Identification of Handwritten Digits for Hereditary Cancer Test Request Forms

Brandon Northrup
Western Governors University
001177877

Letter of Transmittal

January 13, 2021

Richard Smith
Genetique
640 Gardner Village Way
Langer, MA, 02139

Dear Mr. Smith,

I am writing to you about an on-going issue involving human error in reading test request forms (TRFs). Too often, a number may be entered incorrectly, causing a patient to potentially be given improper treatment or not receive insurance coverage. This has become a problem for several different medical companies, but I believe Genetique would be a great place to start solving it.

An application that is capable of accurately reading handwritten digits would be able to read the numbers on TRFs for us, with the option of verification by humans. This application will be a standalone web application and will be built and tested within 6 months.

This solution would allow for your company to gain improved accuracy in patient results, which may lead to happier patients, improving the business as a whole. This type of application also comes with the potential to drastically lower human labor costs, depending on your company's preference on verification.

The total funding requirement for this project is estimated at $30,000 for development and integration, with an additional $1,200 yearly for maintenance. All tools and licenses that will be used are free.

My relevant academic history consists of a Bachelor's Degree in Computer Science, with multiple certifications from CompTIA, Microsoft, ITIL, and Udacity. I also have 8 years of experience in software development, 2 of those years being involved in machine learning.

Thank you.

Sincerely,

Brandon Northrup
Software Engineer
BNorthrup

Project Recommendation

Problem Summary

It is a relatively common occurrence that medical data is either entered into a system incorrectly or misinterpreted by humans, which has the potential for drastic consequences; not just for the patients, but for the companies as well. Many studies have been done to demonstrate this problem. “Error rates detected by the double-entry method ranged from 2.3 to 26.9%. Errors were due to both mistakes in data entry and to misinterpretation of the information in the original documents.” (Saveli I. Goldberg, Ph.D; Andrzej Niemierko, Ph.D; Alexander Turchin, MD, MS, 2008).

The consequences of incorrect data can range from something small, such as an employee spending a few seconds to a few minutes correcting a mistake that they notice, to something large, like a patient receiving improper, slowed, or uncovered treatment. This can impact a patient’s life and/or cost a company a large amount of time and money. “Forgetting to record test results in the EHR system slows down a patient’s treatment process and can spell trouble for audits and compliance.” (Chris Thompson, 2018)

Taking steps toward automation is an excellent way to help prevent these issues.

Application Benefits

This application will be referred to as the Digit Recognition App, or DRA for short. DRA, in conjunction with existing hardware and software, is considered an automated system. Automation is quickly becoming the norm for simple tasks, and there are many reasons it is becoming so popular. First off, automated systems are generally much more accurate than humans, and they will either stay consistent or get better over time, depending on whether or not machine learning is involved. This leads to increased throughput, which will allow for decreased labor costs. Company reputation is also very important, and there is no better way to improve it than to satisfy your customers. The improved accuracy and turnaround time will lead to happier customers and more business.

Application Description

DRA will use deep learning (a subset of machine learning) and computer vision to accurately recognize and identify digits 0 through 9. It will take an image that has been scanned in by an Accessioning team member, alter it using the model that has been built for it, and read the numbers. At this point, there are two options to proceed based on company preference. DRA can be changed to enter this number into a CRM application automatically, or an employee can verify that the number DRA predicted is correct.

It is typical for medical companies to have double data entry, as this often provides much more accurate data. Because of this, it is understandable that a company may not want to rely solely on one application to determine numbers in paperwork. However, using DRA as either the first or second enterer for a TRF can still save plenty of time and money.

Data Description

The data that will be used for the training of the deep learning model will come from the MNIST dataset. "The MNIST database (Modified National Institute of Standards and Technology database) of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. Additionally, the black and white images from NIST were size-normalized and centered to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels." (Wan, Li; Matthew Zeiler; Sixin Zhang; Yann LeCun; Rob Fergus, 2013)

As mentioned in the quote, the images that are processed are each a 28 by 28 pixel image. This gives us an image of a number that is a total of 784 pixels. These pictures being in grayscale allows for the model to learn based on what are referred to as "weights". These weights are based on how light or dark a pixel is, ranging in decimal format from 0 to 1. The model learns by creating an array of these pixels and comparing this array to the actual number in the image. To put it simply, the model keeps several different arrays in "memory" for each number to represent all of the different ways a number can be written. The trained model uses probability to predict future images that are passed to it. If an image's pixel array is closest to, say, a 7 that has been stored in the model's memory, then the model will predict that it is a 7.

Objective and Hypothesis

The objective of this project is to implement a system that will be able to effectively move the responsibility of reading and entering numbers on TRFs from humans to machines.

My hypothesis is that using the MNIST dataset with an efficient deep learning model will lead to a 99% or better accuracy, leaving relatively little room for error. This is considerably better than even double data entry, which is very rarely more than 97.7% accurate, and frequently under 95%. It is likely that the only time this trained model will not be able to accurately predict a number is in situations where even humans will stare directly at it and question what exactly they are looking at.

Methodology

Requirements for this project are straightforward and are not likely to change. Due to this, and seeing as this project will not take a particularly long time, using the traditional Waterfall methodology seems like it will be the best route.

Phase 1: Requirements

Requirements will be defined as specifically as possible. Because they are not likely to change, being strict with them will not prove to be a problem. Medical and legal requirements will be important to include.

Phase 2: Design

A blueprint that details the design of the system will be created. The design will include a diagram to map out the code, along with any graphical aspects that will be needed.

Phase 3: Development

Development of the model and an accompanying dashboard will begin.

Phase 4: Testing

The testing will be done by training the model using the MNIST dataset of 60,000 images compared with 10,000 testing images. This will give us the accuracy of the model that can be expected in a business environment. A small portion of Genetique's data entry team will test DRA using fake TRF images to verify the functionality and accuracy.

Phase 5: Deployment

DRA will be deployed to the rest of Genetique's data entry team.

Phase 6: Maintenance

Maintenance will be performed every three months, as well as on an ad-hoc basis. Improvements may be done in the future to include a wider variety of handwriting, such as letters and entire words.

Funding Requirements

- Development: $25,000
- Testing: $5,000
- Maintenance: $1,200 per year

Stakeholder Impact

This project is likely to have many benefits to stakeholders, without any expected detriment. Objectives and expected outcomes are extremely clear without much room for change, meaning that stakeholders will know what they are going to get right from the start. More accurate results, reduced labor costs, and reduced turnaround time are all expected. Data entry for the users of DRA will be made simpler. If everything goes as well as expected with the use of DRA at Genetique, external stakeholders in particular may also benefit from the deployment of it to other companies.

Data Precautions

No sensitive information is used in the dataset that will train DRA. DRA will be taking numbers that are written on TRFs that do contain sensitive information. These numbers will be cropped so that the rest of the TRF is not visible. Not only will the rest of the TRF not be visible, but DRA will be hosted locally using the company's intranet as an added layer of security. By default, DRA will not have access to any databases. All of this allows the application to comply with HIPAA regulations.

As with human error, there will occasionally be errors in the data that DRA predicts. DRA is expected to be more accurate than humans, meaning that no responsibility may be assumed when there is an error. Companies will have the option of using a form of double data entry where DRA assumes the role of either the first or second enterer, improving the accuracy to nearly 100%.

Developer Expertise

The developer has received a Bachelor’s in Computer Science, with multiple certifications from CompTIA, Microsoft, ITIL, and Udacity. They also have 8 years of experience in software development, 2 of those years being involved in machine learning.

Project Proposal

Problem Statement

There are many problems around the world when it comes to healthcare, and many of these problems stem from simple things such as mistakes by humans in data entry. My goal is to create a deep learning application that will be implemented with your existing hardware and software to begin automation of the TRF reading process. This application, called DRA (Digit Recognition App), will include a dashboard that can be used for analytics, including graphs, plots, and accuracy verification.

Customer Summary

The customers of DRA will initially be the Data Entry team at Genetique. This team is going to assist in the testing of the software, and will be the only team to use it until it is considered a success. If it is successful, DRA may be deployed to other companies and improved to include more features, which will allow the application to be applicable to other departments, thereby increasing the customer base.

The only necessary skills needed to use DRA will be to have the ability to scan images and read numbers. DRA will be as user-friendly as possible.

Existing System Analysis

Currently, the Accessioning team at Genetique uses a batch management system to handle the scanning-in of patient documents and assigning all of these documents into a single patient “case”. These are the documents that contain the numbers that DRA will be predicting.

With the format of the current TRFs being processed, the only page that we actually need to scan and predict for DRA to work successfully is page 2 of each set of documents. This is the page that includes both the patient cancer history and family cancer history. A model of the TRF will be included in DRA so that it knows where exactly we want the numbers to be pulled from. Once a case has been accessioned and is sent to the Data Entry department, DRA will read the numbers on page 2 and output the predictions above the numbers (or next to them, if necessary). If Genetique would prefer for DRA to automatically enter the numbers into their CRM, this may be arranged at a future date.

Data

The only data needed for DRA to function successfully is the MNIST dataset. This dataset consists of 60,000 training images and 10,000 validation images. The training images teach the model that DRA will be using, and the validation images are all images that the model has never seen before. These validation images are predicted by the model, and we are then given an accuracy rating.

This data does not require any collection beyond simply downloading a .csv file or even just including a built-in library in an IDE. MNIST is a public, commonly used dataset for machine learning. The data is not inherently usable, so it will need to be preprocessed. This is done by splitting the images into 28-pixel by 28-pixel image, with each image containing a single number. The images must also be transformed into a binary format, such as grayscale, so that the weights of the pixels in each image are simple to measure.

Project Methodology

Requirements for this project are straightforward and are not likely to change. Due to this, and seeing as this project will not take a particularly long time, using the traditional Waterfall methodology seems like it will be the best route.

Phase 1: Requirements

Requirements will be defined as specifically as possible so that the users get exactly what is needed. Because the requirements are not likely to change, being extremely specific with them will not prove to be a problem. Medical and legal requirements will need to be included. We will take input from the Data Entry team to see what they would like to see from DRA and its dashboard.

Phase 2: Design

A blueprint that details the design of the system will be created. The design will include an ERD to map out classes, methods, etc. along with any graphical aspects that will be needed.

Phase 3: Development

Development of the model and an accompanying dashboard will begin. The model will be built first, seeing as the dashboard is going to be based off of the model's results.

Phase 4: Testing

The testing will be done by training the model using the MNIST dataset of 60,000 images compared with 10,000 testing images. This will give us the accuracy of the model that can be expected in a business environment. A small portion of Genetique's data entry team will participate in acceptance testing using fake TRF images to verify the functionality and accuracy of DRA. Unit testing will be done on the dashboard in particular to verify that each portion of it is fully functional and correct. We will also perform integration testing to verify that DRA works properly with the existing systems.

Phase 5: Deployment

Upon successful testing, DRA will be deployed to the rest of Genetique's data entry team. The dashboard will keep track of the accuracy of DRA's predictions using real TRFs.

Phase 6: Maintenance

Maintenance will be performed every three months, as well as on an ad-hoc basis. Improvements may be done in the future to include a wider variety of handwriting, such as letters, special characters, or possibly entire words.

Project Outcomes

In-process deliverables will include:

- Full-fledged Schedule With Set Deadlines
- Testing Plans
- Dashboard Mockup

Final deliverables will include:

- Functional Predictive Model
- Functional and User-friendly GUI
- Analytical Dashboard
- Installation Guide
- User Guide

Implementation Plan

The plan for implementation is as follows:

- DRA will be added as an automated process in between Accessioning and Data Entry. It will receive a batch of documents from the Accessioning team's batch management system, crop the numbers from page 2, predict them, and then output the predictions for Data Entry.
- This process will be rolled out in two phases:
  - The first phase will be considered the beta test of this project. It will include 20% of both the Accessioning team and the Data Entry team to act as a pilot group. They will use fake TRFs to verify that both ends surrounding the DRA implementation function successfully.
  - The second phase will involve moving everyone from both teams to the new system. Accessioning's process will not actually change, and may not even provide a noticeable difference on their end.
- Accuracy will be measured over time to verify that DRA is remaining consistent and is improving results as expected.

Evaluation Plan

DRA will be evaluated through the accuracy measurements that will be provided in the dashboard. Analysts will also verify the increase in productivity by reviewing the company statistics. As for the patient results, this will be verified by sending out surveys to both doctors and patients, asking for their feedback on their treatment, insurance experience, overall satisfaction, and anything else Genetique would like to add.

Success is determined by DRA's prediction accuracy, ability to increase productivity, and the feedback received from the doctor and patient surveys. If any of these aspects are decreased, DRA will need to be modified.

Resources and Costs

Resources and costs associated with the development and deployment of DRA will be minimal. The following list goes over each of the technologies that will be used and the costs associated with them:

- Anaconda3 2020.11
  - Free
- Jupyter Notebook 6.2.0
  - Free
- Python 3.8
  - Free
- Keras-Applications 1.0.8
  - Free
- Keras-Preprocessing 1.1.2
  - Free
- Ipympl 0.6.3
  - Free
- Tensorflow 2.3.0
  - Free
- Voila 0.2.6
  - Free
- Voila-Material 0.4.0
  - Free
- Pandas 1.2.1
  - Free
- Scikit-learn 0.24.1
  - Free
- Seaborn 0.9.0
  - Free

It is very unlikely that the organization has not already provided employees with adequate resources to run this application. The model training is the only intensive part of DRA, and it only needs to be run once initially, and again after any model alterations. Employees will also not need to train the model themselves. Nearly any typical computer in the last 10 years would be able to easily run DRA. If employees have their own computers to work on, these items can be disregarded in the total cost. The following list includes the hardware and software that will be used to develop, train, and run DRA initially:

- Operating System: Windows 10 Pro, 64-Bit

- CPU: Intel Core i5-9600K
- GPU: MSI GeForce RTX 2060 6GB GDDR6 192-bit
- RAM: 32GB (2x16) Corsair Vengeance LPX 3200

Note: The items in the PC build were able to run the model in about 12 minutes without using the GPU. Using a GPU is generally recommended for Machine Learning, as it speeds up the process significantly. Older CPUs and GPUs may take a very long time to train the model.

As you can see, the primary cost will come from the labor of development and future maintenance. Below you will see the estimated costs:

- Human Resources: 6 months of development and testing
  - $30,000
- Maintenance:
  - $1,200 each year + ad hoc maintenance costs

Timeline and Milestones

#	Req.	Milestone	Resources	Start Date	End Date
1		Requirements	Project Manager Software Developer	January 12, 2021	January 16, 2021
2	1	High Level Design	Project Manager Software Developer UI/UX	January 17, 2021	January 24, 2021
3	1	Low Level Design	Project Manager Software Developer	January 17, 2021	January 25, 2021
4	2, 3	Model Development	Project Manager Software Developer	January 26, 2021	February 7, 2021
5	4	Model Training and Analysis	Project Manager Software Developer Data Analyst	February 8, 2021	February 11, 2021
6	2, 3	Dashboard and GUI Development	Project Manager Software Developer	January 26, 2021	February 15, 2021
7	4, 6	Dashboard and GUI Testing	Project Manager Software Developer QA	February 16, 2021	February 19, 2021
8	6	Integration Testing	Software Developer QA	February 20, 2021	February 23, 2021

9	8	First Rollout Phase	Project Manager Data Entry Team Accessioning Team	February 24, 2021	April 9, 2021
10	9	Second Rollout Phase	Project Manager Data Entry Team Accessioning Team	April 10, 2021	July 10, 2021
11	10	Maintenance Plan	Project Manager Software Developer	July 11, 2021	July 12, 2021

Application Details

Data Methods

The **descriptive data method** used is that of Principal Component Analysis (PCA). PCA is commonly used for data that can be viewed by humans easily in a two- or three-dimensional format, but the data does not have to be in that format to begin with. With the MNIST images, there are actually three dimensions after preprocessing due to the format of the arrays used to discern the images, but I chose to use two principal components for analyzing the data, as it is significantly easier to visualize.

Data can be reduced in dimension by using what is called “flattening”. This gets rid of any unnecessary dimensions for what you are trying to accomplish. In the case of the flattening of the model’s images, it takes the three-dimensional array and converts it into an array that holds just the pixel weight and its location in the image.

The information that comes from the flattening is plotted onto a graph. This graph (seen in the Data Visualization section) demonstrates the difference between the images of the numbers in the dataset. The two components are the variation between each number and their features, and the variation between the same number and the different features it may have, based on how a person writes it.

The **non-descriptive data method** used is that of a Convolutional Neural Network (CNN). CNNs can span several dimensions, though I chose to use a two-dimensional CNN since it is being used for two-dimensional images. The CNN is connected by neurons, and what becomes a neuron in this application is determined by the weight that a pixel in the image has. The higher the weight, the more likely that the pixel will become a relevant neuron.

The way that this is able to determine which numbers are which, and actually train the model to know this information, is through number characteristics. These characteristics include straight lines, curved lines, white space, corners, etc. The model will learn over time that certain numbers have certain features. The thing used to search for these features is called a filter, which is essentially an array of weights. The model learns to recognize these filters as they show up. These filters can even include all of the little nuances associated with writing numbers, such as writing a 7 with or without a line through the middle from left to right.

The way the model processes an image is by using various types of layers. There are many layers that can be used for a CNN, but we chose to use the following in sequential order:

- 1 Convolutional Layer with 32 filters and a 3x3 kernel size
- 1 Convolutional Layer with 64 filters and a 3x3 kernel size
- 1 Convolutional Layer with 128 filters and a 3x3 kernel size
- 1 Pooling Layer (2x2)
- 1 Dropout Layer (0.5)
- 1 Flatten Layer
- 1 Dense Layer (64 neurons)
- 1 Dropout Layer (0.2)
- 1 Dense (10 neurons)/Softmax layer

Using this CNN, the model can be trained over a given amount of epochs. An epoch is a run through all of the input data and back again. The amount of epochs you choose to run is up to you, but there should be a convergence point where the accuracy stops increasing. If you hit this point, it is generally best to stop running more epochs, as this has the potential to overfit the model. Dropout does help prevent overfitting, but that does not mean the accuracy will be able to keep increasing until 100%. It is extremely rare that any sort of CNN-based model will actually reach 100% accuracy with something as unpredictable (especially by a machine) as human handwriting.

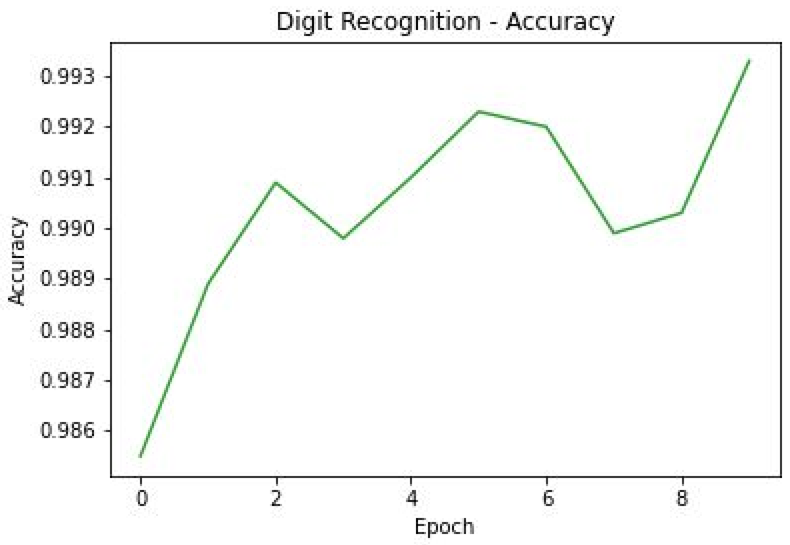
Datasets

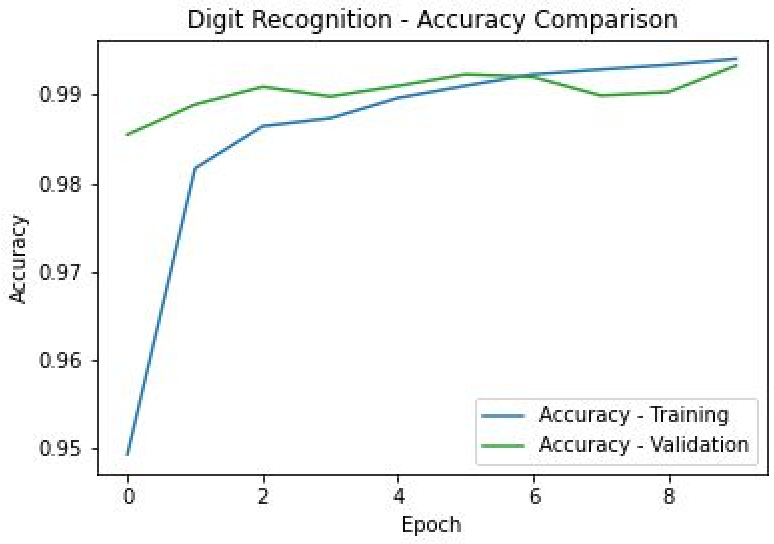
The only dataset used for this project is the MNIST dataset. This dataset consists of 60,000 training images and 10,000 validation images. "This database is well liked for training and testing in the field of machine learning and image processing. It is a remixed subset of the original NIST datasets. One half of the 60,000 training images consist of images from NIST's testing dataset and the other half from Nist's training set. The 10,000 images from the testing set are similarly assembled. The MNIST dataset is used by researchers to test and compare their research results with others." (Wan, Li; Matthew Zeiler; Sixin Zhang; Yann LeCun; Rob Fergus, 2013).

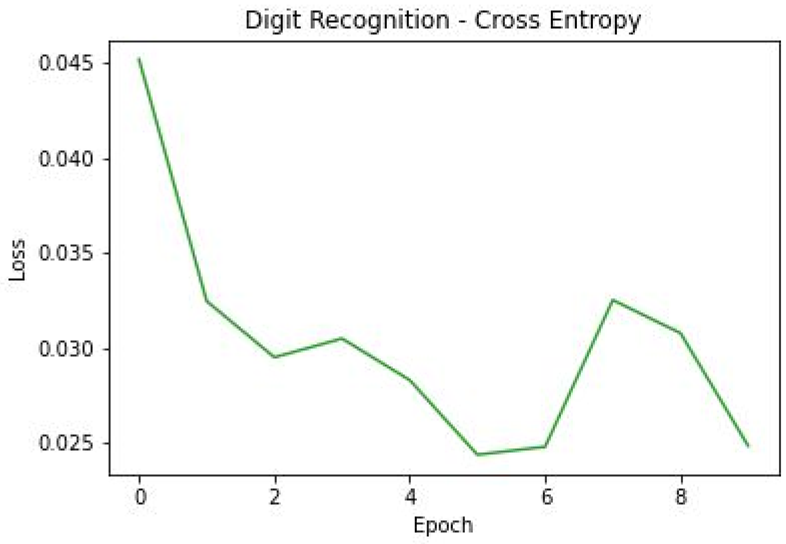
Analytics

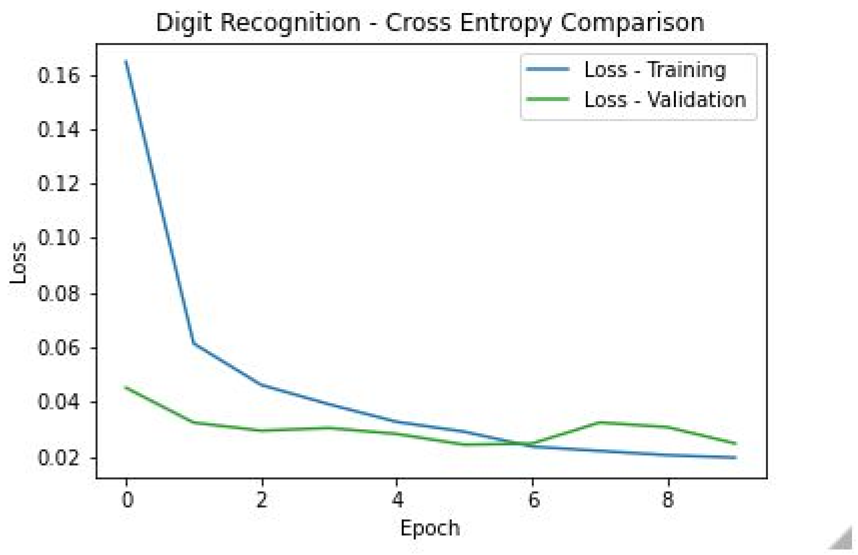
The accuracy and inaccuracy (called loss) of the model can be visualized in the four separate line graphs below. You will notice that the model started off with fairly low accuracy during the training portion of the first epoch, but during the first epoch of validation, it can be seen that it was quickly learning.

The model learned to recognize the training images more and more over time, which is a sign that the model may be overfitting. This is why Dropout was included in the CNN. Dropout is where the model essentially "forgets" some previous information so that it doesn't overfit, or get too used to recognizing only specific features. This makes it more likely to be able to adapt and recognize the nuances previously mentioned. You can see the effect of Dropout in the green lines in the graphs. Each time the accuracy seemed to get lower temporarily, it was simply Dropout and relearning taking place.




Digit Recognition - Accuracy Comparison
Accuracy
Epoch
Accuracy - Training
Accuracy - Validation
0.95
0.96
0.97
0.98
0.99
0
2
4
6
8

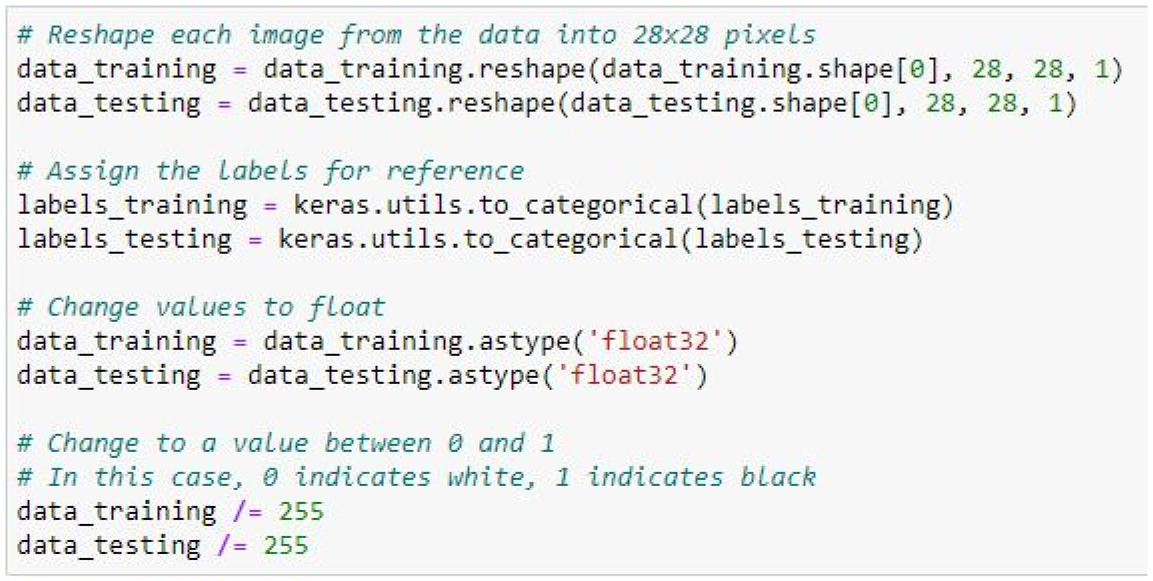

Digit Recognition - Cross Entropy
Loss
Epoch
0.025
0.030
0.035
0.040
0.045
0
2
4
6
8



Data Cleaning

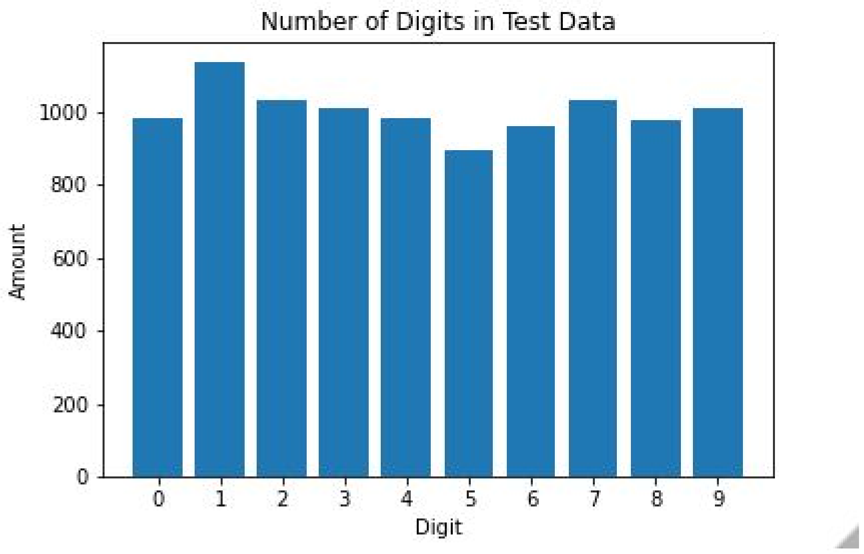
The MNIST dataset has already been cleaned and organized for anyone that wants to use it by size-normalizing and centering the images to fit into a 28x28 pixel bounding box.

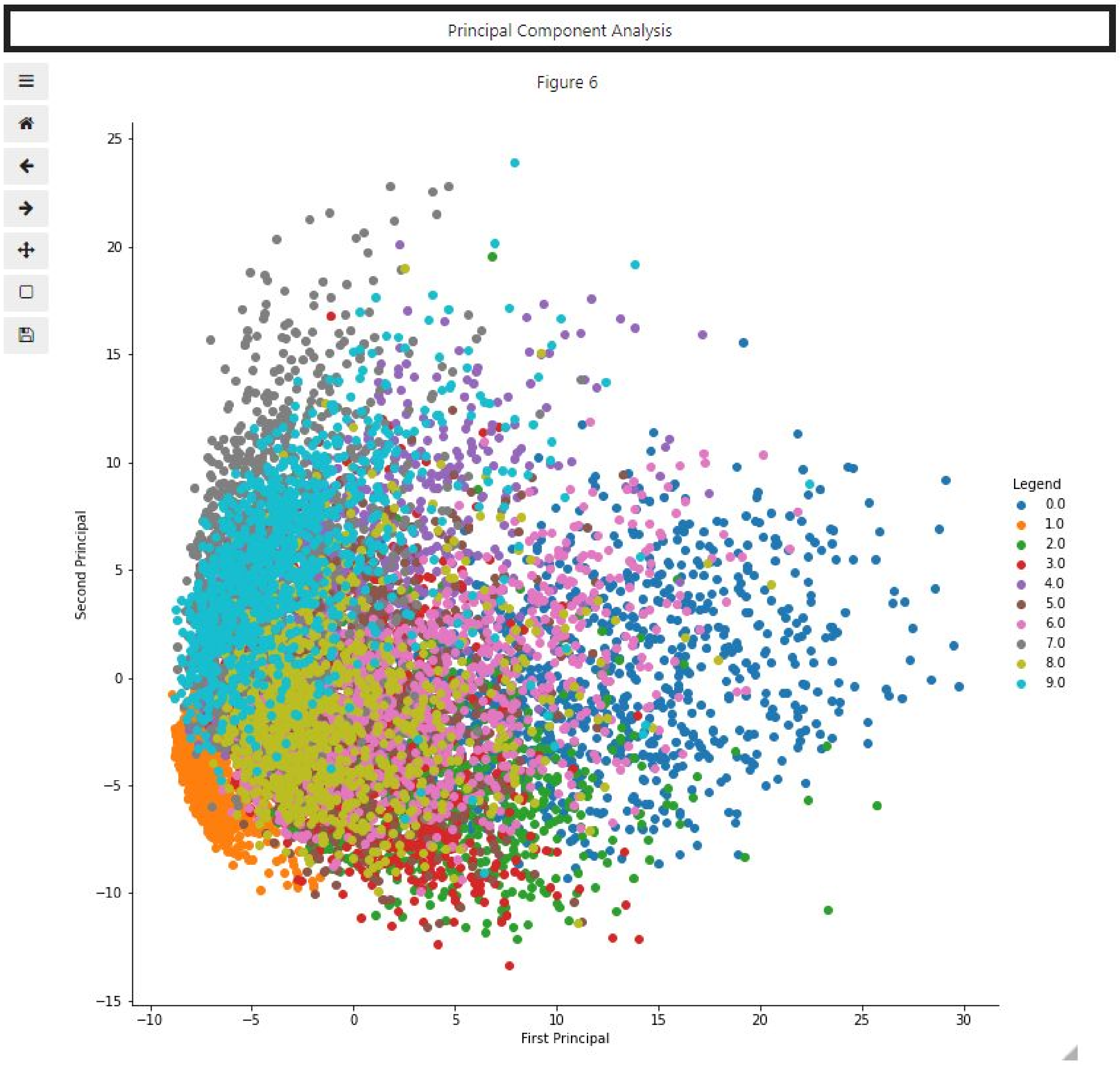
However, the data does need to be preprocessed using the below code before it can be effectively run through the CNN model.

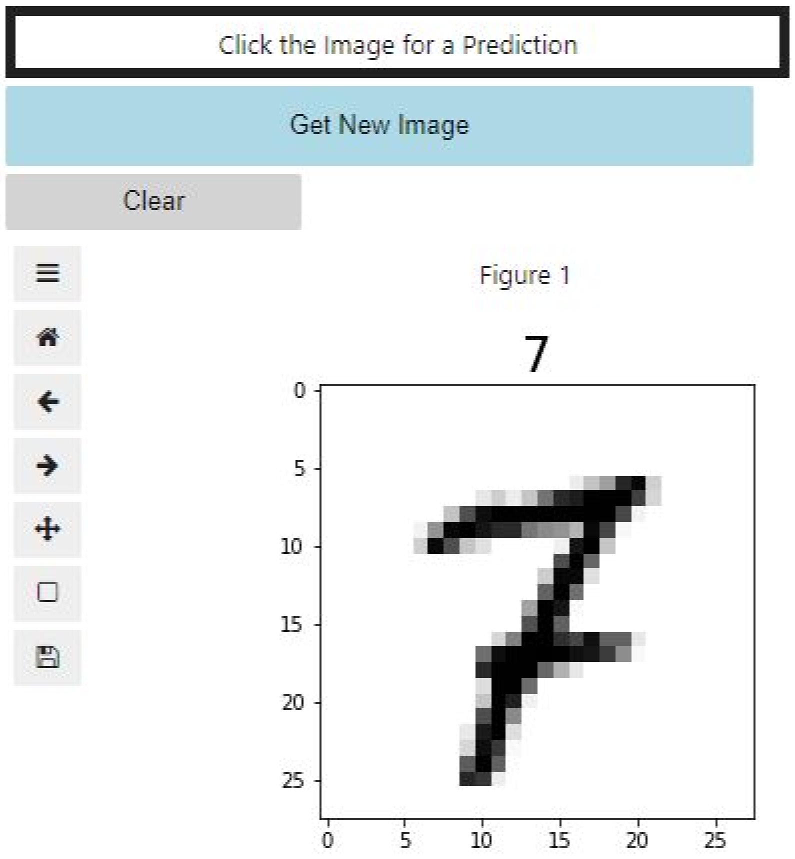

```
# Reshape each image from the data into 28x28 pixels
data_training = data_training.reshape(data_training.shape[0], 28, 28, 1)
data_testing = data_testing.reshape(data_testing.shape[0], 28, 28, 1)

# Assign the labels for reference
labels_training = keras.utils.to_categorical(labels_training)
labels_testing = keras.utils.to_categorical(labels_testing)

# Change values to float
data_training = data_training.astype('float32')
data_testing = data_testing.astype('float32')

# Change to a value between 0 and 1
# In this case, 0 indicates white, 1 indicates black
data_training /= 255
data_testing /= 255
```

Data Visualization

Along with the analytical visualizations shown above, three forms of data visualization are also included in the dashboard. These include a bar graph that shows the amount of images there are for each digit in the MNIST validation dataset, a PCA scatter plot to demonstrate the differences (or similarities) between images and their features, and lastly there is an image of a number from the dataset that can be clicked to predict what the number is. This image comes with two buttons, one of which is labeled “Get New Image” and will grab a new random image from the dataset to be predicted, and one labeled “Clear” that will clear the prediction.

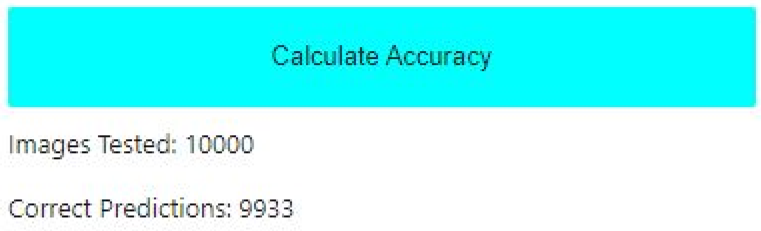



Principal Component Analysis
Figure 6
Second Principal
First Principal
Legend
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
25
20
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
15
20
25
30



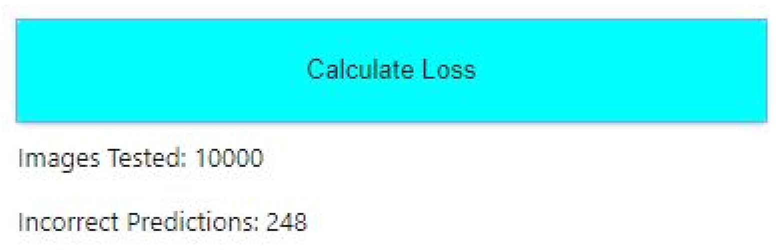
Real-Time Queries

The dashboard includes three forms of queries that users are able to use. First, there is a button labeled “Calculate Accuracy” that allows the user to calculate the accuracy of the trained model. Second, there is a button labeled “Calculate Loss” that allows the user to calculate the loss, or inaccuracy, of the trained model. Third, we have the previously mentioned image button that users may click to predict the number in the image. The “Get New Image” button will pull any random image from the MNIST dataset for users to predict using the model.



Images Tested: 10000

Correct Predictions: 9933



Images Tested: 10000

Incorrect Predictions: 248

Adaptive Element

The model used for DRA is capable of becoming more accurate by taking images of numbers from real TRFs to give it a bigger training and testing pool. However, the cost of this may not be worth the reward, as the model has been shown to be up to 99.3% accurate. It would take a very long time for the model to learn much more with Genetique only receiving around 1000 TRFs each day, with the vast majority of these TRFs containing numbers that are not difficult in any way for the model to predict already.

Outcome Accuracy

DRA's accuracy is determined by how often it correctly predicts an image, with the loss being determined by how often it incorrectly predicts an image. The accuracy and loss can each be evaluated by the user by clicking the buttons on the dashboard previously mentioned.

The parameter for valid output is checked by comparing the prediction of the image to the label of the image. The label is the number that an image has been assigned, and it tells the model what the image actually is.

Security Measures

The security measures necessary will depend on the way a business would like to implement DRA. For Genetique, DRA will be hosted on their intranet so that it is not publicly accessible. DRA will require special permissions and need to be installed by a member of Genetique's IT team, and they will be keeping track of which computers have it installed. Routine maintenance every three months will include verification that DRA is not reading anything beyond the numbers in specified sections of the TRF. A login system for DRA will be developed, and will use the same authentication as Genetique's CRM application.

Product Health Monitoring

Continual use of DRA should not alter anything that would damage its health. However, there is always the possibility for a program to become corrupted for one reason or another.

DRA will be run through Voila. Voila is capable of running Python code in the background using Jupyter Notebook. All Python error messages will be printed to the user, which will allow for constant verification of the product's health. Any error messages that a user receives are logged by default and can be sent with a help ticket to IT, and they will handle it as necessary.

Dashboard

The dashboard contains three sections, each of which serve a different purpose.

The first section simply contains two buttons for evaluation of the model: Calculate Accuracy and Calculate Loss. These buttons, when clicked, will determine the model's accuracy or loss, respectively. The output of these buttons will show the amount of images tested, along with the amount of images either correctly or incorrectly predicted.

The second section contains six graphs for analytical purposes. The first four graphs all have to deal with the accuracy or loss of the model as it trains, from epoch to epoch. Two of these four graphs look at the accuracy and loss during validation, while the other two compare the training accuracy/loss to the validation accuracy/loss. The fifth graph is a bar chart that displays the amount of each digit that exists in the MNIST validation dataset. The sixth graph is a scatter plot using Principal Component Analysis to exemplify the differences and similarities of images of the different numbers, as well as between images of the same number.

Post-Implementation Report

Project Purpose

The purpose of this project, creating an app temporarily referred to as Digit Recognition App (DRA), was to improve the accuracy of numbers entered during the data entry process. Specifically, it was addressing the issue of human error in medical situations. This human error has the potential to lead to a patient receiving improper treatment, not receiving insurance coverage, waiting for a longer turnaround time, among other issues. It can also affect the company itself in multiple ways, from wasting a small amount of resources, to losing a ton of money and gaining a bad reputation.

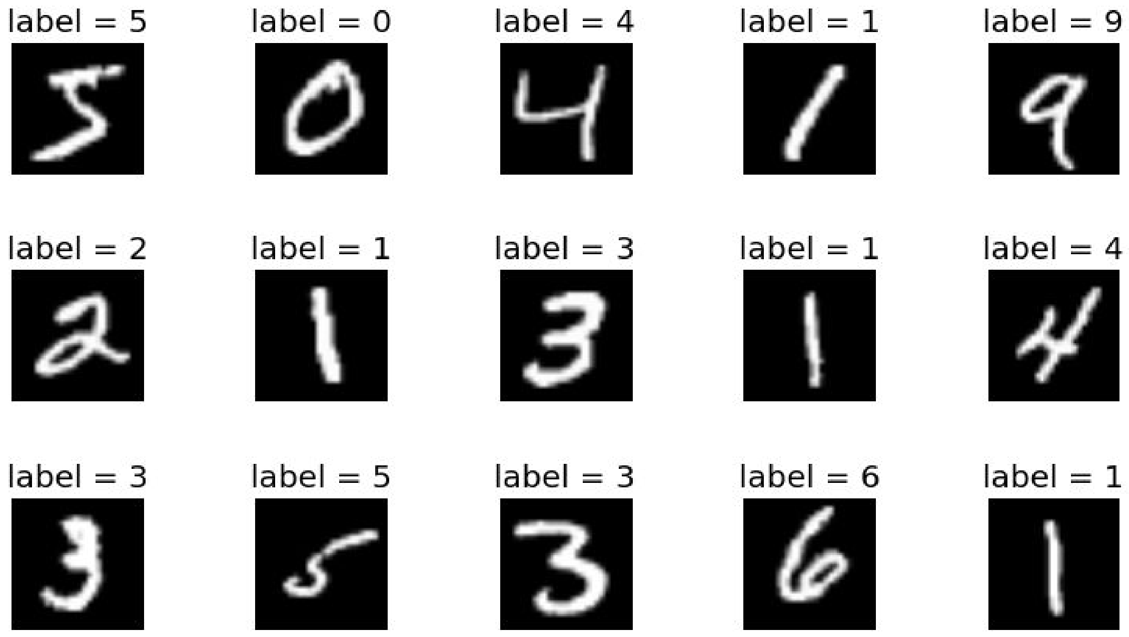
The solution was to automate a portion of the process to allow for more productivity, increased turnaround time, more consistently accurate results, and a better patient experience. Genetique was given the option to have a feature added to DRA that automatically enters the numbers from Test Request Forms (TRFs) into their Customer Relationship Management (CRM) application. They chose to keep a form of double data entry where a human will simply verify that DRA correctly predicted the numbers, as this will provide the most accuracy.

Preventing human errors not only led to resolving the problems previously listed, but patients have been shown to have a better experience with Genetique overall. Genetique has also seen quicker turnaround time and increased accuracy during audits in Data Entry.

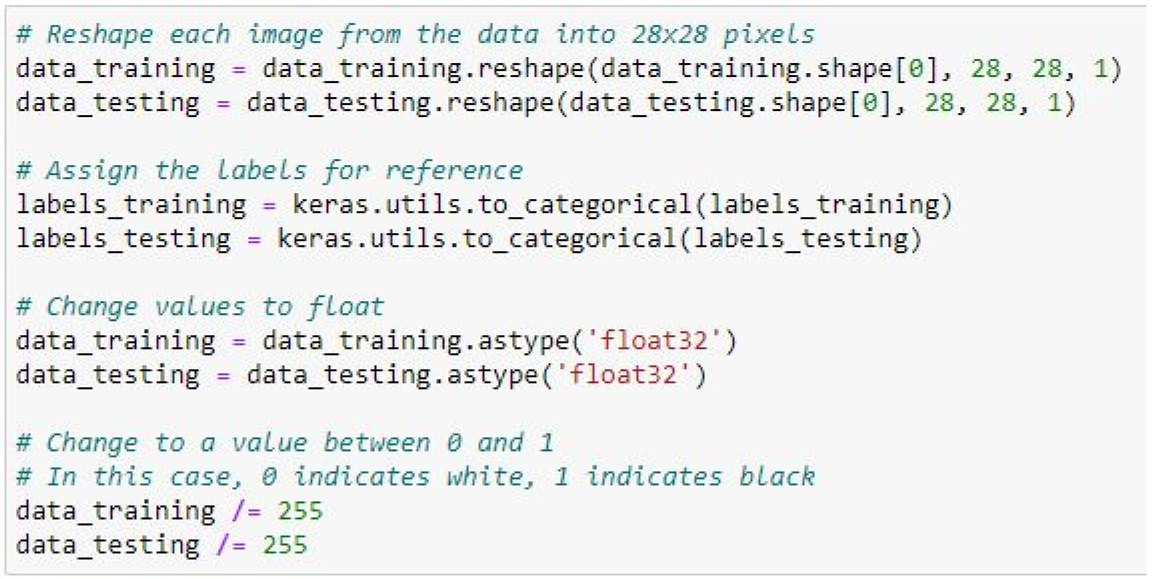
“Delegating an employee’s time to mundane tasks takes time away from activities that improve patient care and drive the organization forward.” (Chris Thompson, 2018)

Datasets

The MNIST dataset was the only dataset used to train the model for DRA. MNIST is regularly used in machine learning, and researchers are often comparing results of trained models to find out what exactly will allow for the most accurate results when using digit prediction. The dataset contains 60,000 images that are used in the training of models, while there are 10,000 more images that are used for testing/validating the model. Below, you can see an example of images from the MNIST dataset, along with their labels that are used to tell the model whether or not its prediction was correct.



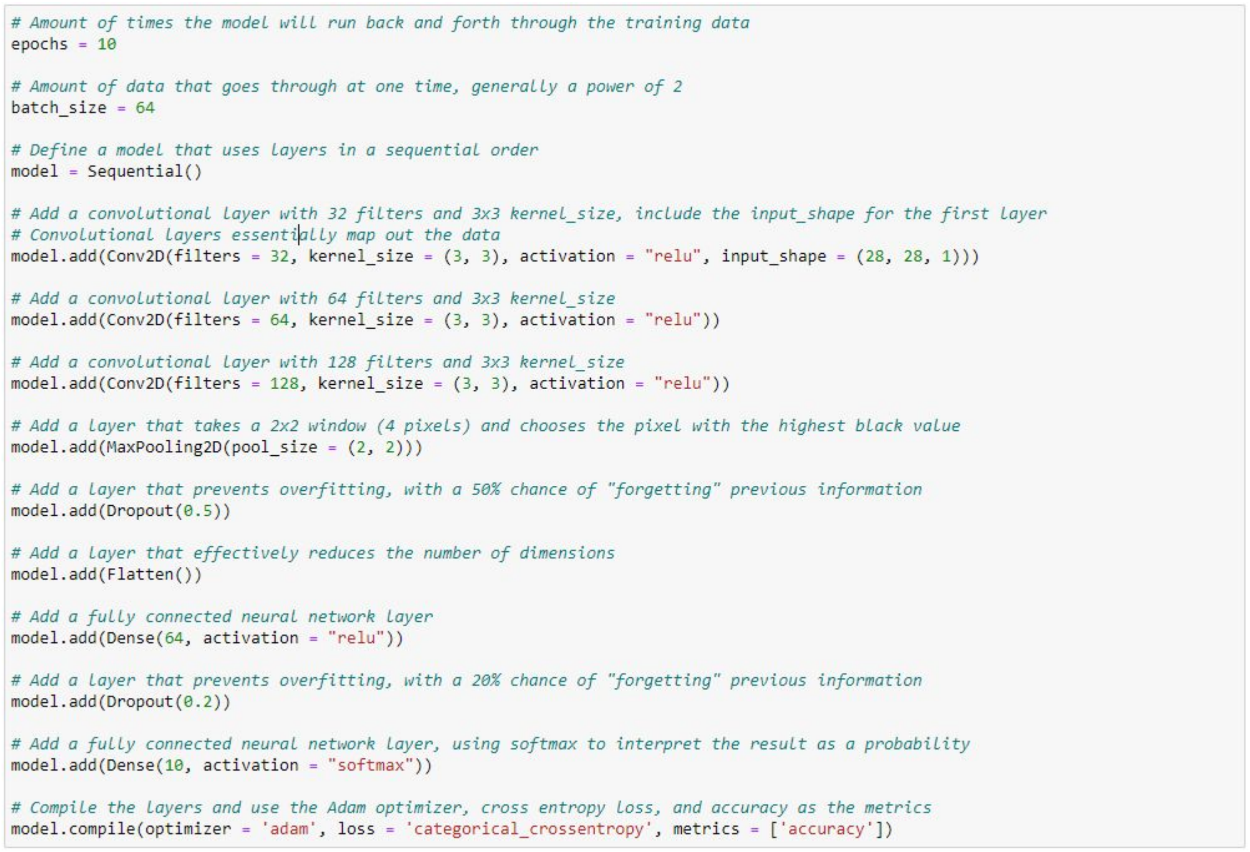
The data comes in a format that is ready to be used, but it still requires preprocessing. This preprocessing is done with the following code (explained below):


```
# Reshape each image from the data into 28x28 pixels
data_training = data_training.reshape(data_training.shape[0], 28, 28, 1)
data_testing = data_testing.reshape(data_testing.shape[0], 28, 28, 1)

# Assign the labels for reference
labels_training = keras.utils.to_categorical(labels_training)
labels_testing = keras.utils.to_categorical(labels_testing)

# Change values to float
data_training = data_training.astype('float32')
data_testing = data_testing.astype('float32')

# Change to a value between 0 and 1
# In this case, 0 indicates white, 1 indicates black
data_training /= 255
data_testing /= 255
```

The first two lines of code reshape the image into what is considered to be a three-dimensional array (not to be confused with a three-dimensional image). It also makes sure that it is in a 28x28 pixel format and is interpreted using a single color channel (grayscale/binary). Next, the labels need to have their indices referenced so that the model does not misinterpret a higher index as being a higher value. Seeing as the images are only of numbers 0-9, it would make no sense for it to think that numbers 10-60000 or 10-10000 are ever accurate predictions. Next, the values are converted to floats and divided by 255. This is because the weights in the array use a number between 0 and 1, with 0 being pure white and 1 being pure black. Dividing by 255 is necessary, because we are using a single color channel rather than 8-bit (256) colors.

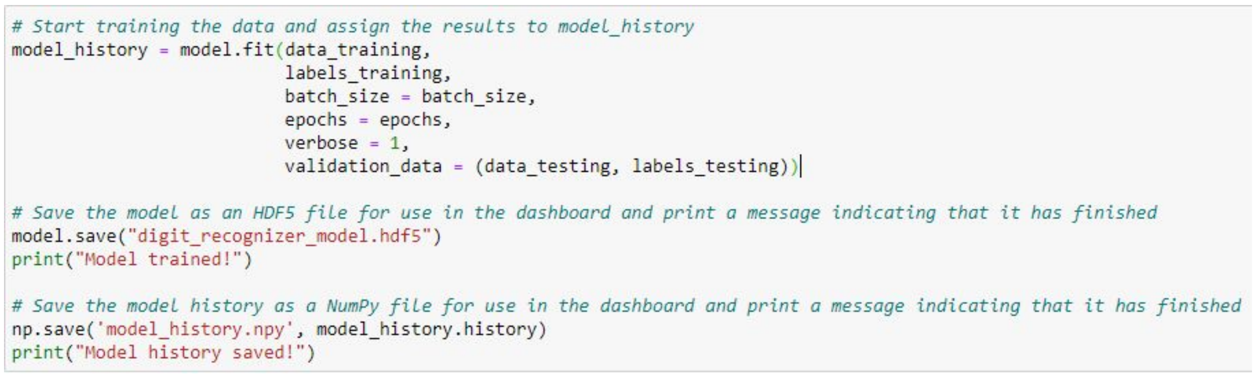
Data Product Code

The model is trained using a two-dimensional convolutional neural network (CNN). It runs through 10 epochs (a run through all of the input data and back again) with a batch size of 64. Each image goes through a sequential order of layers, as shown in the code below.



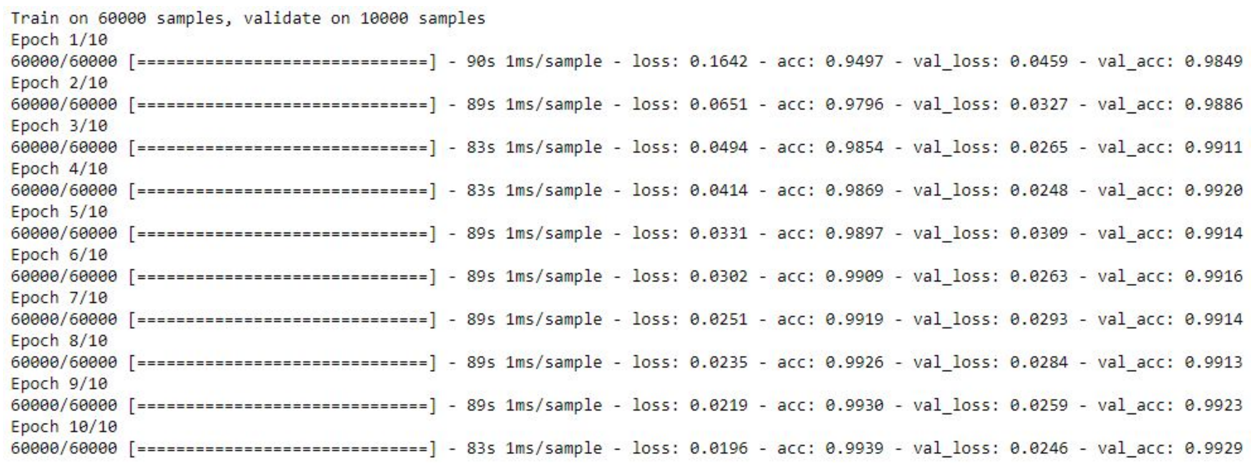
```
# Amount of times the model will run back and forth through the training data
epochs = 10

# Amount of data that goes through at one time, generally a power of 2
batch_size = 64

# Define a model that uses layers in a sequential order
model = Sequential()

# Add a convolutional layer with 32 filters and 3x3 kernel_size, include the input_shape for the first layer
# Convolutional layers essentially map out the data
model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation = "relu", input_shape = (28, 28, 1)))

# Add a convolutional layer with 64 filters and 3x3 kernel_size
model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = "relu"))

# Add a convolutional layer with 128 filters and 3x3 kernel_size
model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = "relu"))

# Add a layer that takes a 2x2 window (4 pixels) and chooses the pixel with the highest black value
model.add(MaxPooling2D(pool_size = (2, 2)))

# Add a layer that prevents overfitting, with a 50% chance of "forgetting" previous information
model.add(Dropout(0.5))

# Add a layer that effectively reduces the number of dimensions
model.add(Flatten())

# Add a fully connected neural network layer
model.add(Dense(64, activation = "relu"))

# Add a layer that prevents overfitting, with a 20% chance of "forgetting" previous information
model.add(Dropout(0.2))

# Add a fully connected neural network layer, using softmax to interpret the result as a probability
model.add(Dense(10, activation = "softmax"))

# Compile the layers and use the Adam optimizer, cross entropy loss, and accuracy as the metrics
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Below, you can see in the code where the training starts. The command “model.fit()” starts the process using the given parameters. After all 10 epochs have gone through the training, the model is then saved as a file named “digit_recognizer_model.hdf5”. Once it has saved, you will get a message telling you that the model has been trained. The output of the training is then saved to a file named “model_history.npy”, with another message letting you know that the history has been saved.



```
# Start training the data and assign the results to model_history
model_history = model.fit(data_training,
                          labels_training,
                          batch_size = batch_size,
                          epochs = epochs,
                          verbose = 1,
                          validation_data = (data_testing, labels_testing))

# Save the model as an HDF5 file for use in the dashboard and print a message indicating that it has finished
model.save("digit_recognizer_model.hdf5")
print("Model trained!")

# Save the model history as a NumPy file for use in the dashboard and print a message indicating that it has finished
np.save('model_history.npy', model_history.history)
print("Model history saved!")
```

Below you can see the epochs that the existing model (digit_recognizer_model.hdf5) went through. This is the information that is saved in model_history.npy.



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 90s 1ms/sample - loss: 0.1642 - acc: 0.9497 - val_loss: 0.0459 - val_acc: 0.9849
Epoch 2/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0651 - acc: 0.9796 - val_loss: 0.0327 - val_acc: 0.9886
Epoch 3/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0494 - acc: 0.9854 - val_loss: 0.0265 - val_acc: 0.9911
Epoch 4/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0414 - acc: 0.9869 - val_loss: 0.0248 - val_acc: 0.9920
Epoch 5/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0331 - acc: 0.9897 - val_loss: 0.0309 - val_acc: 0.9914
Epoch 6/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0302 - acc: 0.9909 - val_loss: 0.0263 - val_acc: 0.9916
Epoch 7/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0251 - acc: 0.9919 - val_loss: 0.0293 - val_acc: 0.9914
Epoch 8/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0235 - acc: 0.9926 - val_loss: 0.0284 - val_acc: 0.9913
Epoch 9/10
60000/60000 [==============================] - 89s 1ms/sample - loss: 0.0219 - acc: 0.9930 - val_loss: 0.0259 - val_acc: 0.9923
Epoch 10/10
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0196 - acc: 0.9939 - val_loss: 0.0246 - val_acc: 0.9929
```

Hypothesis Verification

My original hypothesis was that using the MNIST dataset with an efficient deep learning model will lead to a 99% or better accuracy rate, and that any errors made would likely be the result of outliers that even humans would have a tough time deciphering.

This hypothesis was correct. The accuracy of the model has reached 99.3%, with a very small amount of variation each time it is retrained.

In my hypothesis it was also mentioned that double data entry, which is the most common accurate form of data entry, is likely to be less accurate than the trained model. This model has proven to be more accurate than double data entry, which is often no better than 97.7% accurate. That is an increase of 1.6%. Another way to look at this is that 99.3% accuracy misses only 7 out of 1000 times, whereas 97.7% misses 23 out of 1000 times. That is over three times as many misses, showing just how helpful machine learning can be for something like data entry.

“Human data entry can result in errors that ruin statistical results and conclusions. A single data entry error can make a moderate correlation turn to zero and a significant *t*-test non-significant. Therefore, researchers should design and use human computer interactions that minimize data entry errors.” (Kimberly A. Barchard; Larry A. Pace, 2011)

Effective Visualizations and Reporting

The dashboard contains multiple visualizations of the data and analytics in regard to DRA. The line graphs are a great representation of how the model learned from training over time, and then was able to do particularly well in the testing/validation phase of each epoch. Dropout can be clearly seen in each graph, while still averaging an incline rather than a decline, proving that dropout can be an effective tool to prevent overfitting. You can see in the training accuracy that the model started to plateau at a certain point. This is where convergence happens. The model can only get so accurate before being unable to improve further. Running more epochs can effectively demonstrate this phenomenon.

The bar graph allows you to visualize the amount of each number that the validation set of images has. This tells you the probability of getting a certain number from the dataset when a random one is grabbed.

The PCA scatter plot is an excellent way of showing that there are many ways to write numbers, and that many of these ways of writing can cause issues for both human and machine due to the similarities between them. Any points that overlap have very similar features, and it is somewhat common that they are not even the same number.

The image prediction widget is also a good tool to use to see what exactly the model is capable of predicting, and what it is not capable of predicting. You will not often see an image that cannot be predicted accurately, but when you do, it is usually somewhat obvious why the model may not have been able to accurately predict it. On the other hand, while dropout is useful, it is entirely possible that, if there are not very many examples of a certain way of writing a digit, and dropout occurred with one of these images, that it will continue to predict that writing style incorrectly without more training.

Accuracy Analysis

DRA's model proved to be very effective in predicting the digits in images. The model evaluated to a 99.3% accuracy rating. This means that it is generally only incorrect when there is an outlier that resembles a digit written in a way that it has not seen before, or has only seen minimally.

The analytics data shown in the dashboard, the calculation buttons at the top of the dashboard, and the image prediction widget at the bottom of the dashboard all show accurate representations of what you can expect from DRA.

Application Testing

Testing was done in phases. The first bit of testing done was unit testing with the dashboard. Each different portion had to be tested by itself to make sure it could function as a single unit. Next, we had integration testing. This was done along with the first rollout phase, considered a beta test in this case, by implementing DRA into Genetique's system and having 20% of both the Data Entry and Accessioning departments use the application with fake TRFs. This led into acceptance testing and the second rollout phase. The second rollout phase involved moving the rest of the Data Entry and Accessioning teams to the new system. At this point, they started using DRA with real TRFs. We were keeping a watchful eye on the process, and making sure the accuracy of DRA was consistent.

Many changes were made to the dashboard over time, though it was primarily stylistic choices based on customer feedback. Aside from a few initial installation problems, there were no issues once the first rollout phase started.

Application Files

The file system consists of the following:

- Data (folder)
    - digit_recognizer_model.hdf5
    - mnist_test.csv
    - model_history.npy
- Notebook (folder)
    - MNIST_Model.ipynb
    - Model_Dashboard.ipynb

These files will be organized in a zip file as shown above, separate from this report, for submission.

.ipynb files are Python files used specifically with Jupyter Notebook. The rest of the files are data files to be used with Python code.

Appendices

Installation and User Guide

Both an installation guide and a user guide can be found by using the link below:

User Manual

Summation of Learning Experience

My experience prior to this project consisted of just a few small Python projects and some projects in other object-oriented programming languages. I had absolutely no experience with machine learning itself. While I understood the concept overall and had watched many videos on it in the past, I spent a very large portion of my time on this project just learning about different datasets that I might be able to use, data visualization, and deep learning. Then I started to learn a little bit about pandas, Keras, Tensorflow, Seaborn, CNNs, and more. After that, I decided to use the "Hello, World!" of computer vision, the MNIST dataset, because it seemed like the best way to be introduced to the subject while producing something interesting.

Before starting this project, I was expecting to semi-casually spend just a couple of weeks or so using knowledge that I already had, but that was definitely not the case. It ended up not being casual in the slightest, and I would estimate that 60-70% of the time working on the project was actually spent learning new skills.

Although I wanted it to be a bit easier, I can't help but be thankful for everything I learned in this process. This project, far more than any other I have done so far, contributed to my understanding of what it means to be in a software-oriented field. The idea of needing to be learning constantly can be intimidating, but it is simultaneously very exciting. Overall, this was an extremely interesting and rewarding experience, and I plan to pursue machine learning much further.

References

Goldberg, Saveli I, et al. “Analysis of Data Errors in Clinical Research Databases.” *AMIA ... Annual Symposium Proceedings. AMIA Symposium*, American Medical Informatics Association, 6 Nov. 2008, www.ncbi.nlm.nih.gov/pmc/articles/PMC2656002/.

Thompson, Chris. “The High Cost of Manual Data Entry in Healthcare – And How to Reduce Risk.” *MXOtech, Inc.*, 24 Aug. 2018, www.mxotech.com/2018/05/manual-data-entry-in-healthcare-how-to-reduce-risk/.

Wan, Li, et al. “Python Machine Learning Tutorial.” *Machine Learning with Python: Training and Testing the Neural Network with MNIST Data Set*, 2013, www.python-course.eu/neural_network_mnist.php.

Barchard, Kimberly A., and Larry A. Pace. “Preventing Human Error: The Impact of Data Entry Methods on Data Accuracy and Statistical Results.” *Computers in Human Behavior*, Pergamon, 4 May 2011, www.sciencedirect.com/science/article/abs/pii/S0747563211000707#:~:text=Human%20data%20entry%20can%20result,ruin%20statistical%20results%20and%20conclusions.&text=Visual%20checking%20resulted%20in%202958,correlations%2C%20and%20t%2Dtests.