Date: Friday, 2/5/2021

Goals

Long Term:	**Short Term:**
- To be determined	- Research possible project topics

Progress

Today's Progress:

For our spring semester, we decided to pursue a project that is related to math. At first, we weren't certain if our topic limited us to pure math theories or data analytics. We didn't think proving or demonstrating any mathematical theories would be a sufficient project, so we began to research various data analytics projects. Some examples include comparing the prices of multiple products from different vendors or using past retail store data to predict store sales. Again, the example data science projects we research didn't seem sufficient for a four month project. To make better use of our time, we emailed our advisors, Mr. Levin and Mr. Pandya, about possible math-related research topics. Over the weekend, Mr. Levin responded with a generous list of past math-related projects, providing us with a wide range of ideas to choose from. Next project block, we will examine the list and narrow our choices to a couple of project ideas.

Evidence of Progress

Research:

- Undergraduate Research Projects
  - https://www.math.northwestern.edu/undergraduate/undergraduate-research-projects/index.html
- Making Mathematics: List of Mathematics Research Projects and Student Work
  - http://www2.edc.org/makingmath/mathproj.asp
- Data Analysis Projects - Machine Learning - CMU - Carnegie Mellon University
  - https://www.ml.cmu.edu/research/data-analysis-projects.html
- 6 Interesting Data Science Project Ideas & Examples
  - https://www.springboard.com/blog/data-science-projects/
- Python Machine Learning Project - Detecting Parkinson's Disease with XGBoost

- ➢ https://data-flair.training/blogs/python-machine-learning-project-detecting-parkinson-disease/

Project Topics:

- ❖ Possible Project Topics

Signatures	
 ***Emma Mascillaro***	 ***Kara Pietrowicz***

Date: Tuesday, 2/9/2021

Goals	
Long Term: - To be determined	**Short Term:** - Research possible project topics - Go through list of project topics provided by Mr. Levin

Progress

Today's Progress:

Using the list of project topics from our advisors, we examined each topic to determine which one we would both enjoy working on together, if we can develop a plausible project, and if this project can be completed within our allotted time (early February to mid-June). We first researched optimization projects, such as maximizing parking spaces in a location with oddly shaped flower and tree beds or optimizing the placement of Wifi access points in a home. These examples and other projects did not interest us, so we moved on to environmental simulations. Depending on the purpose of an environmental simulation, multiple variables would have to be accounted for, like temperature, wind, rain, or solar radiation. Combining these conditions and determining the relationship between each one would be tedious, and we would probably lose interest in our project. Also, it's difficult to ascertain if we could complete this type of project before mid-June.

After continuing our process of elimination, we narrowed the list down to data analysis, data encoding, convolutional neural networks, and image recognition. Last year, we created a model in R to determine if there's a correlation between the number of hospitals and the number of COVID-related deaths in each region of the US. From that experience, we both understood that we can only go so far with data analysis, so it was not our first choice for a senior project. For data encoding, we found a couple of example projects, but again, we weren't sure if these projects would interest us or be sufficient for four months. During our research, we came across some Machine Learning projects, such as credit card fraud detection and handwriting recognition. Since we both have taken ML and python courses before, we chose to pursue a handwriting recognition project. Next class, we will learn what handwriting recognition entails and write our project proposal.

Evidence of Progress

Research:

- Undergraduate Research Projects
  - https://www.math.northwestern.edu/undergraduate/undergraduate-research-projects/index.html
- Making Mathematics: List of Mathematics Research Projects and Student Work
  - http://www2.edc.org/makingmath/mathproj.asp
- DataFlair Data Science Project - Detect Credit Card Fraud with Machine Learning in R
  - https://data-flair.training/blogs/data-science-machine-learning-project-credit-card-fraud-detection/
- Data Analysis Projects - Machine Learning - CMU - Carnegie Mellon University
  - https://www.ml.cmu.edu/research/data-analysis-projects.html
- 6 Interesting Data Science Project Ideas & Examples
  - https://www.springboard.com/blog/data-science-projects/
- Encoding Data with R
  - https://www.pluralsight.com/guides/encoding-data-with-r
- Build a Handwritten Text Recognition System using TensorFlow
  - https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5
- Building OCR and Handwriting Recognition for document images.
  - https://blog.usejournal.com/building-ocr-and-handwriting-recognition-for-document-images-f7630ee95d46

Project Topics:

- ❖ Possible Project Topics

Signatures

 ***Emma Mascillaro***	 ***Kara Pietrowicz***

Date: Thursday, 2/11/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Research more on what handwriting recognition entails - Discuss and finalize a feasible project idea - Write and submit project proposal

Progress

Today's Progress:

Handwriting recognition is the ability of a device to recognize and interpret handwritten input from various sources, such as printed physical documents or images. The input is fed to a pattern-recognition software for interpretation, or a real-time recognition that uses a camera for optical scanning.

There are two types of handwriting recognition. First is the older of the two, known as offline handwriting recognition. This is where the handwritten input is scanned or photographed and given to the computer. The second is online, which is where the writing is input through a stylus/touchscreen. This offers the computer more clues about what's being written, such as stroke direction and pen weight.

Overall, handwriting recognition is about allowing the computer to transform handwriting into a format that the computer understands. For example, optical character recognition (OCR) is the use of technology to distinguish printed or handwritten text characters inside

digital images of physical documents, such as a scanned paper document. This is where the computer zooms in on each character and identifies it by comparing it to a database of known characters and words.

There are some issues with handwriting recognition. Obviously, there is a wide range of handwriting, both legible and illegible, making it difficult for programmers to provide enough examples of how every character might look. Characters can appear very similar to each other, decreasing the computer's ability to accurately recognize characters. Cursive handwriting is another challenge for computers. When your letters all connect, computers cannot recognise individual characters as well as with printed handwriting.

In order to accurately recognize different letters, characters, or digits, deep learning and neural networks are used to create a model that allows computers to learn how to recognize different handwritten texts. These subsections of machine learning aim to teach computers to learn by example. The computer will be learning from masses of unstructured (messy) data to draw its own conclusions. For instance, deep learning will use thousands of images of characters for the model. In this model, the computer will learn to spot patterns with certain characters and learn how to classify those images using neural networks.

Neural networks are a way for a computer to process data, and they are trained on the masses of unstructured data. For example, if we want the network to be able to identify cats, we will give it many pictures of cats, and tell it that the correct answer is 'cat'. The network then processes the input and recognises patterns. Then, when it gets fresh input, it compares it to the pattern it found from the trained model.

We decided to dedicate our project to handwriting recognition since both of us have taken Machine Learning and python courses, and we think it's possible to complete this project within our allotted time period. Also, we're very interested in training our model using handwritten text samples collected from our peers. Using our model, we considered developing an app that can interpret handwritten text and possibly solve mathematical problems using a four-function calculator script. If time permits, we could also use our developed app to spellcheck handwritten text. With these ideas in mind, we wrote and submitted our proposal to our advisors for approval.

Evidence of Progress

Research:

- Handwritten Text Recognition using TensorFlow 2.0 | by Arthur Flôr | Medium

- ➢ https://arthurflor23.medium.com/handwritten-text-recognition-using-tensorflow-2-0-f4352b7afe16

❖ Why is handwriting recognition so difficult for AI?

- ➢ https://www.thinkautomation.com/bots-and-ai/why-is-handwriting-recognition-so-difficult-for-ai/

❖ What is OCR (optical character recognition)? - Definition from WhatIs.com

- ➢ https://searchcontentmanagement.techtarget.com/definition/OCR-optical-character-recognition

❖ ELI5: what is deep learning?

- ➢ https://www.thinkautomation.com/eli5/eli5-what-is-deep-learning/

❖ Older Post How to easily do Handwriting Recognition using Deep Learning

- ➢ https://nanonets.com/blog/handwritten-character-recognition/

❖ How to Make an App for Beginners (2020) - Lesson 1

- ➢ https://youtu.be/jniJeamcIUU

❖ Develop Windows 10 applications learning path - Learn

- ➢ https://docs.microsoft.com/en-us/learn/paths/develop-windows10-apps/

❖ MyScript Calculator on the App Store

- ➢ https://apps.apple.com/us/app/myscript-calculator/id1304488725

Proposal:

❖ Mascillaro & Pietrowicz Project Proposal

Signatures	
 ***Emma Mascillaro***	 ***Kara Pietrowicz***

Date: Tuesday, 2/16/2021

Goals	
Long Term: - Write handwriting recognition code to	**Short Term:** - Create Github repository for our code

identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Develop a tentative timeline for our project - Understand the overall process for training an OCR model

Progress

Today's Progress:

Since our project involves a lot of trial and error within the neural networks of our code, we created a Github repository so we can save the original versions of our code while we make adjustments. For our code editor, we will use Visual Studio Code since it allows us to collaborate on the same python file.

Next, we created a tentative timeline for our project so we have an idea of what needs to be accomplished. It includes questions we had about certain aspects of our project that we want to consider during the next four months. For instance, what should we do if the images in the dataset are different sizes? Will this affect the training model? Throughout our project, the timeline will change depending on our conducted research for different aspects of our project.

Before we can begin writing our code, we need to understand the general process of how to train an OCR model. First, we need to import a large classified dataset that contains both training and testing images of handwritten text. Next, we need to perform some operations and preprocess the data in order for the neural network to use the dataset images, such as reshaping the matrix the images are within. After preprocessing, the OCR neural network model needs to be created. Neural networks work well for image classification problems since the images are represented as grid structures. This is when we will have to use trial and error to determine the best use of neural network layers to produce a high accuracy value. Once we train the model, we use the testing images of our dataset to evaluate how well our model works. The testing data was not involved in the training of the model. Therefore, we can see how accurate our model is when given new data. Next project block, we will begin writing our handwriting recognition code.

Evidence of Progress

Research:

- ❖ Mobile App Development Process: A Step-by-Step Guide [2020]
  - ➢ https://www.invonto.com/insights/mobile-app-development-process/

- How to Make an App for Beginners (2020) - Lesson 1
  - https://youtu.be/jniJeamcIUU
- Develop Windows 10 applications learning path - Learn
  - https://docs.microsoft.com/en-us/learn/paths/develop-windows10-apps/
- MyScript Calculator on the App Store
  - https://apps.apple.com/us/app/myscript-calculator/id1304488725
- How I built a handwriting recognizer and shipped it to the App Store
  - https://www.freecodecamp.org/news/build-a-handwriting-recognizer-ship-it-to-app-store-fcce24205b4b/
- What is OCR (optical character recognition)? - Definition from WhatIs.com
  - https://searchcontentmanagement.techtarget.com/definition/OCR-optical-character-recognition
- Deep Learning Project - Handwritten Digit Recognition using Python
  - https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/

Github Repository:

- https://github.com/emascillaro/SeniorProject_Spring

Project Timeline:

- Handwriting Recognition for Mathematical Applications Project Overview

Signatures	
 ***Emma Mascillaro***	 ***Kara Pietrowicz***

Date: Thursday, 2/18/2021

Goals	
Long Term: - Write handwriting recognition code to	**Short Term:** - Begin writing handwriting recognition

identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	code - Take a closer look at the MNIST dataset

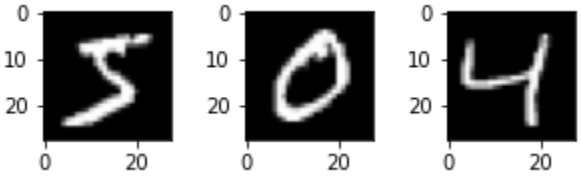
Progress

<u>Today's Progress:</u>

After completing our research on how handwriting recognition code functions, we began to write our OCR code. The following deep learning libraries had to be installed and imported into our file before we could write anything:

- **Numpy**: concentrates on handling extensive multi-dimensional data and the intricate mathematical functions operating on the data.
- **Tensorflow**: allows use to create and train ML models using neural networks.
- **Keras**: features several of the building blocks and tools necessary for creating a neural network, such as neural layers and activation functions (more in depth explanation will be provided later).
- **Pillow**: Python Imaging Library (PIL), which adds support for opening, manipulating, and saving images.

Before we can recognize and transcribe handwritten text, we wanted to start simple and recognize handwritten digits (0-9) from a popular dataset called MNIST. It contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value. Later we will create our own dataset, but we wanted to create a model trained on an acceptable and very common dataset first. Here is an image from MNIST.



```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(x_train.shape, y_train.shape)

# prints 3 images from x_train
for i in range(3):
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(x_train[i], cmap=pyplot.get_cmap('gray'))
  pyplot.show()

#reshapes the matrix of training and testing images
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
```

The above image displays the starter code we wrote. First, we split the MNIST dataset into training data and testing data. The x_train and x_test variables contain the handwritten digit images while the y_train and y_test variables contain the digit labels for each image. We printed 3 images from our training data (as shown previously) so we would know what kind of data our model is being trained on. Lastly, we had to reshape the matrix that contained the training and testing images since neural network models require one more dimension in order to function properly.

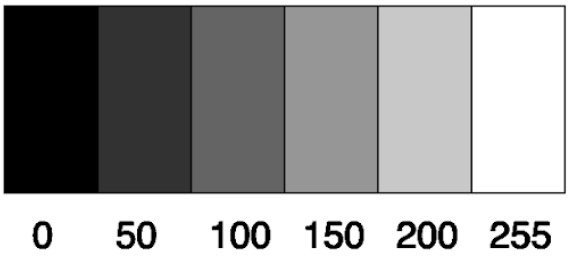
```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('final x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Next, we converted the digit labels in the y_train and y_test vector datasets to binary class matrices. We specified that there are 10 classes of digits, 0 to 9, to create a 60,000 by 10 matrix for the training labels. Each row of the matrix describes the label of the corresponding image with a binary meaning. Below, the corresponding image to this label would be zero since there is a 1 in the 0 position of the matrix.

```
(60000, 10)
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

We then used the astype() function to cast our entire x_train and x_test matrices to float32, which is a number format that represents a wide range of numeric values and occupies 32 bits of computer memory. Then we divide each number in the x_train and x_test matrices by 255 to normalize our data.

Machine learning algorithms tend to perform better or converge faster when the different features (or in our case numbers) are on a smaller scale. Therefore it is common practice to normalize the data before training machine learning models on it. The images of the MNIST dataset are grayscale and the pixels range between 0 and 255 including both bounding values. Every line in the training and testing dataset consists of a handwritten digit image, or 785 numbers between 0 and 255. The first number of each line is the label, i.e. the digit which is depicted in the image. The following 784 numbers are the pixels of the 28 x 28 image with color range between 0 and 255, as shown below. By decreasing our feature scale to 0 to 1 instead of leaving it as 0 to 255, our OCR model will perform better and faster.



Now that we have preprocessed our data, next class we will begin creating our OCR model.

Evidence of Progress

Research:

- How To Build a Neural Network to Recognize Handwritten Digits with TensorFlow
  - https://www.digitalocean.com/community/tutorials/how-to-build-a-neural-network-to-recognize-handwritten-digits-with-tensorflow
- Build a Handwritten Text Recognition System using TensorFlow

- ➢ https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

❖ Older Post How to easily do Handwriting Recognition using Deep Learning

- ➢ https://nanonets.com/blog/handwritten-character-recognition/

❖ Building OCR and Handwriting Recognition for document images.

- ➢ https://blog.usejournal.com/building-ocr-and-handwriting-recognition-for-document-images-f7630ee95d46

❖ OCR: Handwriting recognition with OpenCV, Keras, and TensorFlow

- ➢ https://www.pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/

❖ Deep Learning Project - Handwritten Digit Recognition using Python

- ➢ https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/

❖ Best Python Libraries for Machine Learning and Deep Learning

- ➢ https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

❖ Pillow Tutorial

- ➢ https://zetcode.com/python/pillow/#:~:text=Pillow%20is%20a%20Python%20Imaging,used%20interchange%20and%20presentation%20formats.

❖ Next How to Load and Plot the MNIST dataset in Python?

- ➢ https://www.askpython.com/python/examples/load-and-plot-mnist-dataset-in-python

❖ Machine Learning with Python: Training and Testing the Neural Network with MNIST data set

- ➢ https://www.python-course.eu/neural_network_mnist.php

❖ Prev 2 Easy Ways to Normalize data in Python

- ➢ https://www.journaldev.com/45109/normalize-data-in-python

Github Repository:

❖ https://github.com/emascillaro/SeniorProject_Spring

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 2/23/2021

Goals

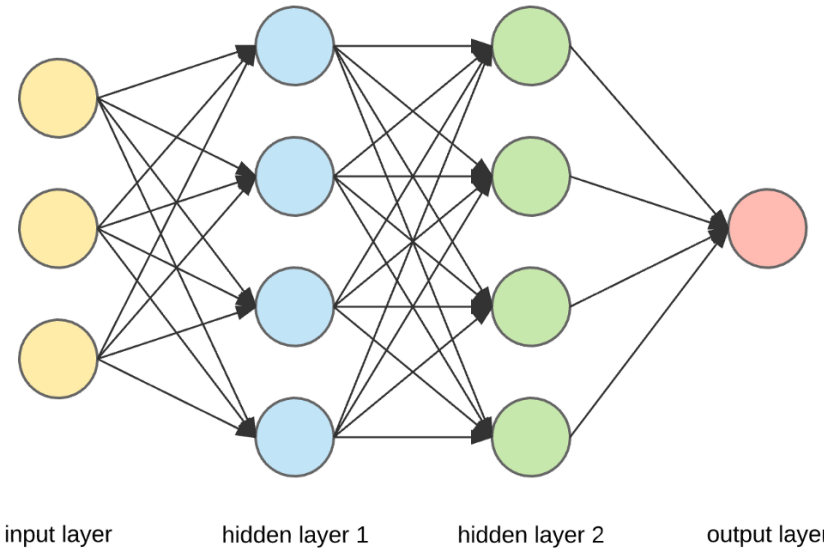
Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Researched Neural Networks and how they function - Begin writing code for model

Progress

Today's Progress:

Neural networks are a subsection of machine learning, in which a computer learns to perform some task by analyzing training datasets. Usually, the components of the datasets have been labeled in advance. An object recognition system, for instance, might be fed thousands of labeled images of cars, houses, coffee cups, and more, and it would find visual patterns in the images that consistently correlate with particular labels. This is called classification. We must transfer our knowledge to our dataset in order for a neural network to learn the correlation between labels and data.

Sometimes we can have "stacked neural networks"; that is, networks composed of several layers. Each layer is made up of nodes, which are places where computation occurs.



An individual node might be connected to several nodes in the layer before it, from which it receives data, and several nodes in the layer after it, to which it sends data, as shown above. Each node connection has a particular weight, which are assigned to a node based on its relative importance against other inputs. When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node “fires,” which means sending the number — the sum of the weighted inputs — along all its outgoing connections to the next layer. The threshold value is determined by an activation function in a layer.

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the input layer and passes through the succeeding layers, getting multiplied and added together in complex ways, until it arrives transformed at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

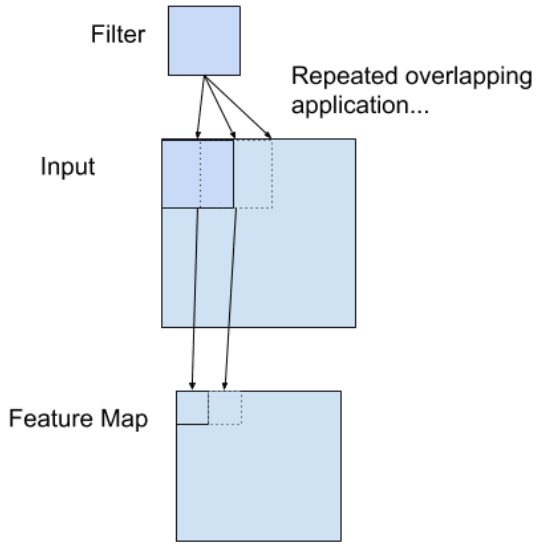
```
batch_size = 128
num_classes = 10
epochs = 10

model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

The above code displays our stacked neural network model that will be trained to classify handwritten digit images with our MNIST training data. Neural networks are defined in Keras as a sequence of layers. The container for these layers is the Sequential class. So we created an instance of the Sequential class. Then we created our layers and added them to the neural network in the order that they should be connected. Num_classess represents the number of classifications for the dataset (0-9 digit labels means that there are 10 classifications). An epoch is the number times that the learning algorithm will work through the entire training dataset. It can consist of one or more batches.

Batch_size denotes the subset size of our training dataset (128 out of 60,000). Each batch trains the model in a successive order, taking into account the updated weights coming from the appliance of the previous batch.

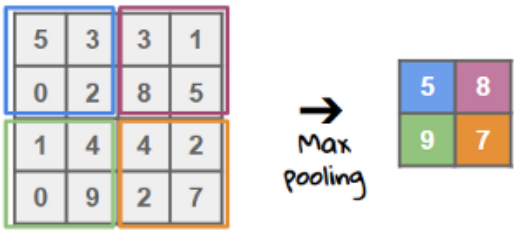
By examining our model, there are five different layers as described below:

Convolutional (Conv2D): applies a filter to an input to create a feature map, which indicates the locations and strength of a detected feature in an input, such as an image.

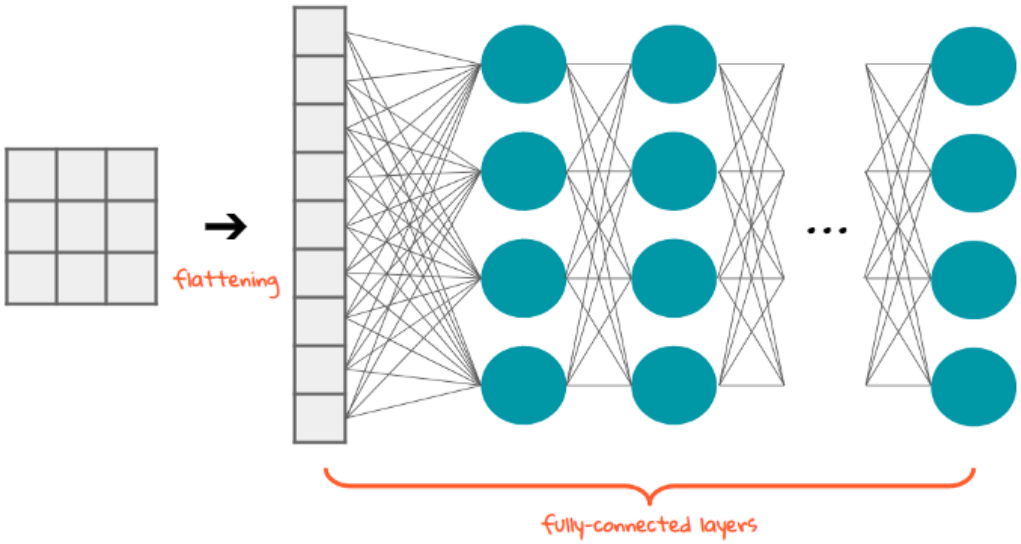


Pooling (MaxPooling2D): Output feature maps record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.

So pooling is required to down sample the detection of features in feature maps. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the model. Basically, it reduces the size of the input data. Max pooling summarizes the most activated presence of a feature in each patch of the feature map. The result of using a pooling layer is a summarized version of the features detected in the input.

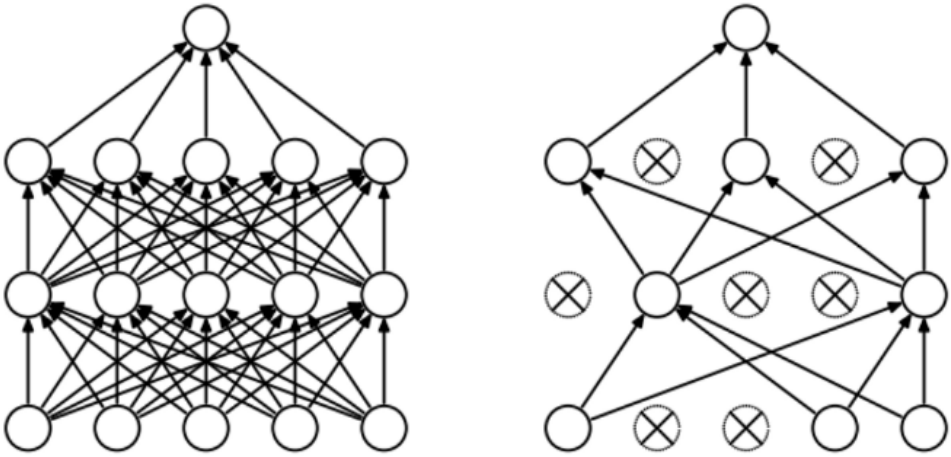


Flattening: converts the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector



Dense: a layer that is connected deeply, which means each node in the dense layer receives input from all nodes of its previous layer. The feature vector created by the Flatten layer is inputted to this layer where final classification occurs.

Dropout: Deep learning neural networks are likely to quickly overfit the training dataset, which results in poor performance when the model is evaluated on new data. Dropout is an approach to regularization (reduces overfitting) in neural networks which helps reduce interdependent learning amongst the nodes.



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

In the above image, part a shows a neural network that is overfitted. So the produced classification output of the model will correspond too closely or exactly to this particular dataset. In part b, the model is reduced to prevent interdependent learning amongst nodes and can fit new data to produce a more accurate classification output.

After researching how various neural networks work, we used some online references to help us develop our model. Next project block, we will begin testing and evaluating the accuracy of our model.

Evidence of Progress

Research:

- A Beginner's Guide to Neural Networks and Deep Learning
  - https://wiki.pathmind.com/neural-network
- Explained: Neural networks
  - https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
- Neural Network: Architecture, Components & Top Algorithms
  - https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/
- Layers in a Neural Network explained
  - https://deeplizard.com/learn/video/FK77zZxaBoI
- 55 Responses to How Do Convolutional Layers Work in Deep Learning Neural Networks?
  - https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
- A Gentle Introduction to Pooling Layers for Convolutional Neural Networks
  - https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/
- The Most Intuitive and Easiest Guide for Convolutional Neural Network | by Jiwon Jeong
  - https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480
- How To Build a Neural Network to Recognize Handwritten Digits with TensorFlow
  - https://www.digitalocean.com/community/tutorials/how-to-build-a-neural-network-to-recognize-handwritten-digits-with-tensorflow

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Wednesday, 2/24/2021

Goals

Long Term:

- Write handwriting recognition code to identify and transcribe written text
- Write four-function calculator scripts
- Develop an app that utilizes all our code to solve mathematical problems

Short Term:

- Deal with TensorFlow version issues

Progress

Today's Progress:

Once we began working through the setup directions, we began to run into a lot of issues with the dependencies having incompatible versions. As Emma is currently using a conda environment with Python3.8 on her Linux device, the first issue she ran into was that Python 3.8 support requires TensorFlow 2.2 or later (she is currently using Tensorflow 2.2), meaning that Tensorflow 2.0 was not available for installation with pip. In order to try to resolve this, Emma created a new conda environment for Python 3, but she still ran into issues with not having the correct version of Tensorflow available in her apt. After doing some research, she read that she would have to downgrade Python again to Python 2.7 in order to be able to install tensorflow==2.0.0, but the other dependencies required her to have at least Python 3. At the same time, Kara had similar issues on her Windows device where she was also struggling to install TensorFlow 2.0.0 as she was running Python 3.9. After attempting to create a conda environment in cmd unsuccessfully, Kara installed Anaconda onto her computer and successfully created a conda environment for Python 3.8 through her Anaconda terminal. By this point, both of us were running into issues installing TensorFlow 2.0.0 with pip, so we did some research and finally were able to successfully install it with the command *conda install tensorflow==2.0.0* instead of with pip.

Evidence of Progress

Research:

- TensorFlow not found Using pip
  - https://stackoverflow.com/questions/38896424/tensorflow-not-found-using-pi
- Installing Conda Packages
  - https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 2/25/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Compile and test accuracy of model

Progress

Today's Progress:

We attempted different neural network layers to provide us with a high validation or testing accuracy. The following code displays our most accurate neural network model.

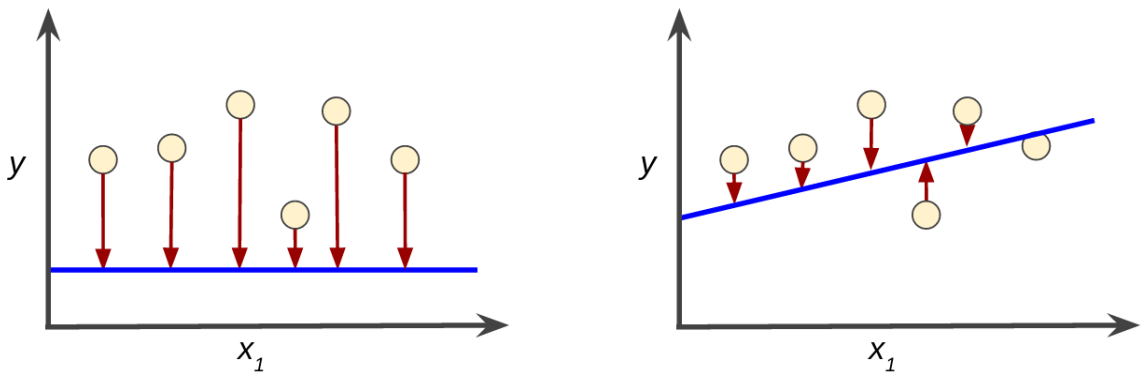
```
batch_size = 128
num_classes = 10
epochs = 20

model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
print("The model has successfully trained")

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

The above code compiles and trains our model on the training datasets with the established epochs and batch_sizes. After the model has trained, the testing datasets will be fed to the model and evaluated for testing loss and testing accuracy.

```
The model has successfully trained
Test loss: 1.2439137697219849
Test accuracy: 0.7865999937057495
Saving the model as mnist.h5
```

We noticed that using only one convolutional layer, one max pooling layer, and 20 epochs provided us with a 78% testing accuracy. This means that when new data is inputted into the model, 78% of the testing data was classified correctly. Testing loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a dataset. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater.



For example, the image to the left shows a high loss model while the image to the right shows a low loss model. The arrows represent loss, and the blue lines represent the model's predictions. Our testing loss was 1.24, which is not an uncommon value. As long as the testing loss decreases as the testing accuracy increases, the model is properly learning how to recognize handwritten digits.

Evidence of Progress

Research:

- Difference between Loss, Accuracy, Validation loss, Validation accuracy in Keras
  - https://www.javacodemonk.com/difference-between-loss-accuracy-validation-loss-validation-accuracy-in-keras-ff358faa
- Descending into ML: Training and Loss | Machine Learning Crash Course
  - https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

Github Repository:

- https://github.com/emascillaro/SeniorProject_Spring

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 3/2/2021

Goals	
<u>Long Term:</u> - Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	**<u>Short Term:</u>** - Improve Log entries to better represent our progress - Create and complete our first project presentation - Gather the necessary research we've completed to discuss in our presentation - Establish our future plans
Progress	

Today's Progress:

Throughout the weekend and the weekdays prior to March 5, we continued to work on our project presentation. This presentation mainly focuses on our overall plans for handwriting recognition and the code we have written so far. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we preprocessed our data, created an OCR model, and calculated the accuracy of our model . We then developed our future plans to show our advisors that we will continue to make progress in our project, such as improving the accuracy of our model and creating a model for handwritten characters. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (6 minutes max) was fairly easy this time, mainly because we focused on just the code we wrote. We were able to present our slides in about 5 minutes while conveying all the necessary information to our advisor and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

First Presentation Link:

- Mascillaro & Pietrowicz Project 3/5/2021

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 3/9/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text	- Watch and critique our first presentation recording

- Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Locate a four-function calculator operator dataset - Consider merging MNIST dataset with math operator dataset - Examine our current ML model

Progress

<u>Today's Progress:</u>

Before continuing our project research, we decided to watch our first presentation recording and take note of what could have been improved to keep in mind for our next presentation (March 25). Overall, the presentation was decent. We organized our slides in a logical order, demonstrated reasonable progress, spoke clearly and loudly, and answered any questions satisfactorily. However, there are some aspects of our presentation that should be addressed and adjusted. While presenting, it is better to speak fluently and not pause in between phrases of our sentences. If we are presenting code, using a textbox to enclose the code we want the audience to focus on might help the audience gain a better understanding of the task our code is performing. Also, whatever is on the slide must be explained; it is not professional to explain one part of our code and ignore the rest. If we are going to ignore some code, then we shouldn't include a picture of that specific code and focus on what's more important.

After critiquing our presentation recording, we began to research a dataset containing basic mathematical operators for addition, subtraction, multiplication, and division. Unfortunately, the Keras library in python does not contain such a dataset. Keras contains the MNIST dataset since it is more prominently used in Machine Learning. Luckily, we located a dataset of JPG files containing the following mathematical symbols: plus sign, minus sign, multiplication sign, division sign, equals sign, and decimal. The number of JPG files for each symbol are not equal, so we will have to either delete or add more files to each symbol so there is an equal number of each symbol. When the number of JPG files for each class (math symbols) are equal, the CNN model will bias towards one class if that class has significantly more samples than other classes.

Most of our time was spent looking for a basic math operator dataset. Before our project period concluded, we considered the following questions:

➔ **Should we merge our MNIST dataset with our math operator dataset?**

Some problems came to mind about merging our two datasets. First, our MNIST dataset is imported from the Keras python library while we had to download JPG files for our math operator dataset. If we want to combine our datasets, we would need to download a MNIST dataset of JPG files. The file type of data inputted into our model doesn't matter; the model can train on any images as long as certain requirements are met for the model's convolutional layers. However, when the model is used to classify a test image (which is separate from training and testing data), the model may not classify the test image if different file types are within the test image's directory. It is not a common problem, but it would be best to avoid this by ensuring that both datasets are of the same file type.

Second, each image in the merged dataset must have the same size and color scheme when presented to the CNN model. The MNIST dataset has white handwritten digits with a black background, but our math operator dataset has black handwritten symbols with a white background. Using the Pillow library in python, we can invert the grayscale values of each image so that we have black handwritten digits with a white background. Next, we need to reshape each image in the combined dataset so they are all the same size. A CNN model cannot train on different sized images.

- ➔ **If we're going to develop a calculator app, is our current CNN model sufficient for our purposes or do we have to make more adjustments (beyond increasing the accuracy of our model)?**

```
The model has successfully trained
Test loss: 1.2439137697219849
Test accuracy: 0.7865999937057495
Saving the model as mnist.h5
```

When new data is inputted into the current model, 78% of the testing data was classified correctly. Testing loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a dataset. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. Of course we need to increase the accuracy of our model in order to better classify random images, but our current model doesn't allow us to input random images for classification. We only used the testing data of MNIST to determine the accuracy of our model. Therefore, it is required that we write python scripts that will allow us to input our own data and use the model to classify our images.

Evidence of Progress

Handwritten Math Symbols Datasets:

- Handwritten math symbols dataset
  - https://www.kaggle.com/xainano/handwrittenmathsymbols
- Data Article Handwritten mathematical symbols dataset
  - https://www.sciencedirect.com/science/article/pii/S2352340916300828
- Handwritten math symbol and digit dataset
  - https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset
- How to invert colors of image with PIL (Python-Imaging)?
  - https://stackoverflow.com/questions/2498875/how-to-invert-colors-of-image-with-pil-python-imaging
- How to combine Keras MNIST dataset with my own MNIST images?
  - https://stackoverflow.com/questions/53253465/how-to-combine-keras-mnist-dataset-with-my-own-mnist-images
- Handwritten math symbol and digit dataset
  - https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset

Combining Datasets:

- Loading in your own data - Deep Learning
  - https://www.youtube.com/watch?v=j-3vuBynnOE
- Python Programming Tutorials
  - https://pythonprogramming.net/convolutional-neural-network-deep-learning-python-tensorflow-keras/

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 3/11/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our	- Locate MNIST dataset with JPG file format - Research how to adjust our model - Begin loading and saving new data

code to solve mathematical problems	

Progress

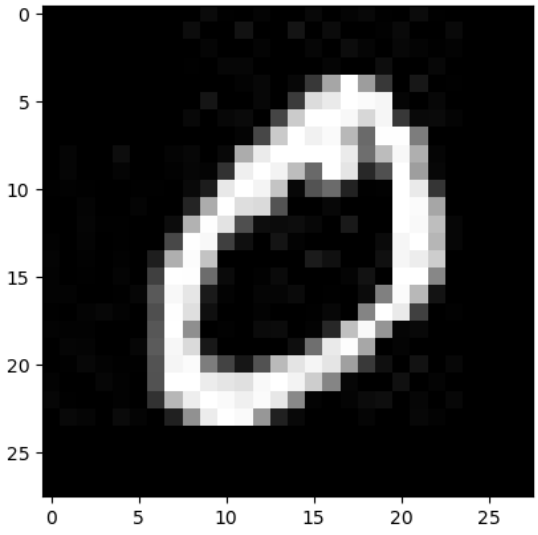
Today's Progress:

Last project block, we discussed if it's necessary to merge our two datasets (MNIST and math operators) and to adjust our CNN model for our future project goals. In general, it is easier and more efficient to train one combined dataset than two different datasets, we just have to be careful about certain properties of our merged dataset, such as image color, image size, image file type, and the frequency of each different image. **We are planning on merging our datasets in the future, but first we need to adjust our model so we can input and classify our own images AND find an MNIST dataset containing JPG files.** It will be easier to write functional code for one dataset instead of a merged dataset. There could be underlying problems with the merged dataset, which would distract us from any issues with our code that need to be addressed. Luckily, we found and downloaded a MNIST JPG dataset online, allowing us to spend more time on writing our python scripts for the CNN model.

Our previous model didn't allow us to merge additional data or test the model on an image separate from the training and testing data. So, we researched and began loading and saving data for our model.

```python
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import random
import pickle
from PIL import ImageOps

# Directories and Categories for MNIST jpg dataset
DATADIR = r"MNIST_Dataset_JPG_format\MNIST_JPG_training"
DATADIR_TEST = r"MNIST_Dataset_JPG_format\MNIST_JPG_testing"
CATEGORIES = ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"]

# Prints one image from training data
for category in CATEGORIES:
  path_train = os.path.join(DATADIR, category)  # path to training data
  for img in os.listdir(path_train):
    img_array = cv2.imread(os.path.join(path_train,img))
    plt.imshow(img_array)
    plt.show()
    break
  break

# Empty lists for training and testing data
training_data = []
testing_data = []
```

First, we noted the paths for our MNIST jpg training and testing files directories and made a list for our class names. We then accessed the path of the training data to read the first image, created an image array of pixel values, and printed the image to see what it looked like and confirm that it was being properly read by our code.



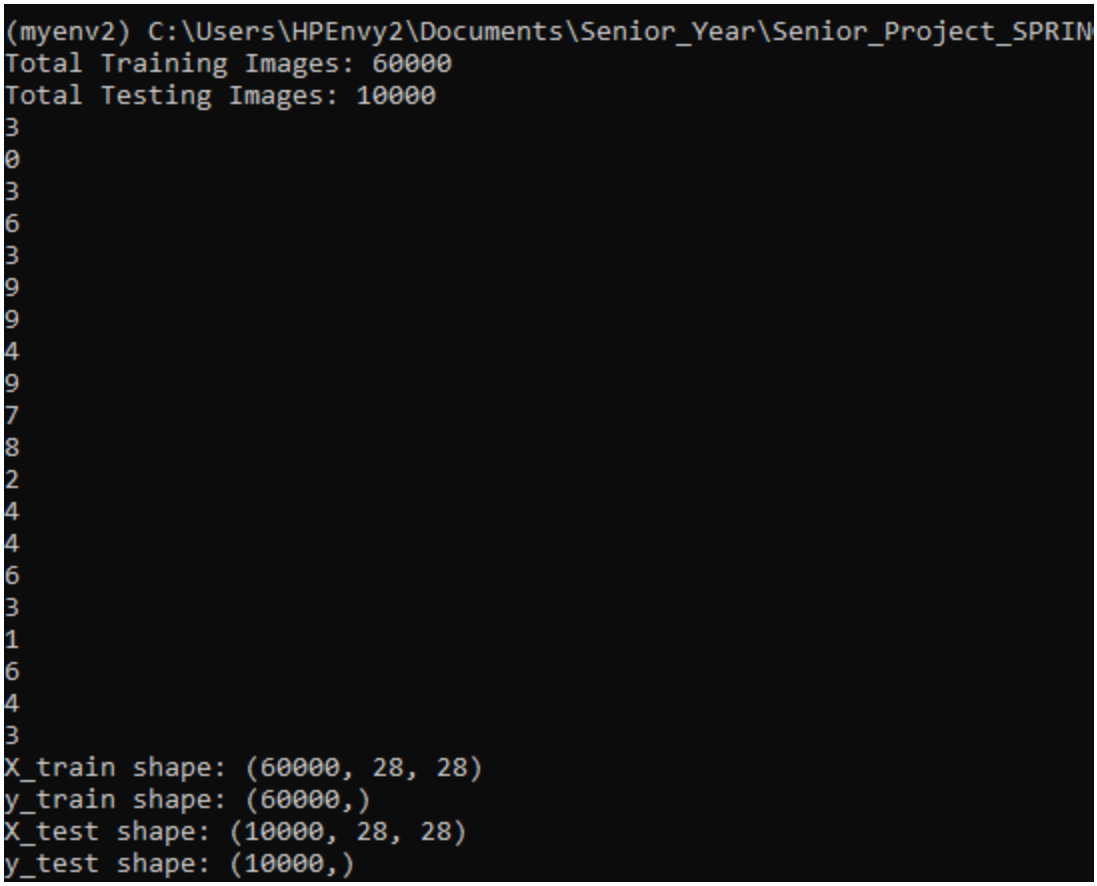
The image is a little blurry but still recognizable as a zero. In order to hold the image pixel arrays of each training and testing image, we created empty training and testing lists.

```
def create_training_data():
    for category in CATEGORIES:
        path_train = os.path.join(DATADIR, category)  # path to training data
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path_train):      #iterate through images in their path, convert to grayscale
          try:
            img_array = cv2.imread(os.path.join(path_train,img), cv2.IMREAD_GRAYSCALE) # array of pixel values from image
            training_data.append([img_array, class_num]) # adds image to training list
          except Exception as e:
            pass

def create_testing_data():
    for category in CATEGORIES:
        path_test = os.path.join(DATADIR_TEST, category)  # path to training data
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path_test):
          try:
            img_array = cv2.imread(os.path.join(path_test,img), cv2.IMREAD_GRAYSCALE)
            testing_data.append([img_array, class_num])
          except Exception as e:
            pass

# Printing the number of images in training and testing data
create_training_data()
print("Total Training Images:", len(training_data))
create_testing_data()
print("Total Testing Images:", len(testing_data))
```

In the above code, we created two functions to iterate through the training and testing images in their paths. We converted the images to grayscale since it speeds up our runtime and color is not a factor in classifying numbers (and symbols for the future) in our dataset. Grayscale images are sufficient, so there is no need to use more complicated and harder-to-process color images. We then created an image array of pixel values for each image and appended the array in either the training or testing list. We implemented a try/except statement to prevent corrupted files from causing errors.

```
#Shuffling training and testing data for proper balance since we iterated over categories
random.shuffle(training_data)
for sample in training_data[:10]:
  print(sample[1])

random.shuffle(testing_data)
for sample in testing_data[:10]:
  print(sample[1])

# Empty training and testing images and classes
X_train = []
y_train = []
X_test = []
y_test = []
#y = np.array(y)

# Building image and class lists
for features, label in training_data:
  X_train.append(features)
  y_train.append(label)
  #np.array((y,label))

for features, label in testing_data:
  X_test.append(features)
  y_test.append(label)
  #np.array((y,label))

# Convert lists into arrays for model
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

After storing the training and testing data, we had to randomize the image pixel arrays inside the lists. Since we iterated through the images by their categories, the class order in the lists were not properly balanced or randomized. We stored the training and testing images and classes within empty lists and converted the lists to arrays for our model to use. The following shapes of each array are shown below.



The total number of training and testing images stored in the lists is presented first. Next, we printed the first ten randomized training and testing image classes. Finally, the training and testing image and class array shapes are shown. For example, there are 60,000 training images within X_train, each reshaped to 28 x 28 pixels. Y_train and y_test are only labels, so the arrays are only one dimension (number of labels).

```
# Training Data Saved and Loaded
pickle_out = open("X_train.pickle", "wb")
pickle.dump(X_train, pickle_out)
pickle_out.close()

pickle_out = open("y_train.pickle", "wb")
pickle.dump(y_train, pickle_out)
pickle_out.close()

pickle_in = open("X_train.pickle", "rb")
X_train = pickle.load(pickle_in)

# Testing Data Saved and Loaded
pickle_out = open("X_test.pickle", "wb")
pickle.dump(X_test, pickle_out)
pickle_out.close()

pickle_out = open("y_test.pickle", "wb")
pickle.dump(y_test, pickle_out)
pickle_out.close()

pickle_in = open("X_test.pickle", "rb")
X_test = pickle.load(pickle_in)

print("Training and Testing Datasets were successfully saved and loaded")
```

Finally, as we know that we will have to go through trial and error with adjusting our model in the future, we used *pickle* to save our data so we don't have to load and compile our data structuring each time.

Evidence of Progress

Research:

- Loading in your own data - Deep Learning
  - https://www.youtube.com/watch?v=j-3vuBynnOE
- Keras - Save and Load Your Deep Learning Models
  - https://www.pyimagesearch.com/2018/12/10/keras-save-and-load-your-deep-learning-models/
- Convert mnist binary on keras datasets to jpeg images.
  - https://gist.github.com/uchidama/40f3076283e485e75bc904193ede6929
- Handwritten math symbol and digit dataset
  - https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset
- Python | os.path.join() method

- ➢ https://www.geeksforgeeks.org/python-os-path-join-method/
- ❖ How to use pickle to save and load variables in Python
  - ➢ https://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/

Github Repository:

- ❖ https://github.com/emascillaro/SeniorProject_Spring

Signatures	
 ***Emma Mascillaro***	 ***Kara Pietrowicz***

Date: Tuesday, 3/16/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Trained and adjusted machine learning model - Research how we use the model to classify our own digit image for next project block

Progress

Today's Progress:

Last period, we successfully loaded and saved the training and testing MNIST jpg datasets. Most of the errors we encountered were trivial. Usually, we had trouble remembering the parameters for the methods used from different python libraries. But, in general, we understood the flow of the code and any compiling errors. Our next python script adjusted our model for better accuracy and saved the model so it can be used later.

```python
import tensorflow as tf;
import numpy as np
import pickle
import random
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import LeakyReLU


X_train = pickle.load(open("X_train.pickle", "rb"))
y_train = pickle.load(open("y_train.pickle", "rb"))
X_test = pickle.load(open("X_test.pickle", "rb"))
y_test = pickle.load(open("y_test.pickle", "rb"))

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)

# Normalizing the data
X_train=X_train/255.0
X_test=X_test/255.0
```

```
X_train shape: (60000, 28, 28, 1)
y_train shape: (60000,)
X_test shape: (10000, 28, 28, 1)
y_test shape: (10000,)
```

In the previous python script, we loaded and saved the training and testing data. In the model script, we loaded the training and testing data for the model to use. Next, we reshaped the matrices that contained the training and testing images since neural network models require four dimensions in order to function properly. The 1 in X_train and X_test shapes indicates that the images are grayscale instead of RGB. Lastly, we divided each number in the X_train and X_test matrices by 255 to normalize our data.

Machine learning algorithms tend to perform better or converge faster when the different features (or in our case numbers) are on a smaller scale. Therefore it is common practice to normalize the data before training machine learning models on it. The images of the MNIST dataset are grayscale and the pixels range between 0 and 255 including both bounding values. Every line in the training and testing dataset consists of a handwritten digit image, or 785

numbers between 0 and 255. The first number of each line is the label, i.e. the digit which is depicted in the image. The following 784 numbers are the pixels of the 28 x 28 image with color range between 0 and 255, as shown below. By decreasing our feature scale to 0 to 1 instead of leaving it as 0 to 255, our CNN model will perform better and faster.

```
31  # OUR model
32  batch_size = 128
33  num_classes = 10
34  epochs = 5
35
36  model = Sequential()
37  model.add(Conv2D(32, kernel_size=(5, 5),input_shape=(28,28,1)))
38  model.add(LeakyReLU(alpha=0.3))
39  model.add(MaxPooling2D(pool_size=(2, 2)))
40  model.add(Flatten())
41  model.add(Dense(128))
42  model.add(LeakyReLU(alpha=0.3))
43  model.add(Dropout(0.3))
44  model.add(Dense(64))
45  model.add(LeakyReLU(alpha=0.3))
46  model.add(Dropout(0.5))
47  model.add(Dense(num_classes, activation='softmax'))
```

The above code displays our **NEW** stacked neural network model that we trained to classify handwritten digit images with our MNIST training data. Neural networks are defined in Keras as a sequence of layers. The container for these layers is the Sequential class. So we created an instance of the Sequential class. Then we created our layers and added them to the neural network in the order that they should be connected. Num_classess represents the number of classifications for the dataset (0-9 digit labels means that there are 10 classifications). An epoch is the number times that the learning algorithm will work through the entire training dataset. It can consist of one or more batches.

Batch_size denotes the subset size of our training dataset (128 out of 60,000). Each batch trains the model in a successive order, taking into account the updated weights coming from the appliance of the previous batch.

The main difference between our new and old model is that we decided to use LeakyReLU as our activation function in the new model instead of ReLU(used in the old model). Rectified Linear Unit, or ReLU, is one of the most common activation functions used in neural networks.

It is added to layers in neural networks to add nonlinearity, which is required to handle more complex and nonlinear datasets.

Without activation functions like ReLU, each neuron in the neural network computes a dot product and adds a bias value before the output is sent to the neurons in the subsequent layer. These mathematical operations are linear in nature. This is not bad if we were training the model against a dataset that is linearly separable.

However, if data is nonlinear, linear neuron outputs ensure that the system as a whole, or the entire neural network, behaves linearly. By consequence, the model cannot handle nonlinear data, such as the MNIST dataset. Activation functions add nonlinearity when placed directly after the neural outputs. Because the mathematical functions used are nonlinear, the output is nonlinear, allowing the system to support nonlinear data.

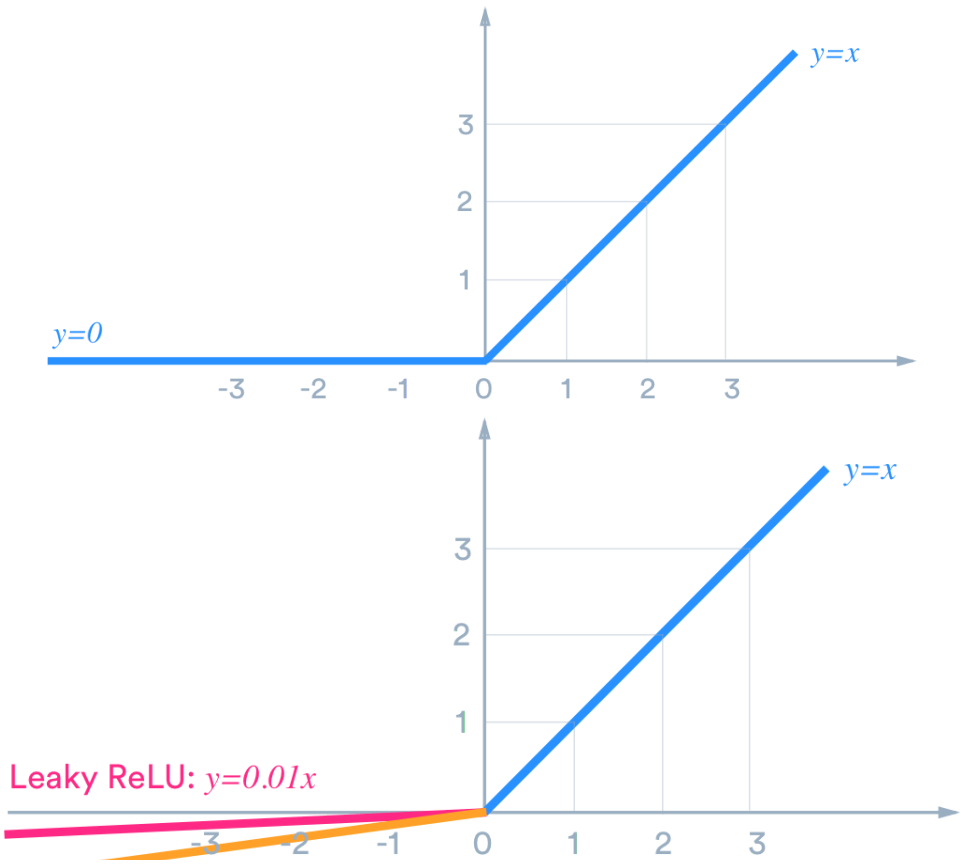
Unfortunately, we learned about the “dying ReLU” problem. ReLU sets the outputs of neurons that are less than zero to zero.

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

This is a problem when neuron outputs arriving at large negative values cannot recover from being stuck at 0. The neuron effectively dies and hence the problem is known as the “dying ReLU problem”. The neurons are especially vulnerable to it when they are not initialized properly or when the data is not normalized very well, causing significant weight swings during the first phases of optimizing the CNN model. Eventually, the neural network essentially stops learning and underperforms.

This is the premise behind Leaky ReLU, one of the possible newer activation functions that attempts to minimize one’s sensitivity to the dying ReLU problem.

$$f(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

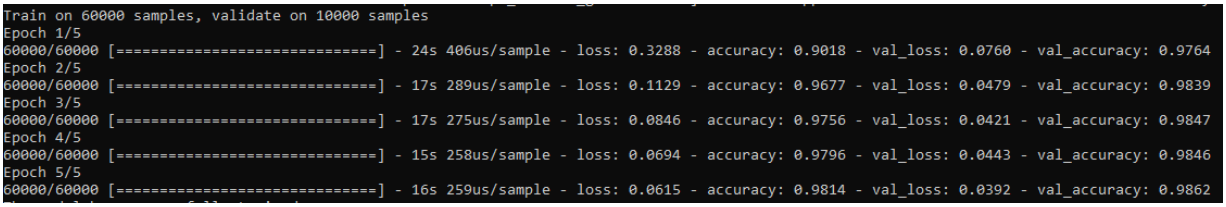


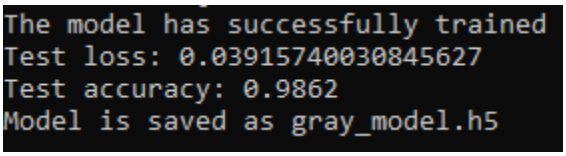
If compared to the traditional ReLU (top graph), for all inputs less than zero in LeakyReLU (bottom graph), the neuron outputs (y-axis) are slightly descending instead of being valued at exactly zero. These small numbers reduce the death of ReLU activated neurons. This way, we can worry less about the initialization of our neural network and the normalization of our data. Although these topics remain important, they are slightly less critical.

```
75   model.compile(loss="sparse_categorical_crossentropy", optimizer= 'adam', metrics=['accuracy'])
76
77   hist = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(X_test, y_test))
78   print("The model has successfully trained")
79
80   score = model.evaluate(X_test, y_test, verbose=0)
81   print('Test loss:', score[0])
82   print('Test accuracy:', score[1])
83
84   model.save('gray_model.h5')
85   print("Model is saved as gray_model.h5")
```

The above code compiles and trains our model on the training datasets with the established epochs and batch_sizes. After the model has trained, the testing datasets will be fed to the model and evaluated for testing loss and testing accuracy. We then saved the model as gray_model.h5 in order to access it without having to recompile when testing new data. H5

files are open-source files which come in handy to store large amounts of numerical data that can also be easily manipulated by Numpy.





In the above two pictures, we compiled our model within 5 epochs and produced a 98.62% test accuracy. This means that when new data is inputted into the model, 98.62% of the testing data was classified correctly. Testing loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a dataset. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. Our testing loss was 0.04, which is definitely better than our previous loss value (1.24). As long as the testing loss decreases as the testing accuracy increases, the model is properly learning how to recognize handwritten digits.

The next step for us is to feed our own handwritten digit image into our model and see if it can be classified correctly. In order to do this, we will have to convert the image into a grayscale pixel array that the model can use to predict the image's class.

Evidence of Progress

Research:

- ❖ in train_validation_split raise ValueError(ValueError: `validation_split` is only supported for Tensors or NumPy arrays, found: (array([[[[0.24705882] · Issue #38613 · tensorflow/tensorflow
  - ➢ https://github.com/tensorflow/tensorflow/issues/38613
- ❖ Leaky ReLU: improving traditional ReLU – MachineCurve
  - ➢ https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/
- ❖ Tensorflow model.fit "use_multiprocessing" "distribution_strategy" "adapter_cls" "failed to find data adapter that can handle" · Issue #35651 · tensorflow/tensorflow
  - ➢ https://github.com/tensorflow/tensorflow/issues/35651
- ❖ How to use your trained model - Deep Learning

- ➢ https://www.youtube.com/watch?v=A4K6D_gx2Iw
- ❖ Handwritten math symbol and digit dataset
  - ➢ https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset
- ❖ TensorFlow TypeError: Value passed to parameter input has DataType uint8 not in list of allowed values: float16, float32
  - ➢ https://stackoverflow.com/questions/44822999/tensorflow-typeerror-value-passed-to-parameter-input-has-datatype-uint8-not-in

Github Repository:

- ❖ https://github.com/emascillaro/SeniorProject_Spring

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 3/18/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Testing Machine Learning model on our own handwritten digits

Progress

Today's Progress:

Last period, we wrote and compiled code to train our model on the MNIST dataset. For the most part, the model layers remained the same as our layers from the first presentation cycle with the exception of changing the ReLU activation function to LeakyReLU. Today, our goal is to use our pretrained model to recognize our own handwritten digits that we feed into it.

```python
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from keras.models import load_model
import os
from keras.preprocessing import image
from PIL import Image, ImageOps

CATEGORIES = ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"]
PATH = r"jpg_photos"
```

First, we created a list with the name labels of each of the categories of data in our dataset and saved it in the CATEGORIES. We also noted the path to our directory with our test images in PATH.

```python
# Formatting image for model
def prepare(path):
    try:
        IMG_SIZE = 28
        img = Image.open(path)
        img = ImageOps.grayscale(img)
        print(img.size)
        resized_img = img.resize((IMG_SIZE, IMG_SIZE))
        resized_img.save('8_resized.jpg')        # save the image as something else, not the name of the original photo
        resized_img.show()
        print(resized_img.size)
        img_array = cv2.imread('8_resized.jpg')
        new_array = img_array.astype('float32')
        new_array = new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
        # new_array = new_array / 255.0
        return new_array
    except Exception as e:
        print(str(e))
```

Next, we wrote a *prepare* function in order to format our images to feed into the model. In order to get the best accuracy, we want our images to have the same color and dimensions as the training images, so we converted our testing images to grayscale and resized them to be 28x28 pixels. Then we created and reshaped a pixel array of the image for the model.

```python
model = tf.keras.models.load_model('gray_model.h5', compile = True)
os.chdir("jpg_photos")
os.system("cd")
prediction = model.predict([prepare('8.jpg')])      #always pass a list
print(CATEGORIES[int(prediction[0][0])])
```

As you can see in the above picture, we then loaded our model that we saved last class, navigated into the directory of our testing images, and made a prediction on our new image after it was processed through the prepare function. In the end, we printed our prediction. For some values, we got the correct prediction, however, we did run into a number of incorrect predictions as well. After we fed the handwritten eight to our model, the model predicted its class to be one. We believe that this might stem from our model layers, photo

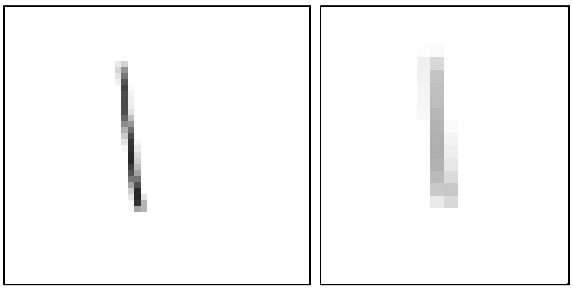
resolution, epochs, batch sizes, and data randomization, which we are currently working on improving.

Picture of "8" and Result:

Original Reshaped

```
Orignal Image: (91, 76)
Resized Grayscale Image: (28, 28)
One
```

Picture of "1" and Result:



Original Reshaped

```
Orignal Image: (79, 58)
Resized Grayscale Image: (28, 28)
One
```

Evidence of Progress

Research:

- Python PIL | ImageOps.grayscale() method
  - https://www.geeksforgeeks.org/python-pil-imageops-greyscale-method/
- OpenCV: Grayscale image displayed in greenish color
  - https://stackoverflow.com/questions/47782311/opencv-grayscale-image-displayed-in-greenish-color
- Image Resizing with Python in Scale: Ultimate Guideline

- ➢ https://www.holisticseo.digital/python-seo/resize-image/
- ❖ Python Image Processing in Python with Pillow
  - ➢ https://auth0.com/blog/image-processing-in-python-with-pillow/
- ❖ Negative transformation of an image using Python and OpenCV
  - ➢ https://www.geeksforgeeks.org/negative-transformation-of-an-image-using-python-and-opencv/
- ❖ I am having trouble with this error (-215:Assertion failed) !ssize.empty() in function 'resize' in opencv
  - ➢ https://stackoverflow.com/questions/52162004/i-am-having-trouble-with-this-error-215assertion-failed-ssize-empty-in-fu
- ❖ Python os.listdir() Method
  - ➢ https://www.tutorialspoint.com/python/os_listdir.htm

Github Repository:

- ❖ https://github.com/emascillaro/SeniorProject_Spring

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 3/23/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Improve Log entries to better represent our progress - Create and complete our second project presentation - Gather the necessary research we've completed to discuss in our presentation - Establish our future plans

Progress

Today's Progress:

Throughout the weekend and the weekdays prior to March 25, we continued to work on our project presentation. This presentation mainly focuses on our new code we have written so far for loading, training, and testing our dataset and new model. We also wrote code so we can feed our own images to our new model and predict the class of the image. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we preprocessed our data, built and calculated the accuracy of our model, and tested new images on our trained model. We then developed our future plans to show our advisors that we will continue to make progress in our project, such as merging symbols into our dataset and performing mathematical operations. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (5 minutes max) was more difficult this time, mainly because we had more code to explain in simple yet sufficient details. We were able to present our slides in about 5 ½ minutes while conveying all the necessary information to our advisor and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

Second Presentation Link:

- Mascillaro & Pietrowicz Project 3/25/2021

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 4/6/2021

Goals

Long Term:

- Write handwriting recognition code to identify and transcribe written text
- Write four-function calculator scripts
- Develop an app that utilizes all our code to solve mathematical problems

Short Term:

- Consider and test possible cause of our incorrect model predictions for different images
- Think ahead and research how our model will predict and print out a given handwritten mathematical expression

Progress

Today's Progress:

Last project cycle, we focused on writing new code for loading, training, and testing our dataset and newly developed model. We also wrote code so we can feed our own images to our new model and predict the class of the image. For some values, we got the correct prediction, however, we did run into a number of incorrect predictions as well. After we fed the handwritten eight to our model, the model predicted its class to be one. We believe that this might stem from our model layers, photo resolution, epochs, batch sizes, and data randomization.

Picture of "8" and Result:

Original Reshaped

```
Orignal Image: (91, 76)
Resized Grayscale Image: (28, 28)
One
```

Picture of "1" and Result:

Original Reshaped

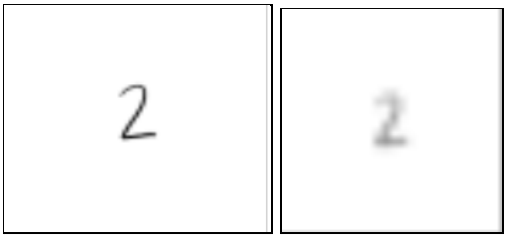
```
Orignal Image: (79, 58)
Resized Grayscale Image: (28, 28)
One
```

We began to adjust our model using simple solutions, such as changing the epochs, batch sizes, and randomizing our data more. Unfortunately, these changes didn't fix our problem. If the model is incorrectly predicting the class of an image, then most likely we have a problem with the model layers or the photo itself. Our model produced about 98% accuracy, which means it correctly classified 98% of the 10,000 provided testing images after the model was trained on 60,000 training images. Due to this high accuracy, we don't believe that our model is the main issue. We attempted to improve the photo resolution of the reshaped images using the Pillow python library. Both project members are not familiar or well-exposed to this image library, so the proper research for the various methods of this library was conducted.

In order to improve the photo resolution of our reshaped images, we attempted to add a quality parameter when we were saving our reshaped image. The image quality is a scale that measures and stores the resolution of an image. It lies in the range of 0 to 100 where 95 is considered the best because 100 can disable some portions of jpeg compression algorithms that result in very large files.

```
resized_img.save('2_resized.jpg', quality=95)
```

Picture of "2" and Result:

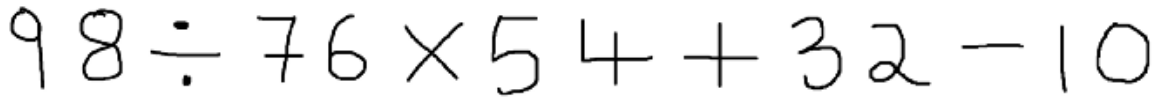


Original Reshaped

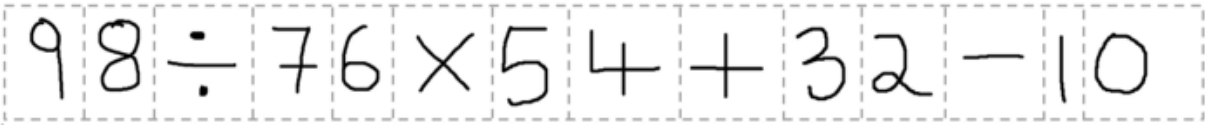
However, from looking at the images of handwritten “2”, the quality parameter did not drastically improve the resolution of the reshaped image. Even though the resolution is still poor, the reshaped image is recognizable as the number 2. So, photo resolution may not be causing our incorrect model prediction issue. Therefore, we will have to continue researching possible causes of our problem.

After spending some time attempting to solve our problem, we decided to take a break from tweaking our ML code and think about our future plans. The next step for the ML component of our project would be to research how our model will predict and print out a given handwritten mathematical expression and adjust the model accordingly.

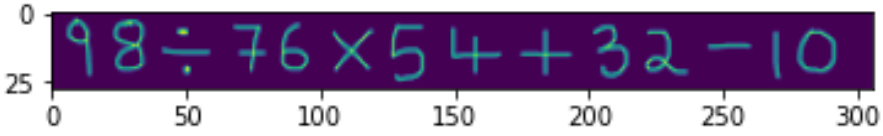
In order for our model to analyze and correctly predict a handwritten mathematical expression, it first needs to be broken down into individual elements that can be identified by the model. Take a look at the following expression.



Before this image is sent to the model, we need to break it down into images of digits and mathematical operations, as shown below.

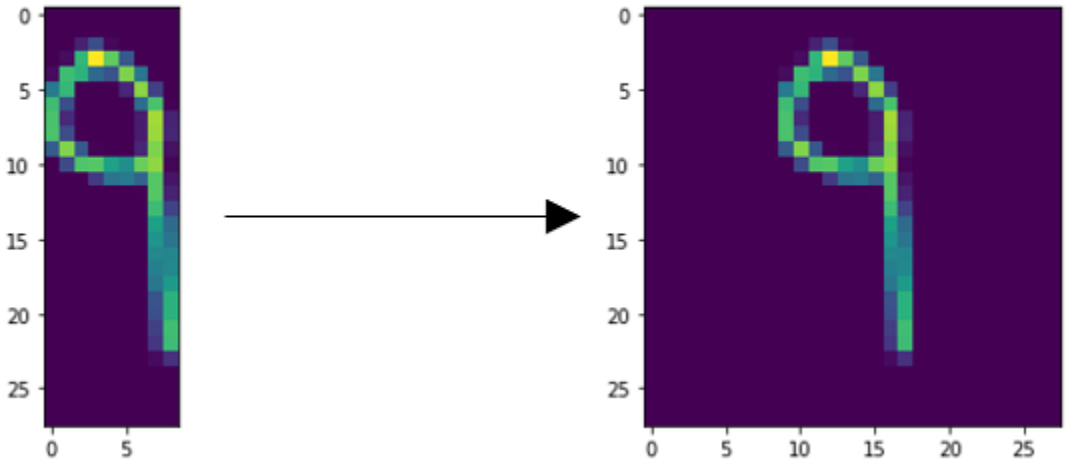


First, the image should be converted to grayscale using Pillow. Then the image will be resized to a height of 28 pixels to keep the same aspect ratio of the model input requirement (28 x 28). The width will be adjusted accordingly. Next, the image will be normalized and converted from an image to a numpy array. The following is the numpy array displayed as the tweaked image.

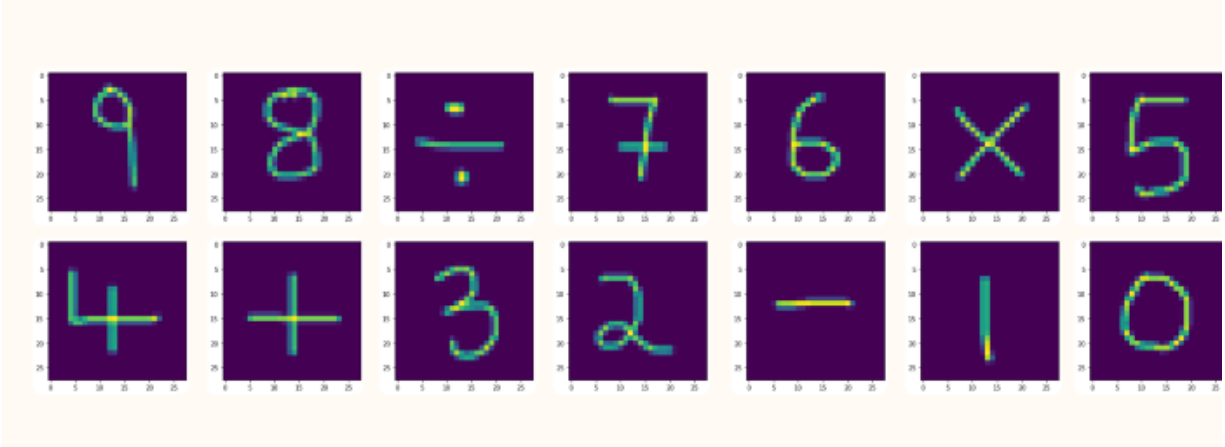


Now the image array will be split into individual element arrays. This is done by searching the image array for successive non zero columns and grouping them to form one element array. These element arrays are all stored in one list. Since there are 14 elements in the mathematical expression, the size of the list is 14.

In order to identify the element using the model, it is important to have the image size of 28 x 28 pixels. The height is already 28 pixels. However, due to the splitting process, the width of each element will not be exactly 28 pixels. Therefore, after the splitting, the width of each element needs to be adjusted to 28 pixels. by adding zero value columns (filler columns) to the element arrays.



This process will be repeated for all mathematical elements in the expression.



Finally, the elements list is converted back into an array of shape (14, 28, 28, 1) to fit the model input requirements of a CNN model [mini_batch_size, height, width, color depth]. The elements array is passed through the model for prediction. The model returns the probabilities of all the number classes (purpose of Softmax function). The class with the highest probability is chosen and assigned to each of the 14 images.

Evidence of Progress

Research:

- ❖ Python PIL | ImageOps.grayscale() method

- ➢ https://www.geeksforgeeks.org/python-pil-imageops-greyscale-method/
- ❖ OpenCV: Grayscale image displayed in greenish color
  - ➢ https://stackoverflow.com/questions/47782311/opencv-grayscale-image-displayed-in-greenish-color
- ❖ Image Resizing with Python in Scale: Ultimate Guideline
  - ➢ https://www.holisticseo.digital/python-seo/resize-image/
- ❖ Python Image Processing in Python with Pillow
  - ➢ https://auth0.com/blog/image-processing-in-python-with-pillow/
- ❖ 2 Replies to “Artificial Intelligence-Based Handwriting Recognition”
  - ➢ https://www.eeweb.com/artificial-intelligence-based-handwriting-recognition/
- ❖ Building a Handwritten Multi-Digit Calculator | by Neerav Gala
  - ➢ https://towardsdatascience.com/building-a-handwritten-multi-digit-calculator-f03cf5028052
- ❖ Python PIL, preserve quality when resizing and saving
  - ➢ https://stackoverflow.com/questions/19388290/python-pil-preserve-quality-when-resizing-and-saving
- ❖ Change image resolution using Pillow in Python
  - ➢ https://www.geeksforgeeks.org/change-image-resolution-using-pillow-in-python/

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 4/8/2021

Goals	
Long Term: - Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	**Short Term:** - Research how to create four-function calculator scripts for ML model to utilize - Write and test four function calculator scripts

Progress

Today's Progress:

For the rest of our project cycle, we decided to focus on our other long term goals and temporarily set aside our ML model.

Our process going about machine learning is filled with trial and error. As all engineers/programmers do, we have to constantly tweak and adjust our algorithms and models. During this process, challenges arise, especially with handling data. When constructing a ML model, it was important for us to remember that real-world data is imperfect, various types of data require different approaches and tools, and there will always be tradeoffs when determining the proper model. It requires creativity, experimentation, and tenacity. Debugging ML models occurs in two cases:

- Our algorithm doesn't work
- Our algorithm doesn't work well *enough*

What's unique about machine learning is that it's 'exponentially' harder to figure out what is wrong when our algorithm doesn't work as expected. Very rarely does an algorithm work the first attempt, so the majority of time spent developing a machine learning model is dedicated to tweaking and sometimes rebuilding the algorithm.

We haven't really worked with much real-world data yet. New images of neatly written digits have been passed to our model for a prediction, but we haven't completely resolved our issue. During the last project cycle, most project blocks were spent constantly adjusting code in order for the model to predict the class of a newly inputted image.

During this project cycle, we don't want to waste too much time on trial and error with our ML model and postpone making progress on the other aspects of our project for it. We WILL fix our ML problem in the future, but we believe we are at a good point to set aside the ML component of our project and begin working on the calculator scripts and app development so we can get an idea of how to incorporate our code with these final project components. We can make some decent progress on the non-ML components since they can be developed and tested partially without our model.

For our calculator scripts, we began by assuming that we would be able to read each character that the user writes and input them into a string. In the case that we end with an array or some other form of data as an output from our ML model, we plan to convert those to a string (the most likely case is that we will have an array that we have to iterate through with a for-each loop in order to build our string). For now, we predefined our string at the

start of the program, but once we improve our ML model, we will import a string from that program instead. For this example, we decided to use an exponential expression to run through.

```
5    equationString = "253^3"
```

Through a set of if and elif statements, we searched our string to determine the type of expression that we were working with as well as the position of the operator within the string.

```
23   elif '^' in equationString:
24       location = equationString.index('^')
25       symbol = "^"
```

With the location of the operator, we went about determining the two numbers present in the expression by creating a substring of all the characters before the operator and another substring of all the characters after the operator. We then cast these two substrings to integers and saved these integers as the numbers we would use to perform the expression calculation.

```
27   #First Number
28   string_numA = equationString[0:location]
29   int_numA = int(string_numA)
30
31   #Second Number
32   string_numB = equationString[location + 1:]
33   int_numB = int(string_numB)
```

Next, we created functions to perform the necessary calculations by taking the input of the first and second numbers as parameters and returning the calculation.

```
48   def exponent (numA, numB):
49       return numA ** numB
```

Finally, dependent on the symbol recognized in the expression, we called and printed the returned value of the corresponding function.

```
63    if symbol == '^':
64        print(exponent(int_numA, int_numB))
```

When we run our program in our terminal, we can see that we have correctly performed and printed the calculation.

```
C:\Users\Family\Documents\Emma\High School\Senior\Projects\Code>python Calc_Scripts.py
16194277
```

Evidence of Progress

Research:

- Building a Handwritten Multi-Digit Calculator | by Neerav Gala
  - https://towardsdatascience.com/building-a-handwritten-multi-digit-calculator-f03cf5028052
- Handwritten Equation Code Explained
  - https://github.com/vipul79321/Handwritten-Equation-Solver/blob/master/CNN%20test.ipynb
- Python String split() Method
  - https://www.w3schools.com/python/ref_string_split.asp
- CONDA CHEAT SHEET
  - https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 4/13/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text	- Begin exploring app development - Research different types of user

- Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	interfaces we can employ - Decide on which user interface to use

Progress

Today's Progress:

After completing our scripts for some basic calculator functions (addition, subtraction, multiplication, division, and exponents), we decided to move on to learning how to develop our user interface today. Initially, we were planning on developing an iOS app, but after doing some research, we decided that we should explore the pros and cons of using other means such as building a website or creating a GUI with the Python library Tkinter.

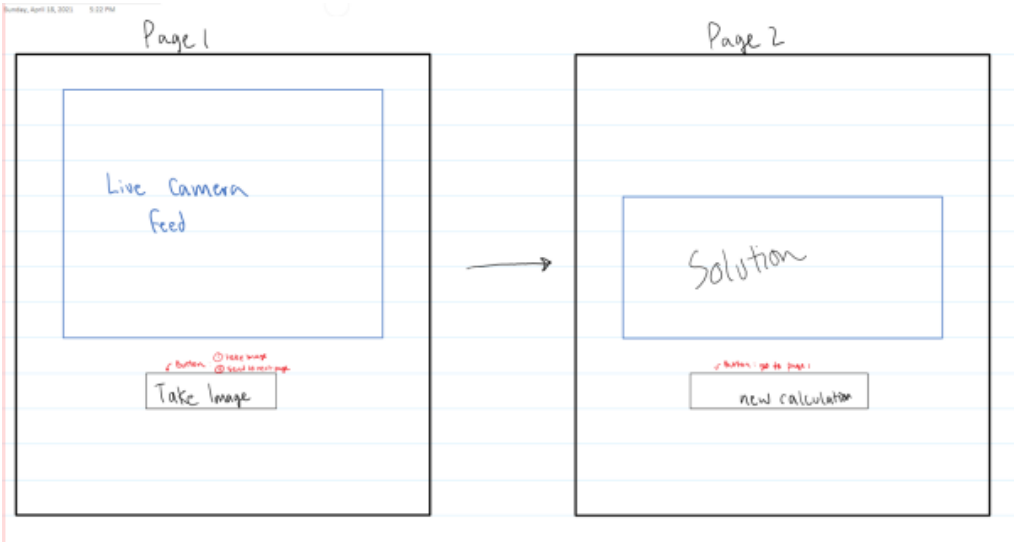
If we were to create an IOS app, we would begin by using Xcode where you design the user interface for your app and write your code that gets turned into an iPhone app which you can publish to the App Store. Xcode is a program you can download for developers to create IOS applications for Mac, iPhone, iPad, Apple TV, and Apple Watch. Within Xcode, we would write our IOS app using Swift, a powerful and intuitive programming language for macOS, iOS, watchOS, tvOS and beyond. The syntax is concise yet expressive, and Swift includes modern features for more advanced applications, such as those involving high quality graphics. When using Swift in Xcode, there will be multiple options for designing your app. There are tabs open for the code editor, customization of the current screen for your app, storage of images to be used in the app, and customization of the app's launch screen.

After watching a couple videos on IOS app development, we realized that Swift and Xcode are more focused on the appearance of an app. Of course, we would focus on writing the code for our app, but when our code would have to change the appearance of the app's launch screen, such as adding a button, sometimes there can be errors with the app's appearance. IOS app development is important to learn, but for the moment, we want to focus on creating a very basic app that can incorporate our python code. If we have time, we might look into using Swift after completing our long term goals. But for the moment, using an IOS platform does not seem like the right choice for our project purpose, especially since we both don't own a functioning Mac and IOS simulators for Windows are not the best to work with.

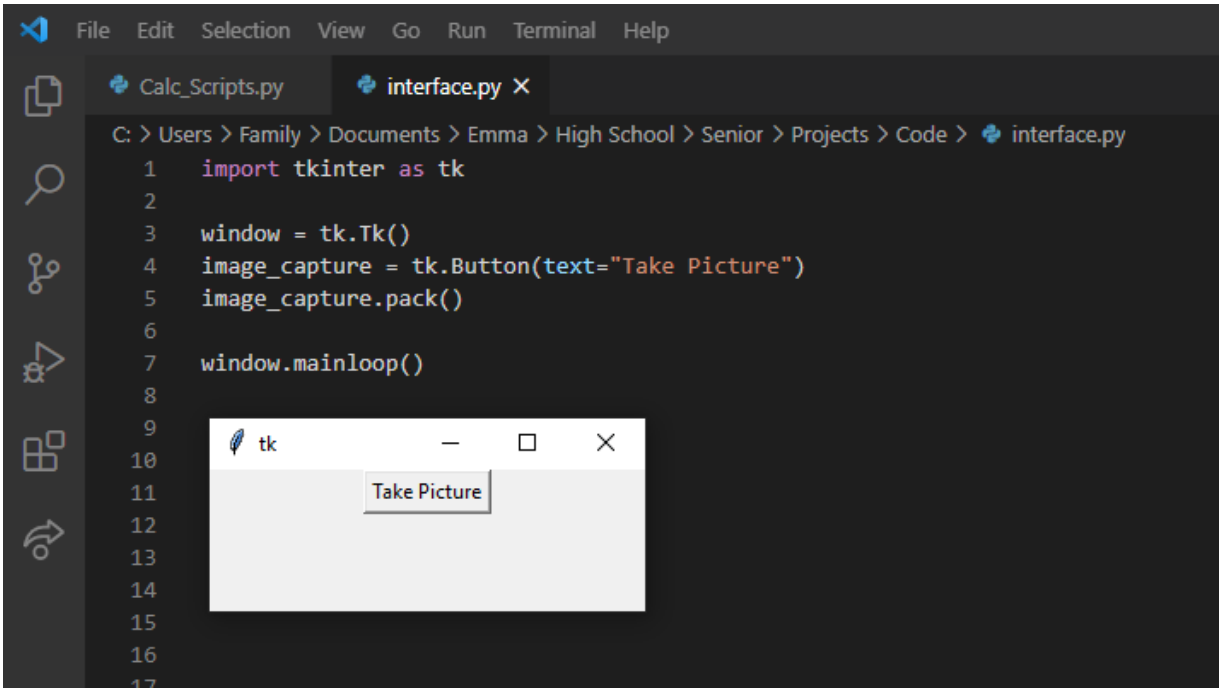
We next considered building a website or exploring how to build a GUI with Python, and we decided that as our project won't need an intricate user interface, the most efficient and practical way for us to build our user interface would be through using Tkinter, a standard Python library for building GUIs.

Some advantages of using tkinter include brevity, cross platform, and simplicity. Python programs using Tkinter can be very brief, partly because of the power of Python, but also due to tkinter. In particular, reasonable default values are defined for many options used in creating a widget, and packing it (i.e., placing and displaying). Second, tkinter provides widgets on Windows, Macs, and most Unix implementations with very little platform-specific dependence. Finally, with very simple syntax, widgets of tkinter are remarkably powerful and very easy to work with. Also, we can continue writing our app in python instead of switching to a new language.

For our app design, we decided that we would need two pages: a page to capture the image and a page to display the solution. On our page to capture the image, we would need to display a live camera feed in order to help users orient their writing with the camera as well as a 'take image' button which will capture a frame from the live camera feed and save it as an image which we can then access from our backend code. On the second page, we would need to have a window for displaying our solution once the backend completes the calculation, and it would also be a nice feature to have a button that allows the user to return to the first page and perform another calculation.



Now that we have decided to use Tkinter to build our user interface, we began exploring tutorials on how to use the package. The first thing that we did was to create a window using the Tk() method and then add in a widget, in this case, a button, as we will be needing a button on in order to allow the user to take a picture. However, Tkinter also has other widget classes such as labels, entries, text, and frames.



We continued to familiarize ourselves with tkinter and its methods that we could use for our app. Next project block, we will use the methods we learned today and more to start constructing our app.

Evidence of Progress

Research:

- Remoted iOS Simulator for Windows - Xamarin
  - https://docs.microsoft.com/en-us/xamarin/tools/ios-simulator/
- How to Make an App for Beginners (2020) - Lesson 1
  - https://www.youtube.com/watch?v=jniJeamcIUU
- Auto Layout Tutorial (2020) - Lesson 2
  - https://www.youtube.com/watch?v=emojd8GFB0o
- Swift
  - https://developer.apple.com/swift/
- Advantages and Disadvantages of Tkinter and wxpython
  - https://www.blog.neudeep.com/howto/advantages-and-disadvantages-of-tkinter-and-wxpython/231/
- 20.1.2. Pros and Cons of Tkinter - Python Programming On Win32 [Book]
  - https://www.oreilly.com/library/view/python-programming-on/1565926218/ch20s01s02.html
- An introduction to Tkinter

➢ https://www.cs.mcgill.ca/~hv/classes/MS/TkinterPres/

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 4/15/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Begin building our chosen user interface - Consider how to incorporate our ML model with our app

Progress

Today's Progress:

After building the basic window last class, we continued to do research on displaying live camera feed inside the window of the user interface and wrote the following code. Here, we first used openCV in order to identify which camera we wanted to use (here we are using an external camera so the parameter is 1, internal camera of laptop would be 0) and then set the dimensions of the frame.

```
17    width, height = 800, 600
18    cap = cv2.VideoCapture(1)
19    cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
20    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
```

Next, we created a window using the Tk() method. We used the .bind() method to bind a Python function (quitting the window) to pressing the escape button. 'Label' is a Tkinter Widget class, which is used to display text or an image, and we utilize it here to give the

window the default name ‘tk’. The pack method tells Tk() to fit the size of the window to the default text ‘tk’.

```
22    root = Tk()
23    root.bind('<Escape>', lambda e: root.quit())
24    lmain = Label(root)
25    lmain.pack()
```

We then wrote the show_frame method in order to display the live camera feed. The first thing that we did was to capture and read the frame. When the image file is read with the OpenCV function .read(), the order of colors is BGR (blue, green, red). On the other hand, in Pillow, the order of colors is assumed to be RGB (red, green, blue). We used cvtColor to convert the color scale of our image from BGR to RGB, allowing us to utilize the pillow to process our image further. Then, we use the .fromarray() to convert the array from openCV to an image object with Pillow. Then, we use ImageTk.PhotoImage to convert the pillow image to an image that is compatible with Tkinkter. Next, we add the Tkinkter compatible image to our labeled window (lmain) and display it. The .after() method simply calls the function show_frame after the given delay in milliseconds (10).

```
28  def show_frame():
29      ret, frame = cap.read()
30      if ret:
31          cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
32          img = PIL.Image.fromarray(cv2image)
33          imgtk = ImageTk.PhotoImage(image=img)
34          lmain.imgtk = imgtk
35          lmain.configure(image=imgtk)
36          lmain.after(10, show_frame)
```

Next, we worked on our function for capturing a still image from a live camera. In OpenCV, to access a camera, we use the method VideoCapture() where the argument inside the parentheses determines which camera we will be using. The computer’s built-in webcam is 0, and an additional camera would be 1. For today, we tested our program with 0, our webcam, but in the future, we would change the camera source to 1. To capture a frame from the camera feed, we used the .read() method which returns a boolean value stating if the frame was captured correctly or not and the still frame which is a numpy array. We then use the cv2.imwrite() method to save our image as a jpg file.

```
36  def capture_image():
37      videoCaptureObject = cv2.VideoCapture(0)
38      ret, frame = videoCaptureObject.read()
39      img = cv2.imwrite("Capture_Image.jpg", frame)
40      videoCaptureObject.release()
```

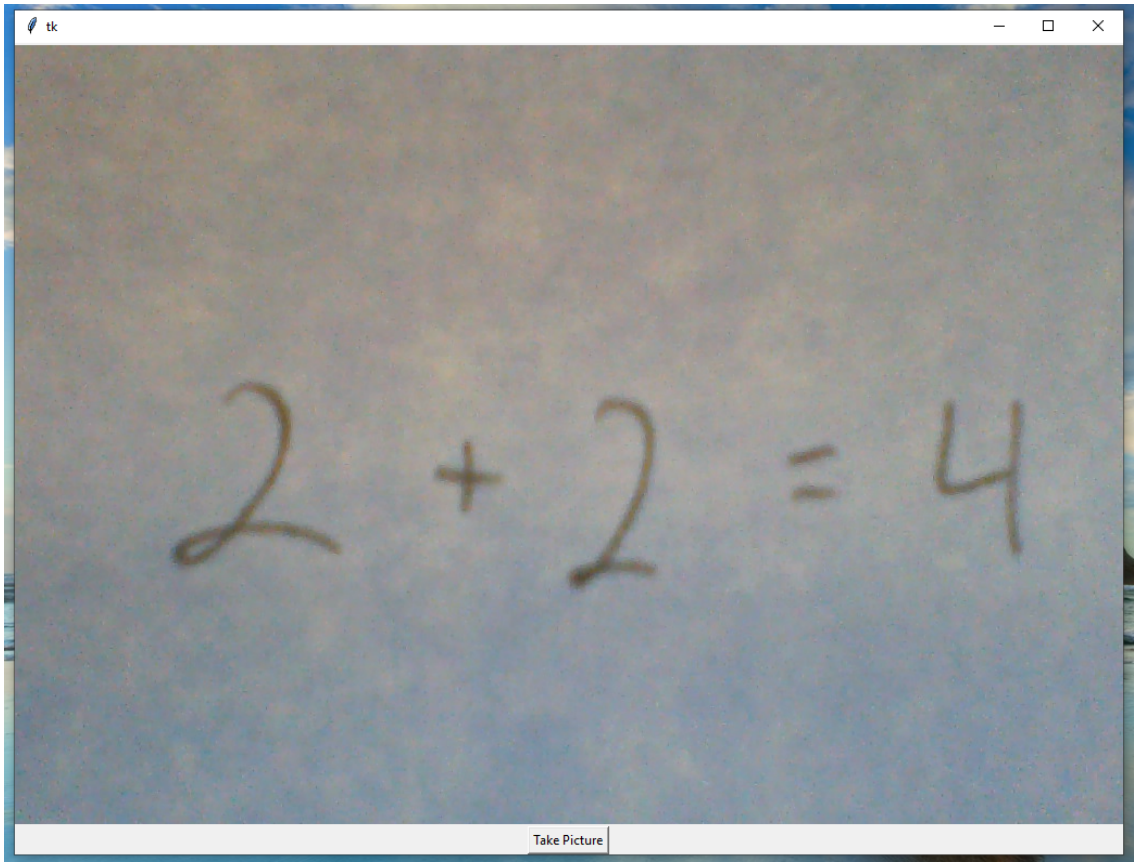
Next, we created a button with the text “Take Picture” which runs the capture_image function when pressed.

```
42  image_capture = tk.Button(text="Take Picture", command = capture_image)
43  image_capture.pack()
```

Finally, we call our show_frame function to display the image and use root.mainloop in order to constantly display and reset the window until the window is closed.

```
37  show_frame()
38  root.mainloop()
```

In the end, our result is as follows: The following displays a handwritten equation image that was taken using our tkinter app that we are currency developing. At the bottom of the image, you can see the “Take Picture” button that allows the user to take a picture of anything using the internal camera of the laptop or a connected external camera.



jpg_photos	3/23/2021 3:06 PM	File folder	
mnist.model	3/21/2021 7:55 PM	File folder	
MNIST_Dataset_JPG_format	3/24/2021 8:30 PM	File folder	
app	4/20/2021 10:50 AM	Python File	2 KB
archive	3/16/2021 10:03 AM	Compressed (zipp...	59,209 KB
calc	4/19/2021 2:53 PM	Python File	2 KB
Capture_Image	4/20/2021 10:36 AM	JPG File	98 KB
gray	3/24/2021 9:10 PM	Python File	4 KB

Evidence of Progress

Research:

- Real Python GUI Programming With Tkinter
  - ➢ https://realpython.com/python-gui-tkinter/
- ValueError: invalid literal for int() with base 10: ''
  - ➢ https://stackoverflow.com/questions/1841565/valueerror-invalid-literal-for-int-with-base-10
- Python String split() Method

- ➢ https://www.w3schools.com/python/ref_string_split.asp
- ❖ To display Webcam in a tkinter display box
  - ➢ https://gist.github.com/aj-ames/ef77f30008faf1cbb85ac90aa3851f11
- ❖ 93 responses to: Displaying a video feed with OpenCV and Tkinter
  - ➢ https://www.pyimagesearch.com/2016/05/30/displaying-a-video-feed-with-opencv-and-tkinter/
- ❖ GUI Programming with Python: Labels in Tkinter
  - ➢ https://www.python-course.eu/tkinter_labels.php
- ❖ tkinter - .after() | tkinter Tutorial
  - ➢ https://riptutorial.com/tkinter/example/22870/-after--
- ❖ Convert BGR and RGB with Python, OpenCV (cvtColor)
  - ➢ https://note.nkmk.me/en/python-opencv-bgr-rgb-cvtcolor/

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 4/20/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Improve Log entries to better represent our progress - Create and complete our third project presentation - Gather the necessary research we've completed to discuss in our presentation - Establish our future plans

Progress

Today's Progress:

Throughout the weekend and the weekdays prior to April 22, we continued to work on our project presentation. This presentation mainly focuses on our new code we have written so far for our calculator scripts and user interface. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we broke down the equation string in order to perform a calculation, accessing and displaying a live camera feed in a Tkinter window, and capturing and saving an image through our user interface. We then developed our future plans to show our advisors that we will continue to make progress in our project, such as going back to improving our machine learning model's prediction, completing our user interface, and upgrading our calculator scripts. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (5 minutes max) was more difficult this time, mainly because we had more code to explain in simple yet sufficient details. We were able to present our slides in about 5 ½ minutes while conveying all the necessary information to our advisor and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

Third Presentation Link:

- Mascillaro & Pietrowicz Project 4/22/2021

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 4/27/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to	- Research and test possible cause of

identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	our incorrect ML model predictions for different images

Progress

<u>Today's Progress:</u>

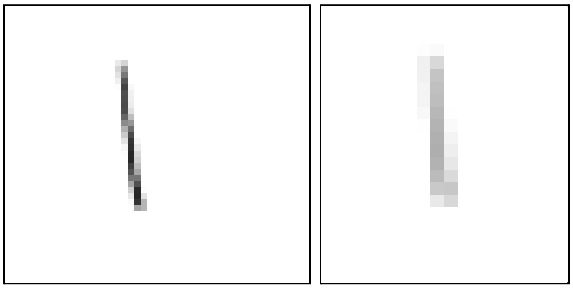
During today's project blocks, we decided to revisit our improper classification issue with our ML model. As a reminder, for some handwritten digit images we got the correct prediction. However, we did run into a number of incorrect predictions as well. After we fed the handwritten eight to our model, the model predicted its class to be one. We believed that this stemmed from our model layers, photo resolution, epochs, batch sizes, and data randomization.

Picture of "8" and Result:

Original Reshaped

```
Orignal Image: (91, 76)
Resized Grayscale Image: (28, 28)
One
```

Picture of "1" and Result:



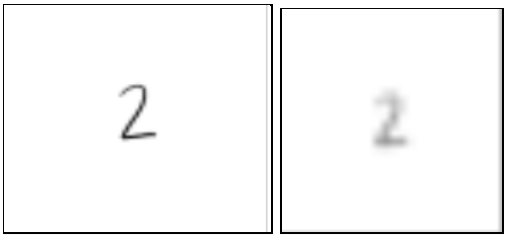
Original Reshaped

```
Orignal Image: (79, 58)
Resized Grayscale Image: (28, 28)
One
```

We adjusted our model using simple solutions, such as changing the epochs, batch sizes, and randomizing our data more. Unfortunately, these changes didn't fix our problem. We attempted to improve the photo resolution of the reshaped images using the Pillow python library. In order to improve the photo resolution of our reshaped images, we attempted to add a quality parameter when we were saving our reshaped image.

```
resized_img.save('2_resized.jpg', quality=95)
```

Picture of "2" and Result:



Original Reshaped

However, from looking at the images of handwritten "2", the quality parameter did not drastically improve the resolution of the reshaped image. Even though the resolution is still poor, the reshaped image is recognizable as the number 2. So, photo resolution may not be causing our incorrect model prediction issue.

We continued to research why a multiclass machine learning model would predict the same class label for every inputted image. Throughout multiple articles, one main cause was constantly discussed: imbalance data.

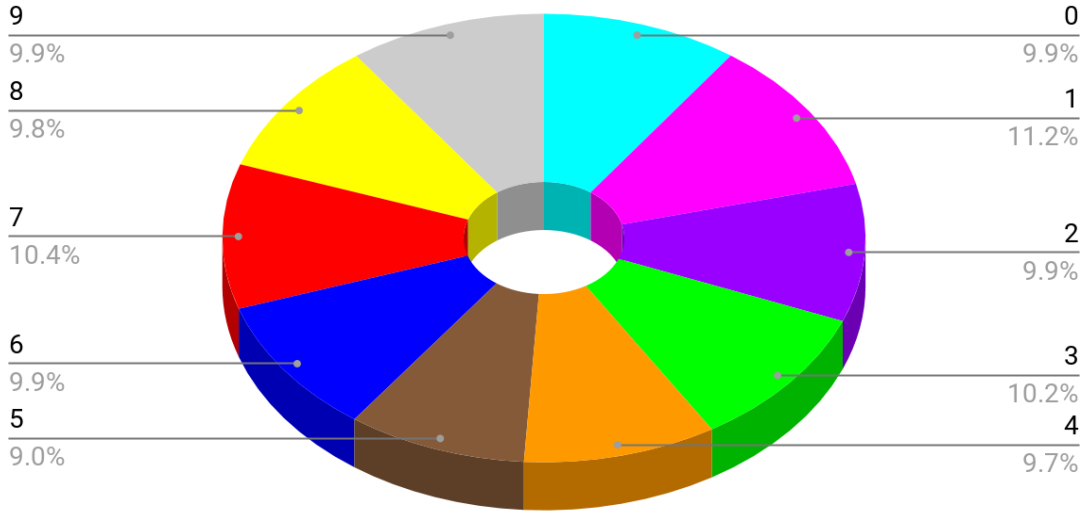
An imbalanced classification problem is an example where the distribution of image data across the known classes is biased or skewed. The distribution can vary from a slight bias to a severe imbalance where there is little image data in the minority class but hundreds, thousands, or millions of image data in the majority class or classes.

Imbalanced classifications pose a challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of image data for each class. This results in models that have poor predictive

performance, specifically for the minority class. This is a problem because typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than the majority class.

At first, we didn't think this was our problem since we are using the MNIST dataset that contains 60,000 training images for a total of 10 digit classes (0 to 9). We assumed that there were 10,000 images per digit class, but when we checked the number of images in each class folder, we noticed an imbalance in our data.

Training Image Data Distribution	
0	5923
1	6742
2	5958
3	6131
4	5842
5	5421
6	5918
7	6265
8	5851
9	5949



From our table and pie chart, we noticed that our image data for the handwritten digit 1 was 2% greater (1,000 images more) than the rest of the image data for our digits. Since we had more handwritten digit images for 1, it made sense that our ML model would always predict the class of our inputted image to be 1. So we began to adjust the training image data for each handwritten digit class to solve our imbalance data problem. Instead of training our model on 60,000 images, we trained the model on 54,000 images, where there are 5,400 images in each handwritten digit class. The images that we removed from each class were saved in a separate file from the training images just in case we would need them in the future.

After balancing our image data, we can see that there's an equal number of images in each digit class. For the rest of the class, we spent our time training our ML model on the balanced data to determine if the newly trained model would correctly classify our inputted handwritten images. Unfortunately, we still received the same incorrect result.

Picture of "8" and Result:

Original Reshaped

```
Orignal Image: (91, 76)
Resized Grayscale Image: (28, 28)
One
```

Most of our project time was dedicated to balancing data and training our model, so for the next project block, we will continue to determine the problem with our model.

Evidence of Progress
Research: ❖ 48 Responses to A Gentle Introduction to Imbalanced Classification ➢ https://machinelearningmastery.com/what-is-imbalanced-classification/ ❖ Neural network always predicts the same class ➢ https://stackoverflow.com/questions/41488279/neural-network-always-predicts-the-same-class ❖ keras model only predicts one class for all the test images ➢ https://datascience.stackexchange.com/questions/45833/keras-model-only-predicts-one-class-for-all-the-test-images
Signatures
 ***Emma Mascillaro***
 ***Kara Pietrowicz***

Date: Thursday, 4/29/2021

Goals	
Long Term: - Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	**Short Term:** - Continue to research possible solutions to our ML model's incorrect class predictions

Progress

Today's Progress:

Last project block, we considered how our imbalance data would affect our ML model's prediction. We spent our time removing image data from each handwritten digit class in order to balance our data and training our model on the newly balanced data. Since our model would always classify an inputted image as the digit 1, it made sense that our imbalance data had image data of 1 that was 2% greater (1,000 images more) than the image

data for any other digit class. However, after we trained our model on balanced data, the model continued to predict each inputted image as the digit 1.

At this moment, both research partners were confused on what exactly the issue was with our ML model's prediction. Did the issue originate from our model or from how the inputted image was prepared before being sent to the model? Our model still produced about 98% accuracy, which means it correctly classified 98% of the 10,000 provided testing images after the model was trained on 54,000 training images. Due to this high accuracy, we didn't believe that our model is the main issue.

So we began to focus on how each inputted image was prepared before it was sent to the model for classification. The following is an image of our **OLD** code that was classifying our inputted images incorrectly.

```
1   import cv2
2   import tensorflow as tf
3   import matplotlib.pyplot as plt
4   import numpy as np
5   from keras.models import load_model
6   import os
7   from keras.preprocessing import image
8   from PIL import Image, ImageOps, ImageChops
9
10  CATEGORIES = ["Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"]
11  PATH = r"jpg_photos"
```

First, we created a list with the class labels of each category of data in our dataset and saved it in the CATEGORIES list. We also noted the path to our directory with our test images in PATH.

```
33  # Formatting image for model
34  def prepare(path):
35      try:
36          IMG_SIZE = 28
37          img = Image.open(path)
38          img = ImageOps.grayscale(img)
39        # img = ImageChops.invert(img)
40          print('Orignal Image:', img.size)
41          resized_img = img.resize((IMG_SIZE, IMG_SIZE))
42          resized_img.save('2_resized.jpg', quality=95)
43          resized_img.show()
44          print('Resized Grayscale Image:', resized_img.size)
45          img_array = cv2.imread('2_resized.jpg')
46          new_array = img_array.astype('float32')
47          new_array = new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
48          return new_array
49      except Exception as e:
50          print(str(e))
```

Next, we wrote a *prepare* function in order to format our images to feed into the model. In order to get the best accuracy, we want our images to have the same color and dimensions as the training images, so we converted our testing images to grayscale and resized them to be 28x28 pixels. Then we created and reshaped a pixel array of the image for the model.

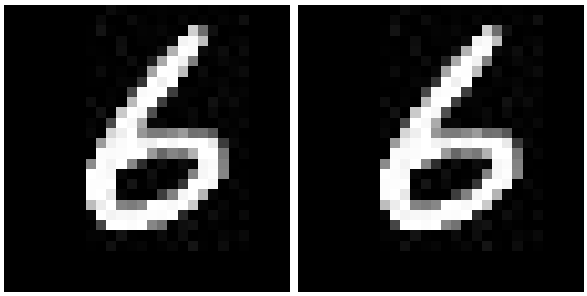
```
83  model = tf.keras.models.load_model('gray_model.h5', compile = True)
84  os.chdir("jpg_photos")
85  os.system("cd")
86  prediction = model.predict([prepare('8.jpg')])      #always pass a list
87  print(CATEGORIES[int(prediction[0][0])])
```

As you can see in the above picture, we then loaded our saved model, navigated into the directory of our testing images, and made a prediction on our new image after it was processed through the *prepare* function .

Usually, we would send a handwritten digit image we created to the model for a prediction. But eventually we began to wonder what would happen if we sent one of the handwritten digit images that we removed from the MNIST training dataset (for balancing purposes) to the model. Would the model correctly predict an MNIST image?

To answer our question, we sent the following MNIST image that was taken out of training data (for balancing purposes) to the model to see what kind of prediction we would receive.

Picture of MNIST "6" and Result:



Original Reshaped

```
Orignal Image: (28, 28)
Resized Grayscale Image: (28, 28)
Zero
```

When we used an image from the MNIST dataset that the model DID NOT train on, the model classified the handwritten 6 as the digit 0. A few things can be noted about this new prediction:

- First, the model is not correctly classifying an image that is from the MNIST dataset. Even though this image was not used to train the model, it has the same format as the images that did train the model. Therefore, it would be expected that the digit 6 image should be classified correctly.
- The prediction was not 1 this time. The non-MNIST images we created for the model were always classified as 1, but the digit 6 image was classified as 0. This could indicate that there's a general problem with the non-MNIST images we create since they all end up with the same classification.
- If the digit 6 image is not correctly classified, there might be something wrong with the *prepare* function we wrote.
- The reshaped image is not blurry or slightly distorted like the previous non-MNIST images we created because the original image is already the correct size for the model. Our non-MNIST images may be classified incorrectly because of how distorted the reshaped images are.

Before attempting to figure out why our non-MNIST handwritten digit images would produce only one classification, we wanted to make sure that our model can correctly classify an image from the MNIST dataset.

For the rest of the project block, we developed a new *prepare* function that would AT LEAST correctly classify any handwritten digit image from the MNIST dataset.

```
43  # Formatting image for model
44  def prepare(path):
45      try:
46          IMG_SIZE = 28
47          img_array = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
48          print('Orignal Image Array Shape:', img_array.shape)
49          new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
50          new_array = new_array.astype('float32')
51          new_array = new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
52          print('Resized Grayscale Image Array for Model:', new_array.shape)
53          return new_array
54      except Exception as e:
55          print(str(e))
```

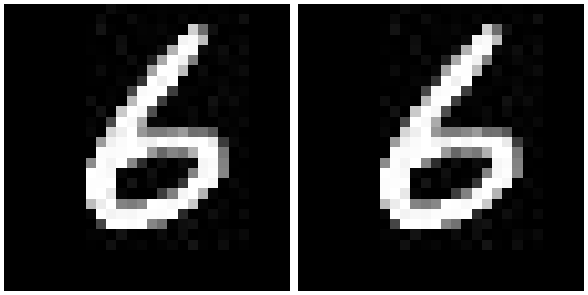
Shown above is our new *prepare* function . In order to get the best accuracy, we want our images to have the same color and dimensions as the training images. Instead of resizing the actual digit image to 28x28 pixels and converting its color scale to grayscale (OLD *prepare* function), we used cv2.imread to immediately create a grayscale image array from the actual image. We then resized the image array to 28x28 pixels and reshaped the 2D pixel array to a 4D pixel array for the CNN model to utilize for its prediction.

```
58  model = tf.keras.models.load_model('gray_model.h5', compile = True)
59  os.chdir("jpg_photos")
60  os.system("cd")
61  prediction = model.predict([prepare('66.jpg')])
62  # print(np.argmax(prediction))
63  pred_name = CATEGORIES[np.argmax(prediction)]
64  print("Model's Prediction:", pred_name)
```

As you can see in the above picture, we then loaded our saved model, navigated into the directory of our testing images, and made a prediction on our new image after it was processed through the *prepare* function. However, we utilized np.argmax. Argmax is an operation that finds the argument that gives the maximum value from a target function. In our case, Argmax is used in machine learning for finding the class with the largest predicted probability.

***The following images were NOT produced by our NEW code, as shown above. These are what the inputted and reshaped images look like when we separately display the image arrays we created in our code.**

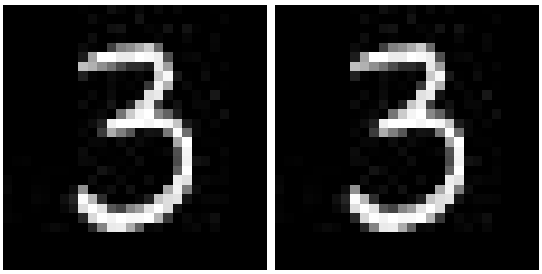
Picture of MNIST "6" and Result:



Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Six
```

Picture of MNIST "3" and Result:

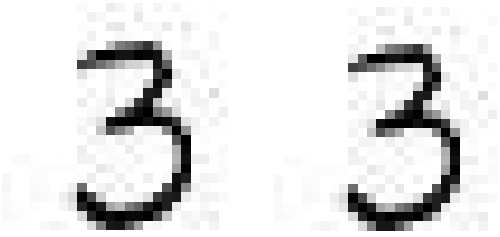


Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Three
```

Luckily, our ML model was able to correctly classify MNIST dataset images using the new *prepare* function we created.

Picture of INVERTED MNIST "3" and Result:

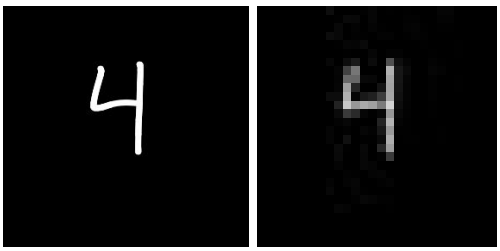


Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Six
```

Out of curiosity, we decided to invert the colors of the MNIST digit 3 to determine if our model can correctly predict the class for an image with a white background and black handwriting. Since the MNIST digit 3 was incorrectly classified when its colors were inverted, we concluded that our model can only classify images if they have a black background with white handwriting. This could also explain why the non-MNIST images we created were previously classified incorrectly (they had a white background with black handwriting).

Picture of NON-MNIST "4" and Result:



Original Reshaped

```
Orignal Image Array Shape: (281, 285)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Nine
```

When we created our non-MNIST image with a black background with white handwriting, the model predicted our digit 4 to be a 9. Thanks to our new *prepare* function , the non-MNIST images we create were no longer classified solely as the digit 1. But there is something else to note. The reshaped image is very blurry and could be considered the digit 9. Therefore, our *prepare* function might be able to help the ML model correctly classify our non-MNIST digit images if we can improve the resolution of the reshaped image.

Evidence of Progress

Research:

- ❖ Keras - How to use argmax for predictions
  - ➢ https://stackoverflow.com/questions/54167910/keras-how-to-use-argmax-for-predictions
- ❖ numpy.argmax — NumPy v1.20 Manual
  - ➢ https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
- ❖ Python OpenCV | cv2.imread() method
  - ➢ https://www.geeksforgeeks.org/python-opencv-cv2-imread-method/
- ❖ Read image grayscale opencv 3.0.0-dev
  - ➢ https://stackoverflow.com/questions/23339315/read-image-grayscale-opencv-3-0-0-dev
- ❖ Python - Color Inversion using Pillow
  - ➢ https://www.geeksforgeeks.org/python-color-inversion-using-pillow/

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Monday, 5/3/2021

Goals

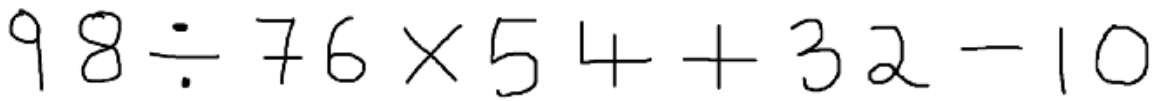
Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Research how our model will predict and print out a given handwritten mathematical expression - Start writing the necessary code for the ML model to utilize

Progress

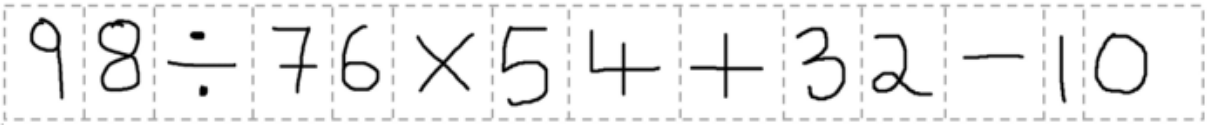
Today's Progress:

During this project cycle, we don't want to waste too much time on trial and error with our ML model and postpone making progress on the other aspects of our project for it. We WILL fix our ML problem in the future, but we believe we are at a good point to set aside the ML component of our project and begin working on the tkinter app development so we can get an idea of how to incorporate our code with these final project components. We can make some decent progress on the non-ML components since they can be developed and tested partially without our model.

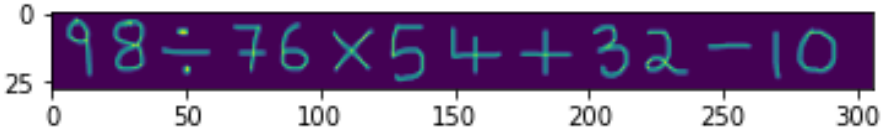
In order for our model to analyze and correctly predict a handwritten mathematical expression, it first needs to be broken down into individual elements that can be identified by the model. Take a look at the following expression.



Before this image is sent to the model, we need to break it down into images of digits and mathematical operations, as shown below.



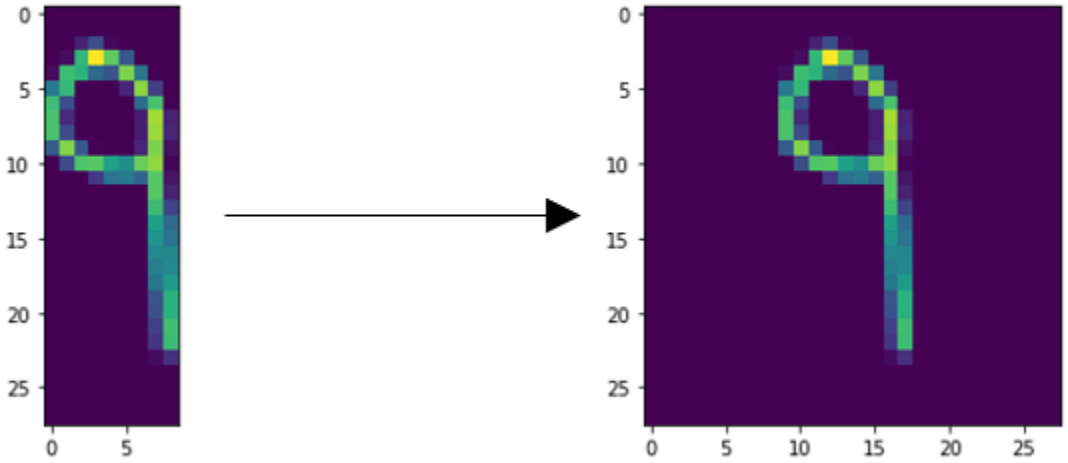
First, the image should be converted to grayscale using Pillow. Then the image will be resized to a height of 28 pixels to keep the same aspect ratio of the model input requirement (28 x 28). The width will be adjusted accordingly. Next, the image will be normalized and converted from an image to a numpy array. The following is the numpy array displayed as the tweaked image.



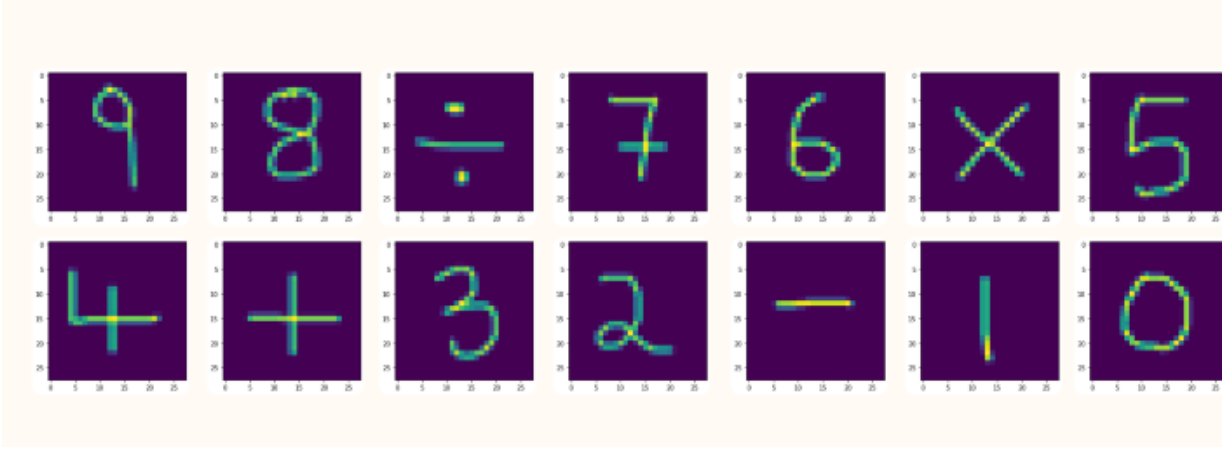
Now the image array will be split into individual element arrays. This is done by searching the image array for successive non zero columns and grouping them to form one element array. These element arrays are all stored in one list. Since there are 14 elements in the mathematical expression, the size of the list is 14.

In order to identify the element using the model, it is important to have the image size of 28 x 28 pixels. The height is already 28 pixels. However, due to the splitting process, the width of

each element will not be exactly 28 pixels. Therefore, after the splitting, the width of each element needs to be adjusted to 28 pixels. by adding zero value columns (filler columns) to the element arrays.



This process will be repeated for all mathematical elements in the expression.



Finally, the elements list is converted back into an array of shape (14, 28, 28, 1) to fit the model input requirements of a CNN model [mini_batch_size, height, width, color depth]. The elements array is passed through the model for prediction. The model returns the probabilities of all the number classes (purpose of Softmax function). The class with the highest probability is chosen and assigned to each of the 14 images.

We spent the rest of the project period writing our bounding box code that would create a bounding box around each digit and operator and save each boxed image for our *prepare* function to use before the images are sent to the model.

```python
import cv2
import os
import numpy as np

img = cv2.imread('box_expression_image.jpg', 0)
path = r"jpg_photos"

cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
ROI_number = 0
```

Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared to the assigned threshold value. If the intensity of a pixel in the input image is greater than a threshold, the corresponding output pixel is marked as white (255, foreground), and if the input pixel intensity intensity is less than or equal to the threshold, the output pixel is marked black (0, background). Thresholding is a very popular segmentation technique, which is used for separating an object from its background. This technique of thresholding is done on grayscale images.

In our code, the function cv2.threshold is used for thresholding:

- **cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)**
    - **source**: Input Image array (must be in Grayscale).
    - **thresholdValue**: Value of Threshold below and above which pixel values will change accordingly.
    - **maxVal**: Maximum value that can be assigned to a pixel.
    - **thresholdingTechnique**: The type of thresholding to be applied.

First, we use the method cv2.imread() to load our mathematical expression image after converting it to grayscale space (passed 0 to indicate grayscale conversion). We set our thresholdValue for our source image array (img) to be 0, which means that all pixel values above 0 will be set to the maxVal (255). Then we specified our thresholdingTechnique to be cv2.THRESH_BINARY + cv2.THRESH_OTSU. Binary Thresholding is where we use our set threshold value. If pixel intensity is greater than the set threshold, that pixel's value is set to 255 (white), or else it is set to 0 (black).

cv2.THRESH_OTSU is an extra flag which specifies the way a threshold should be calculated. Otsu's thresholding is used to perform automatic image thresholding on bimodal images (an image with two distinct pixel values). In the simplest form, the algorithm returns a single

intensity threshold that separates pixels into two classes, foreground (white) and background (black). It can find the optimal threshold value of the input image by going through all possible threshold values (from 0 to 255) from a bimodal image's histogram. First, the input image is processed and the bimodal image's histogram (distribution of pixel values) is obtained. The histogram of such an image contains two clearly expressed peaks, which represent different ranges of intensity values. The core idea is to separate the image histogram into two clusters with a threshold defined as a result of minimization of the weighted variance (an approximation of the spread among the pixel value data points) of these classes. Usually, the threshold value will lie in the middle of both the histogram peak values.

After the threshold value is computed, if the intensity of a pixel in the input image is greater than the calculated threshold, the corresponding output pixel is marked as white (255, foreground), and if the input pixel intensity is less than or equal to the threshold, the output pixel is marked black (0, background).

The reason why we add cv2.THRESH_OTSU to cv2.THRESH_BINARY is because the Otsu thresholding technique has some limitations. The Otsu thresholding technique assumes that there is uniform illumination, the image histogram is bimodal, and there is no noise in the image. If an inputted image violates any of these assumptions, then the Otsu threshold technique will not be used. Instead of using a calculated threshold value based on the inputted image, the threshold value will be the same as what was originally set in the code (thresholdValue = 0 in our code).

Once the thresholding technique is applied to the grayscale image, we save the black and white image obtained after using the thresholding method to img.

```
1   import cv2
2   import os
3   import numpy as np
4
5   img = cv2.imread('box_expression_image.jpg', 0)
6   path = r"jpg_photos"
7
8   cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
9   image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
10  sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
11  ROI_number = 0
```

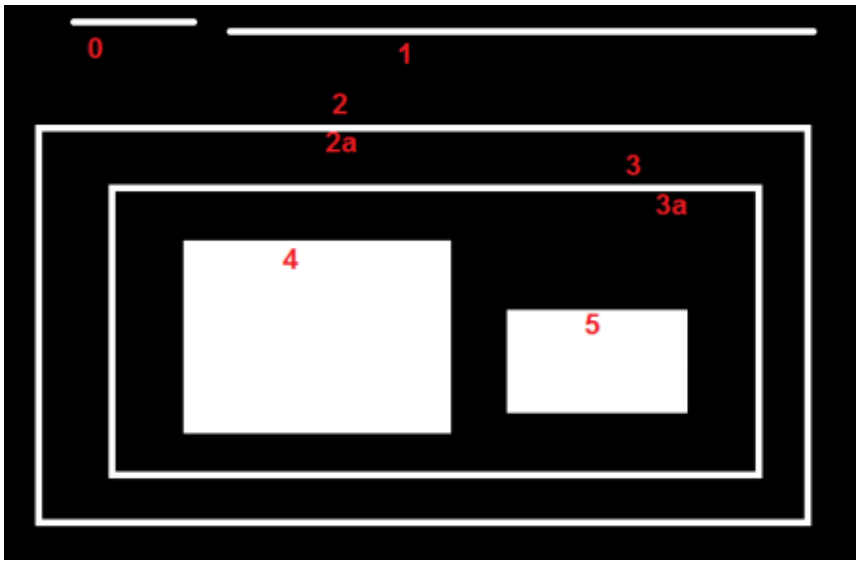
Contours are defined as the line joining all the points along the boundary of an image that all have the same pixel intensity. Contours come in handy for shape analysis, finding the size of the object of interest, and object detection. OpenCV has a findContours() function that helps in extracting the contours from the image. It works best on binary images, so we should first apply thresholding techniques.

We see that there are three essential arguments in cv2.findContours() function. First one is source image, second is contour retrieval mode, third is contour approximation method and it outputs the image, contours, and hierarchy. 'contours' is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object.

We know that contours are the boundaries of a shape with the same intensity. It stores the (x, y) coordinates of the boundary of a shape. But does it store all the coordinates? That is specified by this contour approximation method.

We passed cv2.CHAIN_APPROX_NONE because we need all the boundary points to be stored. For eg, if we had to find the contour of a straight line, we need just two endpoints of that line. This is what cv2.CHAIN_APPROX_SIMPLE does. It removes all redundant points and compresses the contour, thereby saving memory.

But what about hierarchy? Normally we use the cv.findContours() function to detect objects in an image. Sometimes objects are in different locations. But in some cases, some shapes are inside other shapes. When this occurs, we call the outer contour as"parent" and inner contours as "child". This way, contours in an image have some relationship to each other. And we can specify how one contour is connected to each other, like, is it the child of some other contour, or is it the parent etc. Representation of this relationship is called the Hierarchy.



Consider this example. In this image, there are a few shapes which are numbered from 0-5. 2 and 2a denote the external and internal contours of the outermost contour box.

Here, contours 0,1,2 are external, extreme, or outermost. We can say, they are in level 0 hierarchy or simply they are in the same hierarchy level. Next comes contour-2a. It can be considered as a child of contour-2 (or contour-2 is parent of contour-2a). So let it be in level 1 hierarchy. Similarly contour-3 is child of contour-2 and it comes in the next hierarchy. Finally contours 4,5 are the children of contour-3a, and they come in the last hierarchy level.

In our code, we used the contour retrieval mode of cv2.RETR_EXTERNAL, which only retrieves the extreme outer contours. All child contours are left behind. If we were to use this method on the above image, only the extreme or level 0 hierarchy contours would be retrieved (contours 0,1,2).

```
1   import cv2
2   import os
3   import numpy as np
4
5   img = cv2.imread('box_expression_image.jpg', 0)
6   path = r"jpg_photos"
7
8   cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
9   image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
10  sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
11  ROI_number = 0
```

Lastly, we used the sorted() function in python to sort the bounding boxes created each digit and operator. It sorts any sequence (list, tuple) and always returns a list with the elements in sorted manner, without modifying the original sequence.

Syntax : sorted(iterable, key)

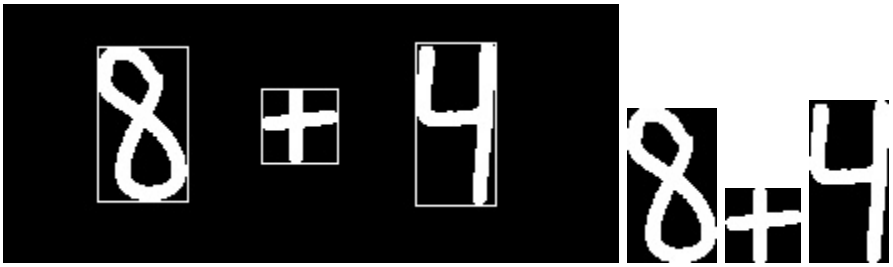
Iterable : sequence (list, tuple, string) or collection (dictionary, set, frozenset) or any other iterator that needs to be sorted.
Key : A function that would serve as a key or a basis of sort comparison.

The line of code enclosed in the red rectangle uses a key that orders the bounding boxes or ROI's (Regions of Interest) by their x-coordinates of the top-left coordinate of the rectangle. This allows us to sort the bounding boxes from left to right (the same manner we use to read mathematical expressions).

```
13  for c in sorted_ctrs:
14      # get the bounding rect
15      x, y, w, h = cv2.boundingRect(c)
16      ROI = img[y:y+h, x:x+w]
17      cv2.imwrite(os.path.join(path, 'ROI_{}.jpg'.format(ROI_number)), ROI)
18      # draw a white rectangle to visualize the bounding rect
19      cv2.rectangle(img, (x, y), (x + w, y + h), 255, 1)
20      ROI_number += 1
21
22  cv2.drawContours(img, contours, -1, (255, 255, 0), 1)
23  cv2.imwrite("output_box_image.jpg",img)
```

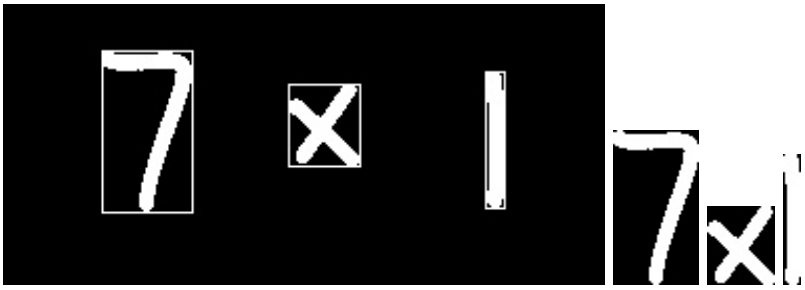
Within the sorted list of bounding box contours, we used the boundingRect() function to calculate the (x,y) top-left coordinate of the rectangle and its width and height (w,h). With these values, another image was created to enclose the ROI (Region of Interest), which would include either a digit or operator. Then the new image is saved to a specific path with a new name, and a white rectangle is drawn on that image to visualize the bounding box or rectangle. The image with a bounding box is then saved as the output_box_image.

Results:

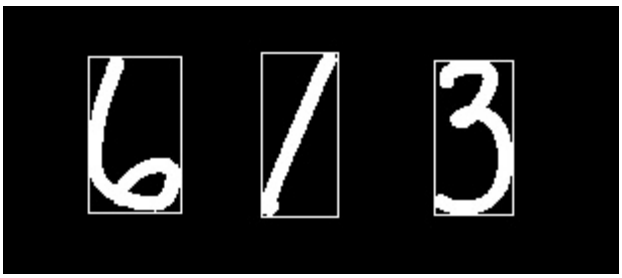


Output_box_image ROI_0 ROI_1 ROI_2

Output_box_image ROI_0 ROI_1 ROI_2

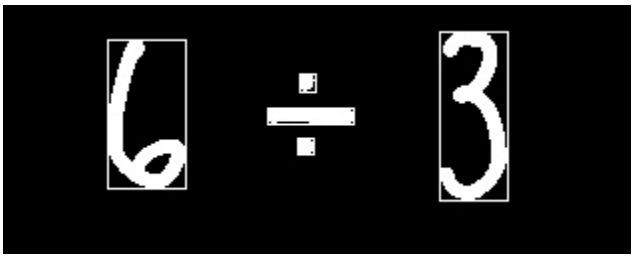


Output_box_image ROI_0 ROI_1 ROI_2




Output_box_image ROI_0 ROI_1 ROI_2

As you can see, our code was able to draw a bounding box around each individual digit and operator. These images can then be sent to our *prepare* function for formatting before they are sent to the model for classification.



Output_box_image

However, when we write the commonly used division operator, five bounding boxes are created instead of 3. This is something we will have to fix later, but at the moment, our calculator scripts are trained to recognize the slash symbol "/" as our division symbol. We can fix this problem after we improve our calculator scripts.

Evidence of Progress

Research:

- cv2.imwrite save to folder Code Example
  - https://www.codegrepper.com/code-examples/python/cv2.imwrite+save+to+folder
- OpenCV - Saving images to a particular folder of choice
  - https://stackoverflow.com/questions/41586429/opencv-saving-images-to-a-particular-folder-of-choice
- Extract bounding box and save it as an image
  - https://stackoverflow.com/questions/13887863/extract-bounding-box-and-save-it-as-an-image
- Text Detection and Extraction using OpenCV and OCR

- ➢ https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr/

❖ Text Detection: Getting Bounding boxes
- ➢ https://stackoverflow.com/questions/50000302/text-detection-getting-bounding-boxes/50004340

❖ Sort images after ROI (Python, OpenCV)
- ➢ https://stackoverflow.com/questions/45000939/sort-images-after-roi-python-opencv

❖ 72 responses to: Sorting Contours using Python and OpenCV
- ➢ https://www.pyimagesearch.com/2015/04/20/sorting-contours-using-python-and-opencv/

❖ Building a Handwritten Multi-Digit Calculator | by Neerav Gala
- ➢ https://towardsdatascience.com/building-a-handwritten-multi-digit-calculator-f03cf5028052

❖ Otsu's Thresholding Technique
- ➢ https://learnopencv.com/otsu-thresholding-with-opencv/

❖ Otsu Thresholding
- ➢ https://hbyacademic.medium.com/otsu-thresholding-4337710dc519

❖ Python | Thresholding techniques using OpenCV | Set-1 (Simple Thresholding)
- ➢ https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/

❖ Image Thresholding — OpenCV-Python Tutorials 1 documentation
- ➢ https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

❖ OpenCV & Python – The Otsu's Binarization for thresholding – Meccanismo Complesso
- ➢ https://www.meccanismocomplesso.org/en/opencv-python-the-otsus-binarization-for-thresholding/

❖ OpenCV 3 Image Thresholding and Segmentation - 2020
- ➢ https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Global_Thresholding_Adaptive_Thresholding_Otsus_Binarization_Segmentations.php

❖ OpenCV: Image Thresholding
- ➢ https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

❖ Python | Thresholding techniques using OpenCV | Set-3 (Otsu Thresholding)
- ➢ https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-3-otsu-thresholding/

- What's the difference between a flag, an option, and an argument? - Unix & Linux Stack Exchange
  - https://unix.stackexchange.com/questions/285575/whats-the-difference-between-a-flag-an-option-and-an-argument
- Basic Image Thresholding in OpenCV | by Anupriyam Ranjit
  - https://medium.com/@anupriyam/basic-image-thresholding-in-opencv-5af9020f2472
- Tag Archives: Limitations of otsu method
  - https://theailearner.com/tag/limitations-of-otsu-method/
- OTSU thresholding
  - http://computervisionwithvaibhav.blogspot.com/2015/10/otsu-thresholding.html
- Find and Draw Contours using OpenCV-Python
  - https://theailearner.com/2019/11/19/find-and-draw-contours-using-opencv-python/
- OpenCV: Contours Hierarchy
  - https://docs.opencv.org/master/d9/d8b/tutorial_py_contours_hierarchy.html
- Find and Draw Contours using OpenCV | Python
  - https://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/
- Contour Features — OpenCV-Python Tutorials 1 documentation
  - https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html
- Sorted() function in Python
  - https://www.geeksforgeeks.org/sorted-function-python/

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 5/4/2021

Goals	
Long Term: - Write handwriting recognition code to	**Short Term:** - Continue working on tkinter GUI for

identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	our model

Progress

<u>Today's Progress:</u>

Last project cycle, we started the code for our tkinter GUI. After building the basic window last class, we were able to display a live camera feed inside the window of the user interface, take a picture of a mathematical expression, and save the picture.

Today, we wanted to improve the GUI so that the user can navigate between multiple pages using buttons on one window. These pages would include a start page with the live camera feed and a solution page where after our model transcribes the handwritten mathematical expression, our calculator scripts would solve the expression and present the solution in the app interface (on the solution page). So far, we have written the following code.

```
10  class Page(tk.Tk):
11      def __init__(self, *args, **kwargs):
12
13          tk.Tk.__init__(self, *args, **kwargs)
14          container = tk.Frame(self)
15
16          container.pack(side ="top", fill="both", expand= True)
17
18          container.grid_rowconfigure(0, weight = 1)
19          container.grid_columnconfigure(0, weight = 1)
20
21          self.frames = {}
22
23          frame = StartPage(container, self)
24
25          self.frames[StartPage] = frame
26
27          frame.grid(row = 0, column = 0, sticky = "nsew")
28
29          self.show_frame(StartPage)
30
31      def show_frame(self, cont):
32
33          frame = self.frames[cont]
34          frame.tkraise()
```

The above code creates the basic structure of a tkinter window. This can be called and used in other classes in order to minimize the amount of repeated code. First, we initialize our class with the init method. Self is required as the first argument for all object initializations. This is so that Python knows which object is being referred to if you have multiple App objects. Args are used to pass an unknown amount of arguments through the method. The difference between them is that args and kwargs is that args are used to pass non-keyworded arguments, where kwargs are keyword arguments. Args are your typical parameters. Kwargs are basically dictionaries.

Next, we define a container, which is responsible for arranging the position of other widgets and holding multiple window frames. To organize the widgets we create, we use layout managers called pack and grid. The pack geometry manager organizes widgets in horizontal and vertical boxes. The grid geometry manager places widgets in a two dimensional grid.

Then we specify a dictionary (self.frames{}) and declare the variable frame to create one window page called StartPage. Finally, we call show_frame, which is a method defined in the next few lines, to display or raise a frame of our choosing, such as StartPage to the front of the one open window.

```
class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        # Start of Our Code #

        width, height = 800, 600
        cap = cv2.VideoCapture(0)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

        lmain = Label(app)  #fix
        lmain.pack()

        def show_image_frame():
            ret, frame = cap.read()
            if ret:
                cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
                img = PIL.Image.fromarray(cv2image)
                imgtk = ImageTk.PhotoImage(image=img)
                lmain.imgtk = imgtk
                lmain.configure(image=imgtk)
                lmain.after(10, show_image_frame)

        def capture_image():
            videoCaptureObject = cv2.VideoCapture(0)
            ret, frame = videoCaptureObject.read()
            img = cv2.imwrite("Capture_Image.jpg", frame)
            #videoCaptureObject.release()

        image_capture = tk.Button(text="Take Picture", command = capture_image())
        image_capture.pack()
        print("Opening Application")
        show_image_frame()
        root.mainloop()
        switch_page = tk.Button(text = "View the solution", command = lambda: controller.show_frame(SolutionPage))
        switch_page.pack()
```

After writing the basic setup of a window page, we created our StartPage class. We initialize our class using our parent or main class (Page()). Then we inserted the live camera feed code we wrote last project cycle so that the StartPage would display a camera for the user to take a picture of a mathematical expression. Overall, we identified the camera to be used, set the dimensions of the frame, created two functions to display the live camera feed and to capture a still image, and created a button which runs the capture_image function when pressed. Finally, we call our show_frame function to display the image and use root.mainloop in order to constantly display and reset the window until the window is closed.

As part of our new code, we created a button that can switch between the StartPage and our Solution page class, which is shown below.

```
76  class SolutionPage(tk.Frame):
77      def __init__(self, parent, controller):
78          tk.Frame.__init__(self, parent)
79          label = tk.Label(self, text = "answer", font = LARGE_FONT)
80          label.pack(pady = 10, padx = 10)
81          return_button = tk.Button(self, text = "Take another Photo", command = lambda: controller.show_frame(StartPage))
82          return_button.pack()
83
84  app = Page()
85  app.mainloop()
```

Again, we initialize our class using our parent or main class (Page()). We also added a label to indicate the answer to the handwritten expression and a return button so that the user can return to the StartPage and take another picture.

After compiling all of our code, we received an error message explaining how “app” is not defined in Line 48 of our code, where we use app to create the basic page setup for our StartPage and live camera feed. This makes sense since app is not defined until the end of all our code, but if we move the position of app, the program gets stuck in an infinite loop where it continuously switches between running StartPage() and Page(). We spent the rest of the project block attempting to solve this problem but were unsuccessful. For the next project cycle, we plan on fixing our GUI code.

Evidence of Progress

Research:

- Tkinter Application to Switch Between Different Page Frames
  - https://www.geeksforgeeks.org/tkinter-application-to-switch-between-different-page-frames/
- Object Oriented Programming Crash course with Python 3 - Tkinter tutorial Python 3.4 p. 2
  - https://www.youtube.com/watch?v=A0gaXfM1UN0&list=PLQVvvaa0QuDclKx-QpC9wntnURXVJqLyk&index=2
- Multiple Windows/Frames in Tkinter GUI with Python - Tkinter tutorial Python 3.4 p. 4
  - https://www.youtube.com/watch?v=jBUpjijYtCk&list=PLQVvvaa0QuDclKx-QpC9wntnURXVJqLyk&index=4

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 5/6/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Improve Log entries to better represent our progress - Create and complete our fourth project presentation - Gather the necessary research we've completed to discuss in our presentation - Establish our future plans

Progress

Today's Progress:

Throughout the weekend and the weekdays prior to May 11, we continued to work on our project presentation. This presentation mainly focuses on our new code we have written so far for our *prepare* function, bounding box images, and user interface. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we improved the model predictions of MNIST dataset images, wrote bounding box code that will be used to save and send images to our *prepare* function, and started to write more code for our GUI. We then developed our future plans to show our advisors that we will continue to make progress in our project, such as going back to improving our machine learning model's prediction, completing our user interface, and utilizing the bounding box images in our *prepare* function. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (5 minutes max) was more difficult this time, mainly because we had more code to explain in simple yet sufficient details. We were able to present our slides in about 5 ½ minutes while conveying all the necessary information to our advisor

and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

Fourth Presentation Link:

- ❖ Mascillaro & Pietrowicz Project 5/11/2021

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 5/13/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Research how to merge bounding boxes - Consider how to fix bounding box around traditional division sign for our code

Progress

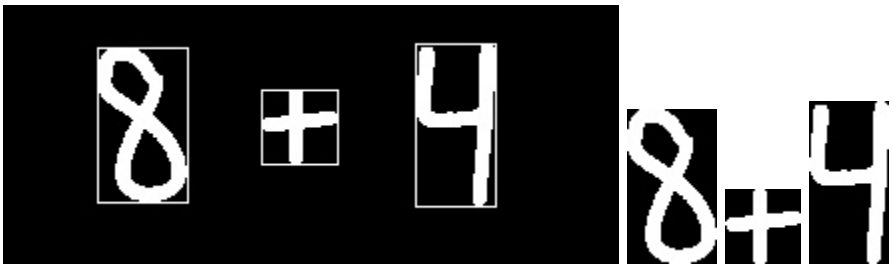
Today's Progress:

Last project cycle, we improved the model predictions of MNIST dataset images, wrote bounding box code that will be used to save and send images to our *prepare* function, and started to write more code for our GUI. Today, we decided to take a look at our bounding box issue.

In order for our model to analyze and correctly predict a handwritten mathematical expression, it first needs to be broken down into individual elements that can be identified by the model.

Within a sorted list of bounding box contours (a Numpy array of (x, y) coordinates of boundary points of the object), we used the boundingRect() function to calculate the (x,y) top-left coordinate of the rectangle and its width and height (w,h). With these values, another image was created to enclose the ROI (Region of Interest), which would include either a digit or operator. Then the new image is saved to a specific path with a new name, and a white rectangle is drawn on that image to visualize the bounding box or rectangle. The image with a bounding box is then saved as the output_box_image.

Results:



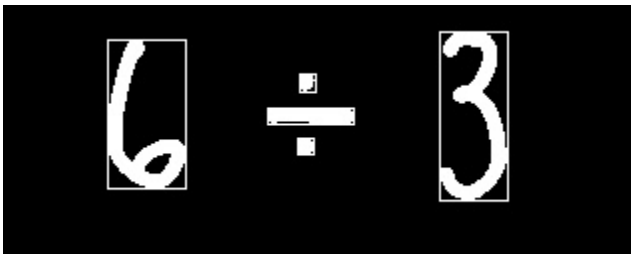
Output_box_image ROI_0 ROI_1 ROI_2

Output_box_image ROI_0 ROI_1 ROI_2

Output_box_image ROI_0 ROI_1 ROI_2

Output_box_image ROI_0 ROI_1 ROI_2

As you can see, our code was able to draw a bounding box around each individual digit and operator. These images can then be sent to our *prepare* function for formatting before they are sent to the model for classification.



Output_box_image

However, when we write the commonly used division operator, five bounding boxes are created instead of 3.

We began to research how we could draw a bounding box around the entire division operator instead of drawing three boxes around the line and dots individually. One possible solution we came across was using the function cv2.groupRectangles().

groupRectangles(rectList, groupThreshold[, eps]) → rectList, weights

The groupRectangles() function expects a list of rectangles that are in the form of [x, y, width, height]. It will return a new list of grouped rectangles where the rectangles that were nearest to each other have been grouped into one rectangle.

The groupThreshold parameter will almost always be 1. If it is set to 0, it's not going to group any rectangles at all. And if it is set to something higher, it's going to require that more rectangles are overlapping/near each other before creating a grouped result for them.

The eps parameter controls how close together the rectangles need to be before they will be grouped together. Lower values require that rectangles be closer together to be merged, while higher values will group together rectangles that are farther away.

Another possible way of merging nearby bounding boxes is to create a list of boxes (List A) using the sorted contours from our code in the previous project cycle. For each box in List A, if a box is within a certain distance of another box, remove those boxes from List A and append them to an empty list (List B). We have to define a distance threshold for merging nearby boxes, which will be trial and error since we are not certain of the exact distance between the dots and line in a traditional division operator.

Next, for List A and List B, we would have to calculate pixel bounds (min x, min y, max x, max y) and use those values to create a new rectangle. As we considered what is in List A, we recalled that our code already calculates the minimum and maximum (x,y) pixel values from our sorted contours. So for List B, we would need to calculate the maximum (x,y) pixel values to create a larger bounding box around the entire division operator.

We continued our research to locate better examples or explanations of how bounding boxes can be merged, but we didn't find many reliable sources of information. We will eventually try the methods described above (we expect the groupRectangle() to be a better solution). For our project, it is better to have our ML model be trained on images of traditional division operators instead of just a slash "/", mainly due to the overwhelming number of training MNIST images for the digit 1 that also look very similar to slashes. We don't want our calculator to classify a division operator as the digit 1 or vice versa.

A more pressing issue is fixing our incorrect ML prediction during this project cycle.

Evidence of Progress

Research:

- Grouping Rectangles into Click Points - LearnCodeByGaming.com
  - https://www.learncodebygaming.com/blog/grouping-rectangles-into-click-points
- groupRectangles - cv2 - Python documentation
  - https://www.kite.com/python/docs/cv2.groupRectangles
- Optical Character Recognition | OCR Text Recognition
  - https://www.analyticsvidhya.com/blog/2020/05/build-your-own-ocr-google-tesseract-opencv/
- OpenCV detecting a single symbol with multiple bounding boxes (this could help with the NON-MNIST images that aren't classified correctly too)
  - https://stackoverflow.com/questions/56105512/opencv-detecting-a-single-symbol-with-multiple-bounding-boxes
- Merge the Bounding boxes near by into one
  - https://stackoverflow.com/questions/55593506/merge-the-bounding-boxes-near-by-into-one
- How to join nearby bounding boxes in OpenCV Python
  - https://stackoverflow.com/questions/55376338/how-to-join-nearby-bounding-boxes-in-opencv-python
- How to merge nearby rectangles
  - https://answers.opencv.org/question/204128/how-to-merge-nearby-rectangles/
- Dilation of Bounding Boxes. How to merge all red boxes into a larger green one? : computervision
  - https://www.reddit.com/r/computervision/comments/emtbut/dilation_of_bounding_boxes_how_to_merge_all_red/
- How to merge nearby rectangles - OpenCV Q&A Forum
  - https://answers.opencv.org/question/204128/how-to-merge-nearby-rectangles/

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 5/18/2021

Goals

Long Term:

- Write handwriting recognition code to identify and transcribe written text
- Write four-function calculator scripts
- Develop an app that utilizes all our code to solve mathematical problems

Short Term:

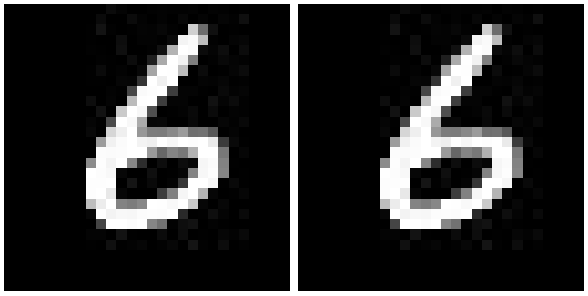
- Fix incorrect ML model predictions on Non-MNIST dataset images
- Combine our calculator scripts with our GUI output screen

Progress

Today's Progress:

During our last project cycle, we developed a new *prepare* function that would AT LEAST correctly classify any handwritten digit image from the MNIST dataset. We noticed that NON-MNIST dataset images (images taken by Kara or Emma) were not correctly classified.

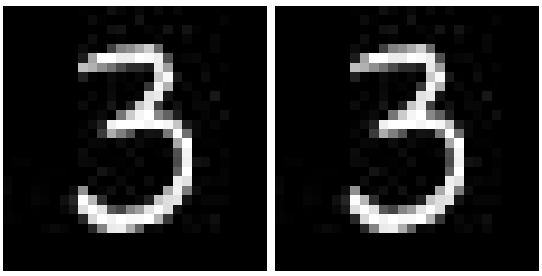
Picture of MNIST "6" and Result:



Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Six
```

Picture of MNIST "3" and Result:

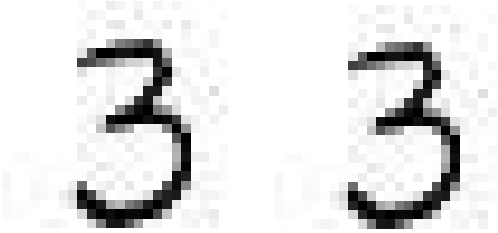


Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Three
```

Our ML model was able to correctly classify MNIST dataset images using the new *prepare* function we created.

Picture of INVERTED MNIST "3" and Result:

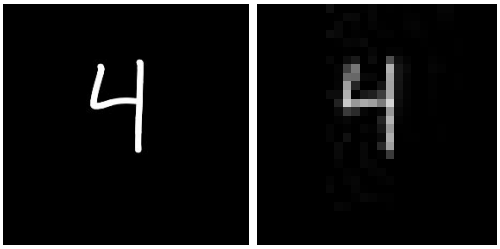


Original Reshaped

```
Orignal Image Array Shape: (28, 28)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Six
```

Since the MNIST digit 3 was incorrectly classified when its colors were inverted, we concluded that our model can only classify images if they have a black background with white handwriting. This could also explain why the non-MNIST images we created were previously classified incorrectly (they had a white background with black handwriting).

Picture of NON-MNIST "4" and Result:



Original Reshaped

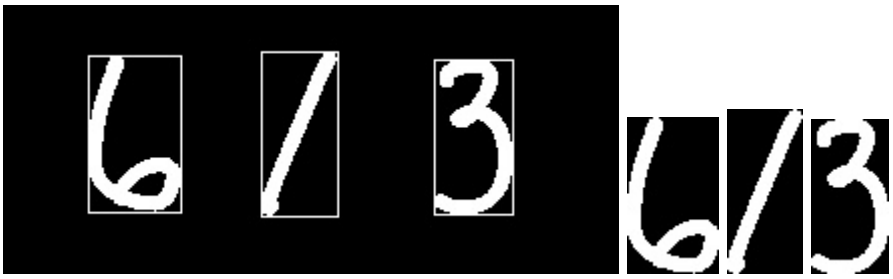
```
Orignal Image Array Shape: (281, 285)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Nine
```

When we created our non-MNIST image with a black background with white handwriting, the model predicted our digit 4 to be a 9. Thanks to our new *prepare* function , the non-MNIST images we create were no longer classified solely as the digit 1. But there is something else to

note. The reshaped image is very blurry and could be considered the digit 9. Therefore, our *prepare* function might be able to help the ML model correctly classify our non-MNIST digit images if we can improve the resolution of the reshaped image.

We did some research to determine how the resolution could be improved, but when we attempted to apply these resolution methods to our images, the image usually remained blurry.

Out of curiosity, we tested our *prepare* function on one of the images saved from our bounding box code (ROI image).



Output_box_image ROI_0 ROI_1 ROI_2

```
Orignal Image Array Shape: (78, 46)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Six
```

```
Orignal Image Array Shape: (82, 38)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Four
```

```
Orignal Image Array Shape: (77, 39)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: Three
```

The non-MNIST images that were subjected to our bounding box code AND prepare function were correctly predicted by our model. We tried different numbers to see if any other problems would arise, but each digit was correctly classified. The mathematical operators obviously weren't classified correctly since our ML model wasn't trained on any operators yet. We first wanted to make sure our model could correctly classify digits before adding operators to our training image dataset (which we will do later). More importantly, why did sending non-MNIST images through our bounding box code fix our incorrect ML prediction issue?

```
import cv2
import os
import numpy as np

img = cv2.imread('box_expression_image.jpg', 0)
path = r"jpg_photos"

cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
ROI_number = 0
```

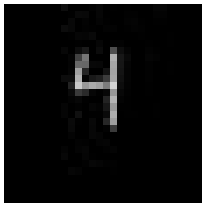
We believe this new result is due to how our code creates a bounding box around each digit in our expression image. In our bounding box code, we used a technique called thresholding, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared to the assigned threshold value. If the intensity of a pixel in the input image is greater than a threshold, the corresponding output pixel is marked as white (255, foreground), and if the input pixel intensity is less than or equal to the threshold, the output pixel is marked black (0, background). Thresholding is a very popular segmentation technique, which is used for separating an object from its background. This technique of thresholding is done on grayscale images. So when we inputted our non_MNIST grayscale images (before bounding box code), it is possible that there wasn't a good enough contrast between the pixels in order for the ML model to correctly predict the digit. When the non-MNIST image undergoes thresholding AND a conversion to grayscale (after bounding box code), the contrast between the digit and background may have been better for the ML model.

```
import cv2
import os
import numpy as np

img = cv2.imread('box_expression_image.jpg', 0)
path = r"jpg_photos"

cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
ROI_number = 0
```

Next in our bounding box code, we located contours, which are defined as the line joining all the points along the boundary of an image that all have the same pixel intensity. Contours

come in handy for shape analysis, finding the size of the object of interest, and object detection. OpenCV has a findContours() function that helps in extracting the contours from the image, and it works best on binary images. By setting these contours around the digit, the prepare function can resize the digit itself instead of resizing the digit and its background. By focusing on just resizing the digit, the ML model can easily predict the class of the digit WITHOUT dealing with the extra noise in the reshaped image (such as having more black background surrounding the digit).

Non-MNIST Image WITH application of bounding box code



Non-MNIST Image WITHOUT application of bounding box code

Just from comparing these two images, it is clear that the digit 4, which had our bounding box code applied to it, has less background or noise, is distinctly focused on the digit, has sharp contrast between the digit and background, and has thicker font for the digit. Meanwhile, the second digit 4, which did NOT have our bounding box code applied to it, is blurry, has a lot of background noise, dull contrast between the digit and background, and has a very thin font for the digit. So we concluded that our model will predict our expression images correctly IF we pass the images to the bounding box code first BEFORE sending them to the *prepare* function for the model.

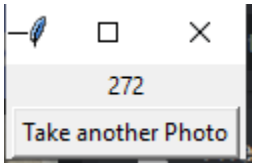
As for our GUI, we focused today on combining our calculator scripts with our output display on the second screen. In order to do this, the first thing that we did was to modify our calc_scripts file by encompassing all of its content in a function (we called it calculation) with its parameter being the equation string. At the end of the function, we modified our series of if statements in order to return the final result of the calculation.

```
51      if symbol == '+':
52          return(add(int_numA, int_numB))
53
54      if symbol == '-':
55          return(subtract(int_numA, int_numB))
56
57      if symbol == '*':
58          return(multiply(int_numA, int_numB))
59
60      if symbol == '/':
61          return(divide(int_numA, int_numB))
62
63      if symbol == '^':
64          return(exponent(int_numA, int_numB))
```

Next, back in our code for our second screen, we temporarily defined our equation string and ran it through our calculation function to get our solution. Then, we adjusted the existing code in the label to display the solution instead of a string.

```
1   import tkinter as tk
2   from Calc_Scripts import calculation
3
4   equationString = "25+247"
5   solution = calculation(equationString)
6   print(solution)
7
8   window = tk.Tk()
9   answer_label = tk.Label(text = solution)
10  answer_label.pack()
11
12  return_button = tk.Button(text = "Take another Photo", command = lambda: controller.show_frame(StartPage))
13  return_button.pack()
14
15  window.mainloop()
```

As a result, we got the following in our new display window:



Evidence of Progress

Research:

- ❖ Extract bounding box and save it as an image

- ➢ https://stackoverflow.com/questions/13887863/extract-bounding-box-and-save-it-as-an-image
- ❖ Text Detection and Extraction using OpenCV and OCR
  - ➢ https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr/
- ❖ Text Detection: Getting Bounding boxes
  - ➢ https://stackoverflow.com/questions/50000302/text-detection-getting-bounding-boxes/50004340
- ❖ Sort images after ROI (Python, OpenCV)
  - ➢ https://stackoverflow.com/questions/45000939/sort-images-after-roi-python-opencv
- ❖ Python | Thresholding techniques using OpenCV | Set-1 (Simple Thresholding)
  - ➢ https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/
- ❖ Image Thresholding — OpenCV-Python Tutorials 1 documentation
  - ➢ https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html
- ❖ OpenCV & Python – The Otsu's Binarization for thresholding – Meccanismo Complesso
  - ➢ https://www.meccanismocomplesso.org/en/opencv-python-the-otsus-binarization-for-thresholding/

Signatures	
	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 5/20/2021

Goals	
Long Term: - Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	**Short Term:** - Create flowchart of overall program - Use number classification in addition to our display window - Begin training our ML model on operators and symbols

Progress

Today's Progress:

Today, as we are beginning to finalize our programs and have them interact in order to perform our calculation from start to finish, we decided to create a flowchart of our program, detailing how information would get from one step to the next.

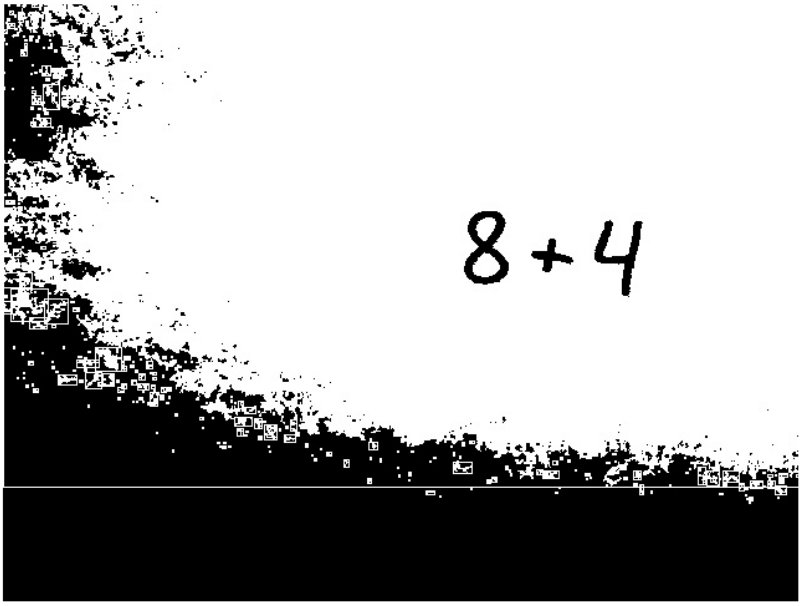
Flowchart of how we plan for our programs to interact:

Run GUI Screen #1 (Capture & save image)
Send image to bounding box code (Separates numbers and operators)
Prepare function (Alters images for ML Model)
ML Model (Classifies numbers within the image and appends predictions to an empty string)
Send expression string to solution window code
Run calculator scripts function (determines solution to the expression)
Display the solution on Screen #2

From here, we decided that the next step we wanted to take in finalizing our code was to combine our image capturing program with our ML model as we had both parts functioning independently at the moment. However, the first issue we ran into here was that we had not yet trained our ML model on our dataset of symbols and operators, so we decided to get that started. Since it can take a few hours to load though, we simultaneously started the process of sending the saved image from our first GUI screen to our bounding box code.

When capturing our image in the first screen, the first inconvenience we ran into was with aligning our equation with the camera so that the only bounding boxes drawn would be around the equation, not any objects in the camera feed's background. Because of this constraint, the paper had to be held very close to the camera (we are running our code on Kara's computer which does not have an external camera) which was mounted on the screen

and difficult to do. However, even when the paper covered the entire frame, due to issues with back lighting on the paper, we were getting an inconsistent background when inverting the colors (Our model is more accurate with white text on a black background) as shown below.

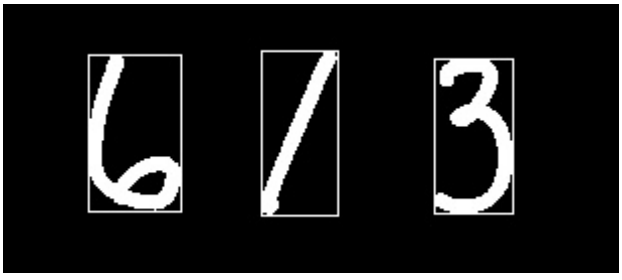


Our solution with this was to write the equation on an iPad, hold the iPad screen up to the computer's camera, and see if that improved the backlighting - which it did. However, because the iPad and the computer screen are both lit, the brightness had to be turned down on both in order to reduce glare and blurriness, making it more difficult to take a picture. We will have to resolve this lighting issue later since we want to at least connect the bounding box code to our *prepare* function and ML model.

By following our flow chart above, we first imported our bounding box code into our script containing the *prepare* function and our compiled ML model.

```python
def box():
    img = cv2.imread('box_expression_5.jpg', 0)        #return to Capture_Image
    path = r"final_images"

    cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU, img)
    image, contours, hier = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    sorted_ctrs = sorted(contours, key = lambda ctr: cv2.boundingRect(ctr)[0])
    ROI_number = 0

    for c in sorted_ctrs:
        # get the bounding rect
        x, y, w, h = cv2.boundingRect(c)
        ROI = img[y:y+h, x:x+w]
        cv2.imwrite(os.path.join(path, 'ROI_{}.jpg'.format(ROI_number)), ROI)
        # draw a white rectangle to visualize the bounding rect
        cv2.rectangle(img, (x, y), (x + w, y + h), 255, 1)
        ROI_number += 1

    cv2.drawContours(img, contours, -1, (255, 255, 0), 1)
    cv2.imwrite("output_box_image.jpg",img)

    return 0
```

We turned our bounding box code into a python function so it would be easier to import and call this code in our script with the *prepare* function and ML model. Overall, this code creates a bounding box around each digit and operator in an expression image and saves each bounded boxed region (ROI or Region of Interest) to an empty folder called “final_images”. Once we fix our issue with the images taken by our GUI, we will replace the “box_expression_5” image to “Capture_Image”, which is the saved photo taken by the GUI’s camera feed. The following ROI images are saved to the “final_images” folder after utilizing our “box_expression_5” image, as shown below.



box_expression_5

ROI_0 ROI_1 ROI_2

Next, we imported our bounding box function into our script with the *prepare* function and ML model.

```
import bound_box

CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

# Calling bound_box function
bound_box.box()
prepared_img_list = []
final_predict_list = []

# (BEST PREPARE METHOD TO USE)
# Formatting image for model
def prepare(path):
    try:
        IMG_SIZE = 28
        img_array = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        print('Orignal Image Array Shape:', img_array.shape)
        new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
        new_array = new_array.astype('float32')
        new_array = new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
        print('Resized Grayscale Image Array for Model:', new_array.shape)
        return new_array
    except Exception as e:
        print(str(e))

# Compile model
model = tf.keras.models.load_model('gray_model.h5', compile = True)
```

We then called our bounding box function box() that saved the previous ROI images to the “final_images” folder and created two lists. The prepared_img_list will contain the image array created from our *prepare* function. The final_predict_list will contain the final model predictions of each image. Our *prepare* function was then defined and our model was compiled.

```
# Switch directory to folder with boxed images from box()
directory = r'final_images'

# Iterate through images in folder and convert to arrays using prepare()
# Add image arrays to empty list
count = 1
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    if os.path.isfile(f) and filename.endswith('.jpg'):
        print("Image #", count)
        count += 1
        prepared_image = prepare(f)
        prepared_img_list.append(prepared_image)


# Iterate through image array list and predict each image
# Add image prediction to another empty list
for i in prepared_img_list:
    prediction = model.predict(i)
    pred_name = CATEGORIES[np.argmax(prediction)]
    final_predict_list.append(pred_name)

# Convert String List to Integer List
final_predict_list = [int(i) for i in final_predict_list]
print(final_predict_list)

# Remove ALL image files from folder
for filename in os.listdir(directory):
   os.remove(os.path.join(directory, filename))
```

Next, we switched to the directory of the “final_images” folder since it contained the bounded boxed digit and operator images. While in the folder, we iterated through the images in order to apply our prepare function to each image and save the outputs of the *prepare* function (image arrays) to the prepared_img_list. Then we iterated through the prepared_img_list in order to send each image array to our ML model, make a prediction, and append the prediction to the final_predict_list. The predictions within the list were then converted from type String to type Integer and printed to the command prompt. Lastly, we removed all the bounded boxed images from the “final_images” folder so that when a new expression is used, the model will only make predictions on the digits and operators of the CURRENT expression image that was sent through the bounding box function.

```
Image # 1
Orignal Image Array Shape: (78, 46)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (82, 38)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (77, 39)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
[6, 4, 3]
```

ROI_0 ROI_1 ROI_2

In the command prompt, the original array shape and resized array shape are presented for each ROI image, along with the final_predict_list. Image #1 and #3 were correctly classified, but Image #2 was not since the ML model was trained on digits, not operators yet.

Evidence of Progress

Research:

- W3School on Python Functions
  - https://www.w3schools.com/python/python_functions.asp
- Delete all files in a directory in Python – Techie Delight
  - https://www.techiedelight.com/delete-all-files-directory-python/
- How to import other Python files?
  - https://stackoverflow.com/questions/2349991/how-to-import-other-python-files
- Python | Converting all strings in list to integers
  - https://www.geeksforgeeks.org/python-converting-all-strings-in-list-to-integers/
- How to invert colors of image with PIL (Python-Imaging)?
  - https://stackoverflow.com/questions/2498875/how-to-invert-colors-of-image-with-pil-python-imaging
- OpenCV: Image file reading and writing
  - https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Friday, 5/21/2021

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Improve Log entries to better represent our progress - Create and complete our fifth project presentation - Gather the necessary research we've completed to discuss in our presentation - Establish our future plans

Progress

Today's Progress:

Throughout the weekend and the weekdays prior to May 25, we continued to work on our project presentation. This presentation mainly focuses on fixing code we have written so far for our ML model, bounding box images, calculator scripts, and user interface. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we improved the model predictions of MNIST dataset images, adjusted bounding box code that will be used to save and send images to our *prepare* function so that it would account for symbols with multiple parts (ex division symbol ÷), and adjusted code for our calculator scripts and GUI in order to integrate them. We then developed our future plans to show our advisors that we will continue to make progress in our project, such as including symbols and operators as items to be classified, completing our user interface, and connecting our user interface to our bounding boxes and ML code. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (5 minutes max) was more doable this time, mainly because we had less explicit code to explain as much of our work was conceptual. We

were able to present our slides in about 5 ½ minutes while conveying all the necessary information to our advisor and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

Fifth Presentation Link:

- ❖ Mascillaro & Pietrowicz Project 5/25/2021

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 5/27/21

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Create our own operator dataset (addition, subtraction, multiplication, and division) - Attempt to combine MNIST dataset with an operator dataset

Progress

Today's Progress:

Last project cycle, we improved our ML model's prediction on the digits 0-9, but since our model was only trained on digits, it still wasn't able to correctly classify mathematical operators. So, to start off our last project cycle, we decided to locate a simple operator dataset, combine it with our MNIST dataset, and train our model on the combined dataset. About a month ago, we found a dataset that contained basic mathematical operators, such as addition, subtraction, multiplication, and division operators. However, even though we thought this operator dataset would work for us, we realized later on that our model will

correctly predict digit images that are written in white handwriting on a black background since the model is trained on images with a black background with white handwriting. So the following images from the operator dataset we found would “confuse” our model if we trained it with these images. If we tried to invert the colors of these images, we would most likely lose important features of the images and result with a very blurry image, which we have noticed in the past when attempting to invert image colors.

We attempted to find a different operator dataset with white handwriting on a black background, but we turned up empty-handed. Instead of wasting time looking for an operator dataset, we decided to create our own dataset. First, we created 30 different images of a handwritten addition sign (five of them are shown below).

Before, we had to adjust the training image data for each handwritten digit class to solve our imbalance data problem early on in the year. So instead of training our model on 60,000 images, we trained the model on 54,000 images, where there are 5,400 images in each handwritten digit class. To keep our data balanced, we would need 5,400 operator images for

addition, subtraction, multiplication, and division. After creating our 30 images, we copied each one and reproduced 180 images in order to create 5,400 operator images to train our model. After combining our datasets, we trained our model and predicted the class for each digit and operator in a simple expression (8 + 4). However, our predictions were drastically incorrect: predicted expression = 8 4 3. Not only was the digit 4 incorrectly predicted as the digit 3, but the addition operator was predicted to be the digit 4. After researching possible causes for this issue, we concluded that since the addition operator images are not 28 x 28 pixels, like the other digit images, the model was not able to properly classify the operators or digits because of the differently sized images in the training dataset.

Next project block, we will try to adjust the size of each addition operator image to 28 x 28 pixels, finish our operator dataset, and train our model.

Evidence of Progress

Research:

- Python Image Processing in Python with Pillow
  - https://auth0.com/blog/image-processing-in-python-with-pillow/#Resizing-Images

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 6/1/21

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Fix problem with training our ML model on the combined MNIST and operator dataset

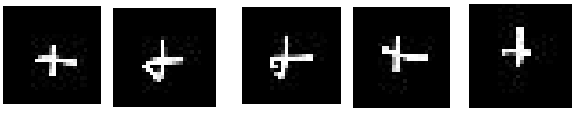
Progress

Today's Progress:

As you can see, each operator image we created is not the same size, but in our MNIST dataset, every digit image is 28 x 28 pixels. While you can train an ML model on different sized images, it is always better for every image to be the same size so the model's predictions are more accurate. Last project block, we already proved that the model can't predict classes well when trained on differently sized images. During this block, we wrote a python script that would iterate through the 5,400 addition operator images we created and resize them all to be 28 x 28 pixels.

```
DATADIR = r"MNIST_Dataset_JPG_format\Operators"
CATEGORIES = ["Plus_two"]   # names of photo files

total_array = []

dir_plus = r"MNIST_Dataset_JPG_format\Operators\resized_plus_two"
dir_minus = r"MNIST_Dataset_JPG_format\Operators\resized_minus_two"
count = 0

for category in CATEGORIES:
  path = os.path.join(DATADIR, category)
  for filename in os.listdir(path):
    if filename.endswith(".jpg"):
        try:
            img_array = cv2.imread(os.path.join(path,filename))
            new_array = cv2.resize(img_array, (28, 28))
            total_array.append(new_array)
        except Exception as e:
            print(str(e))

os.chdir(dir_minus)

for i in total_array:
    image = Image.fromarray(i)
    image.save('test_{}.jpg'.format(count))
    count += 1
```

First, we specified the directory of our addition operator images and the folder that contained the images (Plus_two). We also create an empty array, a count variable, and a variable to represent the directory of an empty folder that would hold all the resized addition operator images (dir_plus). Within a for loop, we joined the directory of the Plus_two folder with all the plus operator images, converted each image to an array, created a new resized array that's 28 x 28 pixels, and added the new array to the empty array total_array. Then we changed the directory of the folder containing the addition operator images to the directory of the empty folder for resized images. Using another for loop, we iterated through total_array, converted each resized array to an image, and saved the resized image within the folder.



After resizing the images, you can see that each operator image is now the same size. We repeated this process with 5,400 subtraction operator images we reproduced from the 30 images we created. After iterating the minus images through our code, we added both the addition and subtraction operator images to the MNIST dataset and trained our ML model.

However, we received an error explaining that we couldn't convert the string "+" or "-" to an integer. Before we added the operators to our MNIST dataset, our model would classify a digit using one of the classes in the following string array.

```
CATEGORIES_digit = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

Numbers that are written as strings can be converted to integers, but mathematical operators can't. We wanted to convert everything to an integer so we could give the predicted expression to our calculator script without any variable type problems. For now, we ignored this problem temporarily and printed the model's prediction. The following table displays our model's results after being trained on digit and operator images (all 28 x 28 pixels in size).

Inputted Expression	Predicted Expression
8 + 4	**8 8 3**
5 - 2	**3 + 2**
82	**82**

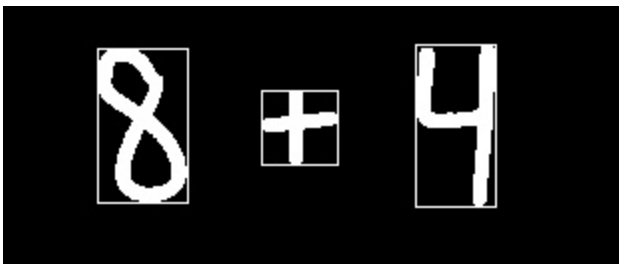
54	**54**
63	**63**
79	**39**

The green highlighted rows were correctly predicted, but the red underlined numbers and operators were incorrectly predicted. For the most part, when there are no operators within the inputted expression, the model almost correctly predicts each digit. For expressions with operators, the operators are either classified as a digit or as an incorrect operator. When we trained our model, we received 99% validation accuracy, which means 99% of the testing images given to the model were correctly predicted.

Since we noticed that sometimes the operator was classified as a digit, we thought it would be better to create an entirely separate ML model to be trained on just operators. So we copied our ML model for MNIST digits, changed the directories and classifications to be just for operators, and trained the new ML model for operators. We still got 99% validation accuracy, but now we had the problem of how to use two models (one for digits only and one for operators only) to predict digits and operators in one expression.

When we were only working with one model that predicted just digits, we would send a picture of an expression to our Bounding Box code. This script drew rectangles on the expression image to enclose the ROI (Region of Interest), which would include either a digit or operator. Then the new image is saved to a specific path with a new name, and a white rectangle is drawn on that image to visualize the bounding box or rectangle. The saved images of the individual digits and operators were then sent to our *prepare* function in order to format our images to feed into the model. In order to get the best accuracy, we want our images to have the same color and dimensions as the training images, so we converted our saved images to grayscale and resized them to be 28x28 pixels. Then we created and reshaped a pixel array of the image for the model to utilize to develop its prediction.

Now that we are working with two models, one for digits only and one for operators only, we have to somehow specify that each image is either an operator or a digit. As we looked at the saved images from our bounding box code, we noticed that the height of the operator image was much smaller than the height of our digits.



Output_box_image ROI_0 ROI_1 ROI_2

Since the height of the operator is always smaller than the height of the boxed digit, we decided to create an array of each image's height in our bounding box function.

While the height of the addition operator is 37 pixels, the height of 8 and 4 are 77 and 81 pixels, respectively. Since we call our bounding box function in our python script with the prepare function and model prediction, we had our bounding box function return the array of heights.

```
CATEGORIES_digit = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
CATEGORIES_operator = ["+", "-"]

# Calling bound_box function
box_heights = bound_box.box()
prepared_img_list = []
final_predict_string = " "
final_predict_list = []

# (BEST PREPARE METHOD TO USE)
# Formatting image for model
def prepare(path):
    try:
        IMG_SIZE = 28
        img_array = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        print('Orignal Image Array Shape:', img_array.shape)
        new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
        new_array = new_array.astype('float32')
        new_array = new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
        print('Resized Grayscale Image Array for Model:', new_array.shape)
        return new_array
    except Exception as e:
        print(str(e))

# Compile model
model_digit = tf.keras.models.load_model('gray_model.h5', compile = True)
model_operator = tf.keras.models.load_model('gray_operators_model.h5', compile = True)
```

First, we created two separate class arrays and compiled both of our models, one for digits and one for operators. Our *prepare* method remained the same.

```
# Switch directory to folder with boxed images from box()
directory = r'final_images'

# Iterate through images in folder and convert to arrays using prepare()
# Add image arrays to empty list
count = 1
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    if os.path.isfile(f) and filename.endswith('.jpg'):
        print("Image #", count)
        count += 1
        prepared_image = prepare(f)
        prepared_img_list.append(prepared_image)

# Iterate through ordered image array list (left to right just like we read math) and predict each image
# Depending on height of orignal boxed image, we either use digit model or operator model
# Add image prediction to another empty list
h_count = 0
for i in prepared_img_list:
    if box_heights[h_count] <= 45:
        prediction = model_operator.predict(i)
        pred_name = CATEGORIES_operator[np.argmax(prediction)]
        #final_predict_string += pred_name
        final_predict_list.append(pred_name)
        print("Model's Prediction:", pred_name)
        h_count += 1
    else:
        prediction = model_digit.predict(i)
        pred_name = CATEGORIES_digit[np.argmax(prediction)]
        #final_predict_string += pred_name
        final_predict_list.append(pred_name)
        print("Model's Prediction:", pred_name)
        h_count += 1
```

We iterated through the saved images from our bounding box code, applied our prepare function to each image, and saved the resulting image arrays in our prepared_img_list. Next, we iterated through the prepared_img_list we created to predict the class of each image using our models. Since our boxed images correlated to the heights in our box_heights array, we could move through the array from left to right, just like we read a mathematical expression, without worrying about whether or not this is the correct height for the saved image. If the height value was less than 45 pixels, we used the operator model to predict the image. Otherwise, we used the digit model to predict the image. The following table displays our two models' results.

Inputted Expression	Predicted Expression
8 + 4	8 + 5
5 - 2	5 + 2
82	82
54	55
63	63
79	39

Some of the digits and operators were still incorrectly classified, but we believed that this was due to how closely the bounding box was drawn around each digit/operator. Some features of the handwritten digit/operator were properly cut off when the bounding box was drawn, so we decided to increase the size of the bounding box around the digit/operator. However, this does NOT affect the heights in our box_height array. The heights within that array were measured BEFORE the bounding boxes were adjusted. That way we can get an accurate height for each operator and digit. The following table display's our two models' results AFTER we increased the size of the bounding box around each digit/operator.

Inputted Expression	Predicted Expression
8 + 4	8 + 4
5 - 2	5 + 2
82	82
54	54
63	63

79	79

After increasing the size of the bounding box, the models correctly classified all the digits and addition operators. The subtraction operator is still not correctly predicted, and so we will have to fix that later.

Evidence of Progress

Research:

- Saving a Numpy array as an image
  - https://stackoverflow.com/questions/902761/saving-a-numpy-array-as-an-image
- Save NumPy Array as Image in Python
  - https://www.delftstack.com/howto/numpy/save-numpy-array-as-image/#use-the-image.fromarray-function-to-save-a-numpy-array-as-an-image
- How to Get and Change the Current Working Directory in Python
  - https://linuxize.com/post/python-get-change-current-working-directory/
- Get and change the current working directory in Python
  - https://note.nkmk.me/en/python-os-getcwd-chdir/
- CV2 Image Error: error: (-215:Assertion failed) !ssize.empty() in function 'cv::resize'
  - https://www.py4u.net/discuss/230433
- Python - Functions
  - https://www.tutorialspoint.com/python/python_functions.htm
- BoundingBox Objects — Planar v0.4 documentation
  - https://pythonhosted.org/planar/bbox.html
- Math operations from string
  - https://stackoverflow.com/questions/9685946/math-operations-from-string

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 6/3/21

Goals	
Long Term: - Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	**Short Term:** - Adjust Bounding Box code to create one rectangle around entire division operator - Test the operator model's prediction on the division operator

Progress

Today's Progress:

Today, we spent the class period focusing on creating a solution to combine the three bounding boxes around the division sign into one bounding box. In order to do this, we used the OpenCV function groupRectangles(). The groupRectangles() function has three arguments which we used: the list of rectangles, the group threshold, and the eps, where the group threshold specifies the number of overlapping rectangles necessary to combine the rectangles and the eps specifies the amount of space allowed between rectangles to consider them close enough to combine. In our case, since we noticed that we only need to combine our bounding boxes vertically in order to create one division symbol bounding box, we decided to increase our y coordinates by 15, used a group threshold of 1, and an eps of 300*.

```
for c in sorted_ctrs:
    # get the bounding rect
    x, y, w, h = cv2.boundingRect(c)
    box_heights.append(h)
    rect = [x, y, w, h] # NC
    rectangles.append(rect) # NC
    shifted_rect = [x,y+dt,w,h]
    rectangles.append(shifted_rect) # NC
    intersection = shifted_rect[1] - rect[1]
    if intersection < 0:
        # Join the two rectangles
        rectangles, weights = cv2.groupRectangles(rectangles, 1, 300)
        rectangles.remove(rect)
    rectangles.remove(shifted_rect)
    ROI = img[y-15:y+h+15, x-15:x+w+15]
    cv2.imwrite(os.path.join(path, 'ROI_{}.jpg'.format(ROI_number)), ROI)
    # draw a white rectangle to visualize the bounding rect
    cv2.rectangle(img, (x-15, y-15), (x + w + 15, y + h + 15), 255, 1)
    ROI_number += 1
```

When we ran this code, we found that we were getting three bounding boxes for the division sign where one of the boxes was the full sign and two of the boxes were partial signs (upper dot & center line and lower dot & center line). We were able to fix this by deleting the two partial signs after they were combined to create the full division symbol.


ROI_1


ROI_2


ROI_3

Evidence of Progress

Research:

- How to join nearby bounding boxes in OpenCV Python
  - https://stackoverflow.com/questions/55376338/how-to-join-nearby-bounding-boxes-in-opencv-python
- OpenCV how to Group Rectangles
  - https://stackoverflow.com/questions/7109267/opencv-how-to-group-rectangles
- Grouping Rectangles into Click Points - LearnCodeByGaming.com
  - https://www.learncodebygaming.com/blog/grouping-rectangles-into-click-points
- Grouping Rectangles into Click Points - OpenCV Object Detection in Games #3
  - https://www.youtube.com/watch?v=m1pbF9BW8tA

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 6/8/21

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Combine GUI with Bounding Box code for our models to predict the class on the inputted image

Progress

Today's Progress:

This project block, we finished creating the operator dataset for addition, subtraction, multiplication, and division. We trained our digit model and operator model and various expression images to see if our models could correctly predict each digit and operator. We also imported different python scripts within certain files so we could run one file within the command prompt instead of us constantly retyping each file related to our model predictions and calculations. Using our flowchart of how our programs interact, we were able to determine which files had to be imported in certain python scripts in order to be compiled.

interface.py
Run GUI Screen #1 (Capture & save image)
gray_three.py
Import **bound_box.py** to send captured image to bounding box code (Separates numbers and operators)
Prepare function (Alters images for ML Model)
ML Model (Classifies numbers within the image and appends predictions to an empty string)
solution_window.py
Send expression string to solution window code
Import **calc.py** and run calculator scripts function (determines solution to the expression)

Display the solution on Screen #2

In the above chart, we first compile our GUI interface script, which captures and saves an image using the computer's camera. At the end of the interface code, we import gray_three.py, which imports our bounding box code to separate and save digit/operator images from the GUI's captured image. It also contains a *prepare* function and ML models to alter the separated image from the bounding box code and make predictions. At the end of our gray_three code, we import solution_window.py, which sends the predicted expression string to the imported calculator scripts function, determines the expression's solution, and displays the solution in a different window. **For now, because it is difficult to take a picture with the computer's camera, we are just inputting an expression image in the bounding box code without using the GUI. Next project block, we will use an external camera to take a picture and have it sent to the bounding box code.**

The following command prompt results display how well our models predicted our expressions and how the calculator scripts developed a solution from the output expression string from our models.

Inputted Expression: 8 + 4

```
Image # 1
Orignal Image Array Shape: (97, 65)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (57, 58)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (101, 60)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: 8
Model's Prediction: +
Model's Prediction: 4
8+4
12
```

Here, the models correctly classified each digit and operator in the inputted expression, and our calculator scripts correctly calculated the solution.

Inputted Expression: 5 - 2

```
Image # 1
Orignal Image Array Shape: (83, 56)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (28, 54)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (89, 76)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: 5
Model's Prediction: +
Model's Prediction: 2
5+2
7
```

Here, the models correctly classified the digits but not the operators in the expression, and the calculatorscripts correctly calculated the solution even though the predicted expression string was incorrect.

Inputted Expression: 7 x 1

```
Image # 1
Orignal Image Array Shape: (97, 63)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (59, 54)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (85, 29)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: 3
Model's Prediction: +
Model's Prediction: 3
3+3
6
```

Inputted Expression: 7 + 1

```
Image # 1
Orignal Image Array Shape: (105, 84)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (58, 60)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (85, 29)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: 3
Model's Prediction: +
Model's Prediction: 8
3+8
11
```

Above, the models' predictions were incorrect. 7 x 1 was predicted to be 3 + 3, and 7 + 1 was predicted to be 3 + 8. Focusing on just the operator model's predictions, it is clear that no matter what operator we inputted, the model would always predict an addition operator. This could be due to the fact that our operator dataset wasn't very diverse. Most operator images were created by reproducing the same operator images over and over. If we trained the model on more diverse images (create 5,400 different addition, subtraction, multiplication, and division operator images each), the model may have developed a better prediction.

For the digit model, we are not completely sure why some digits were identified correctly while others weren't. The process for writing machine learning algorithms is filled with trial and error. As all engineers/programmers do, we have to constantly tweak and adjust our algorithms and models. During this process, challenges arise, especially with handling data. When constructing a ML model, it was important for us to remember that real-world data is imperfect, various types of data require different approaches and tools, and there will always be tradeoffs when determining the proper model. It requires creativity, experimentation, and tenacity. Debugging ML models occurs in two cases:

- Our algorithm doesn't work
- Our algorithm doesn't work well *enough*

What's unique about machine learning is that it's 'exponentially' harder to figure out what is wrong when our algorithm doesn't work as expected. Very rarely does an algorithm work properly on the first attempt, so the majority of time spent developing a machine learning model is dedicated to tweaking and sometimes rebuilding the algorithm.

There's really no simple solution or explanation for why our digit model all of a sudden wasn't able to predict digits as well as it used to. Maybe when we tweaked our python scripts to

accommodate operators, something caused our digit model to develop incorrect predictions for some digits. We spent the rest of the block researching possible causes, but we couldn't find anything meaningful or helpful. If we have time before our presentation, we will attempt to fix this issue. However, from past project cycles, we have learned that fixing an issue with our algorithm can take a while.

Evidence of Progress

Research:

- How to Run One Python Script From Another
  - https://datatofish.com/one-python-script-from-another/
- Building a Handwritten Multi-Digit Calculator | by Neerav Gala
  - https://towardsdatascience.com/building-a-handwritten-multi-digit-calculator-f03cf5028052
- Machine Learning with Python: Training and Testing the Neural Network with MNIST data set
  - https://www.python-course.eu/neural_network_mnist.php
- Previous Article: OCR: Handwriting recognition with OpenCV, Keras, and TensorFlow
  - https://www.pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Thursday, 6/10/21

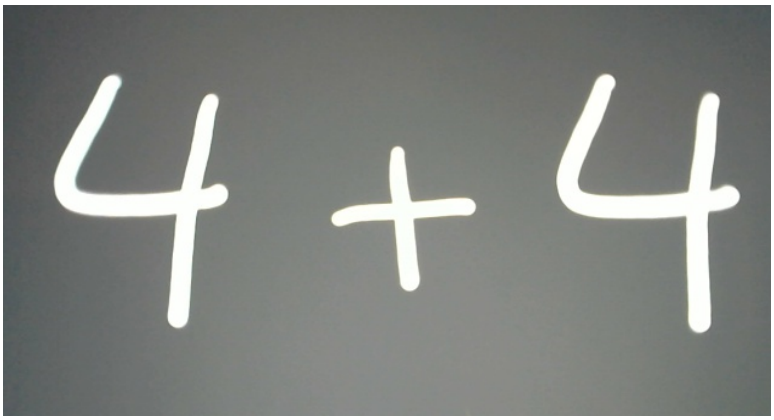
Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Attempt to use an external camera to take pictures using GUI for our models

Progress

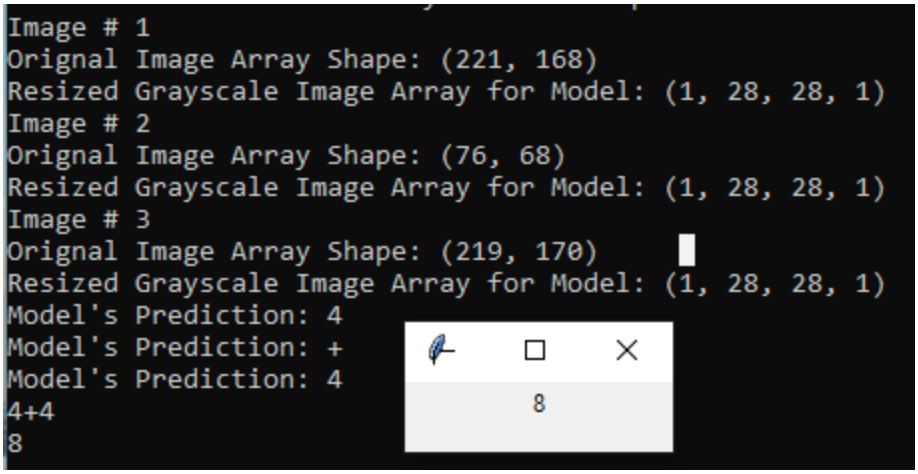
Today's Progress:

Today, we worked on testing our entire program with an external camera to capture an image of our expression. Going into today's project block, we felt confident about our predictions themselves, but we were concerned with how the lighting in the room we capture our image in would affect its visibility and therefore prediction. As it turns out, the lighting wasn't much of an issue as long as we took the images in a dark room in order to avoid glare. Before taking pictures with the external camera, we tried to figure out why some operator images were correctly classified while others were classified as digits. The problem was with our code that determined which model to use (digit or operator) when provided with an image based on the image's height. If the height of the image was less than 45 pixels, the operator model was used to classify the image. Otherwise the digit model was used for classification. We adjusted the height boundary to 60 pixels and began using the external camera.

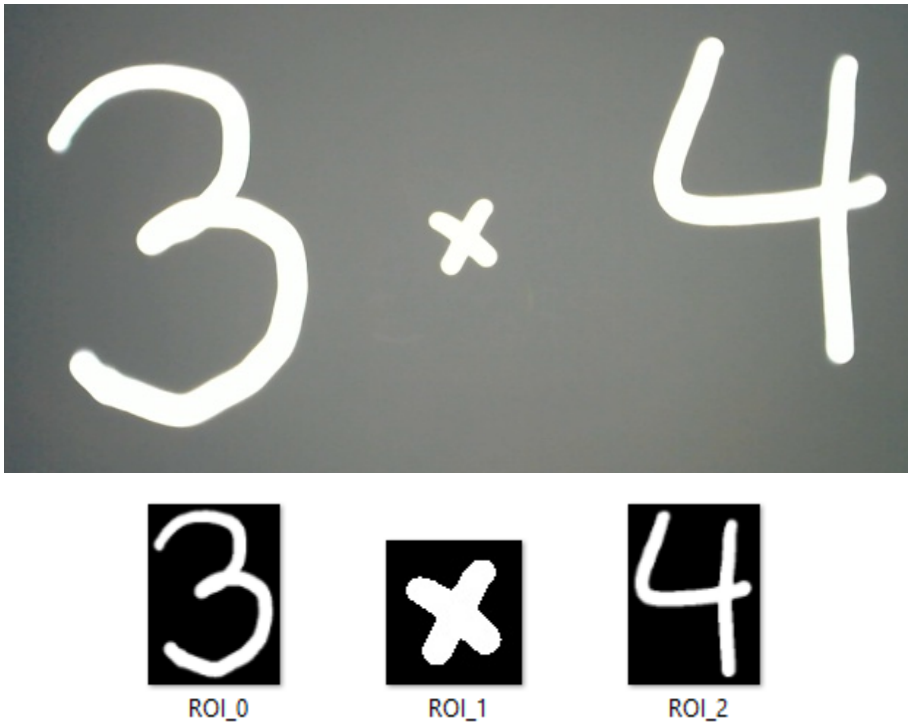


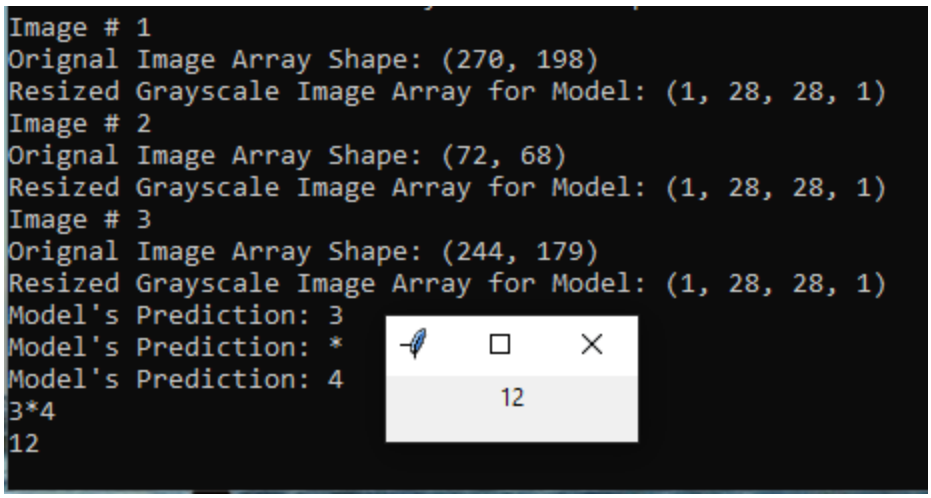
ROI_0 ROI_1 ROI_2

Above is our first attempt at predicting the digits and operators in a picture taken by an external camera AFTER fixing the height specification for the models. Even though the plus sign looks like it is the same height as the digits 4, it is actually less than 60 pixels tall while the digits are over 100 pixels tall (File explorer displayed the pictures incorrectly).

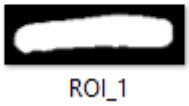


As you can see, once the height problem was fixed, the operator model was used to predict the operator image and succeeded. Also, our calculator and solution window scripts worked properly. You can see the solution window next to the model predictions.





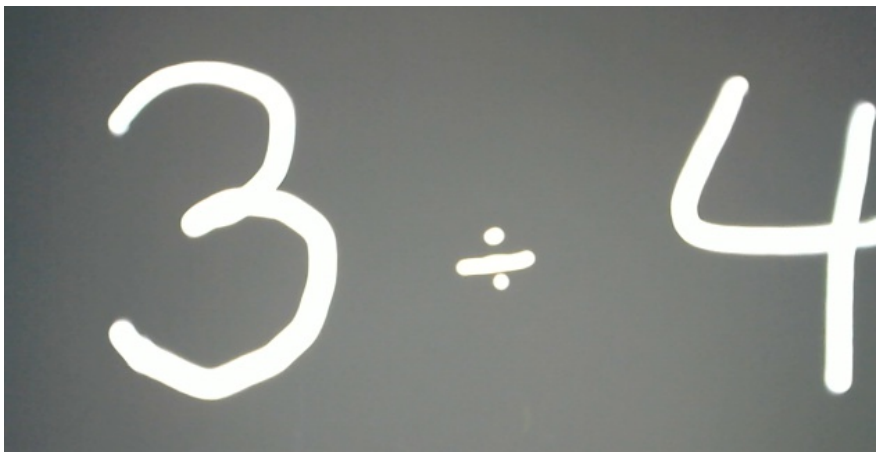
We then tried a multiplication expression and also succeeded with our models, calculator scripts, and solution windows. Our addition and multiplication symbols were properly predicted. However, we ran into issues with subtraction and division. For subtraction, we believe that the model wasn't predicting the operator correctly because all of our training images were square and therefore had significant blank space around the symbol while our testing images were rectangles (cropped closely to the symbol) without much blank space around them.



Testing Subtraction

Training Subtraction (Dataset)

As for division, we were able to get the model to predict the division symbol to be a division symbol, however, because of the way that we approached the group rectangles, we ended up with extra bounding boxes which we did not need, and these extra bounding boxes were consistently being predicted to be other operators, as shown below.




```
Image # 1
Orignal Image Array Shape: (264, 185)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 2
Orignal Image Array Shape: (49, 78)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 3
Orignal Image Array Shape: (33, 36)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Image # 4
Orignal Image Array Shape: (249, 160)
Resized Grayscale Image Array for Model: (1, 28, 28, 1)
Model's Prediction: 3
Model's Prediction: +
Model's Prediction: /
Model's Prediction: 8
3+/8
```

Because of the two images created for the division symbol, our operator model predicted two operators, one division and one addition symbol. We are not exactly certain why 4 was predicted to be an 8. What we have noticed is that if the digit or operator image is not the best (the image isn't slightly cut off or blurry), then the digit model usually incorrectly classifies the digit image.

Overall, our models can correctly classify digits or operators, but it has trouble with division symbols since multiple images are still created for one symbol. Also, the digit images may not be classified correctly if the picture taken using an external camera is not the best.

Evidence of Progress

Research:

- GIMP Downloads
  - https://www.gimp.org/downloads/

Github Repository:

- Spring Semester Handwriting Recognition Project
  - https://github.com/emascillaro/SeniorProject_Spring/tree/main/Python-Project-Handwritten-digit-recognizer/Handwritten%20digit%20recognizer

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***

Date: Tuesday, 6/15/21

Goals

Long Term:	**Short Term:**
- Write handwriting recognition code to identify and transcribe written text - Write four-function calculator scripts - Develop an app that utilizes all our code to solve mathematical problems	- Improve Log entries to better represent our progress - Create and complete our last project presentation - Gather the necessary research we've completed to discuss in our presentation

Progress

Today's Progress:

Throughout the weekend and the weekdays prior to June 16, we continued to work on our project presentation. This presentation mainly focuses on fixing code we have written so far for our ML model, bounding box images, calculator scripts, and user interface. We made sure that our presentation would satisfy all the elements on our project evaluation sheet and that our presentation skills are above average. We gathered the necessary photos of our code to clearly explain to our audience our progress, such as information relating to how we created and trained another ML model for our operator dataset, adjusted bounding box code that will be used to save and send images to our *prepare* function so that it would account for symbols with multiple parts (ex division symbol ÷), and adjusted code for our calculator scripts and GUI in order to integrate them. Finally, we added speaker notes to our presentation to ensure that we touch upon every detail we want to explain to our audience. Practicing how to present without an allotted amount of time (7 minutes max) was more doable this time, mainly because we had less explicit code to explain as much of our work was conceptual. We were able to present our slides in about 6 ½ minutes while conveying all the necessary information to our advisor and audience. Our log demonstrates all the progress we have made in the past few weeks even though our presentation focuses on certain aspects of our progress.

Evidence of Progress

Final Presentation Link:

- FINAL Mascillaro & Pietrowicz Project 6/17/2021
  - https://docs.google.com/presentation/d/1sxvUaBZA8lY3PvVfjbE8oVwVqKAt4E15MQN16SZSmqY/edit?usp=sharing

Signatures

	
Emma Mascillaro	***Kara Pietrowicz***