भारतीय प्रौद्योगिकी संस्थान
काशी हिन्दू विश्वविद्यालय

IIT

INDIAN INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY

Department of Mathematical Sciences

Exploratory Project

CNN-based Classification of Brain Tumors from MRI Data

Presented by Debangi Ghosh
Supervised by Dr. Santwana Mukhopadhyay

Date- 24.11.2023

Acknowledgement

I would like to express my sincere gratitude to all those who contributed to the success of this exploratory project. Special thanks to Dr. Santwana Mukhopadhyay Ma’am for invaluable guidance and unwavering support throughout the journey. I appreciate the invaluable insights gained through this journey, contributing significantly to personal and professional growth. The journey may have been solitary, but the lessons learned and the achievement attained will forever be cherished. Thank you for the unwavering encouragement and inspiration that fueled this endeavor.

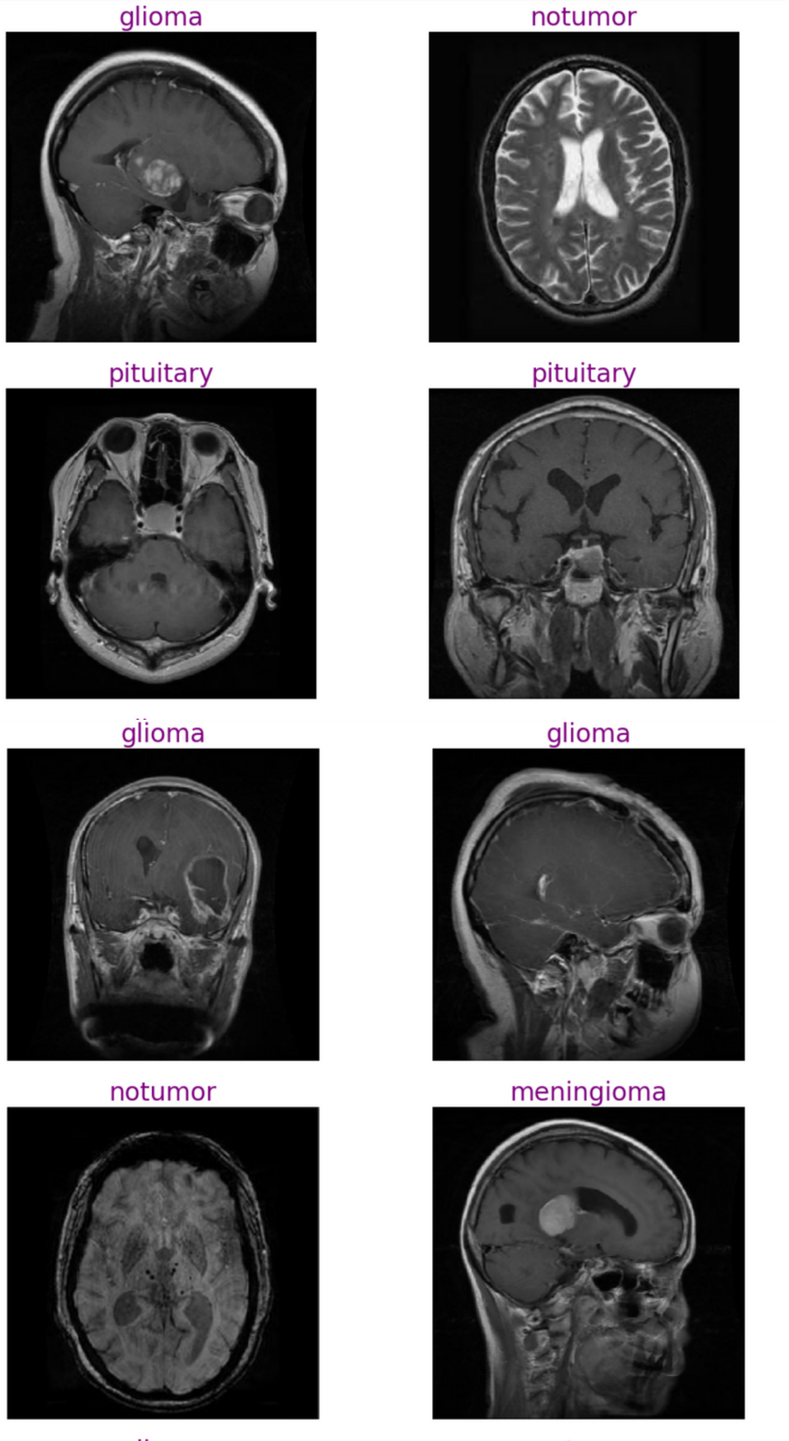
Brain Tumor

What is a brain tumor?

A brain tumor is a collection, or mass, of abnormal cells in the brain. The skull, which encloses your brain, is very rigid. Any growth inside such a restricted space can cause problems. Brain tumors can be cancerous (malignant) or noncancerous (benign). When benign or malignant tumors grow, they can cause the pressure inside your skull to increase. This can cause brain damage, and it can be life-threatening.

The importance of the subject

Early detection and classification of brain tumors is an important research domain in the field of medical imaging and accordingly helps in selecting the most convenient treatment method to save patients life.



The Dataset

This dataset contains 7023 images of human brain MRI images which are classified into 4 classes:

- **Glioma**
- **Meningioma**
- **Pituitary**
- **No tumor**

The train data contains a total of 5712 images and the test data consists of a total of 1311 images which is further divided equally between test setand validation set.

The size of the images in this dataset is different. We can resize the image to the desired size after pre-processing and removing the extra margins. This work will improve the accuracy of the model.

Link to the colab file

Click Here

What does the project encompass?

Data preprocessing

This includes reading of data, splitting of data into train and test set and validation set, and data augmentation

The Model Architecture

This includes model structure, model training and model evaluation followed by model prediction and saving of the CNN based model

Activation Functions

A comparative study of different activation functions such as ReLU, ELU, Tanh, Sigmoid based on loss and accuracy of train and test data.

Data Preprocessing

01

TRAIN DATA EXTRACTION

Paths and Labels Series objectare concatenated in a new DataFrame called Tr_data. This DataFrame contains two columns: Paths, which contains the paths to the images, and Labels, which contains the labels for the images.

02

TEST DATA EXTRACTION

in a similiar manner, Paths and Labels Series object of the test data are concatenated in a new DataFrame called Ts_data consisting of two columns

03

TEST DATA SPLITTING

The test data is split into two sets, test set and validation data set. The validation data is used to adjust the hyperparameters as required and test set is used for testing the model.

04

DATA AUGMENTATION

The code uses Keras' ImageDataGenerator and flow_from_dataframe to create data generators for training, validation, and testing in a deep learning model, incorporating data augmentation for training.

The Model

- **Convolutional Layers:**

1. Sequential convolutional layers with increasing filter sizes (128, 256) and elu activation.
2. Each convolution is followed by max-pooling (2x2) for downsampling.

- **Flatten:**

3. Flattens the output of the convolutional layers to prepare for the fully connected layers.

- **Dense Layers:**

4. Dense (fully connected) layers with decreasing neuron counts (256, 128, 64, 32) and elu activation.
5. Final dense layer with neurons equal to the number of classes, using softmax activation for classification.

- **Input Shape:**

6. Input shape specified by img_shape for the first convolutional layer.

- **Padding:**

7. 'Same' padding used in convolutional layers to maintain spatial dimensions.

- **Model Architecture:**

8. Designed for image classification with a convolutional neural network (CNN) structure.

```
CNN = Sequential([
    Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'elu',
           input_shape = img_shape),
    Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    MaxPooling2D((2,2)),

    Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    MaxPooling2D((2,2)),

    Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    Conv2D(filters = 256, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    MaxPooling2D((2,2)),

    Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    Conv2D(filters = 128, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    MaxPooling2D((2,2)),

    Conv2D(filters = 64, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    Conv2D(filters = 64, kernel_size = (3,3), padding = 'same', activation = 'elu'),
    MaxPooling2D((2,2)),

    Flatten(),

    Dense(256, activation = 'elu'),
    Dense(128, activation = 'elu'),
    Dense(64, activation = 'elu'),
    Dense(32, activation = 'elu'),
    Dense(counter_classes, activation = 'softmax')
])
```

Filter and Feature Map

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Shifted 9 at different position

*

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

Loopy pattern filter

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Loopy pattern filter

*

1	1	1
1	-1	1
1	1	1

→

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55

→

1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

Max pooling →

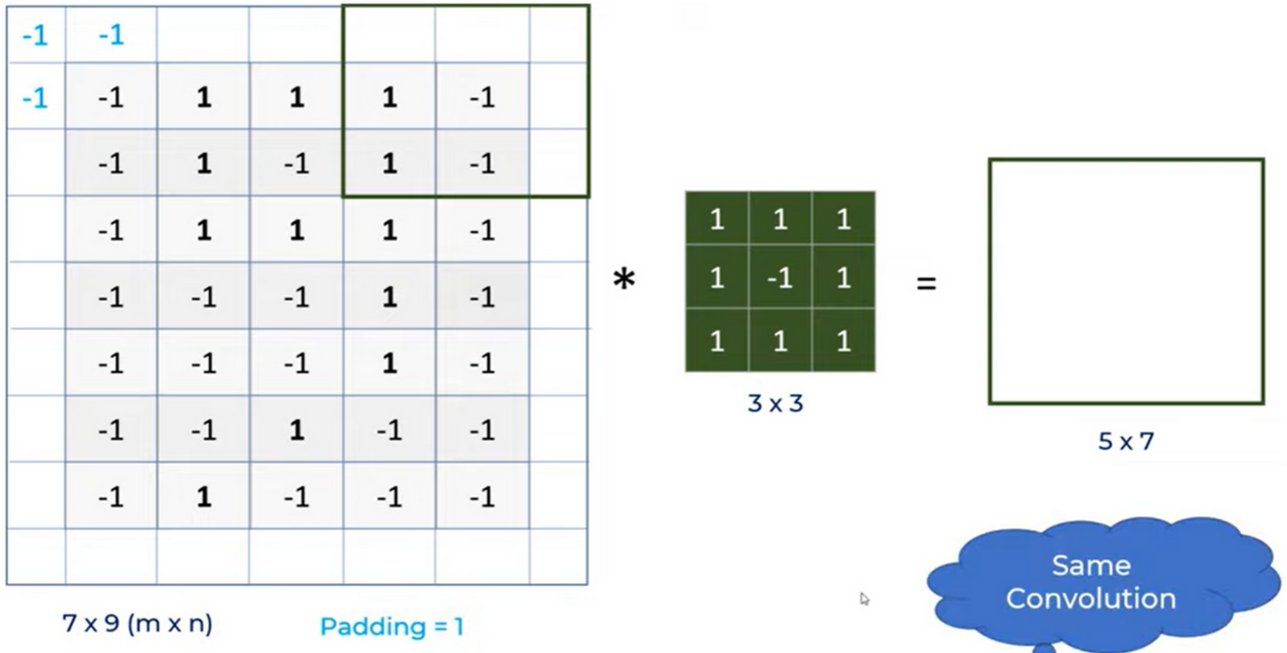
1	0.33
0.33	0.33
0.33	0
0	0

Padding

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

Disadvantage: corner pixels don't contribute as much in feature detection



7 x 9 (m x n)

Padding = 1

3 x 3

5 x 7

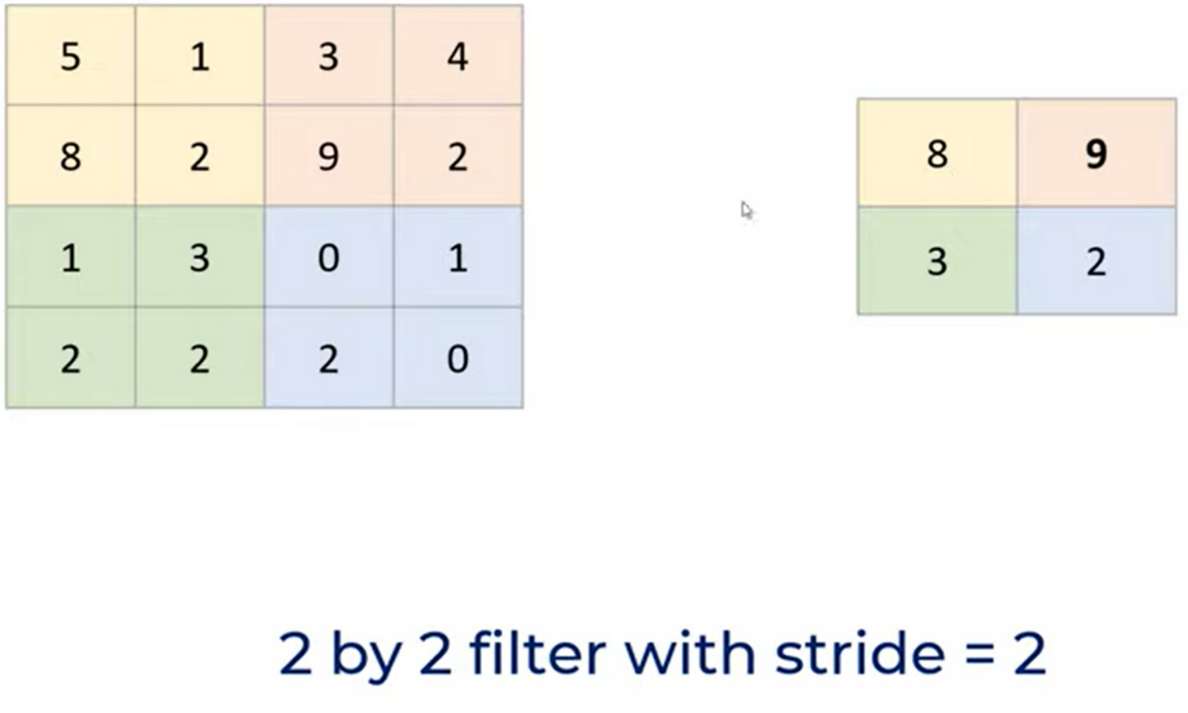
Same Convolution

$(m - f + 1) \times (n - f + 1) = (7-3+1) \times (9-3+1) = 5 \times 7$

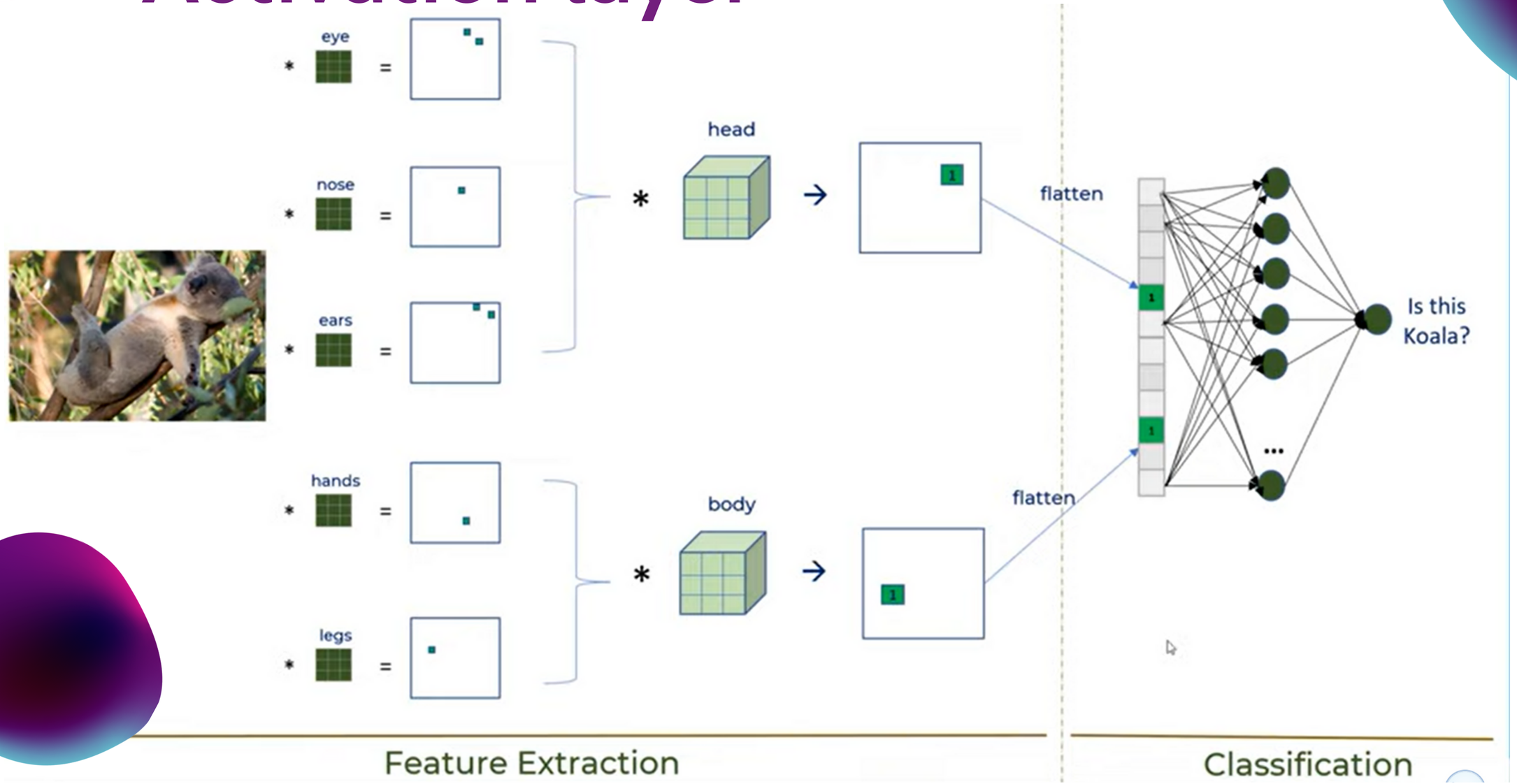
Max Pooling

Advantages of Max Pooling

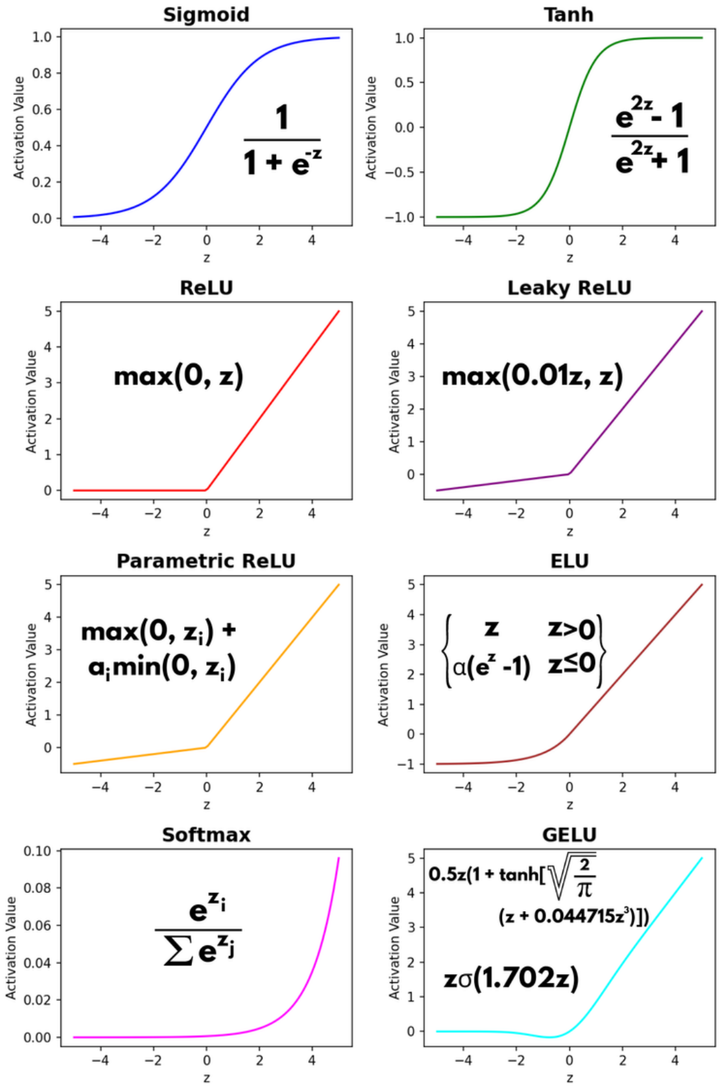
1. Reduces dimension and computation
2. Reduces overfitting as there are less parameters
3. Model is tolerant towards variations and distortions.



Flatten and Dense Activation layer



Activation Functions



The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

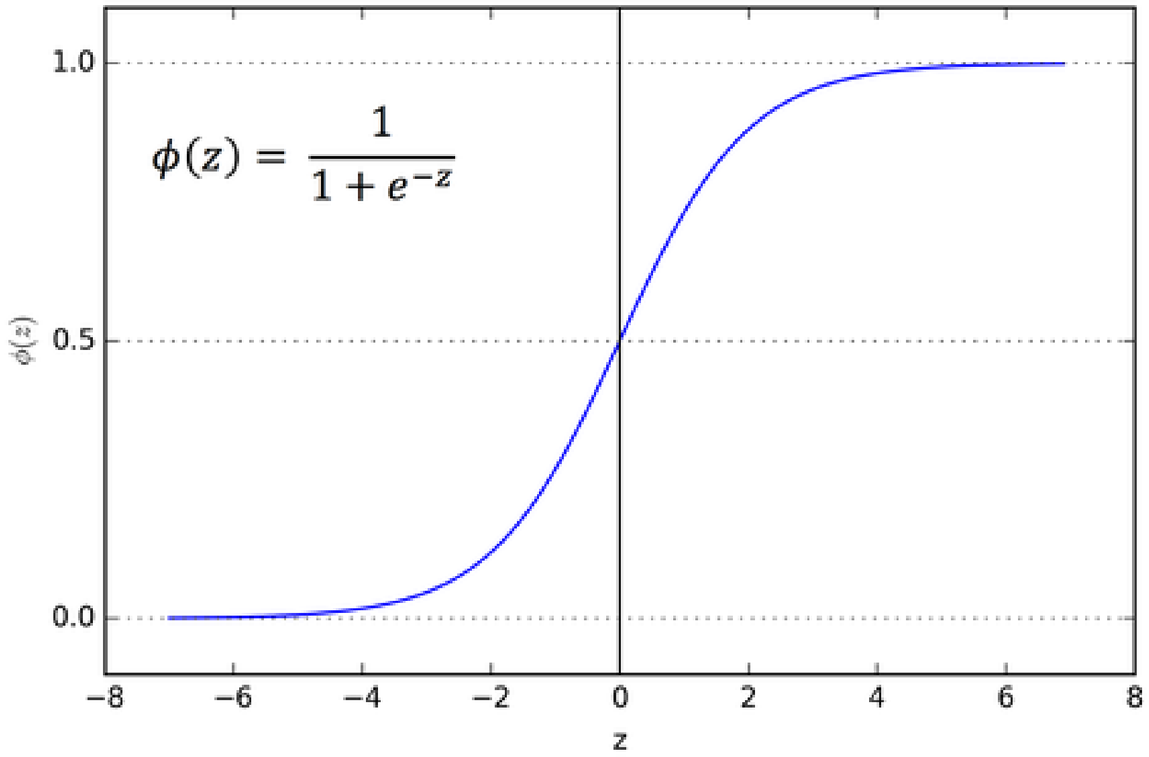
In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Sigmoid activation function

Sigmoid / Logistic

$$f(x) = \frac{1}{1 + e^{-x}}$$

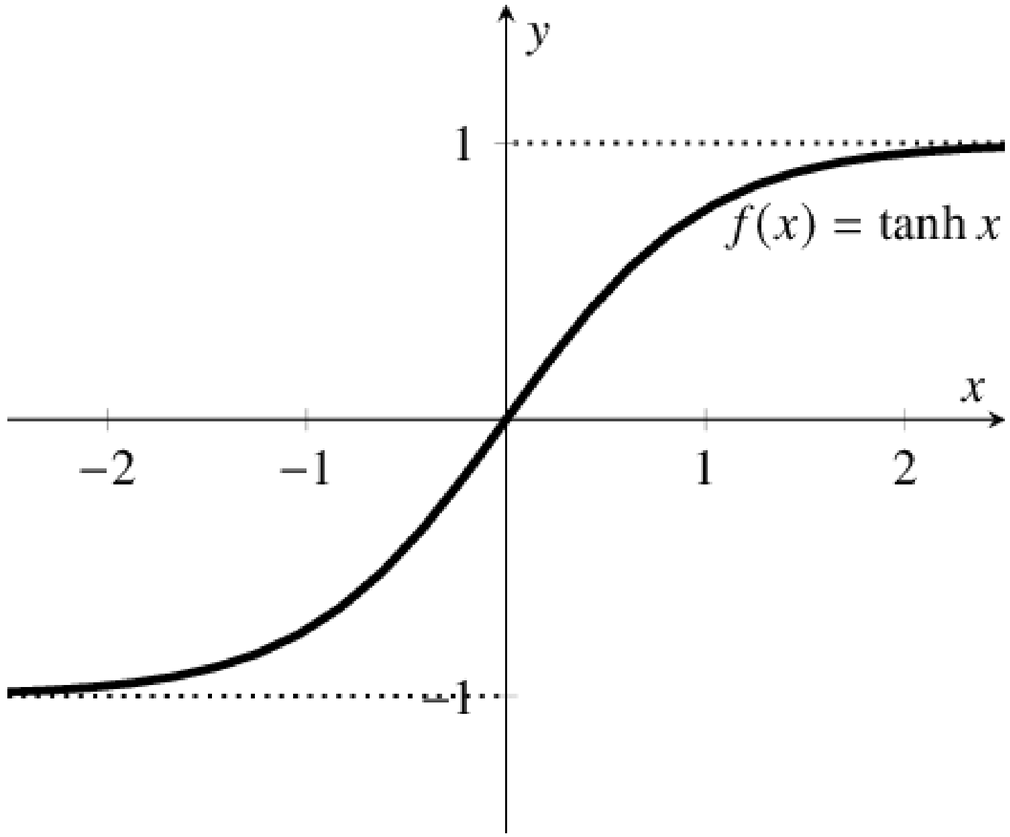


The sigmoid activation function is also called the logistic function.
It is the same function used in the logistic regression classification algorithm.
The function takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

Tanh activation function

$$Tanh$$

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$

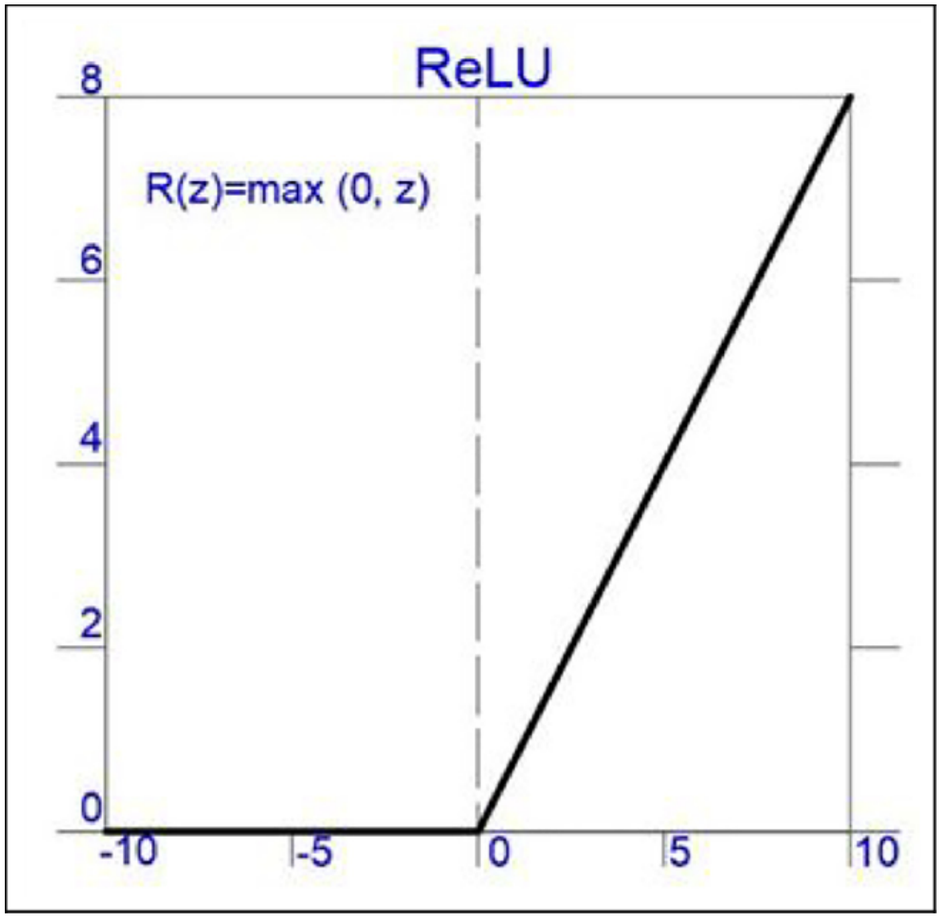


The hyperbolic tangent activation function is also referred to simply as the Tanh (also “tanh” and “TanH“) function.

It is very similar to the sigmoid activation function and even has the same S-shape.

The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

ReLU activation function

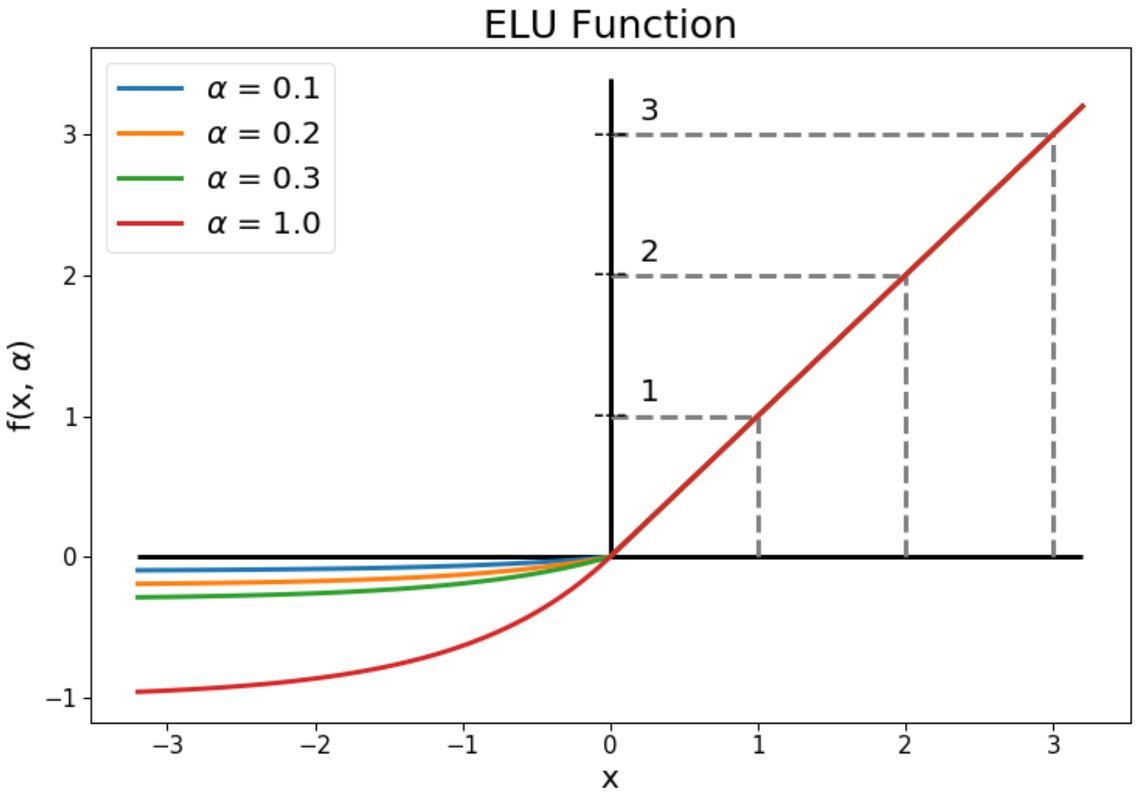


$$ReLU$$

$$f(x) = max\,(0, x)$$

ReLU stands for rectified linear activation unit and is considered one of the few milestones in the deep learning revolution. It is simple yet really better than its predecessor activation functions such as sigmoid or tanh. ReLU function is simple and it consists of no heavy computation as there is no complicated math. The model can, therefore, take less time to train or run.Leaky Relu is a variant of ReLU. Instead of being 0 when z<0, a leaky ReLU allows a small, non-zero, constant gradient α (normally, α=0.01).

ELU activation function



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

The Exponential Linear Unit or ELU is a function that tends to converge faster and produce more accurate results. Unlike other activation functions, ELU has an extra alpha constant which should be a positive number. ELU is very similar to ReLU except for negative inputs. They are both in the identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to -α whereas ReLU sharply smoothes.

Choosing of Activation Functions

There are no hard and fast rules when it comes to choosing activation functions. Experimenting with different activation functions and evaluating their performance on specific problem is often the best approach to find the most suitable one for your neural network. Since, it is a multiclass Classification ELU, ReLU, leaky ReLU are good options.

We can see from the model that:
1. Sigmoid and tanh results in high loss and low accuracy
2. ReLU provides with relatively high accuracy but unpredictable loss due to overfitting.

So, we chose ELU activation function fot this particular dataset.

```
historysigmoid = CNNsigmoid.fit(x = Train, epochs = epochs, verbose = 1, validation_data = Valid, shuffle = False)

Epoch 1/10
286/286 [==============================] - 133s 455ms/step - loss: 1.8340 - accuracy: 0.2556 - val_loss: 1.3828 - val_accuracy: 0.3130
Epoch 2/10
286/286 [==============================] - 129s 450ms/step - loss: 1.3864 - accuracy: 0.2708 - val_loss: 1.3808 - val_accuracy: 0.3130
Epoch 3/10
286/286 [==============================] - 129s 45
Epoch 4/10
286/286 [==============================] - 129s 45
Epoch 5/10
286/286 [==============================] - 133s 46
Epoch 6/10
286/286 [==============================] - 129s 45
Epoch 7/
286/286
Epoch 8/
286/286
Epoch 9/
286/286
Epoch 10
286/286
```

```
historytanh = CNNtanh.fit(x = Train, epochs = epochs, verbose = 1, validation_data = Valid, shuffle = False)

Epoch 1/15
286/286 [==============================] - 138s 477ms/step - loss: 2.5938 - accuracy: 0.2612 - val_loss: 1.3842 - val_accuracy: 0.2336
Epoch 2/15
286/286 [==============================] - 128s 448ms/step - loss: 1.3890 - accuracy: 0.2722 - val_loss: 1.3987 - val_accuracy: 0.2076
Epoch 3/15
```

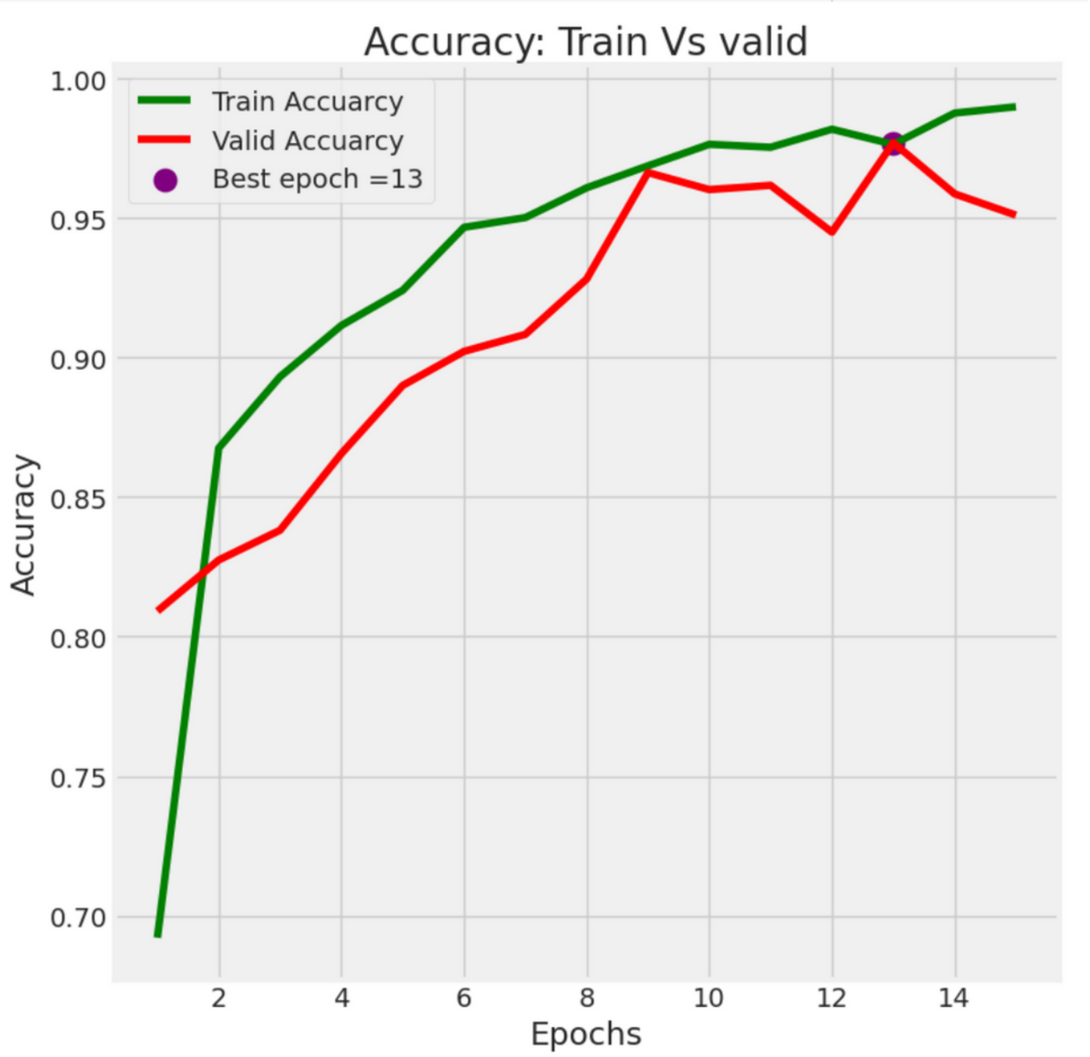
```
historyrelu = CNNrelu.fit(x = Train, epochs = epochs, verbose = 1, validation_data = Valid, shuffle = False)

Epoch 1/15
286/286 [==============================] - 203s 463ms/step - loss: 13.6709 - accuracy: 0.7020 - val_loss: 0.5747 - val_accuracy: 0.7756
Epoch 2/15
286/286 [==============================] - 125s 436ms/step - loss: 0.3051 - accuracy: 0.8964 - val_loss: 0.2702 - val_accuracy: 0.8962
Epoch 3/15
286/286 [==============================] - 126s 439ms/step - loss: 0.1645 - accuracy: 0.9447 - val_loss: 0.2137 - val_accuracy: 0.9298
Epoch 4/15
286/286 [==============================] - 125s 437ms/step - loss: 0.0869 - accuracy: 0.9723 - val_loss: 0.2745 - val_accuracy: 0.9328
Epoch 5/15
286/286 [==============================] - 125s 438ms/step - loss: 0.0602 - accuracy: 0.9806 - val_loss: 0.1729 - val_accuracy: 0.9542
Epoch 6/15
286/286 [==============================] - 125s 437ms/step - loss: 0.0291 - accuracy: 0.9928 - val_loss: 0.2670 - val_accuracy: 0.9466
Epoch 7/15
286/286 [==============================] - 125s 437ms/step - loss: 0.0295 - accuracy: 0.9895 - val_loss: 0.3646 - val_accuracy: 0.9359
Epoch 8/15
286/286 [==============================] - 126s 441ms/step - loss: 0.0197 - accuracy: 0.9942 - val_loss: 0.2782 - val_accuracy: 0.9527
Epoch 9/15
286/286 [==============================] - 126s 440ms/step - loss: 0.0050 - accuracy: 0.9991 - val_loss: 0.2176 - val_accuracy: 0.9557
Epoch 10/15
286/286 [==============================] - 125s 436ms/step - loss: 0.0110 - accuracy: 0.9975 - val_loss: 0.2850 - val_accuracy: 0.9511
Epoch 11/15
286/286 [==============================] - 126s 439ms/step - loss: 0.0138 - accuracy: 0.9960 - val_loss: 0.2577 - val_accuracy: 0.9496
Epoch 12/15
286/286 [==============================] - 125s 438ms/step - loss: 0.0028 - accuracy: 0.9995 - val_loss: 0.3431 - val_accuracy: 0.9542
Epoch 13/15
286/286 [==============================] - 125s 438ms/step - loss: 4.8593e-04 - accuracy: 1.0000 - val_loss: 0.3343 - val_accuracy: 0.9573
Epoch 14/15
286/286 [==============================] - 125s 436ms/step - loss: 9.7907e-05 - accuracy: 1.0000 - val_loss: 0.3435 - val_accuracy: 0.9603
Epoch 15/15
286/286 [==============================] - 126s 439ms/step - loss: 6.6139e-05 - accuracy: 1.0000 - val_loss: 0.3676 - val_accuracy: 0.9573
```

Model Compilation and Training

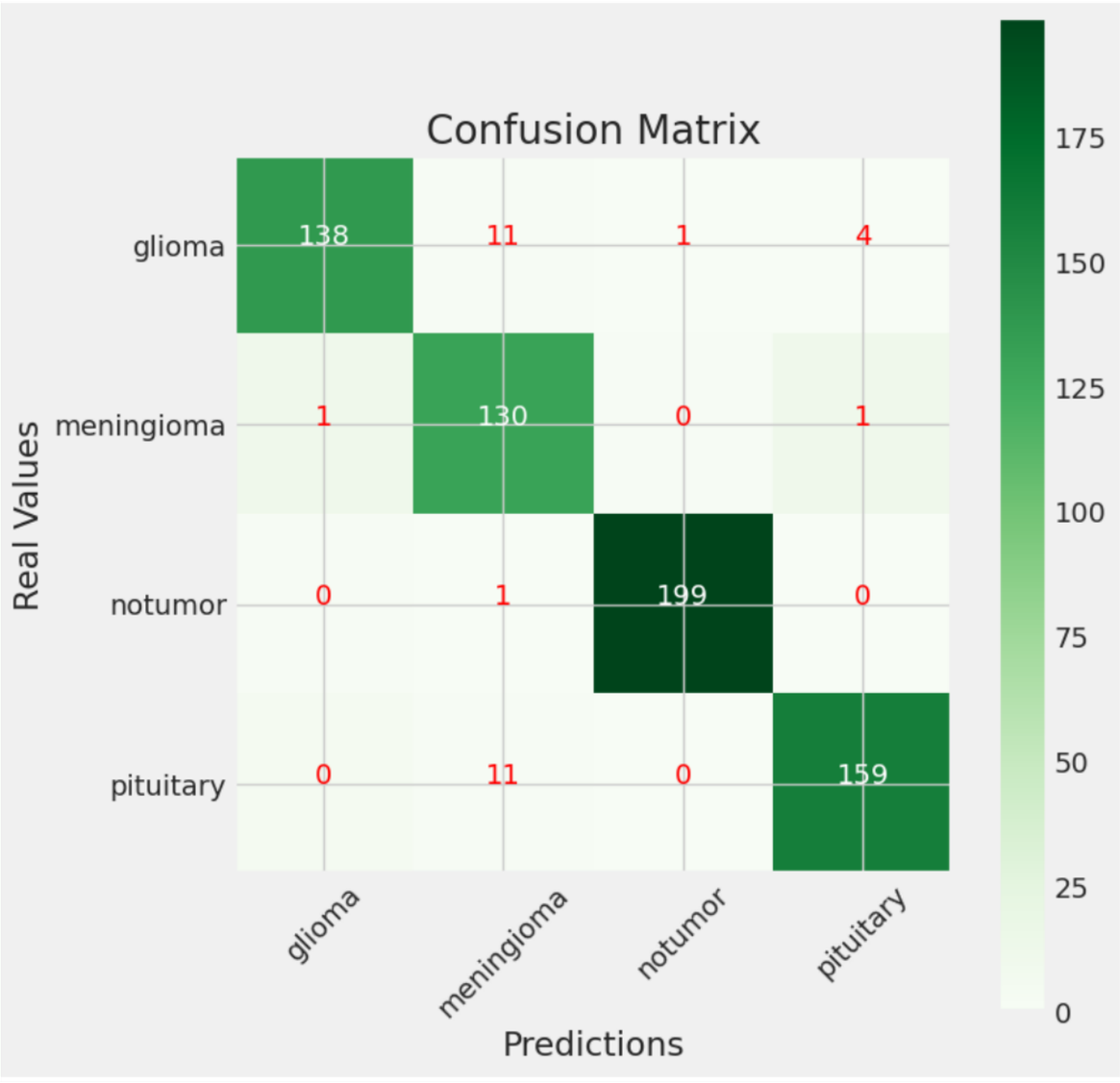
```
epochs = 15
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history= CNN.fit(x = Train, epochs = epochs, verbose = 1, validation_data = Valid, shuffle = False)
```

```
Epoch 1/15
286/286 [==============================] - 234s 808ms/step - loss: 0.8139 - accuracy: 0.6921 - val_loss: 0.5117 - val_accuracy: 0.8092
Epoch 2/15
286/286 [==============================] - 231s 807ms/step - loss: 0.3636 - accuracy: 0.8676 - val_loss: 0.5755 - val_accuracy: 0.8275
Epoch 3/15
286/286 [==============================] - 230s 803ms/step - loss: 0.2980 - accuracy: 0.8932 - val_loss: 0.4667 - val_accuracy: 0.8382
Epoch 4/15
286/286 [==============================] - 230s 805ms/step - loss: 0.2454 - accuracy: 0.9116 - val_loss: 0.3451 - val_accuracy: 0.8656
Epoch 5/15
286/286 [==============================] - 230s 804ms/step - loss: 0.2064 - accuracy: 0.9242 - val_loss: 0.3609 - val_accuracy: 0.8901
Epoch 6/15
286/286 [==============================] - 229s 802ms/step - loss: 0.1526 - accuracy: 0.9468 - val_loss: 0.2322 - val_accuracy: 0.9023
Epoch 7/15
286/286 [==============================] - 231s 806ms/step - loss: 0.1381 - accuracy: 0.9503 - val_loss: 0.2559 - val_accuracy: 0.9084
Epoch 8/15
286/286 [==============================] - 229s 801ms/step - loss: 0.1120 - accuracy: 0.9610 - val_loss: 0.2012 - val_accuracy: 0.9282
Epoch 9/15
286/286 [==============================] - 229s 802ms/step - loss: 0.0858 - accuracy: 0.9688 - val_loss: 0.0996 - val_accuracy: 0.9664
Epoch 10/15
286/286 [==============================] - 229s 800ms/step - loss: 0.0710 - accuracy: 0.9765 - val_loss: 0.1425 - val_accuracy: 0.9603
Epoch 11/15
286/286 [==============================] - 228s 797ms/step - loss: 0.0707 - accuracy: 0.9755 - val_loss: 0.1039 - val_accuracy: 0.9618
Epoch 12/15
286/286 [==============================] - 229s 800ms/step - loss: 0.0545 - accuracy: 0.9820 - val_loss: 0.2000 - val_accuracy: 0.9450
Epoch 13/15
286/286 [==============================] - 229s 800ms/step - loss: 0.0692 - accuracy: 0.9765 - val_loss: 0.0755 - val_accuracy: 0.9771
Epoch 14/15
286/286 [==============================] - 228s 798ms/step - loss: 0.0388 - accuracy: 0.9877 - val_loss: 0.1302 - val_accuracy: 0.9588
Epoch 15/15
286/286 [==============================] - 229s 799ms/step - loss: 0.0286 - accuracy: 0.9900 - val_loss: 0.1476 - val_accuracy: 0.9511
```

Model Compilation and Training



Confusion Matrix



A confusion matrix is a performance evaluation tool in machine learning, representing the accuracy of a classification model. It displays the number of true positives, true negatives, false positives, and false negatives.

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the total number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

The model is then saved and the predictions can be made by providing the path of an image from the test dataset

Link to the colab file

References

1. Brain Tumor MRI dataset- Kaggle
2. CNN in machine learning- GeeksforGeeks
3. Convolutional layers, filters, padding, Maxpooling by Codebasics
4. Activation Functions- GeeksforGeeks
5. Confusion Matrix- StackOverflow

Thank You

–Debangi Ghosh