Classification of Tumor-Containing and Non-Tumor-Containing Brain MRI's With Convolutional Neural Networks

Dillon Bangasser
Patrick Carr
November 30th, 2022

Abstract

Convolutional Neural Networks (CNNs) are machine learning algorithms that have been widely demonstrated for use in computer vision, because of their dimensionality reduction pooling layers that still capture image features. In this study, convolution and pooling layers were manually coded to process MRI images of tumor-containing (TC) and non-tumor-containing (NTC) brains, to be learned by a Keras Sequential, Fully-Connected neural network. Three hidden layers with Relu and Softmax activation functions and 80, 20, and 5 neurons were used and fed to a sigmoid output layer. Sensitivity analyses were performed on the convolution and pooling layer hyperparameters of kernel element size, kernel geometry, and reshaping size. It was found that a 10-filled kernel provided a marginally better accuracy of 79.2%, confirmed with 10-fold validation than 1-filled or 100-filled kernels. When reshaping, it was found that a [300x300] reshape size balanced the high false negative error of the [200x200] dataset and the high false positive error of the [400x400] dataset, with the highest overall test set accuracy of 79.2%. An 'x' shape kernel was found to again be marginally more accurate than a 't' shape kernel, but not significantly more accurate within error. Significant test set accuracy was achieved by the model, but confusion matrices revealed that the model had more false positive error than false negative error, implying bias in the prediction method. The model learning curves revealed high variance, which is an area of improvement for future study.

Introduction

The field of image classification had its origins in the related discipline of computer vision starting in the 1960s, but has only recently entered a period of accelerated progressc[1]. In 2010, the arrival of ImageNet, an online visual database presently comprising more than 14 million annotated images, was accompanied by a number of advancements in quick succession [1, 2]. The innovations emerging at the time took place within a class of deep learning algorithms named convolutional neural networks (CNNs). The key difference between CNNs and conventional feed-forward nets is the convolution layer, which simplifies the original image without the loss of important information by the application of filters [3]. AlexNet was one of the first deep CNNs to attain considerable accuracy in the 2012 ImageNet Large Scale Recognition Challenge, achieving an accuracy of 84.7 % whereas the runner-up only achieved an accuracy of 73.8%. AlexNet, the architecture of which is shown below in Figure 1, was followed by a series of notable breakthroughs in the CNN space including VGGNet, ResNet, and GoogLeNet [4].

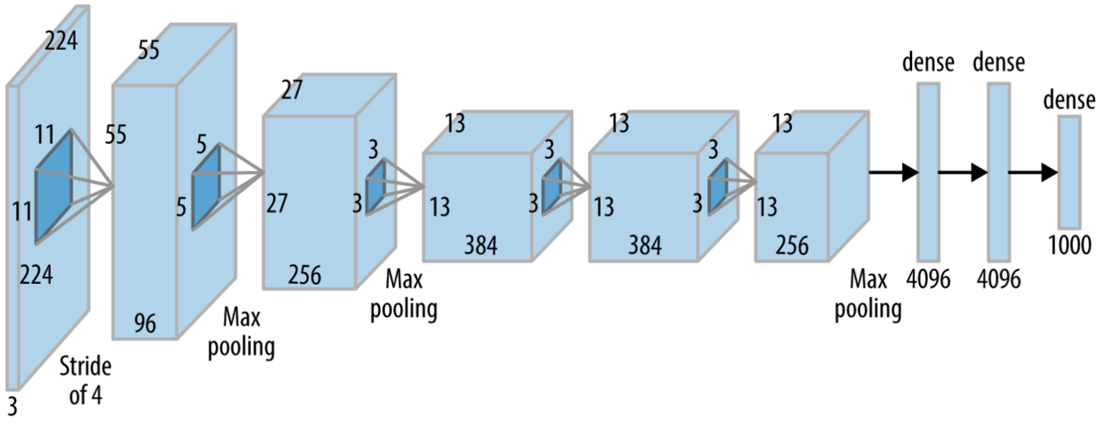

Figure 1. AlexNet Architecture [4].

With such considerable progress in a relatively short period of time, many were inspired by the potential applications of these image classification CNNs. One of the most promising uses for this technology is in medical imaging, where it has the possibility to allow earlier and more accurate detection of diseases such as cancer and Alzheimer's. Although this technology seems to have a budding future within medical imaging, it has yet to be widely adopted because of concerns over the generalizability of the models. Though many algorithms perform very well on the data on which they have been trained, they have not been validated with diverse data sets which would provide confidence for their widespread use. Another cause for hesitation is that many of the models are effectively "black boxes," meaning that experts cannot know how the models arrive at their conclusions [5]. A number of solutions have been suggested in order to address the aforementioned issues. The use of larger datasets has been put forward as a possible remedy for biased models, however, it has been shown this cannot be implemented universally as a simple fix as in some cases upon evaluation with a larger dataset some models have become less accurate [6].Despite the concerns machine learning scientists continue to develop new algorithms which could potentially offer expert-level detection to health systems without the necessary resources to afford a highly-trained physician.

Though the quality and quantity of the data fed to a model is extremely important, the purpose of this study was to improve detection of brain tumors in MRI images based on the modification of the CNN architecture and the preprocessing protocol. This optimization of the

CNN architecture and preprocessing method was performed by considering various metrics from the models such as learning curves, k-fold cross validation accuracies, and confusion matrices.

Methods

First, a dataset was obtained from Kaggle [7], containing horizontal MRI cross-sections of the brain . Each datum was labeled 'yes' or 'no', indicating the presence or absence of a tumor respectively. No other labels or features were included with the data, nor information regarding the origins of the images. While other datasets were explored to use for this study that contained more background information and features associated with each datum, the computer memory and processing power available to the researchers limited the size of the dataset and quality of images that could be stored, imported, and processed effectively. The chosen dataset, titled "Brain MRI Images for Brain Tumor Detection," contained 253 images, with 155 images containing tumors, and 98 images containing no tumor.

The tumor-containing images suggest that an intravenous contrast agent was used to enhance abnormalities in the brain tissue, evident in the significant discoloration of the tumor region relative to other areas of the brain. Images in the dataset showed the results of either T1 or T2 MRI imaging. In T1 imaging, only the fat tissues in the brain are targeted, where in T2 imaging, fat and water compositions are targeted. Examples of these two image types are shown in Figure 2.

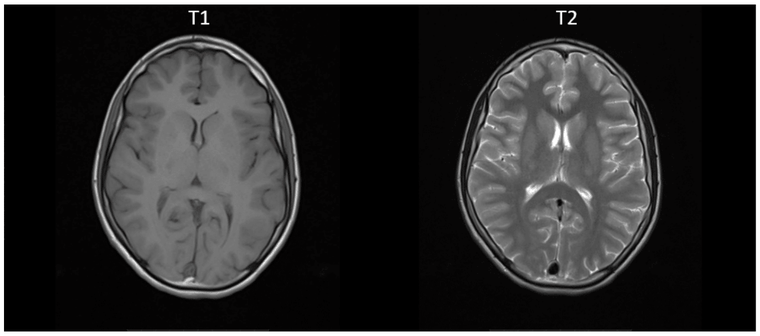

Figure 2. T1 and T2 imaging of the same brain cross-section [8].

As shown in Figure 2, images resulting from T1 imaging are lighter in the brain matter and darker towards the boundaries of the brain. T2 imaging, though, provides higher intensity imaging around the extremities and edges of the gyri, and darker intensities inside the brain. The effect of image type on model accuracy was determined to be beyond the scope of this study, and the combined dataset was used to train a robust model that could learn from either image type. Optimization of the image type hyperparameter is a potential avenue for future study. These images were then uploaded into the convolutional neural network created by the group, whose architecture is shown in Figure 3.

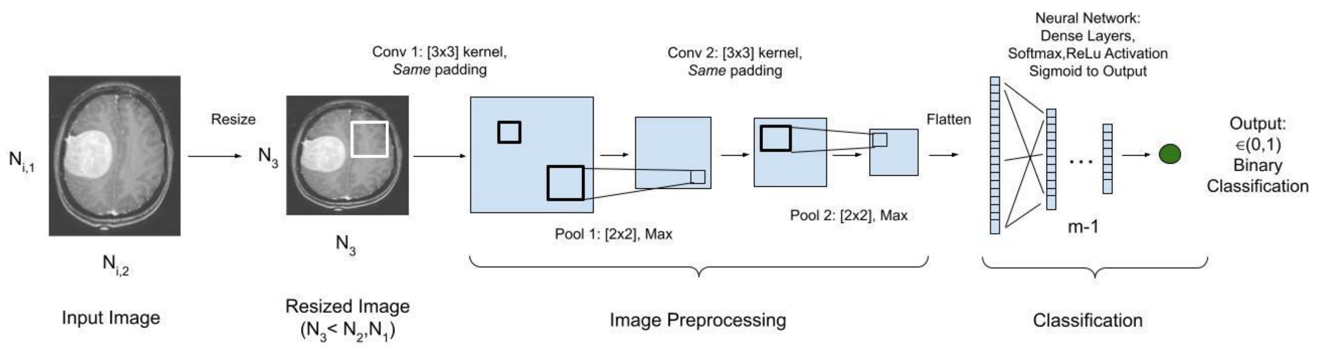
Neural Network: Dense Layers, Softmax,ReLu Activation Sigmoid to Output
Conv 1: [3x3] kernel, *Same* padding
Conv 2: [3x3] kernel, *Same* padding
Resize
Flatten
$N_{i,1}$
N_3
Output: ∈(0,1) Binary Classification
Pool 1: [2x2], Max
Pool 2: [2x2], Max
m-1
$N_{i,2}$
N_3
Input Image
Resized Image ($N_3 < N_2, N_1$)
Image Preprocessing
Classification

Figure 3. Neural Network Architecture

First, the images contained in the dataset were not of uniform size or aspect ratio, so they had to first be resized to create a uniform model input. A 1:1 aspect ratio was chosen for simplicity. Because the aim of the study is classification, not quantification, of tumor tissues, the warp from the resizing should not introduce significant error into the system. The resizing was done with three different final dimensions (N_3 = 200,300,400) to test the sensitivity of the model accuracy on input size, using tensorflow.image.resize(). Some of the images contained multiple layers in the third dimension, like an RGB tensor, while others contained information in only two dimensions. To use the given resize function, the image must be three dimensional, and so a dummy array was added in the third dimension of two dimensional image arrays to fix the processing error. The resizing merged the dummy dimension and actual third dimension from all of the images, resulting in a two dimensional pixel array.

After resizing to a 1:1 image of size (N_3, N_3,1), the images were subjected to a two-dimensional convolution layer (Conv1) with a [3x3] kernel and "same"-type padding. To achieve "same"-type padding, the input image was padded one-deep with zeros, which forces the output of the convolution layer to have the same size as the input. The convolution was manually coded, using for loop iteration and element-wise matrix multiplication to apply and store the kernel. Three different [3x3] kernels were used, in order to evaluate the sensitivity of the model on the kernel contents. A [3x3] kernel was chosen following the advice of TowardsDataScience [3]. These kernels are shown in Table 1.

Table 1. Kernel Matrices for Conv1 and Conv2 Layers

Kernel Label	Kernel Contents
1x	[[1,0,1] [0,1,0] [1,0,1]]
10t	[[0,10,0] [10,0,10] [0,10,0]]
10x	[[10, 0 ,10] [0 , 10 , 0] [10, 0 , 10]]
100x	[[100, 0 ,100] [0 , 100 , 0] [100, 0 , 100]]

The 1x kernel was chosen as a standard group kernel, which amplifies the current pixel intensity value with its diagonal neighbors, which are outside the directions of motion of the kernel, giving it more explorative properties while representing the image element itself. The 1t kernel was chosen as a delocalization kernel, assigning each element's value to the intensity of the cells around it. This exclusively weights clusters over noise, and prioritizes the area around an element, not the element itself. The sensitivity of the model accuracy on the kernel shape was later evaluated.

Finally, the 100x kernel was chosen as an amplification kernel. It had the same "x-shape" as the 1x kernel, but was scalar-multiplied by a factor of 100. This increased the difference in orders of magnitude between high intensity clusters and medium/low intensity clusters. The performance of this dataset relative to the 1x kernel was later evaluated.

After the first convolution, the data were sent to a [2x2] maximum pooling layer (Pool1), cutting the dimensions of the image in half to ($N_3/2$, $N_3/2$, 1). This was also manually coded, using for loop iterations and the numpy library. After Conv1 and Pool1, the data were passed through Conv2 and Pool2, which had the same properties and unit operations as the first convolution and pooling layers, respectively. The final size of the convolved/pooled image before further processing was ($N_3/4$, $N_3/4$, 1).

Before entering a neural network, the image array has to be flattened into a single dimension. The ($N_3^2/16$, 1) row vector functioned as the input layer to the neural network. The row vector for each image was normalized with the pixel intensity mean and standard deviation, and train_test_split() from scikit-learn was used to generate a training set and validation set, using a random state of 42, a test size of 0.30, and stratification of labels. A sequential model from the keras library was used. Three hidden layers were used to train the model, in order to optimize model accuracy and performance. The hidden layers used the Dense() layer from the keras package, with the first layer using the ReLu activation function and subsequent layers using the SoftMax activation function, which is best for classification objectives. The number of neurons were 80, 20, and 5 respectively [9].

The output layer was always a one neuron layer, with a sigmoid activation function. As such, the training images were assigned labels of 1 if the image had a tumor and 0 if the image did not contain the tumor. The output value was taken to be 1 if greater than or equal to 0.5, and 0 if less than 0.5, which was used as the model prediction. After the model was trained with the training set and associated neural network hyperparameters, the confusion matrix was constructed by comparing the model predictions with the known data labels, as shown in Table 2.

Table 2. Construction of Confusion Matrix

Outcome	Predicted Label	Actual Label
True Positive	1	1
False Positive	1	0
True Negative	0	0
False Negative	0	1

By generating the confusion matrix, in addition to the overall accuracy, the accuracy for positive images and negative images was calculated, by dividing the true predictions over the total number of positive and negative images respectively. This method of analysis gives insight into the bias of the model and classification strength.

Finally, K-fold validation (k=10) was used to get an average accuracy with associated variance for each dataset/hyperparameter trial, using the Kfold function contained in the scikit-learn package. Learning curves were also generated to assess model bias and variance by evaluating model loss with increasing numbers of data points.

The Jupyter Notebook containing the above operations is located in Appendix A.

Results and Discussion

First, the input and output images from the convolution and pooling layers (CPLs) were compared, in order to make sure that the convolution and pooling strategies and hyperparameters were effective. CPL inputs and outputs for images not containing tumors are shown in Figure 4.

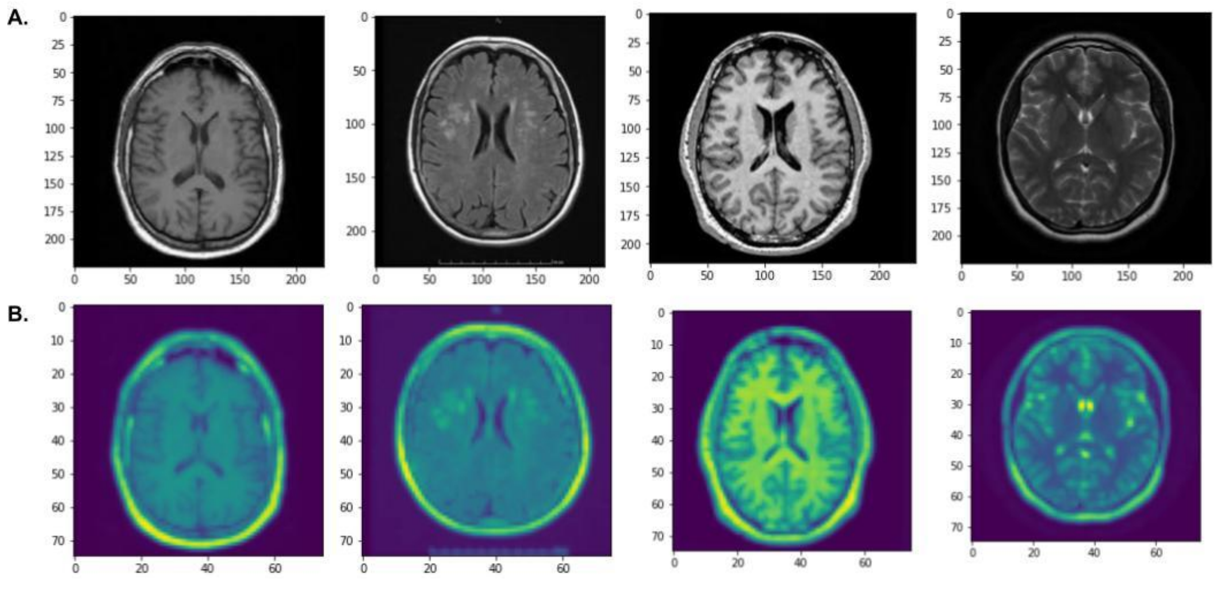
Figure 4. A) Original images from Kaggle dataset showing various MRIs without tumors, including both T1 and T2 MRI types. B) Associated convolved images with arbitrary intensity colormap.

As shown in Figure 4, the manual convolution and pooling algorithms worked as predicted, and effectively reduced the problem dimensions. Most notably, the convolved images in Figure 4-B are blurred relative to the input images. This is beneficial for further analysis, because the noise has been significantly reduced, and the off-target loci, like the folds of the brain, or gyri, have been smoothed. Thus, for the non-tumor-containing (NTC) images , the blurry convolved images take on a very symmetrical, uniform appearance, a potential source of differentiation for the model versus the tumor-containing images, shown in Figure 5.

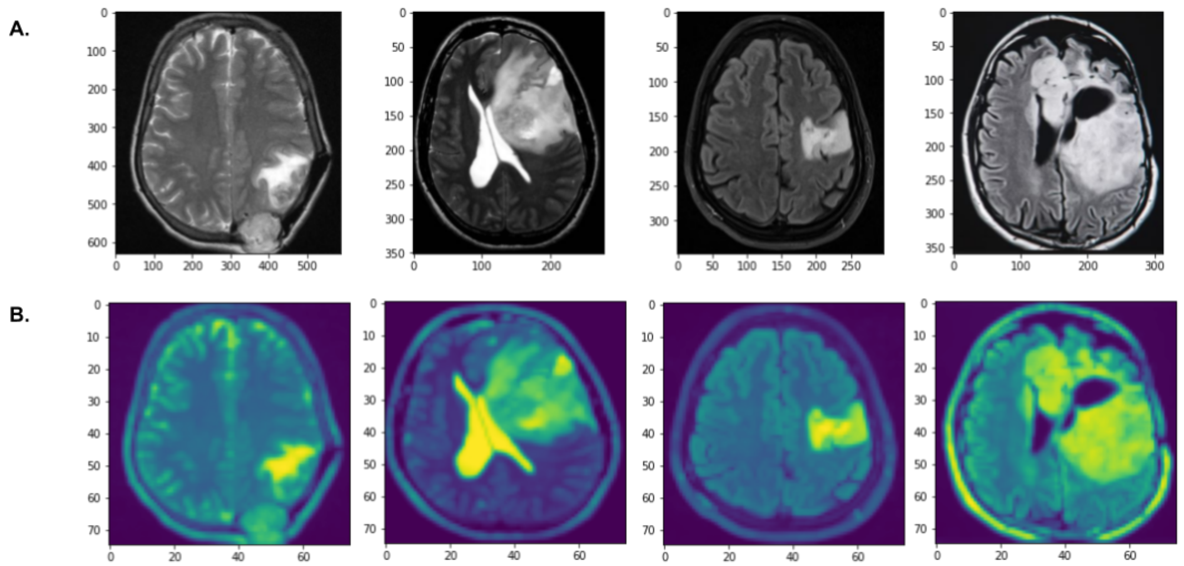
Figure 5. A) Original images from Kaggle dataset showing various MRIs containing tumors, including both T1 and T2 MRI types. B) Associated convolved images with arbitrary intensity colormap.

The convolution of tumor-containing (TC) images was also effective. Because the tumors are local masses of high pixel intensity, the CPLs amplified tumor loci more than other parts of the brain, resulting in convolved images where the contrast between the tumor and the rest of the brain is much larger than in the original image. Similar to the NTC convolutions, the output is blurred, reduced in dimension, and contains less noise.

In order to better understand the model and how to improve it, the flattened feature vectors were investigated. Since the neural network does not see the image, but instead a row vector of pixel intensity values, the row vector intensities were plotted for pattern recognition, as shown in Figure 6.

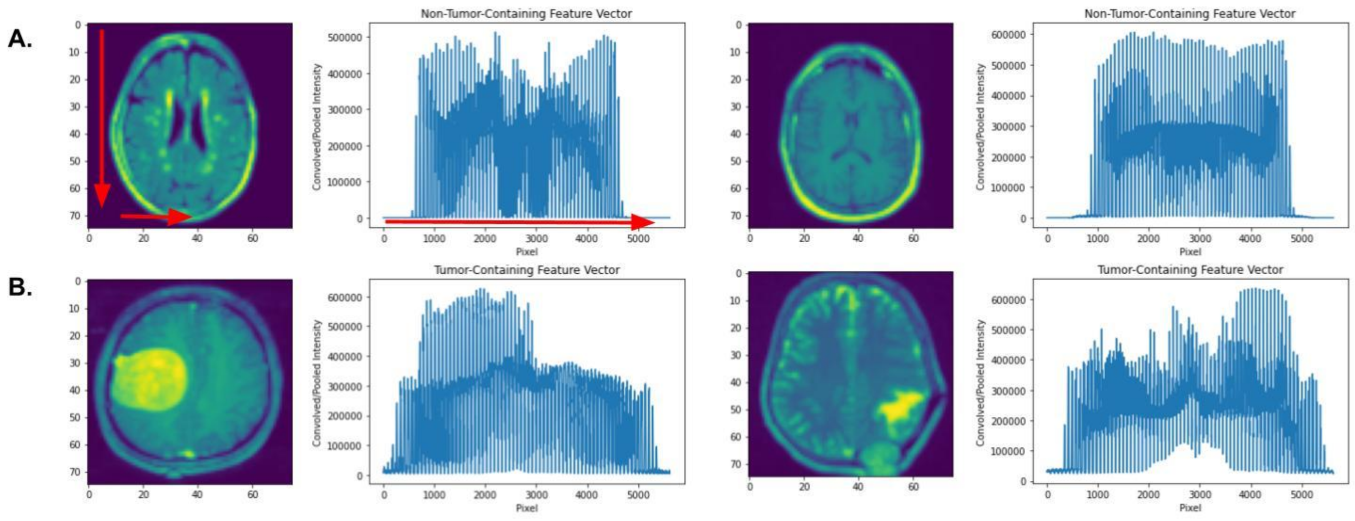

Figure 6. A) Pixel intensity values in NTC convolved, flattened feature vectors for pattern recognition. The flattened vector shown in the graph was created by stacking the columns from left to right, as indicated by the red arrows. B) Pixel intensity values in TC convolved, flattened feature vectors.

Theoretically, a healthy brain is symmetrical across the vertical axis, and so when the columns are stacked across the symmetry, they should produce symmetrical intensity patterns. As can be seen in Figure 6-A, the typical NTC, convolved image produces a relatively symmetrical intensity pattern along the cerebral fissure that divides the left and right hemispheres. For a brain containing a tumor, that symmetry is broken, as can be seen in the pixel intensities in Figure 6-B. The tumors show up as asymmetric regions with amplified intensities relative to the rest of the graph. This offers a potential insight into the mechanism by which the neural network weights different pixel regions in order to classify an image; in addition to this, refining models to seek this pattern could be a future direction of study for quicker and less computationally involved screening.

Once the efficacy of the CPLs in creating better input layers was evaluated, a sensitivity analysis was performed on the dependence of model accuracy on the kernel scalar amplification. Small kernel elements do not significantly amplify features of the image during convolution, which may cause the model to overlook important image features; inversely, large kernel elements could potentially over-amplify parts of the image, which may blind the model to important parts of the image. Thus, k-fold validation was used to test the effect of uniform kernels with element values of 1, 10, and 100, as previously shown in Table 1, in order to optimize that hyperparameter of the manual convolution. Each kernel tested had the "x" geometry, and the images underwent the same [300x300] resizing. The average test set accuracies and associated uncertainty of the 10-fold validation are shown in Figure 7.

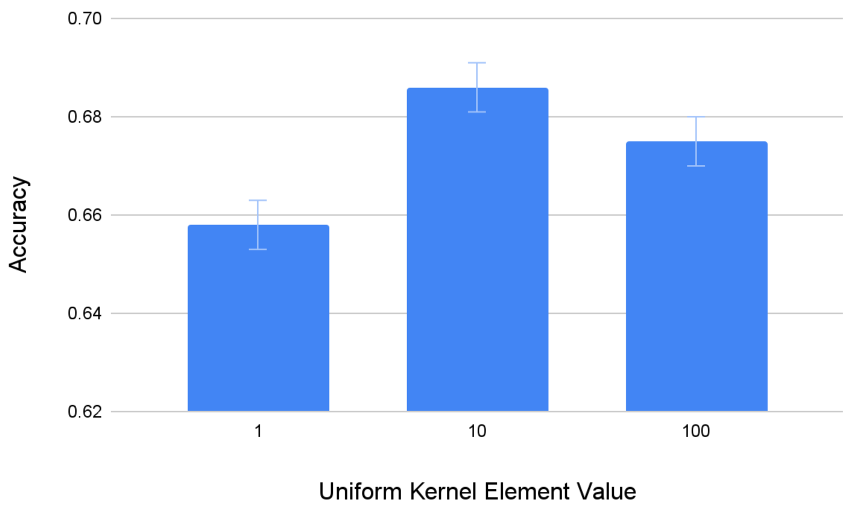

Figure 7. Sensitivity of Model Accuracy on Kernel Magnitude

The results of the 10-filled trial provided the highest overall accuracy, which confirms the expectations that under-amplification and over-amplification are detrimental to model accuracy. The uncertainties of each accuracy are large relative to the value of the difference between the accuracies of the 10 and 100 trials, but both are significantly more accurate than the 1 trial. In addition to this, the 10-filled kernel had the highest test set accuracy (from the 30% train/test split) at 79.2%, with the 1 and 100 trials having overall accuracies of 77.9% and 75.3% respectively. This slightly differs from the k-fold results, since the 1-filled kernel performed better than the 100-filled kernel. This could potentially be attributed to the large test set sizes.

In addition to this, the confusion matrices were constructed for each trial by comparing the given labels with the trained model predictions. These results are shown in Table 3.

Table 3. Test Set Confusion Matrices from the Kernel Element Sensitivity Analysis

Element Value	True Positives	True Negatives	False Positives	False Negatives
1	42	18	12	5
10	42	19	11	5
100	40	18	12	7

The confusion matrices did not largely vary between trials, but the 10-filled kernel again slightly outperformed the other kernels. The model correctly identified one more negative image than both of the other trials, and two more positive images than the 100-filled trial, when data convolved with the 10-filled kernel. Each dataset had significantly more false positive identifications than negative identifications, indicating that the model leaned more toward a positive classification when unsure. However, in diagnostic applications, it is preferable to have the model be over-cautious and label false positives than under-cautious and miss positive diagnoses with false negative identifications. The source of this model bias is unknown, and a potential avenue for future study.

The second sensitivity analysis was performed on the dependence of the model accuracy on the image resizing shape. The three image resize shapes used were [200x200], [300x300], and [400x400]. Reducing the size of an image in turn reduces the number of features that the model must handle, but comes at the cost of a potential loss of important information. Therefore, this analysis was meant to illustrate the optimal resize shape of the images. The average test set accuracies and associated uncertainty of the 10-fold validation are shown in Figure 10.

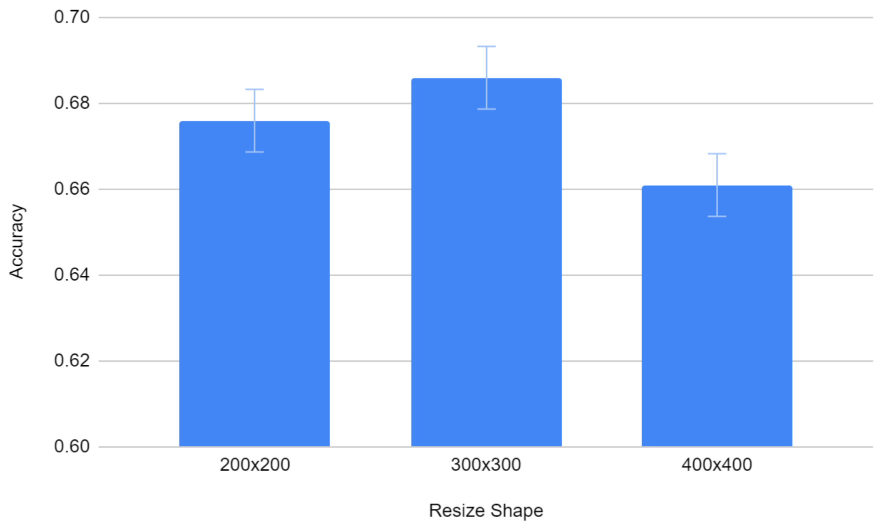

Figure 8. Sensitivity of Model Accuracy on Resize Shape

The figure above does not seem to show that there is a resize shape with a 10-fold validation accuracy that is significantly higher than the rest. The confusion matrices were calculated for all three image resize shapes as well and results are shown in the Table 4 below.

Table 4. Test Set Confusion Matrices from the Resize Shape Sensitivity Analysis

Resize Shape	True Positives	True Negatives	False Positives	False Negatives
200x200	38	20	10	9
300x300	42	19	11	5
400x400	44	16	14	3

There seems to be a fair degree of variance between the confusion matrices for the three image resize shapes. Though all three have a higher number of false positives than false negatives, notably, the [200x200] image resize shape only has one more false positive than false negatives. It is preferable to have more false positives than false negatives since it is better for a non-cancerous patient to receive a positive result than it is for a cancerous patient to receive a negatice result. The overall test accuracies for [200x200], [300x300], and [400x400] image resize shapes were 75.3%, 79.2%, and 77.9%, respectively. Therefore, the [300x300] image resize shape seems to be the most accurate image resize shape.

The final sensitivity analysis was also performed on the dependence of the model accuracy on the shape of the kernel by comparing the 10x and 10t kernels. The resize shape was [300x300] for both models.This analysis was meant to elucidate whether a standard group kernel or a delocalization kernel was preferable. The average test set accuracies and associated uncertainty of the 10-fold validation are shown in Figure 9.

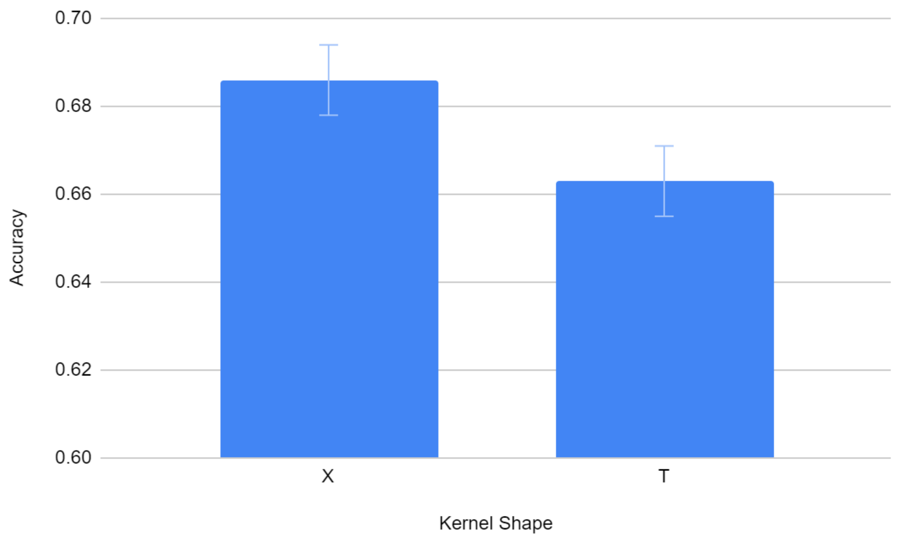

Figure 9. Sensitivity of Model Accuracy on Kernel Shape

The “x-shape” kernel has 10-fold validation accuracy that is significantly higher than that of the “t-shape” kernel. Perhaps the “x-shape” kernel is more adept at emphasizing edges and boundaries within the original image, potentially contributing to a more accurate detection of tumors. The confusion matrices were calculated for both kernel shapes as well and results are shown in the Table 5 below.

Table 5. Test Set Confusion Matrices from the Kernel Shape Sensitivity Analysis

Kernel Shape	True Positives	True Negatives	False Positives	False Negatives
X	42	19	11	5
T	41	19	11	6

Though the 10-fold validation accuracy was significantly higher for the “x-shape” kernel, the confusion matrices for the two shapes suggest that there is not much of a difference in accuracy between the two kernel shapes. The two models only disagreed about the classification of a single image.Again in both cases there were more false positives than false negatives, which, as has been discussed, is preferable to the alternative. The overall test accuracy was 79.2% for the “x-shape” kernel and 77.9% for the “t-shape” kernel. Therefore the shape of the kernel does not seem to have a significant effect on the accuracy of the model.

As shown above, the model is able to provide accurate data, but has a few limitations. First, with the size of the given dataset, different models were found to only disagree on one or two images, which could either be attributed to lack of sensitivity on the hyperparameters tested or random deviations in the model training. Thus, the model should be tested further with much larger datasets to allow for more statistically significant hyperparameter optimization. In addition to this, the model made more errors when identifying negative images than positive images, as the false positive rate was higher than the false negative rate in all trials but one. A potential explanation for this error is the amount of TC and NTC data used. 155 TC images were used while 98 NTC images were used. Thus, the model prioritized learning the TC images and minimized error in the TC images versus the NTC images. This also suggests that the model wasn’t completely effective in distinguishing tumors from healthy areas that happen to have clusters of high intensity. If this were the case, then the model would be able to identify most of the positive images correctly, but would also identify areas in negative images that looked like tumors. This is further supported by the model learning curves shown in Figure 10.

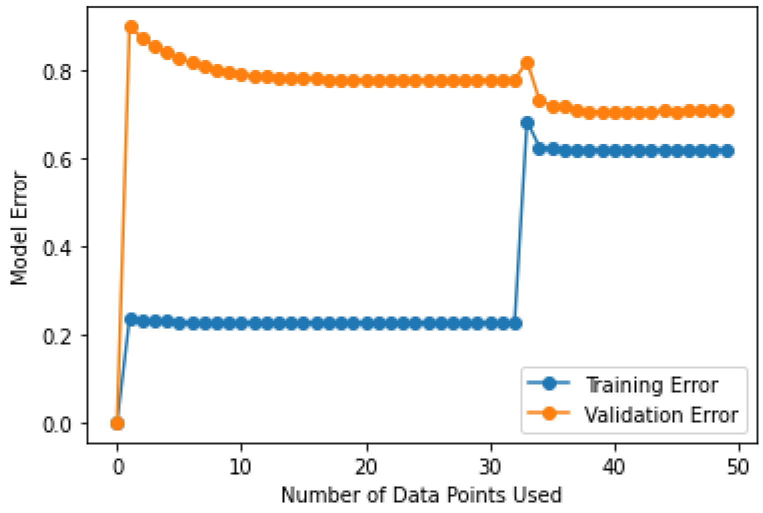

Figure 10. Optimized Model Learning Curves

The learning curves shown in Figure 10 indicate that the model has high bias, since the training error and validation error converge to a high model error relative to the initial validation size. This implies that the model may not be complex enough to completely represent the system and identify tumors, which is to be expected, since the model was theorized to work by simple pattern recognition in brain symmetry. Another area of future research would be to make the model more complex with other biological features or parameterization that give the model more information about how to tell whether high intensity pixel clusters in the images are healthy or tumors.

Conclusion

In conclusion, the group was able to manually code a convolutional neural network to accurately identify the presence of tumors in MRI images, with an optimized accuracy of 79%, and found that the model accuracy is not significantly sensitive to the model hyperparameters tested. The success of this application can be improved upon in future studies that continue to optimize other hyperparameters, use larger datasets, and implement more complex CNN libraries and algorithms that require higher processing power. In addition to this, the model's high variance should be studied to reduce the memorization of the model and increase true model learning, which will increase test set accuracies. The study of the use of computer vision in medical diagnostic technology, while already somewhat established, will continue to revolutionize medicine and treatment, as machine learning methods progress to learn more than humans can from patient information and testing.

Citations

1. *A history of computer vision & how it lead to 'vertical ai' image recognition*. Pulsar Platform. (2019, March 26). Retrieved November 30, 2022, from https://www.pulsarplatform.com/blog/2018/brief-history-computer-vision-vertical-ai-image-recognition/
2. ImageNet. (n.d.). Retrieved November 30, 2022, from https://www.image-net.org/
3. Saha, S. (2022, November 16). *A comprehensive guide to Convolutional Neural Networks-the eli5 way*. Medium. Retrieved November 30, 2022, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
4. Anwar, A. (2022, January 22). *Difference between alexnet, vggnet, ResNet and inception*. Medium. Retrieved November 30, 2022, from https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96
5. NIH. (2022). *Can artificial intelligence help see cancer in new ways?* National Cancer Institute. Retrieved November 30, 2022, from https://www.cancer.gov/news-events/cancer-currents-blog/2022/artificial-intelligence-cancer-imaging
6. Varoquaux, G., &amp; Cheplygina, V. (2022, April 12). Machine Learning for Medical Imaging: Methodological Failures and recommendations for the future. Nature News. Retrieved November 30, 2022, from https://www.nature.com/articles/s41746-022-00592-y
7. Chakrabarty, N. (2019, April 14). *Brain MRI images for Brain tumor detection*. Kaggle. Retrieved November 30, 2022, from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection/code
8. Interpretation·, R. H. R. (2021, November 12). *The basics of MRI interpretation: Radiology*. Geeky Medics. Retrieved November 30, 2022, from https://geekymedics.com/the-basics-of-mri-interpretation/
9. Krishnan, S. (2022, August 6). *How do determine the number of layers and neurons in the hidden layer?* Medium. Retrieved November 30, 2022, from https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3#:~:text=The%20number%20of%20hidden%20neurons%20should%20be%202%2F3%20the,size%20of%20the%20input%20layer.

Appendix A: Jupyter Notebook

Identification of Tumors in Brain MRI's via Classification in Convolutional Neural Networks

Import Libraries and Mount Google Drive

```
# Import necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.optimize as opt

import sklearn
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import tensorflow as tf

from sklearn.model_selection import learning_curve
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import KFold

import os
import cv2 as cv
from PIL import Image
import pprint
```

```
# Mount Drive
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

Convolution and Pooling: Feature Matrix Output

```
path_yes = '/content/drive/MyDrive/College/04 Senior/Machine Learning/Final Project/Data

path_no = '/content/drive/MyDrive/College/04 Senior/Machine Learning/Final Project/Datas

yes_imgs = os.listdir(path_yes)
```

```

no_imgs = os.listdir(path_no)

n_yes = len(yes_imgs)

n_no = len(no_imgs)

print(f'# Yes: {n_yes} # No: {n_no}  # Total: {n_yes+n_no}')
```

```
#### Generate Yes Feature Matrix ####

# Conv1 --> Pool1 --> Conv2 --> Pool2

n_yes = len(yes_imgs)

n_no = len(no_imgs)

final_pixlength = 75*75

yes_array = np.zeros((final_pixlength,n_yes))
no_array = np.zeros((final_pixlength,n_no))

for i in range(n_yes):

  img = Image.open(path_yes + '/' + yes_imgs[i])

  image_data = np.array(img)

  if len(image_data.shape) == 2:
    image_data = np.expand_dims(image_data,2)


  image_data = tf.image.resize(image_data,(300,300))

  image_data = image_data[:,:,0]

  kernel_n = 3

  filter = np.array([[0,10,0],
                    [10,0,10],
                    [0,10,0]])

  m,n= image_data.shape

  skip = 0

  conv1_shape = np.array([m,n])

  conv1 = np.zeros((conv1_shape[0],conv1_shape[1]))

  image_datap = np.pad(image_data,(1,1),'constant')
```

```

for k in range(conv1_shape[1]):

  for j in range(conv1_shape[0]):

    current_slice = image_datap[j:j+3,k:k+3]

    #print(current_slice)

    conv1[j,k] = np.sum(current_slice*filter)


m,n = conv1.shape

pool1 = np.zeros((int(m/2),int(n/2)))

for k in range(int(m/2)):

  for j in range(int(n/2)):

    pool1[k,j] = np.max(conv1[2*k:2*k + 2, 2*j:2*j+2])


m,n = pool1.shape

conv2_shape = np.array([m,n])

conv2 = np.zeros((conv2_shape[0],conv2_shape[1]))

pool1p = np.pad(pool1,(1,1),'constant')

for l in range(conv2_shape[1]):

  for p in range(conv2_shape[0]):

    current_slice = pool1p[p:p+3,l:l+3]

    #print(current_slice)

    conv2[p,l] = np.sum(current_slice*filter)



m,n = conv2.shape

###

pool2 = np.zeros((int(m/2),int(n/2)))

for k in range(int(m/2)):

  for j in range(int(n/2)):

    pool2[k,j] = np.max(conv2[2*k:2*k + 2, 2*j:2*j+2])


yes_array[:,i] = pool2.flatten()
```

```
# Validate and Save Yes Data
print(yes_array.T.shape)
plt.plot(yes_array.T[8,:])
```

```
pd.DataFrame(yes_array).to_csv("/content/drive/MyDrive/College/04 Senior/Machine Learnin
```

```
#### Generate No Feature Matrix ####

# Conv1 --> Pool1 --> Conv2 --> Pool2

for i in range(n_no):

  img = Image.open(path_no + '/' + no_imgs[i])

  image_data = np.array(img)

  if len(image_data.shape) == 2:
    image_data = np.expand_dims(image_data,2)


  image_data = tf.image.resize(image_data,(300,300))

  image_data = image_data[:,:,0]

  kernel_n = 3

  filter = np.array([[0,10,0],
                     [10,0,10],
                     [0,10,0]])

  m,n= image_data.shape

  skip = 0

  conv1_shape = np.array([m,n])

  conv1 = np.zeros((conv1_shape[0],conv1_shape[1]))

  image_datap = np.pad(image_data,(1,1),'constant')

  for k in range(conv1_shape[1]):

    for j in range(conv1_shape[0]):
```

```

    current_slice = image_datap[j:j+3,k:k+3]

    #print(current_slice)

    conv1[j,k] = np.sum(current_slice*filter)


m,n = conv1.shape

pool1 = np.zeros((int(m/2),int(n/2)))

for k in range(int(m/2)):

  for j in range(int(n/2)):

    pool1[k,j] = np.max(conv1[2*k:2*k + 2, 2*j:2*j+2])


m,n = pool1.shape

conv2_shape = np.array([m,n])

conv2 = np.zeros((conv2_shape[0],conv2_shape[1]))

pool1p = np.pad(pool1,(1,1),'constant')

for l in range(conv2_shape[1]):

  for p in range(conv2_shape[0]):

    current_slice = pool1p[p:p+3,l:l+3]

    #print(current_slice)

    conv2[p,l] = np.sum(current_slice*filter)



m,n = conv2.shape

###

pool2 = np.zeros((int(m/2),int(n/2)))

for k in range(int(m/2)):

  for j in range(int(n/2)):

    pool2[k,j] = np.max(conv2[2*k:2*k + 2, 2*j:2*j+2])

  no_array[:,i] = pool2.flatten()
```

```
# Validate and Save No Data
print(no_array.T.shape)
plt.plot(no_array.T[3,:])
```

```
pd.DataFrame(no_array).to_csv("/content/drive/MyDrive/College/04 Senior/Machine Learning
```

```
# Compare original and Convolved Images
num = 6

plt.imshow(no_array.T[num,:].reshape(75,75))
plt.show()

img = Image.open(path_no + '/' + no_imgs[num])
plt.imshow(img)
plt.show()

plt.imshow(yes_array.T[num,:].reshape(75,75))
plt.show()

img = Image.open(path_yes + '/' + yes_imgs[num])
plt.imshow(img)
plt.show()
```

Featurization, Normalization, and Neural Network Training

```
# Import Feature Matrix From Selected Dataset
nodata = pd.read_csv(r'/content/drive/MyDrive/College/04 Senior/Machine Learning/Final P
yesdata = pd.read_csv(r'/content/drive/MyDrive/College/04 Senior/Machine Learning/Final
```

```
# Check if orientation is correct
print(nodata.shape)
print(yesdata.shape)
```

```
# Reorient if necessary; Careful to only run once per desired transposition
nodata = nodata.T
yesdata = yesdata.T
```

```
# Experimental flip of flattening axis
yesdatat = np.zeros((yesdata.shape))
for i in range(yesdata.shape[0]):

  yesdatat[i,:] = np.array(yesdata.iloc[i,:]).reshape(20,20).T.flatten()

nodatat = np.zeros((nodata.shape))
for i in range(nodata.shape[0]):

  nodatat[i,:] = np.array(nodata.iloc[i,:]).reshape(20,20).T.flatten()

yesdata = pd.DataFrame(yesdatat)

nodata = pd.DataFrame(nodatat)
```

```
# Visualize row vector symmetry
w = 6

yesar = np.array(yesdata)
noar = np.array(nodata)

plt.plot(yesar[w,:])
plt.show()
plt.plot(noar[w,:])
plt.show()

```

```
# Construct feature matrix and label vector

X_feat = nodata.copy()
X_feat = X_feat.append(yesdata,ignore_index=True).iloc[:,1:]

X_feat.tail()

yes_label = np.ones(len(yesdata.iloc[:,0]))
no_label = np.zeros(len(nodata.iloc[:,0]))

y = np.append(no_label,yes_label)

print(y,y.shape)
```

```
# Normalize features
means = X_feat.mean(axis=1)
stds = X_feat.std(axis=1)

m,n = X_feat.shape

X_norm = np.zeros((m,n))

for i in range(m):
  X_norm[i,:] = (X_feat.iloc[i,:] - means[i]*np.ones(n))/ (stds[i]*np.ones(n))

pd.DataFrame(X_norm.T).describe()
```

```
# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.30,stratify=y

# Validate Stratification
plt.plot(y_train,'ro',markersize=2)
plt.plot(y_test,'bo',markersize=2)
```

```
# Function to build and adjust neural network/hyperparameters

def construct_model():

  NN = Sequential()
  NN.add(Dense(80,activation='softmax',use_bias=False))
  #NN.add(Dense(40,activation='softmax',use_bias=False))
  #NN.add(Dense(20,activation='softmax',use_bias=False))
  NN.add(Dense(5,activation='softmax',use_bias=False))

  NN.add(Dense(1,activation='sigmoid'))

  # Compile the model
  NN.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

  return NN
```

```
# Train model and plot accuracy/loss
# Plot code from https://vitalflux.com/python-keras-learning-validation-curve-classifica

model = construct_model()

history = model.fit(X_train, y_train, validation_data = (X_test,y_test),epochs=100, verb

history_dict = history.history

print(history_dict.keys())
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) + 1)
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Plot the model accuracy vs Epochs

ax[0].plot(epochs, accuracy, 'bo', label='Training accuracy')
ax[0].plot(epochs, val_accuracy, 'b', label='Validation accuracy')
ax[0].set_title('Training & Validation Accuracy', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=16)
ax[0].set_ylabel('Accuracy', fontsize=16)
```

```
ax[0].legend()

# Plot the loss vs Epochs

ax[1].plot(epochs, loss_values, 'bo', label='Training loss')
ax[1].plot(epochs, val_loss_values, 'b', label='Validation loss')
ax[1].set_title('Training & Validation Loss', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=16)
ax[1].set_ylabel('Loss', fontsize=16)
ax[1].legend()
```

Learning Curve

```
# Learning Curve Function

def learningcurve(X,y,X_val,y_val,m):
  '''
  Function to calculate training and validation error
  Inputs:
  X=training set features (assumes already has columns of 1)
  y=training set data
  X_val = validation features (assumes already has columns of 1)
  y_val = validation data

  Output:
  [error_train, error_val] = array with training and validation errors as a function of
  '''
###BEGIN SOLUTION

  assert (m <= len(y_val)), 'M is larger than the validation set'

  error_train = np.zeros(m)
  error_val = np.zeros(m)

  for i in range(1,m):

    history = model.fit(X_train[:m], y_train[:m], validation_data = (X_test[:m],y_test[:

    history_dict = history.history

    loss_values = history_dict['loss']
    val_loss_values = history_dict['val_loss']

    error_train[i] = loss_values[-1]
    error_val[i] = val_loss_values[-1]

  return [error_train,error_val]
```

```
# Generate Learning Curves

error_train,error_val = learningcurve(X_train,y_train,X_test,y_test,10)

plt.plot(error_train,'o-')
plt.plot(error_val,'o-')
plt.xlabel('Number of Data Points Used')
plt.ylabel('Model Error')
plt.show()
```

```
  # Plot model accuracy
  plt.plot(history.history['accuracy'])
```

Confusion Matrices

```
# Confusion Matrix Training

y_pred_train = model.predict(X_train)

for i in range(len(y_pred_train)):

  if y_pred_train[i] <0.5:

    y_pred_train[i] = 0

  elif y_pred_train[i] >=0.5:

    y_pred_train[i] = 1

print(y_pred_train.shape)

#plt.plot(y_pred_train,'o')
plt.plot(y_train,y_pred_train,'o')

pos = np.where(y_train==1)[0]
neg = np.where(y_train==0)[0]

pos_pred = np.where(y_pred_train >= 0.5)[0]
neg_pred = np.where(y_pred_train < 0.5)[0]

true_pos = len(np.intersect1d(pos,pos_pred))

true_neg = len(np.intersect1d(neg,neg_pred))

false_pos = len(np.intersect1d(neg,pos_pred))

false_neg = len(np.intersect1d(pos,neg_pred))

```

```
print(f'True Positive: {true_pos}, True Negative: {true_neg}, False Positive: {false_pos

print(f'Positive Identification Rate: {100*true_pos / (true_pos + false_neg): .2f}%')

print('')

print(f'Negative Identification Rate: {100*true_neg / (true_neg + false_pos): .2f}%')
```

```
# Confusion Matrix Test Data

y_pred_test = model.predict(X_test)

pos = np.where(y_test==1)[0]
neg = np.where(y_test==0)[0]

for i in range(len(y_pred_test)):

  if y_pred_test[i] <0.5:

    y_pred_test[i] = 0

  elif y_pred_test[i] >=0.5:

    y_pred_test[i] = 1

pos_pred = np.where(y_pred_test >= 0.5)[0]
neg_pred = np.where(y_pred_test < 0.5)[0]

plt.plot(y_test,y_pred_test,'o')
plt.show()

true_pos = len(np.intersect1d(pos,pos_pred))

true_neg = len(np.intersect1d(neg,neg_pred))

false_pos = len(np.intersect1d(neg,pos_pred))

false_neg = len(np.intersect1d(pos,neg_pred))

print(f'True Positive: {true_pos}, True Negative: {true_neg}, False Positive: {false_pos

print(f'Positive Identification Rate: {100*true_pos / (true_pos + false_neg): .2f}%')

print('')

print(f'Negative Identification Rate: {100*true_neg / (true_neg + false_pos): .2f}%')
```

K-Fold Validation

```
# K Fold Validation

# Code from https://vitalflux.com/k-fold-cross-validation-python-example/

model = construct_model()

strtfdKFold = StratifiedKFold(n_splits=10)
kfold = strtfdKFold.split(X_train, y_train)
scores = []

for k, (train, test) in enumerate(kfold):
    model.fit(X_train[train, :], y_train[train])
    score = model.evaluate(X_train[test, :], y_train[test])[0]
    scores.append(score)
    print('Fold: %2d, Accuracy: %.3f' % (k+1, score))

print('\n\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
```

✓ 44s completed at 2:22 AM