KARNAVATI UNIVERSITY

UNITEDWORLD INSTITUTE OF TECHNOLOGY(UIT)

Summative Assessment (SA)

Submitted BY

Niraj Sharma

(Enrl. No.: 20220701012)

Course Code and Title: 21BAIS99E43 – Artificial Neural Network

B.Sc. (Hons.) Computer Science / Data Science / AIML

IV Semester – Dec– April 2024

UIT

Contents

Abstract

One of the more serious illnesses that can affect both adults and children is a brain tumor. Eighty to ninety percent of primary tumors of the Central Nervous System (CNS) are brain tumors. Approximately 11,700 people receive a brain tumor diagnosis each year. For those with a cancerous brain or central nervous system tumor, the 5-year survival rate is roughly 34% for men and 36% for women. There are various classifications for brain tumors, including benign, malignant, pituitary, and others. The patients' life expectancy should be increased by using appropriate care, advance planning, and precise diagnosis. Magnetic Resonance Imaging is the most effective method for identifying brain tumors (MRI). The scans produce a massive amount of image data. The radiologist looks over these pictures. Because brain tumors and their characteristics are so complex, a manual examination can be prone to errors.

The use of artificial intelligence (AI) and machine learning (ML) to automate classification techniques has continuously demonstrated higher accuracy than manual classification. Therefore, it would benefit doctors worldwide to propose a system that uses Deep Learning Algorithms to perform detection and classification. These algorithms include Convolution Neural Network (CNN), Artificial Neural Network (ANN), and Transfer Learning (TL).

Objective of the project

The objective of this project is to determine the diagnosis of brain tumor given the MRI. The model can predict up to 3 variations of tumor. The project is built upon the concept of neural networks. A custom made convolutional neural network is implemented to diagnose the tumor.

Convolutional Neural Network

The extended form of artificial neural networks, known as convolutional neural networks (CNNs), is primarily used to extract features from grid-like matrix datasets. For instance, visual datasets with a lot of data patterns, like pictures or videos. The input layer, pooling layer, convolutional layer, and fully connected layers are some of the layers that make up a convolutional neural network. The input image is processed by the Convolutional layer to extract features, the Pooling layer reduces computation by downsampling the image, and the fully connected layer generates the final prediction. The network uses gradient descent and backpropagation to discover the best filters.

Layers

- Conv2D - A tensor of outputs is produced by convolving the layer input with the convolution kernel created by this layer.
- MaxPooling2D - A convolution neural network architecture's max pooling layer is an essential part that lowers the data's dimensions while assisting the neural network in extracting significant features from the input.
- Dense - The feature map that the flatten layer received from the max-pooling layer is transformed into a format that the dense layers can comprehend.
- Dropout - In order to avoid overfitting, a dropout layer in neural networks randomly sets input units to 0 during training. By excluding specific nodes from different training runs, this procedure creates the illusion that they are not a part of the network architecture.
- Dense - A dense layer is one that has strong connections to the layer before it, meaning that all of the neurons in that layer are connected to every other neuron in that layer.

Activation Function

- Relu - Rectified linear units, or ReLUs for short, are non-linear activation functions that are used in deep neural network machine learning. Another name for it is the function that activates rectifiers.
- Softmax - softmax activation function transforms a vector of real numbers into a probability distribution. In a multiclass classification problem, the output layer activation is the softmax function. The softmax function and the sigmoid function are comparable, but the softmax function adds up everything in the raw output in the denominator.

Loss Function

An event or the values of one or more variables are mapped onto a real number that intuitively represents some "cost" connected to the event by a

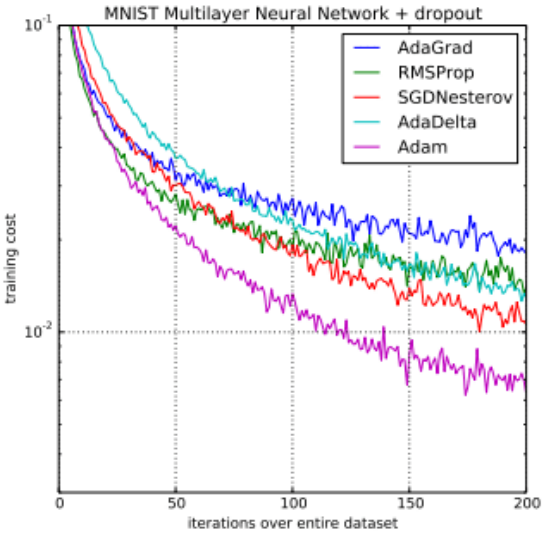
loss function, also known as a cost function. The goal of an optimisation problem is to reduce a loss function.

- Categorical cross entropy - In machine learning, categorical cross-entropy is a frequently utilised loss function, especially in classification tasks where the output variable is a categorical variable with multiple classes. It calculates the discrepancy between the classes' actual probability distribution and their anticipated probability distribution. The formula looks at how well the computer's guesses match the correct answers, and it tries to adjust the computer's guesses so that they become better over time.

Optimizers

Algorithms or techniques known as optimizers are employed to maximise production efficiency or minimise an error function, also known as a loss function. Mathematical functions called optimizers rely on the learnable parameters of a model, such as weights and biases. Optimizers aid in reducing losses by understanding how to alter the neural network's weights and learning rate.

Adam - The Adam optimizer, also known as "Adaptive Moment Estimation," is an iterative optimisation algorithm that reduces the loss function when neural networks are being trained. Adam can be thought of as a stochastic gradient descent with momentum combined with RMSprop. The adaptive gradient algorithm, or AdaGrad, enhances performance on problems with sparse gradients (such as computer vision and natural language problems) by maintaining a per-parameter learning rate.
Root Mean Square Propagation, which additionally sustains per-parameter learning rates adjusted according to the mean of the weight's gradients' recent magnitudes (i.e., the rate of change). This indicates that the algorithm performs well in non-stationary, online scenarios (noisy, for example).
Adam understands the advantages of RMSProp in addition to AdaGrad.



CNN Model

Data Collection

```
X_train = []
Y_train = []
image_size = 150
labels = ['glioma_tumor','meningioma_tumor','no_tumor','pituitary_tumor']
for i in labels:
    folderPath = os.path.join('../input/brain-tumor-classification-mri/Training',i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size,image_size))
        X_train.append(img)
        Y_train.append(i)

for i in labels:
    folderPath = os.path.join('../input/brain-tumor-classification-mri/Testing',i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size,image_size))
        X_train.append(img)
        Y_train.append(i)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
```

This Python code is used for loading and preprocessing image data for a machine learning task, specifically for a brain tumor classification problem. The images are stored in different folders based on their labels, which are 'glioma_tumor', 'meningioma_tumor', 'no_tumor', and 'pituitary_tumor'.

The code starts by initializing two empty lists, X_train and Y_train. X_train will hold the image data, and Y_train will hold the corresponding labels.

The image_size variable is set to 150, which will be used to resize all images to a standard size of 150x150 pixels. This is important because machine learning models usually require input data to be of the same size.

The code then iterates over each label. For each label, it constructs a path to the directory holding the images for that label in the 'Training' folder. It then iterates over each image file in that directory. Each image is read into memory using cv2.imread(), resized to 150x150 pixels using cv2.resize(), and then appended to the X_train list. The corresponding label is also appended to the Y_train list.

The same process is repeated for the images in the 'Testing' folder. However, it's worth noting that these images are also being appended to the X_train and Y_train lists, which might be a mistake. Typically, you would want to separate your training and testing data.

Finally, the X_train and Y_train lists are converted to NumPy arrays using np.array(). This is done because NumPy arrays provide more functionality and are more efficient than Python lists when performing mathematical operations, which are common in machine learning tasks.

Preprocessing

```
X_train,Y_train = shuffle(X_train,Y_train,random_state=101)
X_train.shape
```

```
(3264, 150, 150, 3)
```

The shuffle() function from sklearn's utils module is used to shuffle the X_train and Y_train arrays. Shuffling the data is important in machine learning to ensure that the training process is not biased by the order of the data. This can help improve the model's ability to generalize from the training data to unseen data.

The random_state parameter is set to 101. This is used for initializing the internal random number generator, which will decide the randomness of the shuffling. By setting a specific random_state, the shuffle operation will produce the same result every time it is run. This can be useful for creating reproducible results in experiments.

After shuffling, the shape of the X_train array is printed out using X_train.shape. This is likely done to verify that the shuffling operation has not changed the number of samples or features in the training data. The shape of a NumPy array is a tuple that gives the size of each dimension of the array.

```
X_train,X_test,y_train,y_test = train_test_split(X_train,Y_train,test_size=0.1,random_state=101)
```

The train_test_split function shuffles the dataset and then splits it into two parts. The first two arguments to the function, X_train and Y_train, are the feature matrix and the target array, respectively. These are the data that you want to split.

The test_size parameter is set to 0.1, meaning that 10% of the data will be used for the test set and the remaining 90% will be used for the training set. This is a common split ratio in machine learning, although the exact ratio can vary depending on the size and nature of the dataset.

The random_state parameter is set to 101. This is used to seed the random number generator used for shuffling the data, ensuring that the same train/test split is produced every time the code is run. This can be useful for creating reproducible results in experiments.

The function returns four arrays: the training data (X_train), the testing data (X_test), the training labels (y_train), and the testing labels (y_test). These are then used for training and evaluating the machine learning model.

```
y_train_new = []
for i in y_train:
    y_train_new.append(labels.index(i))
y_train=y_train_new
y_train = tf.keras.utils.to_categorical(y_train)

y_test_new = []
for i in y_test:
    y_test_new.append(labels.index(i))
y_test=y_test_new
y_test = tf.keras.utils.to_categorical(y_test)
```

This Python code is used to convert the categorical labels in the training and testing sets into numerical form, and then into one-hot vectors, for a machine learning task.

The code starts by initializing an empty list, y_train_new. It then iterates over each label in y_train, finds the index of that label in the labels list using the index() method, and appends that index to y_train_new. This effectively converts the categorical labels into numerical form, where each unique label is represented by a unique integer.

The y_train list is then replaced with y_train_new. After this, the tf.keras.utils.to_categorical() function is used to convert the numerical labels in y_train into one-hot vectors. One-hot encoding is a process of converting integer labels into binary vectors where the index of the integer label is marked with a 1, while all other positions in the vector are filled with 0s. This is a common way to represent categorical data in machine learning.

The same process is repeated for the y_test list. The end result is that both y_train and y_test are transformed from lists of categorical labels into 2D arrays of one-hot vectors.

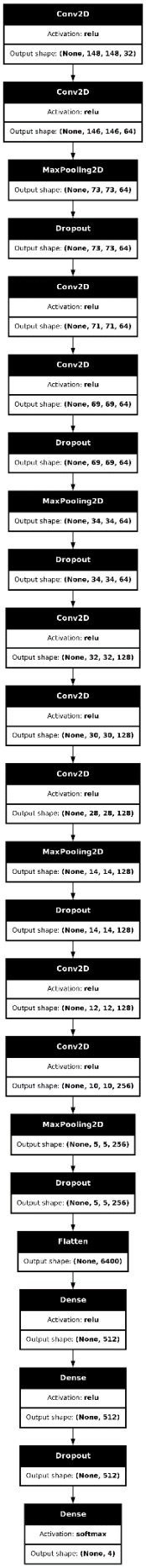
Model architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
conv2d_1 (Conv2D)	(None, 146, 146, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 73, 73, 64)	0
dropout (Dropout)	(None, 73, 73, 64)	0
conv2d_2 (Conv2D)	(None, 71, 71, 64)	36,928
conv2d_3 (Conv2D)	(None, 69, 69, 64)	36,928
dropout_1 (Dropout)	(None, 69, 69, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 34, 34, 64)	0
dropout_2 (Dropout)	(None, 34, 34, 64)	0
conv2d_4 (Conv2D)	(None, 32, 32, 128)	73,856
conv2d_5 (Conv2D)	(None, 30, 30, 128)	147,584
conv2d_6 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_3 (Dropout)	(None, 14, 14, 128)	0
conv2d_7 (Conv2D)	(None, 12, 12, 128)	147,584
conv2d_8 (Conv2D)	(None, 10, 10, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 256)	0
dropout_4 (Dropout)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 512)	3,277,312
dense_1 (Dense)	(None, 512)	262,656
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 4)	2,052

Total params: 4,447,044 (16.96 MB)

Trainable params: 4,447,044 (16.96 MB)

Non-trainable params: 0 (0.00 B)



```
model = Sequential()
model.add(keras.Input(shape=(150,150,3)))
model.add(Conv2D(32,(3,3),activation = 'relu'))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(4,activation='softmax'))
```

```
model.summary()
```

The model is initialized as a Sequential model, which is a linear stack of layers. This means that each layer in the model has exactly one input tensor and one output tensor.

The first layer is an Input layer with shape (150,150,3), which means the model expects input images to be 150x150 pixels with 3 color channels (RGB).

The next layers are Conv2D layers, which apply convolution operations to the input data. These layers have 32, 64, 64, 128, 128, 128, 128, and 256 filters respectively, and use 3x3 kernels. The activation function for these layers is 'relu', which stands for Rectified Linear Unit, a common activation function in deep learning models.

MaxPooling2D layers are used after some of the Conv2D layers. These layers downsample the input along its spatial dimensions (height and width) by taking the maximum value over an input window for each channel of the input. The window size for these layers is 2x2.

Dropout layers are used to prevent overfitting. These layers randomly set a fraction of the input units to 0 at each update during training time. The fraction rate is 0.3 in this model.

After the last MaxPooling2D layer, a Flatten layer is used to flatten the input. This means that it is converted from a 2D matrix (or higher-dimensional tensor) into a 1D vector.

Finally, three Dense layers are used. The first two have 512 units and use the 'relu' activation function. The last Dense layer has 4 units and uses the 'softmax' activation function, which makes it suitable for multi-class classification. The output of this layer will be a vector of 4 probabilities summing to 1, each indicating the predicted probability of the corresponding class.

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

The loss argument is set to 'categorical_crossentropy', which is a common loss function for multi-class classification problems. It measures the dissimilarity between the predicted probability distribution and the true distribution. In other words, it tries to minimize the difference between the model's predictions and the actual labels.

The optimizer argument is set to 'Adam'. Adam is a popular optimization algorithm in deep learning. It adapts the learning rate for each weight in the model individually and computes adaptive learning rates for different parameters. This can lead to better results in less time compared to other optimization algorithms.

The metrics argument is set to ['accuracy']. This means that the model will evaluate and report its performance in terms of accuracy, which is the proportion of correct predictions out of total predictions. This is a common metric for classification problems.

Model Training

```
history = model.fit(X_train,y_train,epochs=20,validation_split=0.1)
```

```
Epoch 1/20
 1/83 ━━━━━━━━━━━━━━━━━━━━ 43:13 32s/step - accuracy: 0.2188 - loss: 7.6404
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1712216890.336956     107 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime of th
W0000 00:00:1712216890.363181     107 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step - accuracy: 0.2808 - loss: 3.6988
W0000 00:00:1712216916.309888     107 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
W0000 00:00:1712216917.212971     107 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
83/83 ━━━━━━━━━━━━━━━━━━━━ 61s 361ms/step - accuracy: 0.2815 - loss: 3.6774 - val_accuracy: 0.5034 - val_loss: 1.1899
Epoch 2/20
W0000 00:00:1712216920.036396     108 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
83/83 ━━━━━━━━━━━━━━━━━━━━ 31s 67ms/step - accuracy: 0.5104 - loss: 1.1146 - val_accuracy: 0.5986 - val_loss: 0.8737
Epoch 3/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.6245 - loss: 0.8667 - val_accuracy: 0.6327 - val_loss: 0.8007
Epoch 4/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.6631 - loss: 0.7529 - val_accuracy: 0.7347 - val_loss: 0.7072
Epoch 5/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 68ms/step - accuracy: 0.7379 - loss: 0.6413 - val_accuracy: 0.7381 - val_loss: 0.6146
Epoch 6/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 68ms/step - accuracy: 0.7626 - loss: 0.5871 - val_accuracy: 0.8163 - val_loss: 0.5083
Epoch 7/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 69ms/step - accuracy: 0.8031 - loss: 0.4687 - val_accuracy: 0.7925 - val_loss: 0.5298
Epoch 8/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 68ms/step - accuracy: 0.8209 - loss: 0.4591 - val_accuracy: 0.8333 - val_loss: 0.4102
Epoch 9/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 69ms/step - accuracy: 0.8304 - loss: 0.4342 - val_accuracy: 0.8537 - val_loss: 0.3914
Epoch 10/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.8860 - loss: 0.3046 - val_accuracy: 0.8435 - val_loss: 0.3939
Epoch 11/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.9023 - loss: 0.2499 - val_accuracy: 0.8878 - val_loss: 0.3332
Epoch 12/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 71ms/step - accuracy: 0.8924 - loss: 0.2849 - val_accuracy: 0.8810 - val_loss: 0.3073
Epoch 13/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 71ms/step - accuracy: 0.9344 - loss: 0.1917 - val_accuracy: 0.9150 - val_loss: 0.2781
Epoch 14/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 71ms/step - accuracy: 0.9341 - loss: 0.1695 - val_accuracy: 0.8946 - val_loss: 0.3751
...
Epoch 19/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.9606 - loss: 0.1061 - val_accuracy: 0.9184 - val_loss: 0.2814
Epoch 20/20
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.9578 - loss: 0.1177 - val_accuracy: 0.8946 - val_loss: 0.2726
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

The fit method takes several arguments. The first two, X_train and y_train, are the feature matrix and the target array for the training data, respectively.

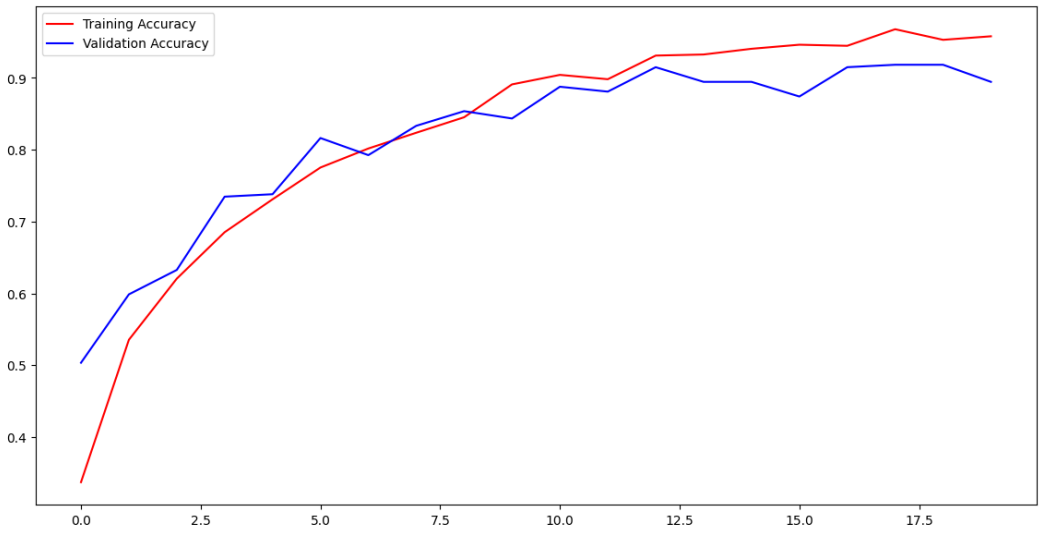
The epochs argument is set to 20, which means that the entire dataset will be passed through the model 20 times. In each epoch, the model's weights are updated to minimize the loss function.

The validation_split argument is set to 0.1, which means that 10% of the training data will be set aside and used as validation data. After each epoch, the model's performance is evaluated on this validation data, which provides a check on overfitting. The validation data is not used to train the model's weights.

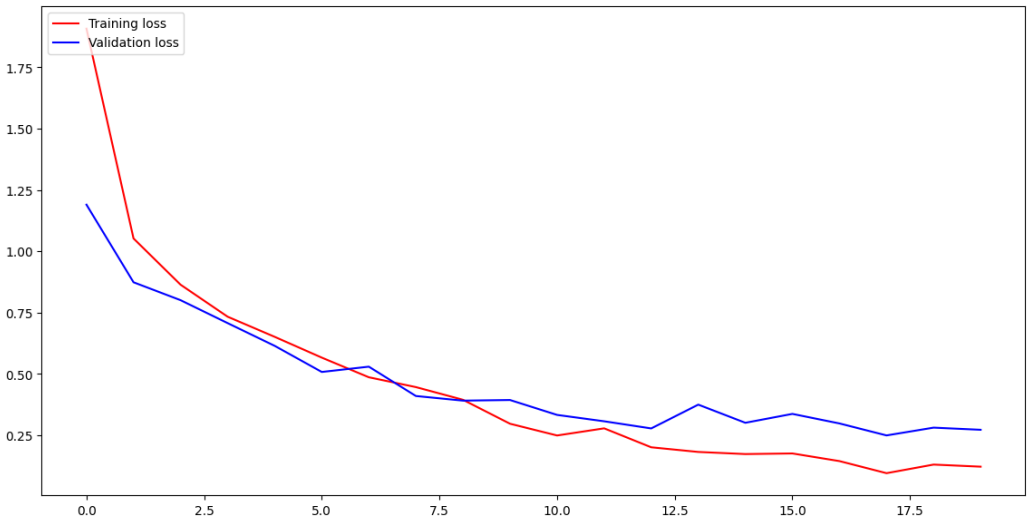
The fit method returns a History object, which is being assigned to the variable history. This object has a history attribute that is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values

Model Evaluation

- Training accuracy vs validation accuracy



- Training loss vs validation loss

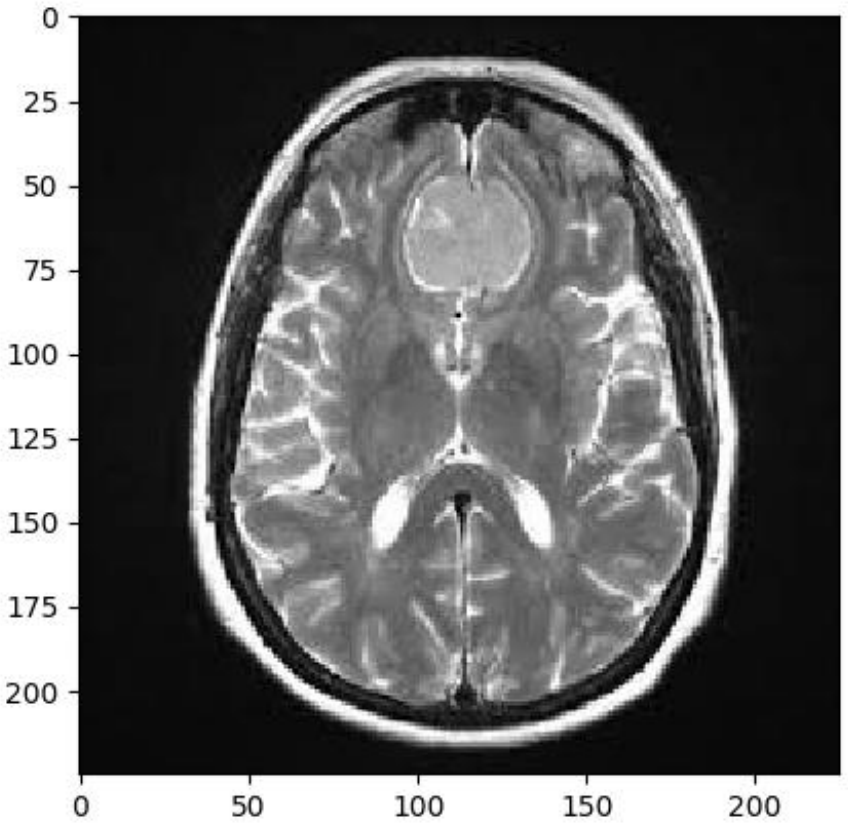


```
from tensorflow.keras.preprocessing import image
img = image.load_img('/kaggle/input/brain-tumor-classification-mri/Testing/meningioma_tumor/image(97).jpg')
plt.imshow(img,interpolation='nearest')
plt.show()
```

The load_img function from the image module is then used to load an image from a file. The file path is provided as a string. The loaded image is assigned to the variable img.

Next, the imshow function from the matplotlib.pyplot module (aliased as plt) is used to display the image. The interpolation parameter is set to 'nearest', which means that the value of a pixel in the displayed image will be the same as the nearest pixel in the input image. This is one of several methods that can be used to determine the display value of pixels when the image is shown at a size other than its original resolution.

Finally, the show function from the matplotlib.pyplot module is called to display the figure. This will open a window with the image displayed.



```
a=model.predict(img_array)
indices = a.argmax()
if indices == 0:
    print("Gliomar Tumor. chemo needed soon")
elif indices == 1:
    print("Meningioma Tumor")
elif indices == 3:
    print("Pituitary Tumor")
else:
    print("No Tumor. Cancer free :)")
```

The model.predict function is used to generate predictions for the input image. The output of this function is a probability distribution over the four classes, which is stored in the variable a.

The argmax function is then used to find the index of the maximum value in a, which corresponds to the most likely class according to the model. This index is stored in the variable indices.

A series of if and elif statements are then used to print a message based on the value of indices. Each value corresponds to a different class: 0 for "Gliomar Tumor", 1 for "Meningioma Tumor", and 3 for "Pituitary Tumor".

The line of code in question is the else clause of this series of statements. If indices is not 0, 1, or 3, then this line of code will be executed, and "No Tumor. Cancer free :)" will be printed. This implies that the model has classified the image as not containing a tumor.

Project Implementation

Integration into Hospital Workflow:

1. **Pre-existing Radiology Infrastructure:** Ideally, the CNN system would integrate seamlessly with existing PACS (Picture Archiving and Communication System) used by radiologists to store and access patient scans. This would minimize workflow disruption.
2. **User Interface Design:** The radiologist interface should be intuitive and efficient. It should allow for:
    - Uploading de-identified patient MRI scans.
    - Visualizing uploaded scans.
    - Interacting with the CNN model (e.g., initiating analysis, viewing results).
    - Integrating CNN results with other patient data for a comprehensive view.

Result Presentation and Interpretation:

1. **Clear and Concise Output:** The CNN should provide clear probability scores for tumor presence and potentially tumor type (if applicable).
2. **Emphasis on Diagnostic Aid:** The system should emphasize that the CNN's role is to assist with diagnosis, not replace it. The final call rests with the radiologist who can consider the CNN's output alongside their expertise and other patient information.

Prioritization and Treatment Planning:

1. **Prioritization based on Risk:** The CNN's output, particularly for highly aggressive tumors, can help prioritize patients who may need faster evaluation and treatment.
2. **Treatment Planning Support:** The system could potentially aid treatment planning by providing tumor location and size information extracted by the CNN (depending on model capabilities).

Scalability and Future Advancements:

1. **Modular Design:** The system should be modular to allow for future integration of additional AI models for tumor segmentation, treatment response prediction, etc.
2. **Continuous Learning:** The CNN model should be continuously updated with new data to improve accuracy and adapt to evolving tumor characteristics.

Challenges and Considerations:

- **Data Security and Privacy:** Robust security measures are essential to protect sensitive patient data throughout the entire process.

- **Regulatory Compliance:** Obtaining regulatory approval from relevant healthcare authorities is crucial before hospital deployment.
- **Radiologist Training and Education:** Training radiologists on interpreting and leveraging CNN outputs effectively is necessary for successful integration.

By addressing these challenges and thoughtfully integrating the CNN system into the existing hospital workflow, this project has the potential to significantly improve brain tumor diagnosis efficiency and potentially patient outcomes. However, it's important to remember that CNNs are still under development in the medical field, and responsible implementation with a focus on ethical considerations and human expertise is paramount.

Conclusion

This project enabled me to gain a deeper knowledge on Machine learning and mainly on convolutional neural networks and to predict diseases through training a model on brain images. Gained the fundamental understanding on building a neural network and tuning its parameters and layers to attain the best results.

- The model can be further tuned and trained on a larger dataset to attain near perfect results.
- Scope of ML in the medical field is very vast and helpful for people to get better healthcare.

Sources

- https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- https://www.pinecone.io/learn/softmax-activation/