Accurate Cancer Prediction Using AI

Team 47

Advisor: Dr. Ashraf Gaffar

Jack Sebahar, Isaiah Mundy, Helen Lau, Mason Wichman, Lal Siama, Nicholas Otto

sdmay24-47@iastate.edu

https://sdmay24-47.sd.ece.iastate.edu

Table of Contents

Introduction

The current challenge in cancer diagnosis and prediction lies in the limited ability to accurately foresee its occurrence and recurrence and give related diagnoses. Traditional diagnostic methods may not be as precise as desired, and human doctors, despite their expertise, face challenges in providing consistently accurate predictions. Recent research suggests that artificial intelligence (AI) has the potential to significantly improve predictive accuracy, surpassing the capabilities of human doctors. The goal of this project is to develop a simple AI model, leveraging machine learning and neural networking tools such as Tensorflow and Keras, to enhance cancer predictions, providing a valuable asset in the realm of medical diagnosis.

Goals

Goal 1: Develop ML model to predict likely prognosis of sample data

The first goal relates to the development of the model. We want to see if we can develop a machine learning that, when given input data in the same format as our sample set, can predict a cancer patient's prognosis. Throughout the semester, we will employ numerous strategies relating to data preprocessing and different neural network architectures in attempts to improve the accuracy of the model.

Goal 2: Create application allowing user to interact with model

The next goal involves creating a web application capable of allowing the user to make predictions with our model. The user should be able to run their cellular data, similar to the samples we were provided, through our model and return a prediction giving them an estimate of their prognosis. The model will be deployed on the cloud and interact with the data via an endpoint.

Goal 3: Cloud Platform Comparison

Lastly, we want to create some comparisons between two cloud platforms: Amazon Web Services and and Google Cloud Platform. We will train and deploy the same model on both platforms and compile some comparisons between the two, such as price, performance, and usability differences. This will help determine if there are any notable differences between the platforms

Intended Users and Use Cases

Intended Users:

- Medical professionals such as oncologists and healthcare providers seeking enhanced predictive tools for cancer outcomes.
- Patients or other users with access to pathology data that may want to inquire about their prognosis

Intended Uses:

- Assisting medical professionals in making more accurate and timely cancer-related predictions.
- Serving as a supplementary tool for informed decision-making in cancer treatment.

Our application is not meant to replace oncologists, but rather provide a tool they can use to assist their detection capabilities.

Revised Design

Requirements

Functional

- The model must be capable of making predictions based on user input data
- The model must be trained to accept data in the form of the sample data
- The model must be deployed on a cloud platform
- The application must allow the user to upload their own data
- The application must retrieve prediction from cloud hosted model
- The application must display the prediction to the user

Resource

- Cellular spectra data in csv format
- Cloud platform (likely Google Cloud and Amazon Web Services)
- Tensorflow machine learning library
- Keras neural network API
- Web application framework (likely Python Flask)

UI

- The application shall present information in an organized manner)
- The application should lack any spelling errors
- Each page of the application shall be easy to read
- Pages should be intuitive to find the information you are looking for
- Pages within the application shall be easily accessible by use of navigational menus and buttons

Performance

- The application should run at a user-friendly speed, the user should not have to wait more than a minute for page navigation or model predictions

Legal

- Comply with data privacy and security regulations when handling medical data
- Ensure only patients and medical experts ever see data

Maintainability

- The application shall support the addition of new or changed information
- documentation shall be provided to the client including user guides to understand the app's functionality

Testing

- The model shall be tested thoroughly to ensure it has a desirable level of accuracy
- The application shall be tested thoroughly to ensure it has a high level of reliability, usability, and accuracy

Engineering Standards

- HIPAA - This standard ensures the protection of patients' medical data. Since the project deals with medical information, we will need to ensure patient privacy and security
- IEEE 1058: It provides guidelines for the preparation of software project management plans.
- IEEE 3051 - This standard revolves around the trustworthiness of Artificial Intelligence and Autonomous Systems, and provides guidelines for ensuring the trustworthiness and reliability of AI systems. This standard is particularly relevant to projects in critical domains like healthcare.
- IEEE 3123 - This standard provides clear definitions for terminology utilized in artificial intelligence and machine learning
- IEEE 12207 - This standard defines the processes involved in the software development life cycle. It provides a framework for managing software projects and ensuring the quality and reliability of software systems.
- ISO 13485:2016 - This standard discusses quality management systems regarding medical devices. Since our AI is intended for medical purposes, there is a high level of quality that is expected.

Security Concerns and Countermeasures

The largest security concern of ours is data vulnerability since we are working with sensitive medical information. At no point throughout our project, should anyone besides us, the patients, or medical professionals ever see or have access to the data. In order to protect the data, we have implemented the following countermeasures:

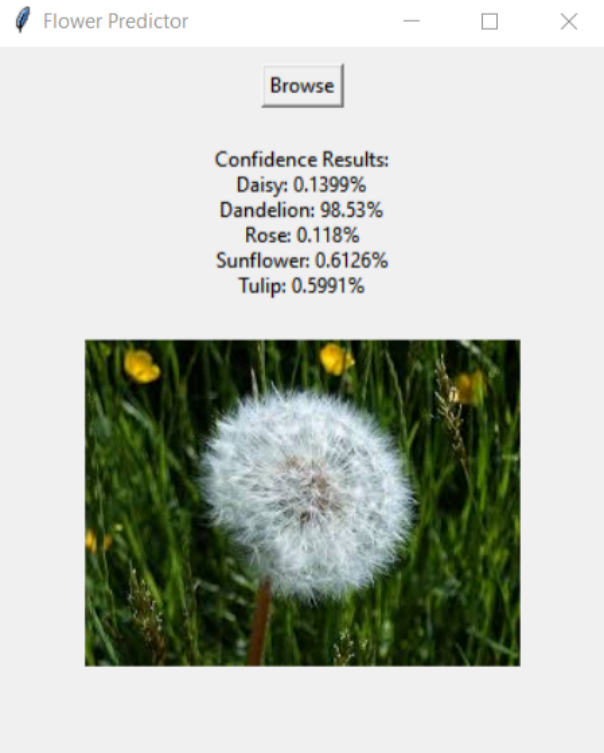
1. Data is only transferred physically. This means that we use a physical connection such as a flash drive when moving the data between the group members. The data is never emailed or otherwise shared over the internet.
2. Data is only stored in safe, encrypted locations. For example, when developing the model in Google Colab, we only store the data in Google Drive, which is encrypted in rest and in transit. The same applied in AWS with Sagemaker
3. The web application is implemented using HTTPS. HTTPS incorporates SSL/TLS (Secure Sockets Layer/Transport Layer Security) encryption, which encrypts the data being transmitted between the user's browser and the website's server.
4. Google and AWS authentication credentials. Both of these platforms have a high level of security naturally integrated. Nobody can access any endpoints or storage buckets unless they obtain the credentials to the project. Credentials were never shared outside of the group and are stored on a secure device

Design Evolution from 491

The core goals of our project have not changed since 491. However, there were some design changes we did need to make from 491.

In order to familiarize ourselves with basic machine learning concepts and Google Colab, we made a basic flower classification model using Tensorflow. The model was capable of classifying different types of flowers from images. This provided a good outline of common machine-learning techniques for beginners. Although this model was a good starting point, we had to deeply refine it to work with our use case.

Another design change we implemented was how our application was designed. It was initially created as a desktop application using the Python Library “Tkinter,” and was intended to work with our flower classification model. This means the model metadata was built directly into the client side of the application. We learned that this was insufficient for the project because the application needed to be a web application that referenced the model deployed on the cloud. This promoted higher scalability if the model needed to be improved, nothing had to be changed on the client side. We decided to implement the web application using the Python Flask Framework, which allowed much of our code to be reused.



Here is what the early design of our application looked like with the flower classification model. This proved to be a good outline for our current application design and was a good starting point with machine-learning.

Implementation Details

Data

Our data set is comprised from patients who were cancer recurrent to some extent. It is an encoded cell image of either a cancerous or benign structure. The exact implications are beyond the scope of the project. The data was given to us in the form of comma separated value files (csv). Each sample corresponds to a single patient and contains 2449 vectors. The x column of data is generally larger and the y column is generally smaller. These values have some sort of representation of the patient at a cellular level. Each patient also has a corresponding survival number, which represents how many months they survived. It is unknown if there is a correlation between the cell data and the outcome, but it was our goal to see if a machine learning algorithm could expose one.

Sample 1			
2.919128473	1.24E-01		
2.916994473	1.24E-01		
2.914860473	1.24E-01		
2.912726473	1.24E-01		
2.910592473	1.24E-01		
2.908458473	1.24E-01		
2.906324473	1.24E-01		
2.904190473	1.24E-01		
2.902056473	1.24E-01		
2.899922473	1.24E-01		
2.897788473	1.24E-01		
2.895654473	1.24E-01		
2.893520473	1.24E-01		
2.891386473	1.24E-01		
2.889252473	1.24E-01		
2.887118473	1.24E-01		

num	sample	Survival
0	1	21
1	2	213
2	3	100
3	4	54
4	5	24
6	6	21
8	7	56
10	8	135
12	9	32
13	10	102
14	11	78
15	12	93
16	13	120

Here, we have a fabricated data set to give a representation of what our data looks like. The image on the left represents a single patient's data of coordinate pairs. On the right, we have the "master" spreadsheet containing the survival number for each patient.

Data Preprocessing

When initially working with the data, we performed some preprocessing to filter out two types of unusable data. The first was a number of incomplete samples either containing only patient samples, or only length of life data. The second was corrupted patient samples that were missing chunks or included corrupted strings. For this data preprocessing, we used a series of Python scripts to filter out unusable samples and transform usable data, which left us with 340 usable samples. More data processing methods that applied to error reduction will be described in our testing section.

Model

For our model, we decided to use a Sequential Keras model, which is essentially a linear stack of layers with each layer containing one input tensor and one output tensor. We decided on a sequential model because it is very straightforward to define and allows for easy prototyping and experimenting with different architectures. Given the experimental nature of our project, we thought this would be very valuable. The Sequential model is also very efficient in training which was very useful considering the amount of experimentation we performed when testing different model architectures.

Ultimately we decided on a “base” model consisting of 4 dense layers containing 64, 32, 16, and 1 unit, respectively. The input tensor shape was (1, 2449, 2), meaning each data point has 2449 data features, with each containing 2 values (the coordinate pair in the data sample). The output tensor shape was (2449, 1), meaning that we want to return a single prediction for each set of 2449 data features. This directly corresponds to the data we were working with. We found that this architecture was most effective in achieving a faster accuracy convergence while providing a low error compared to other model architectures.

Deployment

Once our model was trained, we were able to save the model’s metadata as a zip file. The model’s metadata includes its weights and graphs, which are Tensorflow specific values for describing the model. Once we had the metadata saved, we could deploy the model on the different cloud platforms. On Google Cloud, we used Vertex AI, which is Google’s most powerful and currently supported AI platform. In Vertex AI, our model imported under the Tensorflow framework version 2.13. The model was then deployed to an endpoint with 1 compute node. Once the model was deployed, we were provided with a python sample request that allowed us to make calls to the endpoint within our application

In AWS, we used Sagemaker to deploy the model. The process for doing so was essentially the exact same. The model was deployed under the Tensorflow framework. Once deployed, we were provided with a python sample request that we could use in our application code to infer the endpoint and make predictions

Application

Our application was designed as a web application using the Python Flask framework. The UI was written in HTML and gives the user brief instructions on how to utilize the application. All they have to do is click on the button to upload their CSV data into the application. From there, an endpoint request is made to our deployed model on the cloud, and the user’s data is run through the model. A prediction is then returned to the user with an estimate of their prognosis.

The application is really only in charge of preprocessing the data in a way that the model expects. It iterates through the csv file that the user provides and extracts all of the coordinate points into a pandas data frame. It then performs min-max normalization on the columns and creates a new instance of data to be fed into the endpoint. The endpoint is requested with the normalized data points and a prediction is fetched. Once a prediction is received, the UI is updated with the prediction for the user to view. Due to how the predictions are formed, we are only concerned with the first value in the predictions list, so we do some formatting to ensure that only the first prediction is displayed to the user.

Cloud Comparison

An important goal of our project was to produce the application on various cloud platforms, namely Google Cloud (GCP) and Amazon Web Services (AWS) and gauge performance differences and other useful metrics. The purpose of this was to determine what platform was most suitable for a project like ours. As part of our analysis, we have divided it into two categories: training and cost.

Training

Training our model on both platforms essentially involved the exact same steps. Both AWS and GCP provide free Jupyter notebooks in the form of Colab for GCP and Sagemaker for AWS. Both of these notebook platforms allow you to store data, design, and train a model for free using the platform's resources. The final iteration of our model was able to train all 100 epochs in Colab in roughly 51 seconds. The same model was trained in Sagemaker in approximately 50 seconds. All iterations of our model had similar training times with a difference of only a few seconds. This slight difference can likely be attributed to human error in timing or resource contention on the cloud platform. These results are to be expected since we were training the exact same model on essentially the same platform (Jupyter).

In order to ensure we were comparing apples to apples, we evaluated the mean squared error on both models.

```
9/9 [==============================] - 0s 48ms/step - loss: 3106.2073
Epoch 100/100
9/9 [==============================] - 1s 57ms/step - loss: 3111.6924
3/3 [==============================] - 0s 13ms/step - loss: 2651.8625
Mean Squared Error on Test Data: 2651.862548828125
```

The mean squared error on the Sagemaker version of the model was roughly 2651.

```
9/9 [==============================] - 1s 100ms/step - loss: 3060.8843
Epoch 100/100
9/9 [==============================] - 1s 75ms/step - loss: 3029.5840
3/3 [==============================] - 0s 16ms/step - loss: 2773.2251
Mean Squared Error on Test Data: 2773.22509765625
```

The mean squared error on the Colab version of the model was roughly 2773.

The mean squared error will be slightly different every time you train the model due to random initialization in the neural network and data shuffling before each epoch. In this example, the mean squared error is very high due to this being an early iteration of our design. In our case, the

difference between the two is negligible and demonstrates that the same model is being trained properly on the two platforms. Overall, there was no significant difference between set-up and training time between the two platforms.

Cost

Development on both Colab and Sagemaker is free to use. The real cost concerns come with deploying the models, as both platforms require a fee to deploy the models for online predictions.

GCP:

When deploying an endpoint in GCP Vertex AI, you'll be charged based on how many API calls you make. According to Google, for 0-2M API calls per month, you'll be charged $0, for 2M-1B API calls per month, you'll be charged $3.00 and for 1B+ API calls per month, you'll be charged $1.50. You'll also be charged while your model is deployed. In our case, we used Vertex AI with text based data, which incurs $0.05 per hour of deployment. You also get charged for various other features you use on the cloud such as cloud storage and cloud logging. We had one model saved in our storage bucket, one model created, and one model deployed to an endpoint for approximately 6 days and to this point, have been charged exactly $13.85. Google does provide $300 free for 90 days when you create a new Google Cloud account

AWS:

AWS Sagemaker has a similar pricing scheme to GCP. Depending on your deployed model's memory, instance, and CPU size, you'll be charged at different rates per hour while your model is deployed. For example, an "ml.t3.medium" instance, which has a vCPU of 2 and 4 GiB of memory is $0.05 per hour the instance is deployed. AWS also charges per use of its storage platform S3. The first 50 TB / month is $0.023 per GB and the price slowly increases as you increase the usage size. AWS provides what they call a "free tier" to new users, which provides customers the ability to explore and try out AWS services free of charge up to specified limits for each service for up to 12 months. Unfortunately, while this free tier is active, AWS does not show you any charges, so we could not find specifics for our project.

Overall, the pricing schemes are very similar between the two platforms and both provide a free trial for new users. This is to be expected since Amazon and Google are competitors, so the platform of choice for a user really boils down to preference or specific use case.

Model Testing

Model Error Reduction Methods:

1. Normalization (Min-Max, Standardization, Robust)
2. Identifying peaks of data samples
3. Compartmentalization of data points in each sample
4. Unsupervised learning (autoencoding data)

Results

We first tried normalizing our data using 3 different methods: min-max, standardization, and robust. Min-max normalization is a technique used to transform numerical features to a specified range. Standardization, also known as Z-score normalization, is a data preprocessing technique used to transform numerical features so that they have a mean of 0 and a standard deviation of 1. Lastly, Robust normalization scales features using statistics that are robust to the presence of outliers in the data. All of these methods resulted in marginally better results, with min-max normalization generally outperforming the others by a small amount. We also found normalization was most effective with the entire dataset and less effective with compartmentalized samples.

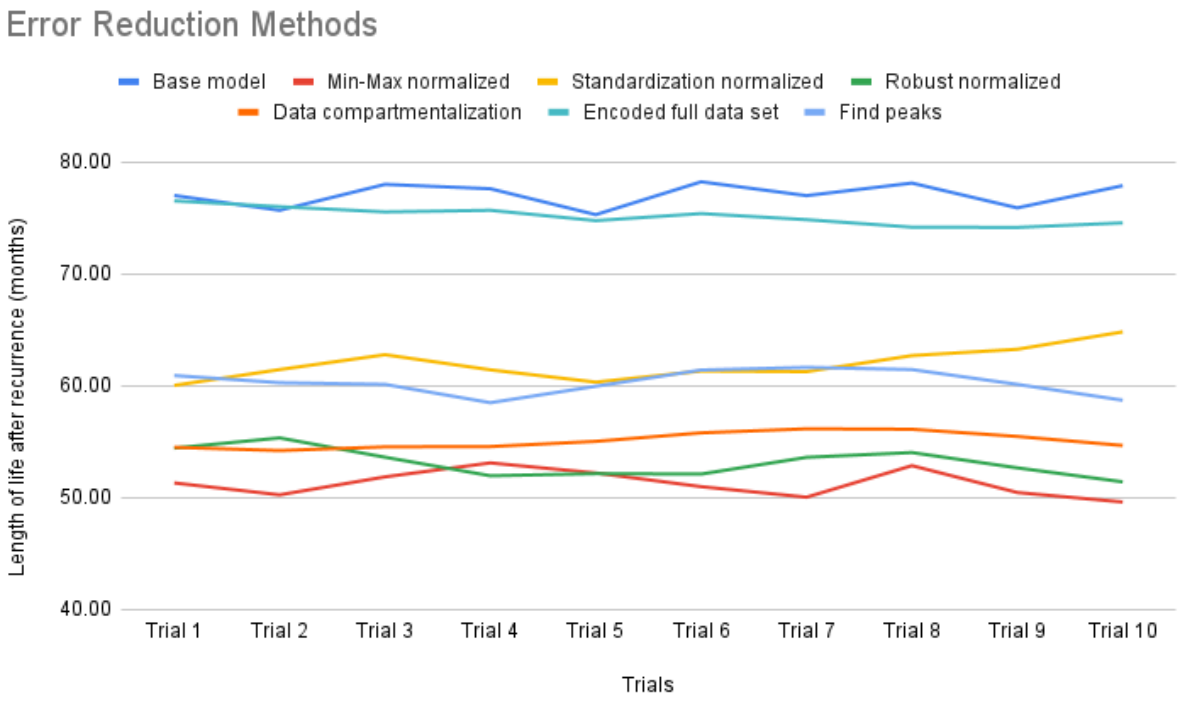
Another method we employed was finding the peaks of each sample. It was thought that the peaks had more significance in the outcome than other portions of the data. This was done by treating each sample as a graph and finding the highs and lows. We devised a simple Python method for extracting the peaks of each datapoint in the preprocessing step. This resulted in a lower error than our base model but was less successful than normalization.

The next method we tried was compartmentalizing our data into different sections such as the first 250 data points, the first 500 data points, etc. We believed that not every single data point was important in indicating the patient's outcome, so we wanted to see if only certain sections of the data lead to a more accurate prediction. From this, we were able to see similar decreases in error as we saw in normalization, however, using compartmentalization and normalization together didn't show any significant improvements over using either of them separately.

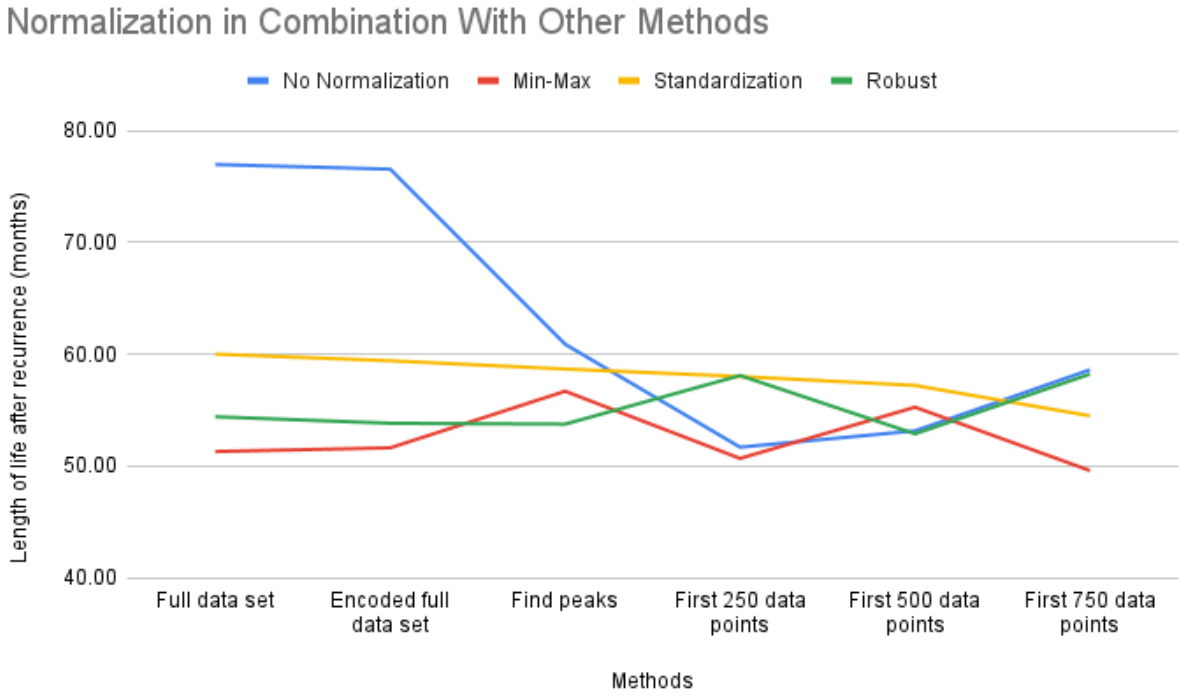
Our last error reduction strategy was using unsupervised learning in the form of an autoencoder. Unsupervised learning is a machine learning strategy in which the model looks for patterns without explicit supervision or labeled responses. The type of unsupervised learning we employed was autoencoding, in which the model compresses input data into a lower-dimensionality representation, in our case, in hopes of revealing some prominent pattern

or significant feature in our data. Overall, this didn't produce any significant improvement over our base model and was found to be inconclusive.

In the end, we determined that simplifying the data in some way, whether through normalization or compartmentalization, was the best way to reduce overall error. Within those methods, we saw improvements from our base model in every strategy and the most successful were min-max normalization and compartmentalizing the first 250 data points of each sample. However, we did not find that any of these reduction methods proved to be significantly more effective in combination than separately.



This chart gives a quantified representation of our testing findings through various trials of our different preprocessing methods. We tested each data preprocessing method through 10 trials and compared the mean absolute error of the model for each. It is clear that min-max normalization consistently produced the best mean absolute error of about 50 months through the 10 trials.



Next, we wanted to test combining multiple methods. Each preprocessing method was combined once with each of the normalization methods. It was evident that each preprocessing method had the lowest mean absolute error when combined with min-max normalization, but was not significantly different than with min-max normalization by itself.

Application Testing

End-to-End testing

We tested the overall behavior of the application via end-to-end testing, in which sample data was input into the application, and a prediction was returned. The prediction was then compared to a raw prediction request directly on our model in Google Cloud. The purpose of this test was to ensure that the results the user got are the same as expected and data was not getting corrupted or otherwise messed up throughout the course of data transfer.

We ran 15 sample data files through our application and noted the prediction that came with it. We took those same 15 data files and ran them directly into our model code on Colab using the model.predict() function built into Tensorflow. Upon comparing the results, we were able to verify that the predictions returned were expected, indicating that the application could reliably be used to serve predictions.

Stress Testing

One question we had surrounding the performance of our application regarded how many predictions could be made at once. It's understood that this number may vary depending on factors such as the instance type, model complexity, and resource allocation. Because of this, we wanted to get a good idea of how many predictions could be made at once before performance starts to decline. The simplest way to do this was to make numerous requests simultaneously from different instances of our application.

With a single instance of the application running, a prediction is returned to the user within a second. We were able to run the application with 5 users requesting predictions simultaneously. This had no evident effects on latency in returning predictions. We have reason to believe that there is a point in which too many users making predictions at once would cause latency due to resource contention. However, we can't feasibly determine its limits with our setup. In a real world scenario, it's safe to assume that the model would be deployed on a high-performance node or split across multiple endpoints to reduce resource consumption and be able to serve predictions to numerous users at once.

Broader context

Area	**Description**	**Examples**
Public health, safety, and welfare	For many software projects, there is an expected level of quality since many of them deal with safety, health, and welfare, which can directly impact people	Our product directly impacts public health by providing a tool that can aid in the diagnosis of cancer.
Global, cultural, and social	It is important to consider factors such as accessibility, language barriers, and cultural sensitivities when designing and employing a piece of software	Our project aims to benefit communities worldwide by offering a technology-driven solution for cancer diagnosis but does not necessarily reflect the values, practices or aims of any specific cultural group.
Environmental	Some software can have a direct impact on environmental processes	We don't expect our project to have much of a direct or indirect environmental

		impact.
Economic	Software projects can be expensive and have issues that can cost money. It is important to make a design as cost-efficient as possible. Some software can also be used to help decrease spending on other things.	As our project could potentially lead to improved patient outcomes in the future, this could result in reduced healthcare costs.
Ethical	There are certain ethical considerations to be made with our project since we are dealing with confidential data.	Since we are dealing with real patient medical data, we are legally bound to ensure this data remains confidential to only ourselves and medical professionals.
User Experience	There must be a quality user experience when the user is interacting with our product.	Since the users will be allowed to upload medical data via a front-end application, this process must be seamless and have no flaws or security leaks.
Continuous Improvement	Software must incorporate continuous improvements as new information surfaces and bugs are discovered in the design.	If new cancer diagnosis techniques get discovered or new medical data become available, we need to incorporate them to ensure our model maintains the highest accuracy possible.

Conclusions

We have designed an application that can retrieve predictions from a hosted machine-learning model that we developed. The model was developed using the Keras neural network framework and numerous data processing techniques.

We were able to prove that several data preprocessing techniques were useful in reducing the error of our model. However, we were never able to obtain consistently accurate predictions. Across our entire data set, the average survival number was 148.46 months. Throughout all of our testing, every prediction was between 138 - 153 months, which indicates that the model seems to be predicting near the average survival number every single time. This was always the case, no matter what changes we made to the model architecture or preprocessing. This leads us to believe that there is potentially no distinct correlation between the spectra data and the

survival data and the changes we observed were from the different preprocessing methods. There may be more advanced methods of neural network design and data preprocessing that we are unaware of that could expose a correlation, but given our experience and timeframe, we were not successfully able to prove any meaningful connection. There's also a chance that this use case is too specific and complex for machine learning to handle.

An important aspect of the project that we felt was detrimental to our success was our understanding of the sample data. It was explained to us that we should simply treat the data as "some data from some sensor" which represents some encoded image of a cell and that its implications were beyond the scope of the project. However, we felt that if we were given some more specifics about what the data actually represented, it could have yielded better results. For example, the added context could have led to better reference material and more complex data preprocessing methods.

Future steps:
Currently, our model's accuracy is unfit to provide value to users in a realistic setting. The predictions would have to be vastly more reliable to be fit for a medical device. However, the functionality of our application works for our use case, so only real updates would have to be made in the model's accuracy. Some techniques we anticipate that could be useful include:

1. Data pre-processing method we have not tried. We employed several methods such as normalization, data scaling, extracting peaks, and data encoding. There's a chance that one of these methods can be refined into providing something useful, but perhaps there's a different pre-processing method we are unfamiliar with that would lead to better results.
2. Advanced neural network techniques. Our work with the Keras framework has only spanned the last semester and a half. As such, there is still much for us to learn in terms of model architecture. Perhaps there is an advanced technique that involves number of layers, types of layers (e.g., convolutional, recurrent), and layer sizes that we have yet to try that can expose a potential correlation within the data.

Only a medical professional would be able to determine if there was a realistic correlation within the data, and perhaps spending more time working with the dataset would have proven useful for us. For now, the data remains a mystery. As machine learning evolves and becomes more sophisticated a project of this magnitude will only become more plausible.

References

"API Documentation | Tensorflow v2.14.0," *Tensorflow*. http://www.Tensorflow.org/api_docs (accessed Sep. 15, 2023).

"Google Cloud Platform Overview," *Google Cloud*. http://cloud.google.com/docs/overview (accessed Sep. 20, 2023).

"Graphical User Interfaces with Tk," *Python Documentation*. http://docs.python.org/3/library/tk.html (accessed Nov. 10, 2023).\

"IEEE Code of Ethics," *IEEE*, 2019. http://www.ieee.org/about/corporate/governance/p7-8.html (accessed Sep. 20, 2023).

"ISO Code of Ethics and Conduct," *ISO*, Jan. 01, 1970. http://www.iso.org/publication/PUB100011.html (accessed Sep. 20, 2023).

"Keras: Deep Learning for humans," *Keras.io*, 2018. http://Keras.io (accessed Sep. 20, 2023).

Machine Learning Service - Amazon Sagemaker - AWS, aws.amazon.com/sagemaker/. Accessed 2024.

O. for C. Rights (OCR), "Health Information Privacy," *HHS.gov*, Jun. 09, 2021. http://www.hhs.gov/hipaa/index.html

"Welcome to Flask¶." *Welcome to Flask - Flask Documentation (3.0.x)*, flask.palletsprojects.com/en/3.0.x/. Accessed 2024.

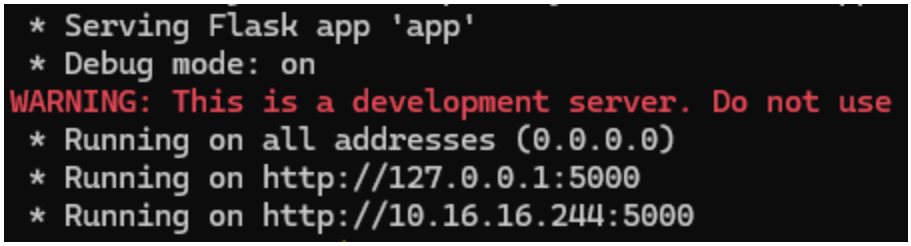
Appendix 1 - Operation Manual

Serving Application:

In order to serve predictions, the Flask app must be deployed on a device. The source code for the application can be found at:

https://git.ece.iastate.edu/sd/sdmay24-47/-/tree/main/WebApp?ref_type=heads

Once the user has cloned the project, they need to run the application with the command "flask run -h 0.0.0.0." This assumes the user has already set up a Python and Flask environment. The application is designed to run on all available IP addresses of the device. Once running, it will tell the user what IP addresses the app is running on.



Once the application is running, a user can visit the given https address to make a prediction as long as they are on the same network.

Retrieving Predictions:

This instruction manual will demonstrate to a user how to interact with our model to return a prediction once the application is running

Step 1: The user needs to gather their spectra data.

Our model is designed to return predictions based on a collection of spectra vectors. The provided data must be in csv format, with the first two columns occupied by the points

Step 2: Navigate to the appropriate address.

Since the address at which the app is deployed can vary based on the current connection of the host device, it's important to consult the previous section about deploying the device. For example, if the IP address is "10.26.45.1111," the user needs to visit "https://10.26.45.111:5000." The user will then be directed to the application

Step 3: Import CSV spectra data

There is a button located in the middle of the application, named "upload CSV." Upon clicking this button, the user's file system will be automatically opened. The user must locate their saved CSV file for processing and select it.

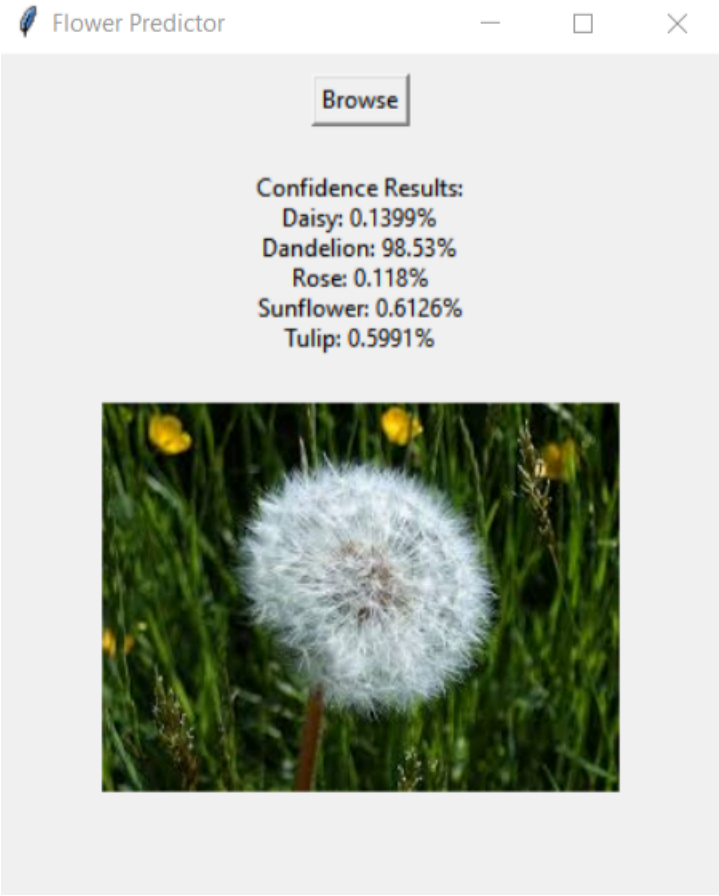
Step 4: View prediction

A section labeled "Prediction" is located towards the bottom of the application. Once the user uploads their CSV data, a prediction should immediately be returned to the user. This prediction represents their prognosis as a number of months based on our model

Step 6: Repeat

The user can upload as much data as they want in a single session. Each time they upload a new file, a new prediction will be returned .

Appendix 2 - Initial version of design



Here is what the early design of our application looked like with the flower classification model. This proved to be a good outline for our current application design and was a good starting point with machine-learning.

```
from google.colab import files
import os
import tensorflow as tf
assert tf.__version__.startswith('2')

from mediapipe_model_maker import image_classifier

import matplotlib.pyplot as plt

image_path = tf.keras.utils.get_file(
    'flower_photos.tgz',
    'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
    extract=True)
image_path = os.path.join(os.path.dirname(image_path), 'flower_photos')


print(image_path)
labels = []
for i in os.listdir(image_path):
  if os.path.isdir(os.path.join(image_path, i)):
    labels.append(i)
print(labels)


NUM_EXAMPLES = 5

for label in labels:
  label_dir = os.path.join(image_path, label)
  example_filenames = os.listdir(label_dir)[:NUM_EXAMPLES]
  fig, axs = plt.subplots(1, NUM_EXAMPLES, figsize=(10,2))
  for i in range(NUM_EXAMPLES):
    axs[i].imshow(plt.imread(os.path.join(label_dir, example_filenames[i])))
    axs[i].get_xaxis().set_visible(False)
    axs[i].get_yaxis().set_visible(False)
  fig.suptitle(f'Showing {NUM_EXAMPLES} examples for {label}')

plt.show()
```

```
data = image_classifier.Dataset.from_folder(image_path)
train_data, remaining_data = data.split(0.8)
test_data, validation_data = remaining_data.split(0.5)

spec = image_classifier.SupportedModels.MOBILENET_V2
hparams = image_classifier.HParams(export_dir="exported_model")
options = image_classifier.ImageClassifierOptions(supported_model=spec, hparams=hparams)

model = image_classifier.ImageClassifier.create(
    train_data = train_data,
    validation_data = validation_data,
    options=options,
)

loss, acc = model.evaluate(test_data)
print(f'Test loss:{loss}, Test accuracy:{acc}')

model.export_model()

!ls exported_model
files.download('exported_model/model.tflite')
```

Here is the code that powered our first model design. It was a basic Tensorflow image classification model with 80% training, 10% testing, and 10% validation data

Appendix 3 - Relevant Code

Model

The code used to train our base model consists of a few main parts.

```
'''Iterates over every file uploaded, parses each into a separate dataframe and puts it in a list'''
# Directory containing CSV files
directory = '.'
# Lists all files in the directory
file_list = sorted(os.listdir(directory))

# Initializes an empty list to store DataFrames
dataframes = []

# Iterates over each file in the directory
for file_name in file_list:
    if file_name.endswith('.csv'):
        # Constructs the full file path
        file_path = os.path.join(directory, file_name)
        # Reads the CSV file into a DataFrame and appends it to the list
        df = pd.read_csv(file_path, header=None, names=['X', 'Y'])
        df = min_max_normalize(df)
        dataframes.append(df)
```

In the first section, we iterated through each of our samples and parsed them into separate Pandas dataframes.

```
'''Extracts survival values from common_numbers_data.xlsx'''
# Loads the Excel file into a pandas DataFrame
df = pd.read_excel('common_numbers_data.xlsx')

# Extracts the values from the "sample" column into a list
survival_values = df['Survival'].tolist()
sample_endings = df['sample'].tolist()
```

Next, we iterated through our survival data and split it into the patient numbers and their corresponding survival values.

```
# Creates an empty list to store individual dataframes and numbers
data = []

# Iterates over each dataframe and its corresponding dependent value
for i in range(0, len(dataframes)):
    cur_df = dataframes[i] #Gets the current dataframe
    dep_val = survival_values[i]  # Gets the corresponding dependent value from the list

    # Creates a new DataFrame with one row containing the current dataframe and its dependent value
    row_data = pd.DataFrame({'DataFrame': [cur_df], 'Dependent': [dep_val]})

    # Appends the row to the data list
    data.append(row_data)

# Combines all the rows into a single DataFrame
result_df = pd.concat(data, ignore_index=True)
```

We then used those patient numbers to match survival data to samples and compiled them into one larger dataframe to train the model.

```
X = result_df['DataFrame'].apply(lambda x: x.values).tolist()
y = result_df['Dependent'].tolist()

# Splits the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Converts lists to numpy arrays
X_train = tf.constant(X_train)
X_test = tf.constant(X_test)
y_train = tf.constant(y_train)
y_test = tf.constant(y_test)

# Builds and trains the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(2449, 2)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100)

# Evaluates the model
loss = model.evaluate(X_test, y_test)
print("Mean Squared Error on Test Data:", loss)
```

Finally, we split our final dataframe into input and output variables, divided them into training and testing data, defined our model, and trained it on our data.

Application

```python
from google.protobuf import json_format
from google.protobuf.struct_pb2 import Value
from flask import Flask, render_template, request
from sklearn.preprocessing import MinMaxScaler
import os
import pandas as pd
import numpy as np

app = Flask(__name__)
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]=r"path/to/json"

@app.route('/')
def index():
    return render_template('index.html', predictions=None)

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return render_template('index.html', predictions=None)
    file = request.files['file']
    if file.filename == '':
        return render_template('index.html', predictions=None)
    if file:
        # Preprocess the uploaded CSV file
        data = pd.read_csv(file, header=None, names=['X', 'Y'])

        # Min-max normalization
        scaler = MinMaxScaler()
        columns_to_normalize = ['X', 'Y']
        data[columns_to_normalize] = scaler.fit_transform(data[columns_to_normalize])

        # Conver to numpy
        data = data.values

        # Construct model input instance
        instance = []
        for i in range(len(data)):
            instance.append([data[i][0], data[i][1]])

        # Reference the existing endpoint by its ID or name
        endpoint_id = '5558750359012245504'

        # Get the endpoint object
        endpoint = aiplatform.Endpoint(endpoint_id)

        # Get predictions from the existing endpoint
        predictions = endpoint.predict(instances=[instance])
        first_prediction = predictions[0][0][0] if predictions else None

        print(first_prediction)

        return render_template('index.html', predictions=first_prediction)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True, ssl_context=('cert.pem', 'key.pem'))
```

This is the code to our Flask application. It is responsible for processing the data points within the uploaded csv file. It then correctly parses tha data into an endpoint instance to get a prediction from our model. Once a prediction is returned, only the first prediction updated into the html template.

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Mulish:ital,wght@0,200..1000;1,200..1000&display=swap"
    rel="stylesheet">
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
        integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
    <title>Accurate Cancer Prediction</title>
    <style>
        body {
            background-color: #29355D;
            /* Set background color */
            color: #FFFF;
            /* Set text color*/
            margin: 0;
            /* Remove default margin */
            padding: 0;
            /* Remove default padding */
            font-family: Mulish;

        }

        .container {
            text-align: center;
            font-size: 19px;
            padding: 20px;
            /* Add padding for better readability */
        }

        .instructions {
            text-align: left;
            display: inline-block;
            margin: 0 auto;
            font-size: 19px;
        }

        .button-container {
            text-align: center;
            margin-top: 20px;

            /* Adjust as needed */
        }
    </style>
</head>

<body>
    <nav class="navbar" style="background-color: #e3f2fd;">
        <!-- Navbar content -->
      </nav>
    <div class="container">
        <h1>Accurate Cancer Predictor</h1>
        <h2>How to use:</h2>
```

```
        <div class="instructions">
            <ol>
                <li>Gather CSV pathology data</li>
                <li>Upload the data using the button</li>
                <li>Cancer predictor will return your prediction!</li>
            </ol>
        </div>
        <div class="button-container">
            <form id="uploadForm" method="POST" enctype="multipart/form-data" action="/predict">
                <input id="fileInput" type="file" name="file" style="display: none;">
                <button id="uploadButton"  type="button" class="btn btn-outline-light">Upload CSV</button>
            </form>
        </div>
    </div>

!-- </body> -->

head>
    <title>Predictions</title>
/head>

!-- <body> -->
    <div class="container">
    <h1>Predictions:</h1>
        {% if predictions %}
                {% for prediction in predictions %}
                    Based on our model, we predict a prognosis of {{ prediction }} months
                {% endfor %}
            {% else %}
                No predictions yet. Please upload a CSV file.
            {% endif %}

    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
        integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
        crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
        integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
        crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
        integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
        crossorigin="anonymous"></script>
    <script>
        document.getElementById('uploadButton').addEventListener('click', function() {
            document.getElementById('fileInput').click();
        });

        document.getElementById('fileInput').addEventListener('change', function() {
            document.getElementById('uploadForm').submit();
        });

        document.getElementById('fileInput').value = ''; //clears form to allow for resubmission
    </script>

    </div>
/body>

/html>
```

The previous two images are our html template. It initializes all of the buttons, text and colors in the UI. Once the user uploads data and gets a prediction, it takes them to a new page that displays the prediction. The user can upload as many predictions as they desire in one session. No data is saved in the application