

# **Machine Learning Project Report**

## **Brain Stroke Prediction**

### **(UML501)**

### **Fifth-Semester**

**Submitted by:**

<b>Jatin Thakur</b>	<b>102116096</b>
<b>Jasmeet Kaur</b>	<b>102116124</b>

**BE Third Year, CSE**

**Submitted To:**

**Dr. Anjula Mehto**



**Computer Science and Engineering Department**  
**Thapar Institute of Engineering and Technology, Patiala**

**November 2023**

## **TABLE OF CONTENTS:**

1. Introduction
2. Problem Statement
3. Dataset
4. Methodology
5. Results
6. Conclusion
7. References
8. Github repository link

## **INTRODUCTION:**

The Brain Stroke CT Image Dataset from Kaggle provides normal and stroke brain Computer Tomography (CT) scans. The dataset presents very low activity even though it was uploaded more than 2 years ago. It may be probably due to its quite low usability. The challenge is to get some interesting results, i.e., to try to perform brain stroke detection, even from this low-quality CT scan dataset.

The followed approach is based on the usage of a 3D Convolutional Neural Network (CNN) in place of a standard 2D one. 2D CNNs are commonly used to process both grayscale (1 channel) and RGB images (3 channels), while a 3D CNN represents the 3D equivalent since it takes as input a 3D volume or a sequence of 2D frames, e.g. slices in a CT scan.

## **PROBLEM STATEMENT:**

Brain strokes are critical medical emergencies that require prompt diagnosis. Early detection of strokes is crucial for effective medical management and minimizing potential long-term damage. Medical imaging, such as Computed Tomography (CT) scans, plays a pivotal role in the diagnosis and assessment of stroke cases.

Brain strokes are a leading cause of mortality and disability worldwide, emphasizing the critical need for early and accurate prediction methods. This project addresses the challenge of timely detection by leveraging Convolutional Neural Networks (CNNs) for brain stroke prediction.

The conventional methods for stroke prediction often rely on time-consuming and resource-intensive processes, including manual image analysis and complex medical assessments. This project aims to overcome these limitations by developing a CNN-based predictive model that can efficiently analyze neuroimaging data, such as MRIs or CT scans. By automating the analysis through deep learning techniques, we strive to enhance the speed and precision of stroke prediction.

### **Key Components:**

- **Dataset:** The Brain Stroke CT Image Dataset comprises CT scans from individuals with normal brain conditions and those affected by strokes. The dataset is split into training and validation sets.
- **Image Preprocessing:** Prior to model training, the CT scan images undergo preprocessing steps such as normalization, resizing, and noise removal. These steps aim to enhance the quality and uniformity of input data, facilitating effective learning by CNN.
- **3D Convolutional Neural Network (CNN):** The solution's core lies in implementing a 3D CNN architecture.

- **Training and Validation:** The model is trained on the labeled training dataset, optimizing its parameters to classify CT scans into normal or stroke categories accurately. The validation set is used to assess the model's generalization performance and identify potential overfitting.
- **Performance Metrics:** The model's performance is evaluated using various metrics such as accuracy, precision, recall, and area under the receiver operating characteristic curve (AUC-ROC).

#### Expected Outcomes:

- **Accurate Prediction:** The CNN model is expected to achieve a high level of accuracy in distinguishing between normal and stroke cases based on CT scan images.
- **Robustness to Noise:** The model should demonstrate robustness to variations and noise within the dataset, considering the potential challenges associated with low-quality CT scans.

## **DATASET:**

The success of any machine learning project, particularly one involving Convolutional Neural Networks (CNNs), depends on the quality and diversity of the dataset used for training and evaluation. In this project, a meticulously curated dataset of neuroimaging scans will be employed to develop and validate the CNN model for brain stroke prediction.

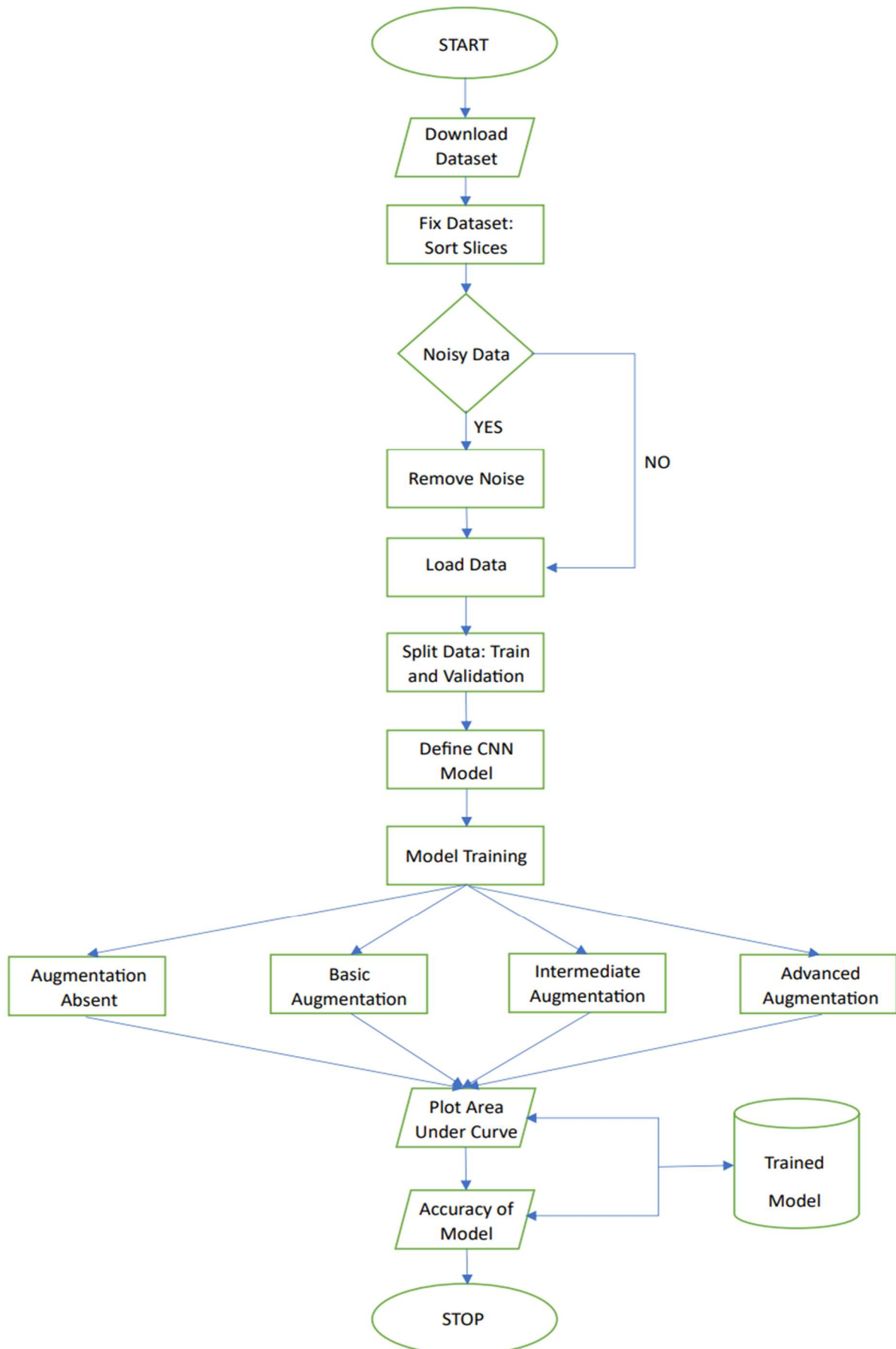
The dataset has been taken from Kaggle. It comprises a diverse collection of computed tomography (CT) scans sourced from reputable medical repositories. Positive cases include images of individuals diagnosed with brain strokes, while negative cases encompass scans of individuals without stroke occurrences. It improves the model's ability to detect subtle patterns of depression.

Link:

[https://www.kaggle.com/datasets/afridirahman/brain-stroke-ct-image-dataset?select=Brain Data Organised](https://www.kaggle.com/datasets/afridirahman/brain-stroke-ct-image-dataset?select=Brain+Data+Organised)

## METHODOLOGY:

- FLOWCHART:



- **MODEL USED:**

The primary object of this project is to build a predictive model that can detect brain stroke by using **Convolutional Neural Networks (CNNs)**.

Generally, Neural networks are composed of 3 types of layers: **a single Input layer, Hidden layers, and a single output layer**. Input layers are made of nodes, which take the input vector's values and feed them into the dense, hidden layers.

The number of hidden layers could be quite large, depending on the nature of the data and the classification problem.

Input values are transmitted forward until they reach the Output layer. The Output layer is composed of nodes associated with the classes the network is predicting.

Convolutional layers are the building blocks of CNNs. These layers are made of many filters, which are defined by their width, height, and depth. Unlike the dense layers of regular neural networks, Convolutional layers are constructed out of neurons in *3-Dimensions*. Because of this characteristic, Convolutional Neural Networks are a sensible solution for image classification.

A Convolutional Neural Network (CNN) is a type of deep learning neural network that is well-suited for image and video analysis. CNNs use a series of convolution and pooling layers to extract features from images and videos, and then use these features to classify or detect objects or scenes.

Reshape Layer:

This layer is used to add an extra dimension (channel) to the input data. The target shape is set to (width, height, depth, 1), indicating a 3D volume with one channel.

Layers used to build CNN:

A complete Convolution Neural Network architecture is also known as convnets. CNN is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers:

- **Input Layers:**

It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images.

- **Convolutional Layers:**

This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. It slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred to as feature maps.

- **Activation Layer:**

By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. It will apply an element-wise activation function to

the output of the convolution layer. Some common activation functions are RELU:  $\max(0, x)$ , Leaky RELU, etc. The volume remains unchanged.

- Pooling layer:

This layer is periodically inserted in the convnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling.

1. Max Pooling Layers:

MaxPooling layers take two input arguments: kernel width and height, and stride(the distance the filter moves at a time. A filter with a *stride* of 1 will move over the input image, 1 pixel at a time).

2. Global Average Pooling Layer:

With Global Averaging, the feature maps' dimensions are reduced drastically by transforming the 3-dimensional feature stack into a 1-dimensional vector. The values of all pixels are averaged and outputted to a single node in a 1-dimensional vector. This results in the dimensions (, K) where K is the total number of feature maps.

- Flattening:

The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

- Fully Connected Layers:

It takes the input from the previous layer and computes the final classification or regression task.

The vector input will pass through two to three — sometimes more — dense layers and pass through a final activation function before being sent to the output layer. The activation function used for prediction does not need to be a rectified linear unit. Selecting the right activation function depends on the type of classification problem, and two common functions are:

1. **Sigmoid** is generally used for binary classification problems, as it is a logistic function.
2. **Softmax** ensures that the sum of values in the output layer sums to 1 and can be used for both binary and multi-class classification problems.

- Output Layer:

The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

#### ModelCheckpoint:

Saves the best weights for the model to a file after each epoch

Filters:

Convolutional Layers are composed of weighted matrices called Filters, sometimes referred to as kernels. Filters slide across the image from left to right, taking as input only a subarea of the image (the receptive field). The depth of a filter is equal to the number of filters in the convolutional layer.

Epoch:

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model. Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

Keras:

A great way to classify images is to build a convolutional neural network (CNN). The Keras library in Python makes it pretty simple to build a CNN.

### **Image Augmentation:**

- Rotation:

This augmentation introduces random rotations to the image, mimicking potential variations in the orientation of brain structures in CT scans. The rotation factor specifies the range of degrees by which the image can be rotated.

- Flip:

Vertical flipping involves flipping the image upside-down. This can be beneficial as brain structures might appear differently in scans with varied orientations.

- Shift:

Horizontal and vertical shifts simulate variations in the positioning of the brain within the image. This is achieved by translating the image by a fraction of its size both horizontally and vertically.

- Zoom:

Zooming in and out provides the model with variations in scale. This is useful for capturing details at different levels of magnification that may be present in the CT scans.

- Shear:

Shearing deforms the image along the x and/or y-axis. In the context of brain stroke prediction, this can account for potential distortions or tilting of the brain structures in the scans.

- Brightness:



Adjusting brightness contributes to the model's ability to handle different lighting conditions in scans. It ensures that the model is not overly sensitive to variations in illumination.

- Contrast:

Contrast adjustment enhances or diminishes the difference between pixel intensities. This is valuable for training the model to identify subtle contrasts in brain structures that may indicate stroke-related abnormalities.

```
# Define the model
model = keras.Sequential()
model.add(keras.Input((width, height, depth)))

# (Optionally) Add augmentation layers
if augmentation:
    if rotation:
        model.add(layers.RandomRotation(factor=(-0.125, 0.125), fill_mode='constant', fill_value=0))
    if flip:
        model.add(layers.RandomFlip(mode='vertical'))
    if shift:
        model.add(layers.RandomTranslation(height_factor=0.2, width_factor=0.2, fill_mode='constant', fill_value=0))
    if zoom:
        model.add(layers.RandomZoom(height_factor=0.15, fill_mode='constant', fill_value=0))
    if shear:
        model.add(keras_cv.layers.RandomShear(x_factor=(0, 0.3), y_factor=(0, 0.3), interpolation="bilinear", fill_mode="nearest", fill_value=0.0))
    if brightness:
        model.add(layers.RandomBrightness(factor=0.1, value_range=[0.0, 1.0]))
    if contrast:
        model.add(layers.RandomContrast(factor=(0, 1.2)))
```

```
# Add a dimension to perform 3D convolutions
model.add(layers.Reshape(target_shape=(width, height, depth, 1)))

model.add(layers.Conv3D(filters=64, kernel_size=3, activation="relu"))
model.add(layers.MaxPool3D(pool_size=2))
model.add(layers.BatchNormalization())

model.add(layers.Conv3D(filters=64, kernel_size=3, activation="relu"))
model.add(layers.MaxPool3D(pool_size=2))
model.add(layers.BatchNormalization())

model.add(layers.Conv3D(filters=128, kernel_size=3, activation="relu"))
model.add(layers.MaxPool3D(pool_size=2))
model.add(layers.BatchNormalization())

model.add(layers.Conv3D(filters=256, kernel_size=3, activation="relu"))
model.add(layers.MaxPool3D(pool_size=2))
model.add(layers.BatchNormalization())

model.add(layers.GlobalAveragePooling3D())
model.add(layers.Dense(units=512, activation="relu"))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(units=1, activation="sigmoid"))

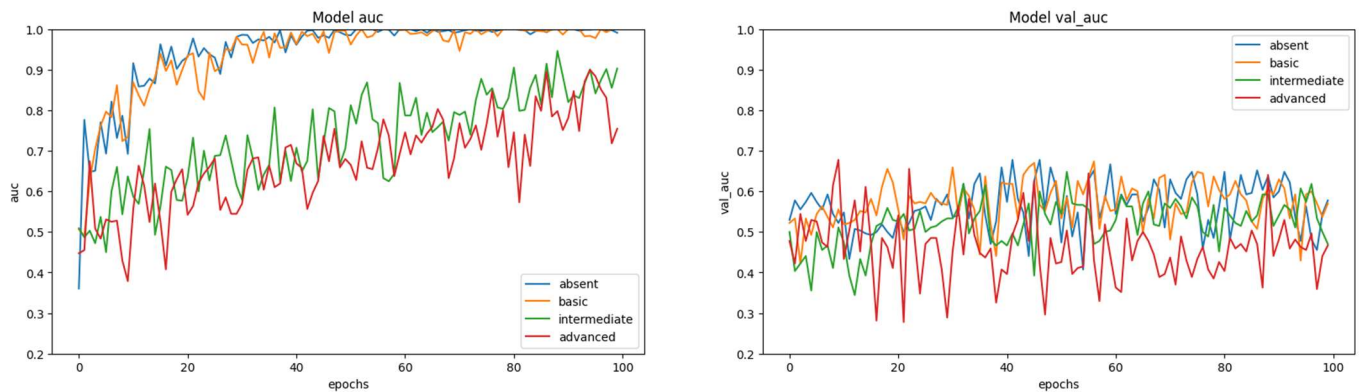
# Define the optimizer
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=decay_steps, decay_rate=decay_rate, staircase=True
)
```

Fig. Model Defination

## RESULTS:

Training and Validation accuracy of the model is shown in the figure below.

- For Absent Augmentation: Training accuracy is 99% and validation accuracy is 62.50%.
- For Basic Augmentation: Training accuracy is 99.75% and validation accuracy is 72.22%.
- For Intermediate Augmentation: Training accuracy is 90.91% and validation accuracy is 57.75%.
- For Advanced Augmentation: Training accuracy is 80.18% and validation accuracy is 47.78%.

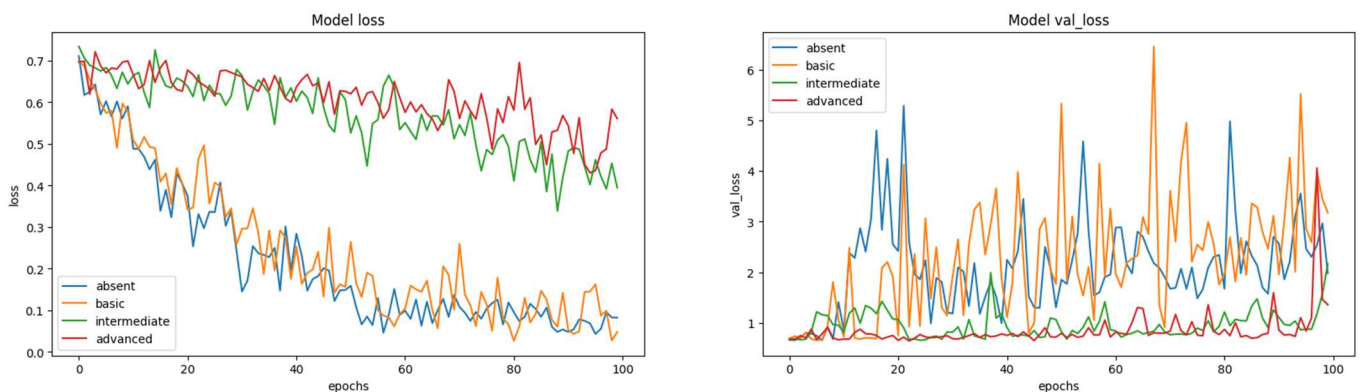


```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
ax = ax.ravel()

for i, metric in enumerate(["auc", "val_auc"]):
    for label, history in performance.items():
        ax[i].plot(history.history[metric])
        ax[i].set_title("Model {}".format(metric))
        ax[i].set_xlabel("epochs")
        ax[i].set_ylabel(metric)
        ax[i].set_ylim([0.2, 1])
        ax[i].legend(performance.keys())
```

Fig. Training and Validation Accuracy

The following graphs depict the model loss and validation loss. For all the different types of augmentations (absent, basic, intermediate, advanced), the training loss is less than the validation loss.



```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
ax = ax.ravel()

for i, metric in enumerate(["loss", "val_loss"]):
    for label, history in performance.items():
        ax[i].plot(history.history[metric])
        ax[i].set_title("Model {}".format(metric))
        ax[i].set_xlabel("epochs")
        ax[i].set_ylabel(metric)
        ax[i].legend(performance.keys())
```

Fig. Training and Validation Losses

## FOR ABSENT AUGMENTATION:

```
# Load best weights.
model.load_weights("ct-scan-brain-stroke-detection-001-0.6037.h5")
prediction = model.predict(np.expand_dims(x_val[0], axis=0))[0]
scores = [1 - prediction[0], prediction[0]]

class_names = ["normal", "stroke"]
for score, name in zip(scores, class_names):
    print(
        "This model is %.2f percent confident that CT scan is %s"
        % ((100 * score), name)
    )
```

1/1 [=====] - 1s 557ms/step  
 This model is 57.07 percent confident that CT scan is normal  
 This model is 42.93 percent confident that CT scan is stroke

## FOR BASIC AUGMENTATION:

```
# Load best weights.
model.load_weights("ct-scan-brain-stroke-detection-019-0.6667.h5")
prediction = model.predict(np.expand_dims(x_val[0], axis=0))[0]
scores = [1 - prediction[0], prediction[0]]

class_names = ["normal", "stroke"]
for score, name in zip(scores, class_names):
    print(
        "This model is %.2f percent confident that CT scan is %s"
        % ((100 * score), name)
    )
```

1/1 [=====] - 0s 34ms/step  
 This model is 1.84 percent confident that CT scan is normal  
 This model is 98.16 percent confident that CT scan is stroke

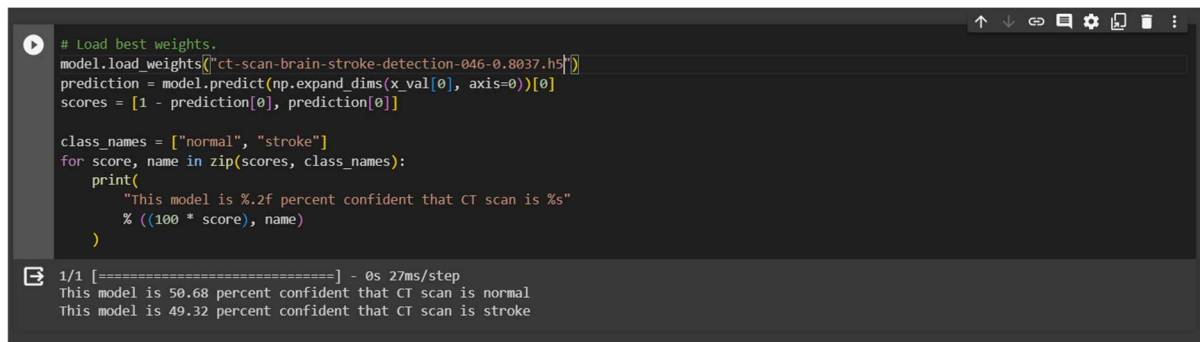
## FOR INTERMEDIATE AUGMENTATION:

```
# Load best weights.
model.load_weights("ct-scan-brain-stroke-detection-026-0.7074.h5")
prediction = model.predict(np.expand_dims(x_val[0], axis=0))[0]
scores = [1 - prediction[0], prediction[0]]

class_names = ["normal", "stroke"]
for score, name in zip(scores, class_names):
    print(
        "This model is %.2f percent confident that CT scan is %s"
        % ((100 * score), name)
    )
```

1/1 [=====] - 0s 21ms/step  
 This model is 0.05 percent confident that CT scan is normal  
 This model is 99.95 percent confident that CT scan is stroke

## FOR ADVANCED AUGMENTATION:



```
# Load best weights.
model.load_weights("ct-scan-brain-stroke-detection-046-0.8037.h5")
prediction = model.predict(np.expand_dims(x_val[0], axis=0))[0]
scores = [1 - prediction[0], prediction[0]]

class_names = ["normal", "stroke"]
for score, name in zip(scores, class_names):
    print(
        "This model is %.2f percent confident that CT scan is %s"
        % ((100 * score), name)
    )
```

1/1 [=====] - 0s 27ms/step  
This model is 50.68 percent confident that CT scan is normal  
This model is 49.32 percent confident that CT scan is stroke

## CONCLUSION:

To revolutionize the identification of brain strokes, deep learning, and image processing techniques are applied. The primary advantage of this system lies in its ability to streamline the feature engineering process, significantly reducing workload and time consumption. By leveraging Convolutional Neural Networks (CNNs), the system excels in the identification and classification of brain strokes. The utilization of advanced computer algorithms, coupled with a substantial dataset, enables the system to match the diagnostic accuracy of medical specialists, thereby elevating standards in the medical and scientific domains.

This research attempts to develop a robust model for predicting brain strokes through the implementation of CNN algorithms. The combination of distinctive features with deep learning has enhanced accuracy, enabling the model to predict a broader spectrum of diseases compared to earlier models. Further enhancement of the model's accuracy is attainable through the integration of superior hardware, sophisticated software, and an expansive dataset.

This systematic approach holds the potential to serve as a benchmark in the early recognition of brain diseases, contributing to significant reductions in both diagnosis and treatment timelines.

## REFERENCES:

1. [\*3D image classification from CT scans\*](#)
2. [\*Data augmentation for medical image analysis in deep learning\*](#)
3. [\*https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10101636\*](https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10101636)
4. [\*https://www.learn datasci.com/tutorials/convolutional-neural-networks-image-classification/\*](https://www.learn datasci.com/tutorials/convolutional-neural-networks-image-classification/)

## GITHUB REPOSITORY LINK:

[\*https://github.com/Jatinthakur-1975/Brain Stroke Prediction\*](https://github.com/Jatinthakur-1975/Brain_Stroke_Prediction)

[\*https://github.com/Jasmeet-Kaur16/ML-Project\*](https://github.com/Jasmeet-Kaur16/ML-Project)