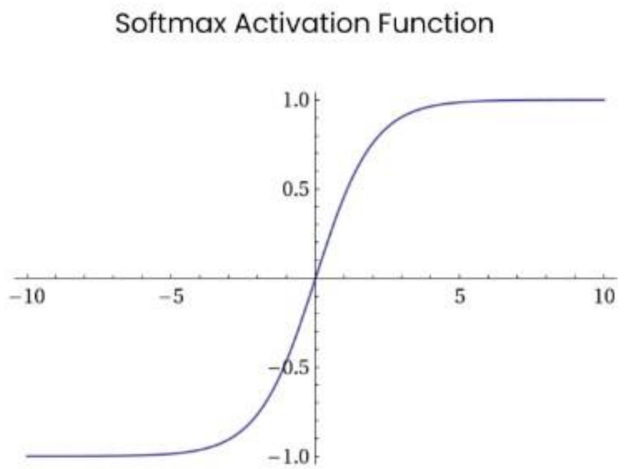
```
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1) #
```

4.2 Training The Model

The neural network is composed of 3 layers, the first 2 hidden layers consist of 2025 neuron while the output layer contains 32 output neurons as they are 32 classes. The choice of 2025 is that it’s the same number of the pixels as suggested by some experts. The hidden layers are using Relu function as an activation function which is a function that fires when the value is postive otherwise, it assigns it to zero.



The output layer uses sofmax as an activation function which calculates the probability for each weight and chooses the highest one.



The optimizer used is adam optimizer which is considered an algorithm that accelerates the gradient decent algorithm by taking the average of the two gradient decents used. The loss is determined by the sparse categorical crossentropy which computes the loss bamong the classes and the labels.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(2025, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(2025, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(32, activation=tf.nn.softmax))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
    model.fit(x_train, y_train_nums, epochs=5)

Epoch 1/5
5465/5465 [==============================] - 282s 51ms/step - loss: 1.4541 - accuracy: 0.5773
Epoch 2/5
5465/5465 [==============================] - 281s 51ms/step - loss: 0.8027 - accuracy: 0.7626
Epoch 3/5
5465/5465 [==============================] - 288s 53ms/step - loss: 0.6237 - accuracy: 0.8137
Epoch 4/5
5465/5465 [==============================] - 286s 52ms/step - loss: 0.5004 - accuracy: 0.8504
Epoch 5/5
5465/5465 [==============================] - 268s 49ms/step - loss: 0.4243 - accuracy: 0.8717

<keras.callbacks.History at 0x17168d96eb0>
```

The model is trained with 87% of accuracy which is doable for the purpose of this model.

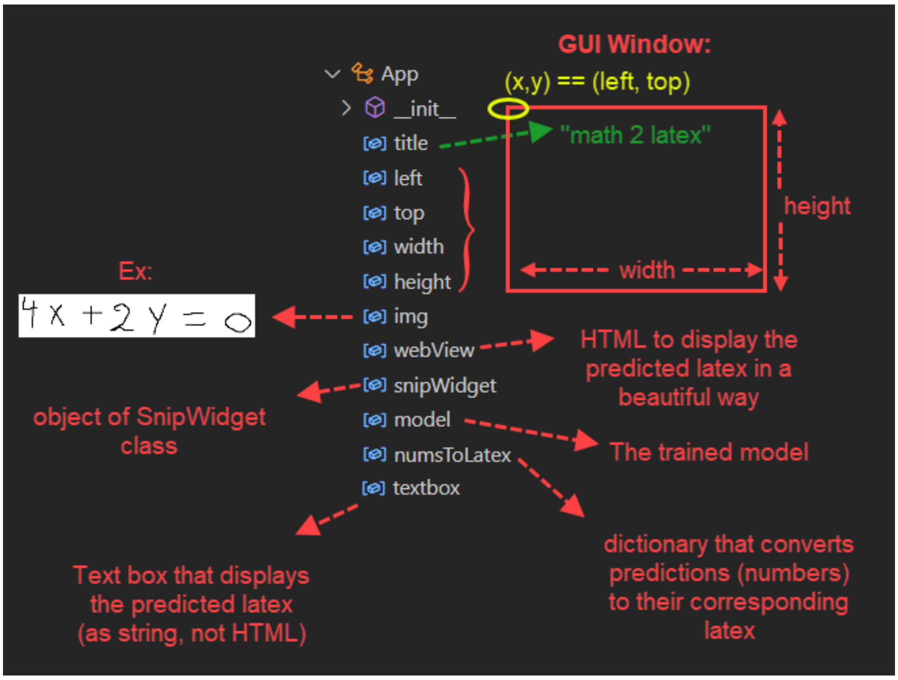
4.3 Developing The GUI

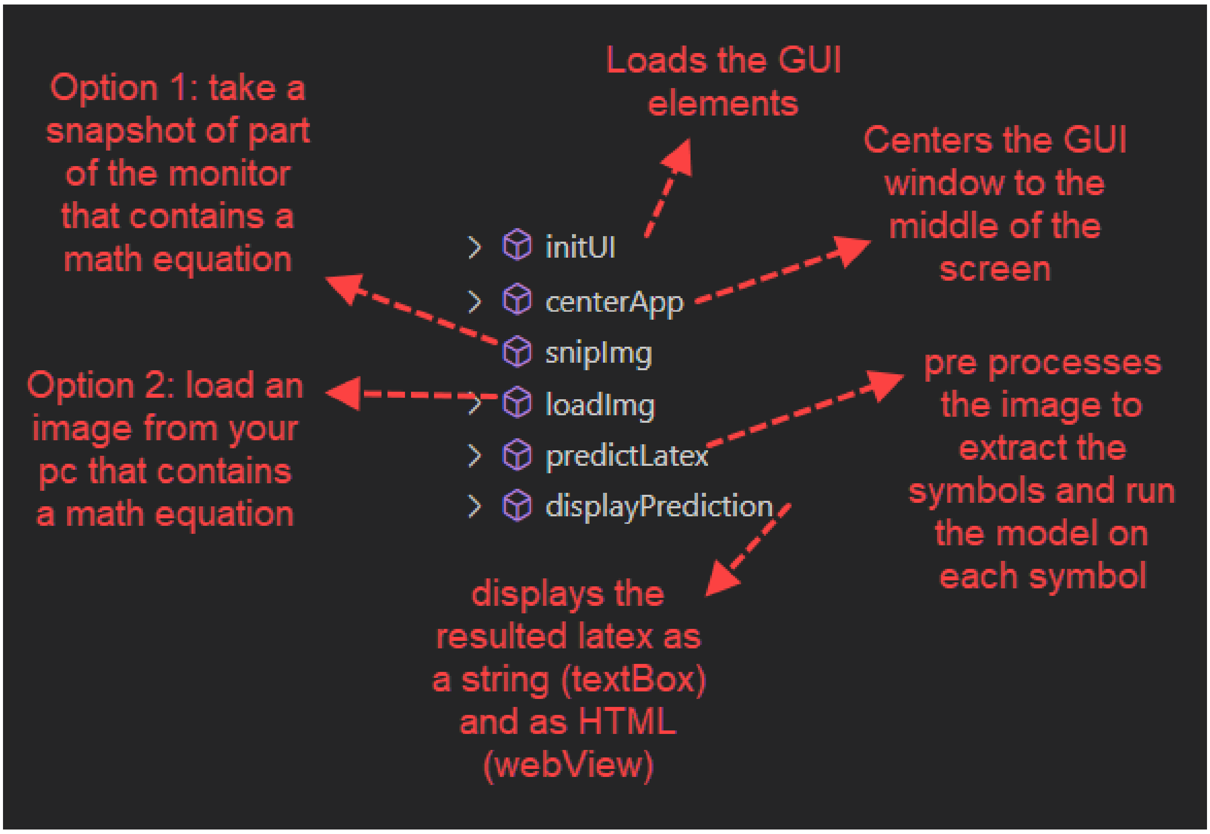
∵ The code of the GUI is a lot more than the rest of the files, comments have been written in the gui.py file that will facilitate understanding the details of the “App” class and the “snipWidget” class.

∴ Only a visualization of the main logic points of “gui.py” will be described:

“App” Class

the represents the class’s attributes while the represents the class’s methods:





“App” object (the GUI) and the info displayed on the monitor (directory folders on the pc, any other windows opened for snipping, etc) → Environment

snipImg() & loadImg() → Sensor (Perceptor) Functions

predictLatex() → Agent Function

displayPrediction() → Actuator Function

Code:

```
class App(QMainWindow): # Using QMainWindow as super class because it conta

    def __init__(self):
        super().__init__()
        self.title = 'Math 2 Latex'
        self.left = 10
        self.top = 10
        self.width = 700
        self.height = 400
        self.img = None
        self.webView = None
        self.textbox = None
        self.snipWidget = SnipWidget(self) # object of SnipWidget Class, "s
        self.model = keras.models.load_model("ThennModel")
        with open("numsToLatex.pickle", 'rb') as f:
            self.numsToLatex = pickle.load(f)
        self.initUI()
```

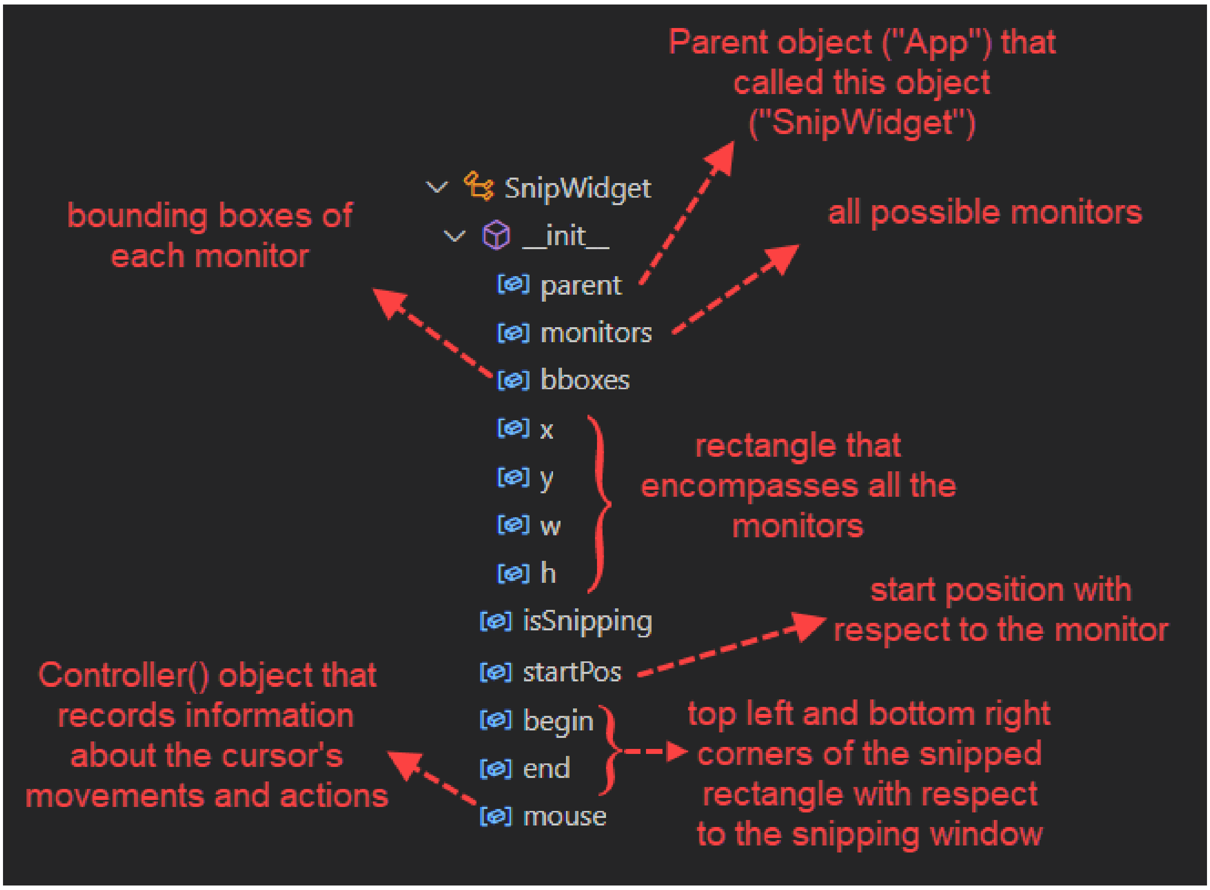
```python
    def initUI(self):
        self.setWindowTitle(self.title)
        QApplication.setWindowIcon(QtGui.QIcon('resources/Pi-Black.svg')) #
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.centerApp() # user defined

        # Create LaTeX display
        self.webView = QWebEngineView()
        self.webView.setHtml("")
        self.webView.setMinimumHeight(40)

        # Creates Textbox
        self.textbox = QTextEdit(self)
        self.textbox.textChanged.connect(self.displayPrediction)
        self.textbox.setMinimumHeight(40)

        # Creates snip button
        btnSnip = QPushButton('Snip', self)
        btnSnip.setToolTip('This is to snip an image of a math equation')
        btnSnip.setMinimumHeight(40)
        btnSnip.clicked.connect(self.snipImg)

        # Creates load image button
        btnLoad = QPushButton("Load Image", self)
        btnLoad.setToolTip('This is to load an image from a folder locally'
        btnLoad.setMinimumHeight(40)
        btnLoad.clicked.connect(self.loadImg)

        # Create Vertical & Horizontal layouts to main window (centralWidge
        centralWidget = QWidget()
        centralWidget.setMinimumWidth(200)
        self.setCentralWidget(centralWidget)

        vBox = QVBoxLayout(centralWidget)
        vBox.addWidget(self.webView, stretch=4)
        vBox.addWidget(self.textbox, stretch=2)

        hBox = QHBoxLayout()
        hBox.addWidget(btnSnip)
        hBox.addWidget(btnLoad)
        vBox.addLayout(hBox)

        settings = QFormLayout()
        vBox.addLayout(settings)

        self.show()
```

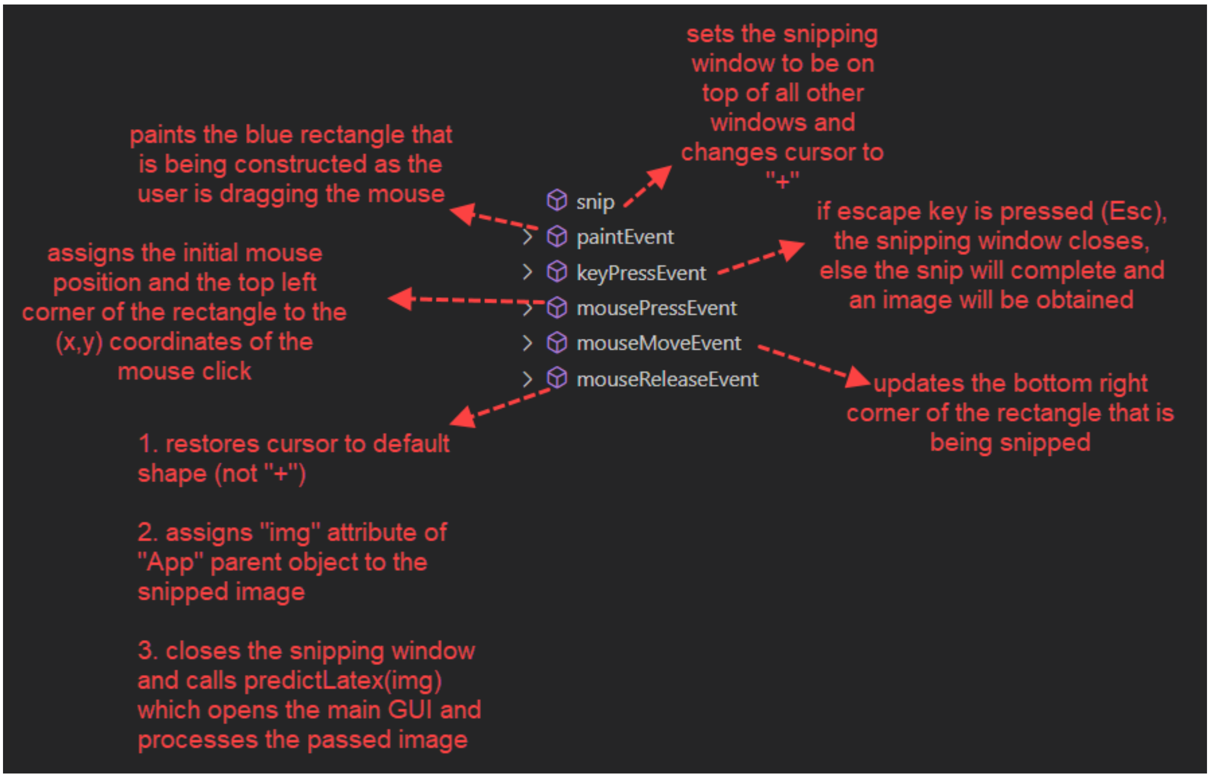
```python
    def centerApp(self): # centers application
        qtRectangle = self.frameGeometry() # retrieves geometry of the wind
        centerPoint = QDesktopWidget().availableGeometry().center() # gets
        qtRectangle.moveCenter(centerPoint) # moves created window to cente
        qPoint = qtRectangle.topLeft()
        self.move(qPoint) # moves current application's window to created w


    @pyqtSlot() #decorator function that runs some built-in code (used for
    def snipImg(self):
        self.close()
        self.snipWidget.snip()

```

```
    @pyqtSlot()
    def loadImg(self):
        currDirectory = os.path.abspath(os.getcwd()) # gets path of current py
        imgsDirectory = os.path.join(currDirectory, "tests") # concatenates te
        fname = QFileDialog.getOpenFileName(self, "Open file",
                                            imgsDirectory,
                                            "Image files (*.jpg *.png)") # "fr
        self.img =  cv2.imread(fname[0]) # setting the img attribute in case i
        self.predictLatex(self.img)

    def predictLatex(self, img=None):
        self.show() # Displays the main GUI window after it has been closed by
        symbols = extractSymbols(imgOrig=img, showSteps=True, medFilter=True)
        prediction = ""
        for symbol in symbols:
            label = np.argmax(self.model.predict(symbol))
            latex = self.numsToLatex[label]
            print(latex) # Debugging
            prediction += latex + ' '
        prediction = prediction.replace('\X', 'X')
        print(prediction)
        self.displayPrediction(prediction)
```

```
    @pyqtSlot()
    def displayPrediction(self, prediction = None):
        if prediction is not None:
            self.textbox.setText("${equation}$".format(equation=prediction))
        else:
            prediction = self.textbox.toPlainText().strip('$')
        pageSource = """
        <html>
            <head>
                <script type="text/javascript" src="qrc:MathJax.js"> <!-- if
                </script>
            </head>
            <body>
                <p>
                    <mathjax style="font-size:2.3em">
                        $${equation}$$
                    </mathjax>
                </p>
            </body>
        </html>
        """.format(equation=prediction)

        self.webView.setHtml(pageSource)
```

“SnipWidget” Class





snip() → Sensor (Perceptor) Function

Code:

```python
class SnipWidget(QMainWindow):

    def __init__(self, parent):
        super().__init__()
        self.isSnipping = False
        self.parent = parent # "parent" here is the "App" object that called

        monitors = get_monitors() # gets monitor's x and y position of top l
        bboxes = np.array([[m.x, m.y, m.width, m.height] for m in monitors])
        x, y, _, _ = bboxes.min(0) # retrieves the positions of the smallest
        w, h = bboxes[:, [0, 2]].sum(1).max(), bboxes[:, [1, 3]].sum(1).max(
        self.setGeometry(x, y, w-x, h-y) # sets the new snipping window with

        self.startPos = None
        self.begin = QtCore.QPoint()
        self.end = QtCore.QPoint()

        self.mouse = Controller() # a controller for sending virtual mouse e
```

```python
    def snip(self):
        self.isSnipping = True
        self.setWindowFlags(Qt.WindowStaysOnTopHint) # hints are used to cus
        QApplication.setOverrideCursor(QtGui.QCursor(QtCore.Qt.CrossCursor))

        self.show() # displays the snipping window
```

```python
    def paintEvent(self, event):
        if self.isSnipping:
            brushColor = (51, 153, 255, 100) # red, green, blu
            opacity = 0.3
        else:
            brushColor = (255, 255, 255, 0)
            opacity = 0
        lineWidth = 3

        self.setWindowOpacity(opacity)
        qp = QtGui.QPainter(self)
        qp.setPen(QtGui.QPen(QtGui.QColor('blue'), lineWidth))
        qp.setBrush(QtGui.QColor(*brushColor)) # "*" unpacks th
        qp.drawRect(QtCore.QRect(self.begin, self.end)) # draws
```

```python
    def keyPressEvent(self, event):
        if event.key() == QtCore.Qt.Key_Escape: # if escape key
            QApplication.restoreOverrideCursor() # restores the
            self.close()
            self.parent.show()
        event.accept() # this means to pass the event of the c
```

```python
    def mousePressEvent(self, event):
        self.startPos = self.mouse.position
        self.begin = event.pos() # (x,y) wi
        self.end = self.begin # sets begin
        self.update() # updates the window
```

```python
    def mouseMoveEvent(self, event):
        self.end = event.pos() # changes en
        self.update()
```

```python
    def mouseReleaseEvent(self, event):
        self.isSnipping = False
        QApplication.restoreOverrideCursor() # restores the original cursor inste

        startPos = self.startPos
        endPos = self.mouse.position

        # this is to make sure (x1,y1) is the top left corner and (x2,y2) is the
        x1 = min(startPos[0], endPos[0])
        y1 = min(startPos[1], endPos[1])
        x2 = max(startPos[0], endPos[0])
        y2 = max(startPos[1], endPos[1])

        self.repaint() # same as self.update() but repaint() forces an immediate
        QApplication.processEvents() # function that returns after all available
        self.parent.img = ImageGrab.grab(bbox=(x1, y1, x2, y2), all_screens=True)
        self.close() # closes the snipping window that approximately covers the m
        self.parent.predictLatex(self.parent.img) # calls the predictLatex() func
```

code that runs when typing “python gui.py” in the terminal:

```python
if __name__ == '__main__':
    appPtr = QtCore.QCoreApplication.instance() # this pointer and the if state
    if appPtr is None:
        app = QApplication(sys.argv) # super class
    ex = App() # subclass
    sys.exit(app.exec_()) # executing super class which executes subclass
    # app.exec_() runs a GUI event loop that waits for user actions (events)
    # and dispatches them to the right widget for handling.
```

Basically, the code starts the “App” GUI.

5. Experiments

This project has been experimented with six different attempts of 6 different neural network models without counting the working one to achieve the most accurate one.

5.1 First Attempt

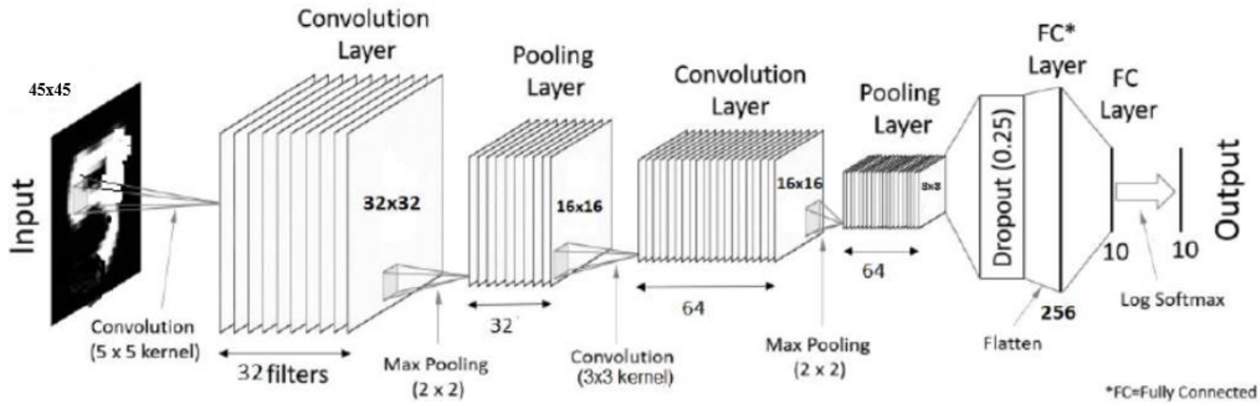
In the first attempt a neural network was applied but with only 128 neurons for the two hidden layers and it turned out that they weren’t enough.

```
# for easier processing: flatten image (e.g. 45x45 will become 1x2025)
model.add(tf.keras.layers.Flatten())
# 128 nodes are chosen as they are a power of 2 (2^7) which makes computation easier, and the images are not larg
# relu is the default activation function to use
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
# add another layer because if you have one, then you're getting linear relations only between the image's featur
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
# number of classifications == number of stored latex strings == len(latexToNums) == 79
# using softmax as it converts the scores to a normalized probability distribution
model.add(tf.keras.layers.Dense(len(latexToNums), activation=tf.nn.softmax))
```

```
# "compiling" means passing the settings for actually optimizing/training the model we've defined
model.compile(optimizer='adam', # same logic as relu, great default optimizer to start with
              loss='sparse_categorical_crossentropy', # A neural network doesn't actually attempt to maximize accuracy
              metrics=['accuracy']) # ratio between the number of correct predictions to the total number of predictio
```

5.2 Second Attempt

The second attempt was to turn directions to the convolutional neural network since it is considered best practice for image data set, it creates filters and




```python
model = Sequential()
model.add(Conv2D(32, (3,3), activation  ='relu', input_shape=(45,45,1)))
model.add(MaxPool2D((2,2)))
#batch normalization, try averagepool
#leak
model.add(Conv2D(48, (3,3), activation='relu'))
model.add(MaxPool2D((2,2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(2025, activation='relu'))
model.add(Dense(79, activation='softmax'))
```

```python
model.compile(optimizer = 'adam', loss='sparse_categorical_crossentropy', metrics = ['accuracy'])
x = model.fit(x_train, y_train_nums, epochs=10, batch_size=128, verbose=2, validation_split=0.1)
```

```
Epoch 1/10
1722/1722 - 386s - loss: 0.6359 - accuracy: 0.8275 - val_loss: 0.2793 - val_accuracy: 0.9140 - 386s/epoch - 224ms/step
Epoch 2/10
1722/1722 - 418s - loss: 0.2386 - accuracy: 0.9268 - val_loss: 0.1760 - val_accuracy: 0.9441 - 418s/epoch - 243ms/step
Epoch 3/10
1722/1722 - 401s - loss: 0.1689 - accuracy: 0.9461 - val_loss: 0.1280 - val_accuracy: 0.9589 - 401s/epoch - 233ms/step
Epoch 4/10
1722/1722 - 423s - loss: 0.1312 - accuracy: 0.9569 - val_loss: 0.1148 - val_accuracy: 0.9613 - 423s/epoch - 246ms/step
Epoch 5/10
1722/1722 - 406s - loss: 0.1091 - accuracy: 0.9634 - val_loss: 0.0922 - val_accuracy: 0.9711 - 406s/epoch - 236ms/step
Epoch 6/10
1722/1722 - 410s - loss: 0.0934 - accuracy: 0.9684 - val_loss: 0.0835 - val_accuracy: 0.9761 - 410s/epoch - 238ms/step
Epoch 7/10
1722/1722 - 428s - loss: 0.0828 - accuracy: 0.9721 - val_loss: 0.0740 - val_accuracy: 0.9781 - 428s/epoch - 248ms/step
Epoch 8/10
1722/1722 - 430s - loss: 0.0739 - accuracy: 0.9752 - val_loss: 0.0722 - val_accuracy: 0.9804 - 430s/epoch - 250ms/step
Epoch 9/10
1722/1722 - 379s - loss: 0.0676 - accuracy: 0.9773 - val_loss: 0.0653 - val_accuracy: 0.9817 - 379s/epoch - 220ms/step
Epoch 10/10
1722/1722 - 363s - loss: 0.0623 - accuracy: 0.9791 - val_loss: 0.0603 - val_accuracy: 0.9833 - 363s/epoch - 211ms/step
```

The images in the dataset are at size 45 x 45 pixels, first a convolutional layer of 32 filters is performed on the image with matrix 3x3 , then a max pooling of 32 layers with matrix 2x2. Then another convolutional layer of 64 filters is added with 3x3 matrix and a 64 Max Pooling layer is applied. Then lastly, a dropout of 0.5 to reduce overfitting.

The problem with this model is that it was too heavy and after all inaccurate. Although the model gave accuracy of 98%, it gave highly inaccurate results. We assumed this may have been caused due to overfitting.

5.3 Third Attempt

```
#model = Sequential()
#model.add(Conv2D(64, (3,3), input_shape=x_train.shape))
#model.add(Activation("relu"))
#model.add(MaxPool2D(pool_size=(2,2)))

#model.add(Conv2D(64, (3,3)))
#model.add(Activation("relu"))
#model.add(MaxPool2D(pool_size=(2,2)))

#model.add(Flatten())
#model.add(Dense(64))
#model.add(Dense(1))
#model.add(Activation('sigmoid'))

#model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
#model.fit(x_train, y_train_nums, batch_size=128, validation_split=0.1)
```

In this attempt, we tried to use sigmoid as the activation function but the model was too heavy to run, and it was discovered later that the sigmoid function is inappropriate for multi-categorical classes.

5.4 Fourth Attempt

```
model = Sequential()
model.add(Conv2D(64, (3,3), activation  ='relu', input_shape=(45,45,1)))
model.add(MaxPool2D((2,2))) #batch normalization, try averagepool
#leak
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPool2D((2,2)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPool2D((2,2)))

#model.add(Conv2D(64, (3,3), activation='relu'))
#model.add(MaxPool2D((2,2)))

model.add(Dropout(0.5))
model.add(Flatten())

model.add(Dense(2025, activation='relu'))
#model.add(Dense(2025, activation='relu'))
BatchNormalization(axis=1)
model.add(Dense(79, activation='softmax'))
```

The fourth attempt was to try and add more layers to the model both convolutional and pooling layers. Nevertheless, the same problem was still present so it's most probable that the overfitting was still existent Nevertheless, we concluded that extracting features from a character will not be appropriate since characters don't have highlighting features to distinct between them and it can be fairly easy to confuse between their features. That's why we decided to turn to the artificial neural networks yet again.

5.5 Fifth Attempt

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(1000, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(1000, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(1000, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(32, activation=tf.nn.softmax))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
5465/5465 [==============================] - 120s 22ms/step - loss: 1.5544 - accuracy: 0.5430
Epoch 2/10
5465/5465 [==============================] - 120s 22ms/step - loss: 0.8703 - accuracy: 0.7398
Epoch 3/10
5465/5465 [==============================] - 134s 25ms/step - loss: 0.6568 - accuracy: 0.8006
Epoch 4/10
5465/5465 [==============================] - 147s 27ms/step - loss: 0.5453 - accuracy: 0.8328
Epoch 5/10
5465/5465 [==============================] - 142s 26ms/step - loss: 0.4646 - accuracy: 0.8571
Epoch 6/10
5465/5465 [==============================] - 133s 24ms/step - loss: 0.4109 - accuracy: 0.8715
Epoch 7/10
5465/5465 [==============================] - 147s 27ms/step - loss: 0.3661 - accuracy: 0.8848
Epoch 8/10
5465/5465 [==============================] - 149s 27ms/step - loss: 0.3264 - accuracy: 0.8965
Epoch 9/10
5465/5465 [==============================] - 152s 28ms/step - loss: 0.2941 - accuracy: 0.9051
Epoch 10/10
5465/5465 [==============================] - 148s 27ms/step - loss: 0.2722 - accuracy: 0.9119
```

We decided to add another layer to the neural network to increase the accuracy but due to the heavy computations and the limited resources, adding 3 layers each has 2025 neurons and an output layer was not possible. So, we downsized it to 1000 neuron each but it was still inaccurate.

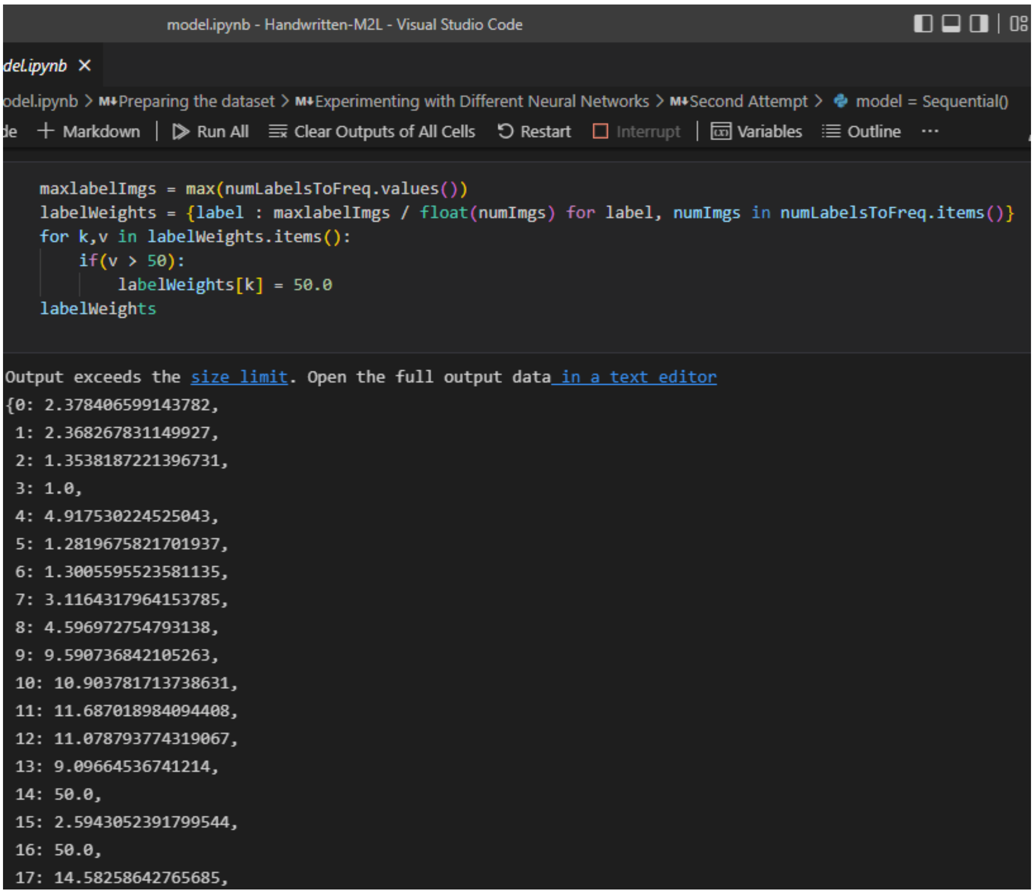
5.6 Sixth Attempt

In the final attempt, we discovered that the dataset is imbalance and that was what caused the inaccuracy although the accuracy of the models was 90+%. The issue was that many of the symbols were predicted as the one symbol that had too many samples in the dataset. So, we decided first to take an equal sample of all classes but some classes had way too few samples to take a sample out of it. Then, w decided to create class weights yet it did not work due to the variance in the sample size of each class.

```
numLabels, counts = np.unique(y_train, return_counts=True)
numLabelsToFreq = dict(zip(numLabels, counts))
numLabelsToFreq
```

```
Output exceeds the size limit. Open the full output data in a text editor
{0: 9577,
 1: 9618,
 2: 16825,
 3: 22778,
 4: 4632,
 5: 17768,
 6: 17514,
 7: 7309,
 8: 4955,
 9: 2375,
 10: 2089,
 11: 1949,
 12: 2056,
 13: 2504,
 14: 320,
 15: 8780,
 16: 173,
 17: 1562,
```

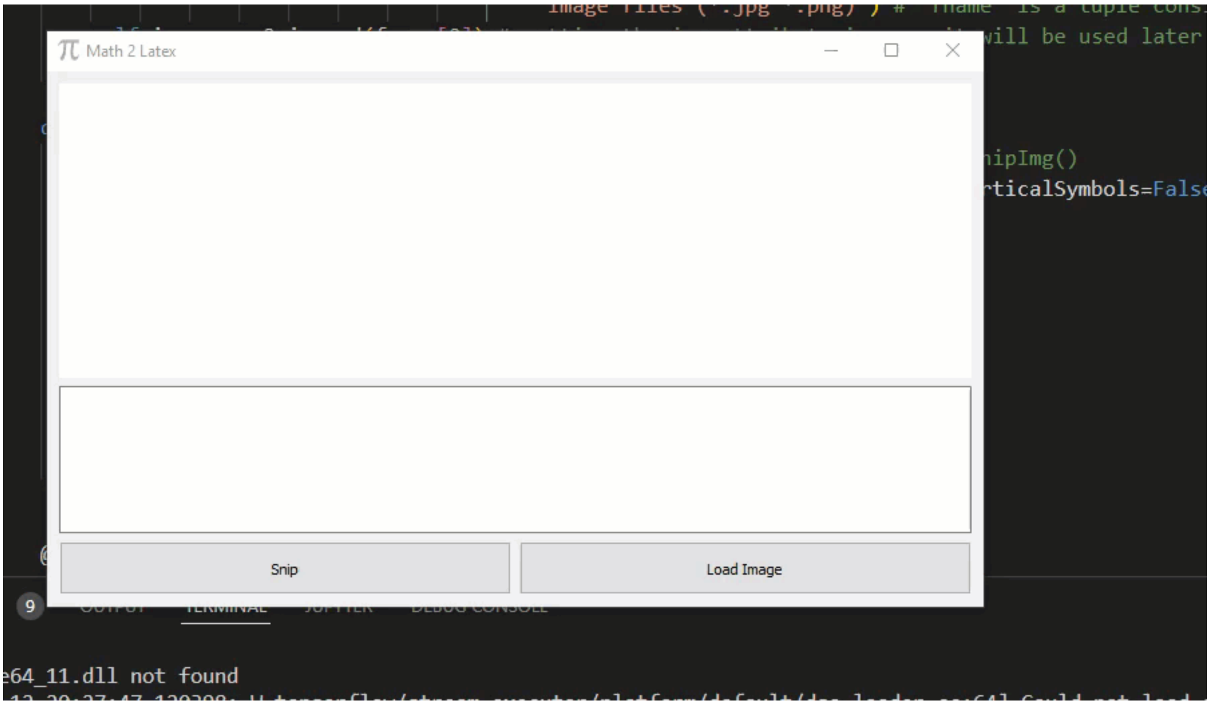
We tried to limit the lovely big class weights to 50 as there were overly big values.



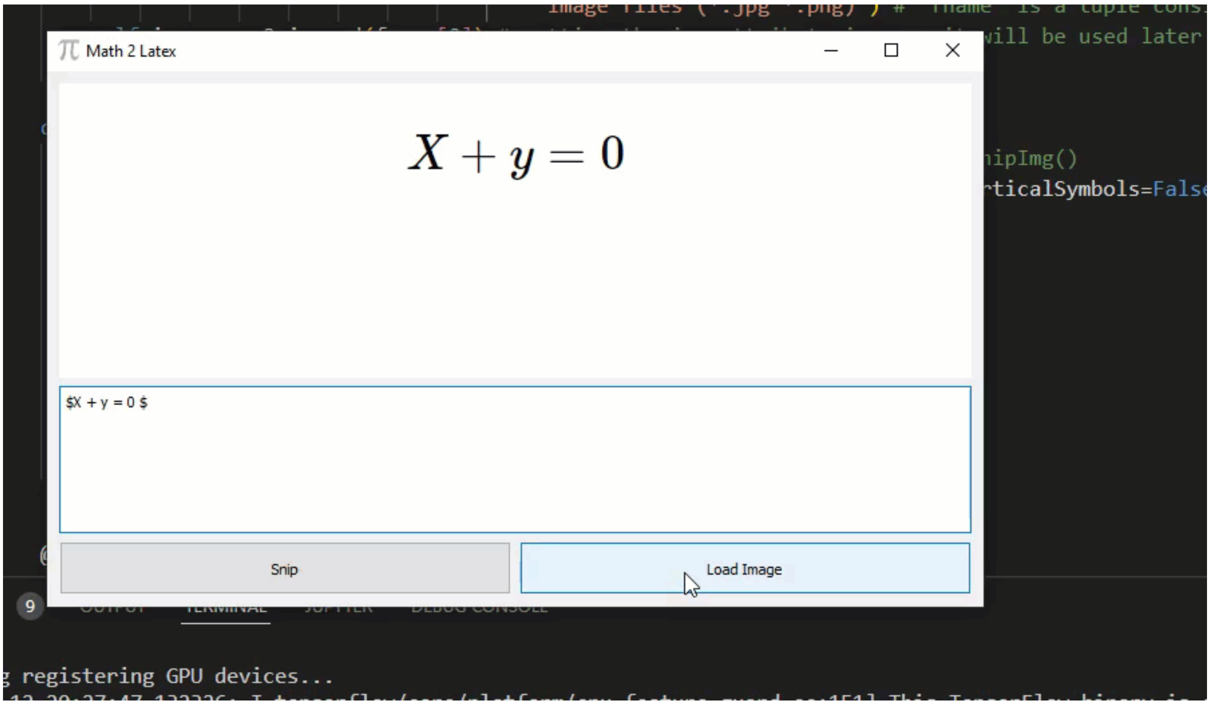
6. Results

Note, these gifs are also named output1.gif, etc in the Handwritten-M2L folder.

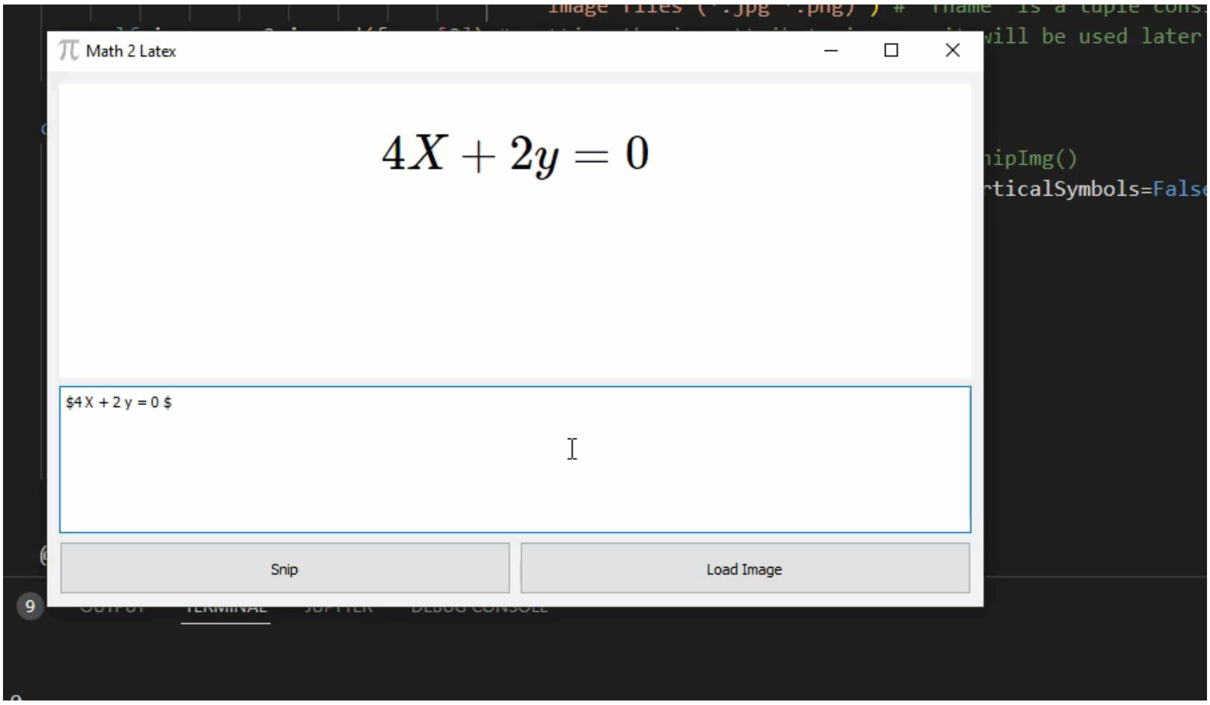
Example on test1.png:



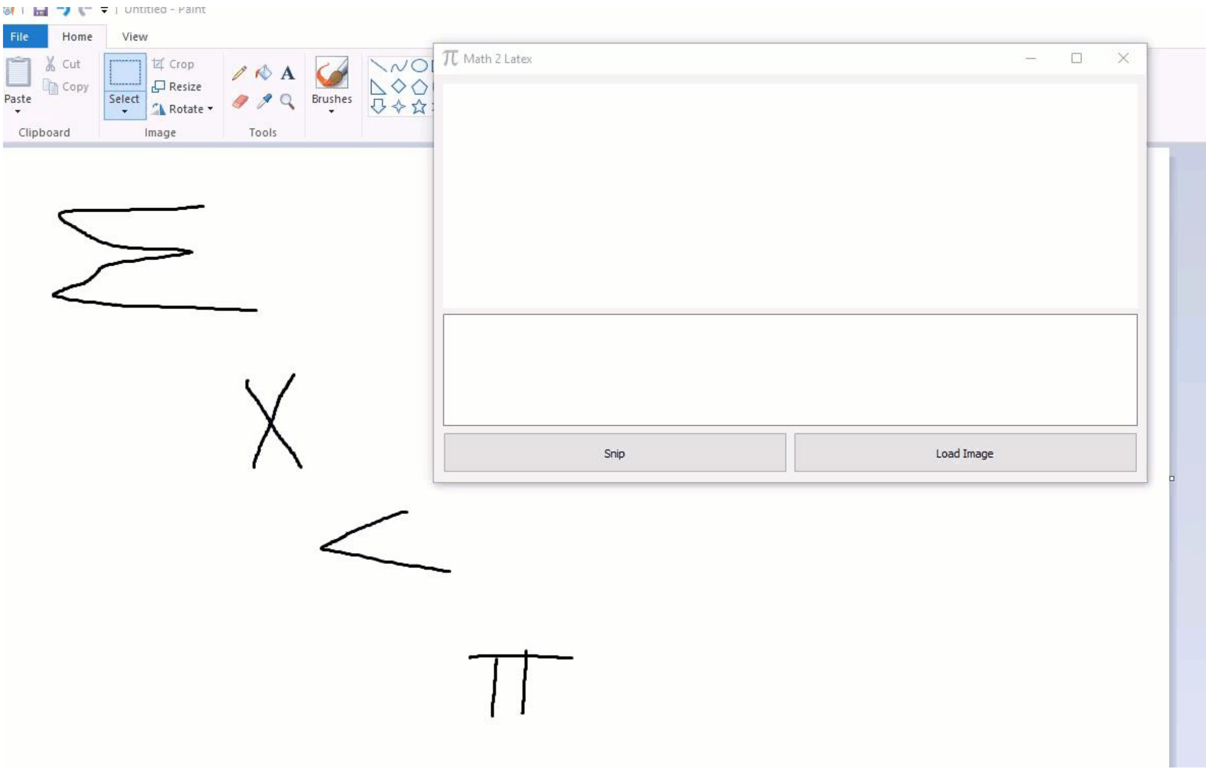
Example on test2.png:



Example on test3.png:



Example on test4.png and snip function (verticalSymbols=True):


Untitled - Paint
File
Home
View
Cut
Copy
Paste
Clipboard
Select
Crop
Resize
Rotate
Image
Tools
Brushes
Math 2 Latex
Snip
Load Image

7. Tools Used

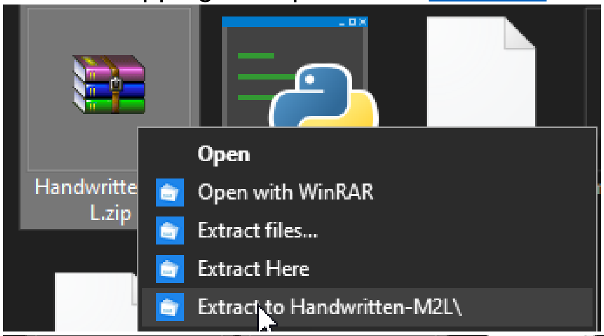
- The language used to implement this project was python, with the help of its many packages to do it.
- Tensor Flow, keras and skilearn for training NN models.
- PyQt5 for GUI
- open CV for displaying and processing images
- Jupyter Notebook
- HTML to display LaTex
- Javascript to elegantly display the LaTex
- VS Code IDE was used throughout the entire project

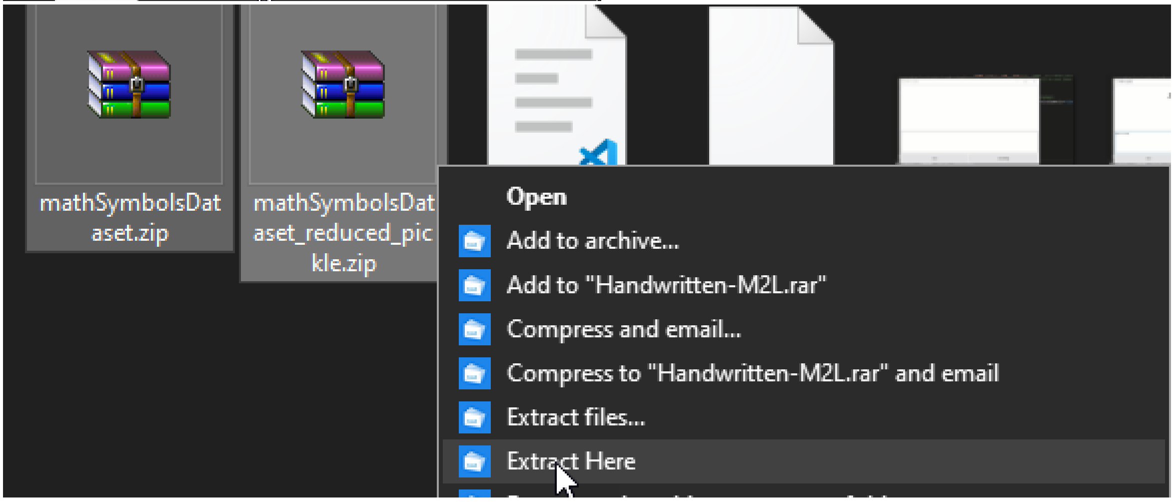
8. How to Use

Finally, to use this application:

1. Pip install everything in the “requirements.txt” (done using pipreqs)
   If some modules still could not be resolved, use “requirementsAll.txt”

2. When unzipping the zip found in [this drive] follow these steps:





3. “python gui.py” in terminal to run the GUI.