Capstone Final Submission

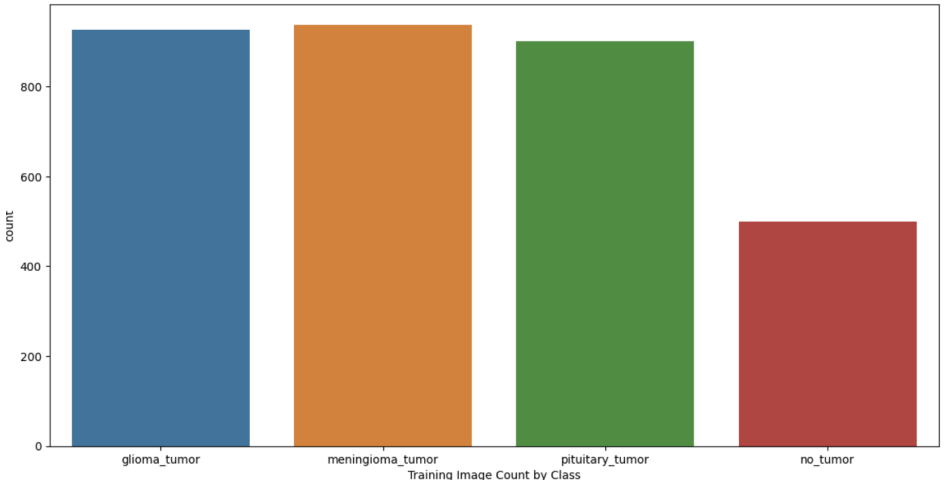
Concept and Objectives

For my capstone project I created a machine learning model for predicting brain tumors in humans using MRI (magnetic resonance imaging) scans. My approach is a supervised learning project using multiclass classification with four potential outputs, using deep learning (a convolutional neural network) for image recognition. The four outputs correspond to three brain tumor types (glioma, meningioma, and pituitary) plus the possibility of no tumor being present in a scan. For both training and testing the model I created I used the Kaggle dataset found at: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri/download?datasetVersionNumber=2

Time is of the essence in determining the health of a patient with potential brain tumors and starting any necessary treatment, with a goal of improving the life expectancy of a patient. However, there are a lot of abnormalities in the sizes and locations of brain tumors, making it really difficult to completely understand the nature of the tumor. Currently MRI scans are the best technique to detect brain tumors. The scans have to be examined by a specialist (ex. neurologist or radiologist). This project uses a trained predictive model to show how machine learning and artificial intelligence can help with three major issues:

1. By reducing the likelihood of human error via an additional review of a scan by an algorithm. The error rate due to human interpretation of radiological scans is roughly 30%, and is largely unchanged over the past 70 years. [1] Interruptions in the workplace contribute to human error, as does how well rested the person reading the scan results is and bias. Outsourcing of the interpretation of these scans may also be a factor.
2. The lack of an available specialist in certain geographic areas.
3. Quicker detection of tumors.

Tumor distribution in training data:



[1] https://www.rsna.org/news/2022/march/human-error-in-radiology

Development of model

For model development I followed a typical machine learning process, first by collecting the data and preprocessing it. For my project I found it worked best to both resize the images to a size known to work best with the underlying model (VGG16) and to augment the number of no tumor images since there were roughly half as many of those as there were images of each of the 3 main tumor types. After splitting the images into train and test groups, I then fine tuned the VGG16 model. This took several iterations and trying out different layers on top of the base VGG16 model. After repeated iterations I developed a model that approached 99% accuracy. The model development work was done in a Jupyter notebook, which lends itself to more rapid iteration of python code for machine learning. GitHub was used for storage and version management of all critical files (https://github.com/stjoha81/Capstone_2023_2024/tree/main).

Finetuned model layers:

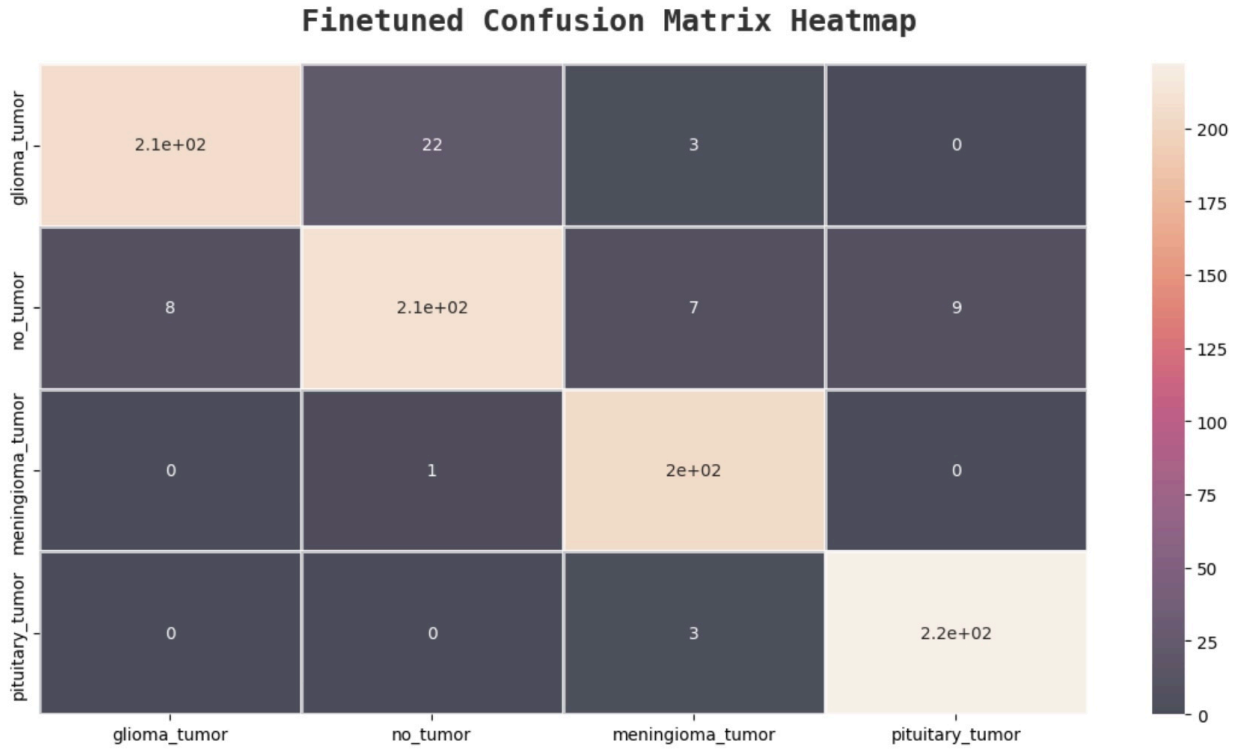
In [13]:

```
# Now create model for fine tuning and add trainable layers to the base VGG16 model.

# Define the Keras model
model2_finetuned = Sequential()

# Add the vgg16 model as the base model.
model2_finetuned.add(base_vgg16_model)

# Flatten to get the right shape?
model2_finetuned.add(Flatten())

# Add a dropout layer to prevent overfitting. Set dropout rate to 0.5.
model2_finetuned.add(Dropout(rate=0.5))

# Add a dense layer to get ouput. Chose softmax activation since this is a multiclassification problem.
model2_finetuned.add(Dense(4, activation='softmax'))

# Summarize the model.
model2_finetuned.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dropout (Dropout)           (None, 32768)             0

 dense (Dense)               (None, 4)                 131076

=================================================================
Total params: 14845764 (56.63 MB)
Trainable params: 131076 (512.02 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

A confusion matrix and classification report were used to evaluate the model.

Confusion matrix:



Classification report:

```
# Classification report.

# 0 - Glioma Tumor
# 1 - No Tumor
# 2 - Meningioma Tumor
# 3 - Pituitary Tumor

finetuned_classification_report = classification_report(y_test,y_predicted)
print(finetuned_classification_report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.89      0.93       232
           1       0.90      0.90      0.90       234
           2       0.94      1.00      0.97       205
           3       0.96      0.99      0.97       225

    accuracy                           0.94       896
   macro avg       0.94      0.94      0.94       896
weighted avg       0.94      0.94      0.94       896
```

Development of Flask web application

After achieving satisfactory testing results (based primarily on confusion matrix data) with the trained model, the next step was to create a Flask application. The Flask application provides a web interface (based on HTML) for the user to upload brains can images, and it serves as a front end to the API calls to the model that process the images, make a prediction for a specific image with respect to tumor classification, and returns the predicted classification result to the web browser

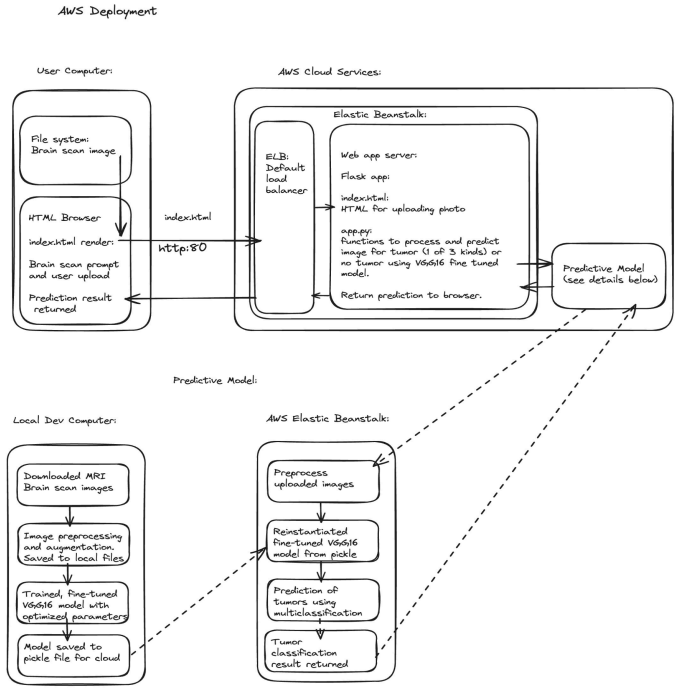
interface. The Flask model was tested repeatedly on my local development machine, an Apple Macbook Pro with an M3 processor.

Cloud Deployment to AWS

For cloud deployment, I chose to use AWS due to its large popularity for machine learning applications. The specific service I chose was Elastic Beanstalk, because it automatically handles a lot of the operational systems necessary for production deployment and monitoring of a web application. Specific functions that Elastic Beanstalk provides include load balancing for request management, relatively straightforward scaling up of the number of environments running the web application, and logging of critical metrics for application health (ex. Memory usage, latency, number of requests, etc.).

Based on a recommendation from my mentor I chose to deploy my capstone application using Docker. Though not covered in the course material, Docker provides a convenient way to package up an application and its dependencies so the application can be deployed on hardware platforms and software operating systems other than the local development environment I used. Using Docker required building a Docker image, and testing it out by running it in a Docker container on my laptop. I then created another Docker image using the same process, but pointed the image to my Docker Hub repository and also used the platform option to specify that I would be using AMD based hardware rather than Apple specific hardware. After I pushed the image to my Docker Hub, I then logged into AWS Elastic Beanstalk and created an environment for the capstone application to run in. Setting up this environment included selecting a JSON based file on my local computer that tells AWS the Docker Hub to pull my image from.

Architecture diagram:



Results

After multiple tries I was able to successfully deploy my capstone application in my AWS environment. It took approximately 20 minutes from when the deployment process started to when my web application was available via a URL that called the Elastic Beanstalk environment I set up, which was running the Docker image of my app that I had pushed to Docker Hub. The app ran successfully, meaning that multiple users were able to upload images and nearly instantaneously get a predicted classification result.

My capstone project demonstrates the ability of machine learning to address the current shortcomings of brain cancer diagnosis that relies solely on human interpretation of an MRI brain scan. The application consistently delivers a highly accurate tumor prediction based on a user supplied image, without the distraction and interruption that a doctor would typically face. It does this in a very rapid manner, eliminating the need to wait hours or days for a preliminary diagnosis. By using a web application for an interface, the application is capable of being used from anywhere in the world and therefore helping to address the lack of adequately trained specialists in parts of the world.

Conclusion and Lessons Learned

I would like to conclude this capstone project final submission by expressing my wonder at training an image recognition model to accomplish a task that may help save people's lives. Simply phenomenal. I would also like to express my sincerest gratitude to my mentor Bikash for his help and guidance throughout this process. He has a tremendous amount of knowledge and readily shares it. A key lesson learned for me is that going forward if I am writing code then I need to pay closer attention to recommended best practices for new technologies as I start to use them, rather than needing to go back and clean up technical details later on. Finally, during the course of this machine learning and AI program I have accomplished what I set out to do: I learned in depth about this rapidly evolving field that has an increasing impact on people's everyday lives, and I got a new job as a product manager. While that job did not specifically require the machine learning skills that I learned in this program, I do believe that I will have opportunities to apply my machine learning skills in the future.