PROJECT REPORT

NAME: S. SACHIN

Medical Image Segmentation using Brain Tumor detection Dataset

AIM

To develop an accurate and robust deep learning model for automatic brain tumor segmentation in MRI images, with the ultimate goal of assisting medical professionals in early diagnosis and treatment planning

---

ABSTRACT:

Our project aims to develop an accurate deep learning model for automatically segmenting brain tumors in MRI images. This model has the potential to assist medical professionals in diagnosing and planning treatments more efficiently, ultimately improving patient outcomes. Through rigorous training and evaluation, our results demonstrate the effectiveness of the model in accurately delineating brain tumors, showcasing its potential for clinical use.

PROCEDURE:

Data acquisition and preprocessing:

Obtain a suitable dataset containing MRI brain images and corresponding tumor segmentation masks.

Preprocess the images and masks by resizing them to a consistent resolution, normalizing pixel values, and addressing any artifacts or noise in the data.

Model selection:

Choose an appropriate deep learning architecture for image segmentation Decide on the number of layers, filter sizes, and other architectural parameters.

Loss function selection:

Select a suitable loss function for the task, such as Cross-Entropy loss, Dice loss, or a combination of multiple losses.

Data augmentation:

Apply data augmentation techniques (e.g., rotation, flipping, and zooming) to the training dataset to increase model robustness.

Training:

Train the model using the training dataset.

Use the validation dataset to monitor the training process and prevent overfitting.

Experiment with different learning rates and optimization algorithms to optimize model convergence.

Evaluation:

Evaluate the trained model on the test dataset using appropriate evaluation metrics (e.g., Intersection over Union, Dice Coefficient, F1-score) to assess its accuracy and performance.

Visualization:

Visualize the segmentation results by overlaying predicted masks on input MRI images.

Documentation and reporting:

Document the model architecture, training process, and evaluation results.

TOOLS AND LIBRARIES USED:

Visualkeras: This library allows you to visualize Keras model architectures.

Opencv (Cv2): OpenCV is a library for computer vision tasks. It's often used for image manipulation and processing.

Seaborn And Matplotlib: These libraries are used for creating data visualizations, including plots and charts.

Pandas: Pandas is a powerful library for data manipulation and analysis. You can use it to work with tabular data.

NumPy: NumPy is essential for numerical operations, especially when working with arrays and matrices.

PIL (Python Imaging Library): PIL is used for opening, manipulating, and saving image files.

Scikit-Learn: Scikit-learn is a machine learning library that provides tools for classification, regression, clustering, and more. In your code, you are importing functions for confusion matrices and classification reports.

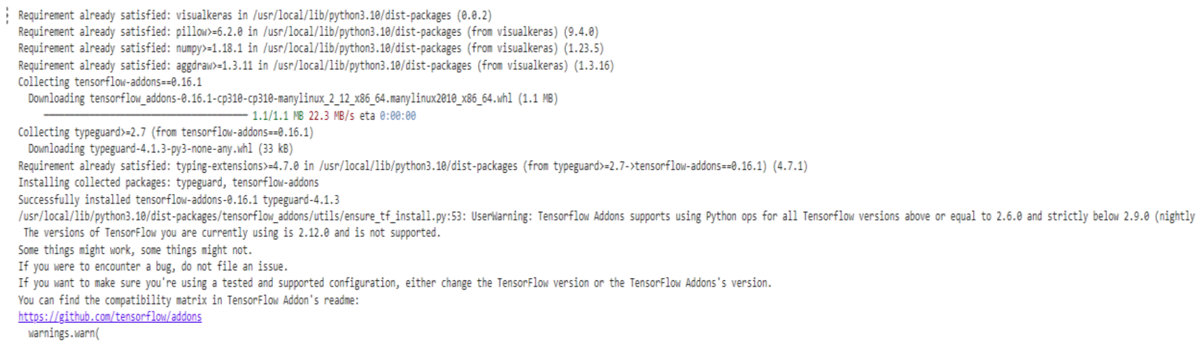
Tensorflow-Addons: An extension library for TensorFlow that provides additional functionality, such as callbacks and optimizers.

Tensorflow: TensorFlow is one of the most popular deep learning frameworks used for building and training neural networks.

IMPLEMENTATION:

IMPORTING NECESSARY LIBRARIES

```
!pip install visualkeras
import os
import warnings
import itertools
import cv2
import seaborn as sns
import pandas as pd
import numpy  as np
from PIL import Image
from sklearn.utils import class_weight
from sklearn.metrics import confusion_matrix, classification_report
from collections import Counter
!pip install tensorflow-addons==0.16.1
import tensorflow as tf
import tensorflow_addons as tfa
import visualkeras
import plotly.express as px
import matplotlib.pyplot as plt
from sklearn.metrics import multilabel_confusion_matrix

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import plot_model
from tensorflow.keras import layers
from tensorflow.keras import regularizers
from sklearn.model_selection   import train_test_split
from keras.preprocessing.image import ImageDataGenerator

warnings.filterwarnings('ignore')
%matplotlib inline
```



```
Requirement already satisfied: visualkeras in /usr/local/lib/python3.10/dist-packages (0.0.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.23.5)
Requirement already satisfied: aggdraw>=1.3.11 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.3.16)
Collecting tensorflow-addons==0.16.1
  Downloading tensorflow_addons-0.16.1-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 22.3 MB/s eta 0:00:00
Collecting typeguard>=2.7 (from tensorflow-addons==0.16.1)
  Downloading typeguard-4.1.3-py3-none-any.whl (33 kB)
Requirement already satisfied: typing-extensions>=4.7.0 in /usr/local/lib/python3.10/dist-packages (from typeguard>=2.7->tensorflow-addons==0.16.1) (4.7.1)
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.16.1 typeguard-4.1.3
/usr/local/lib/python3.10/dist-packages/tensorflow_addons/utils/ensure_tf_install.py:53: UserWarning: Tensorflow Addons supports using Python ops for all Tensorflow versions above or equal to 2.6.0 and strictly below 2.9.0 (nightly
 The versions of TensorFlow you are currently using is 2.12.0 and is not supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

SETTING UP GENERAL PARAMETERS

epochs = 15

pic_size = 240

np.random.seed(42)

tf.random.set_seed(42)

DATA LOADING, PREPERATION AND VISUALIZATION

```
folder_path = "/content/drive/MyDrive/brain_tumor_dataset"

no_images = os.listdir(folder_path + '/no/')

yes_images = os.listdir(folder_path + '/yes/')

dataset=[]

lab=[]


for image_name in no_images:

   image=cv2.imread(folder_path + '/no/' + image_name)

   image=Image.fromarray(image,'RGB')

   image=image.resize((240,240))

   dataset.append(np.array(image))

   lab.append(0)


for image_name in yes_images:

   image=cv2.imread(folder_path + '/yes/' + image_name)

   image=Image.fromarray(image,'RGB')

   image=image.resize((240,240))

   dataset.append(np.array(image))
```

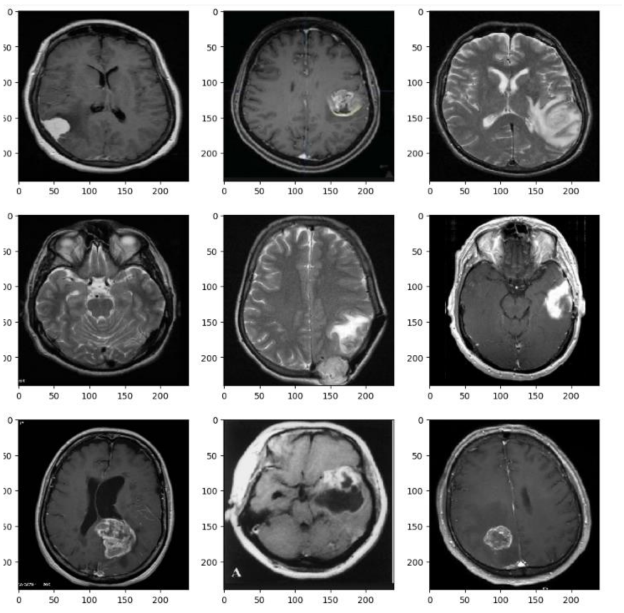
```
    lab.append(1)


dataset = np.array(dataset)
lab = np.array(lab)
print(dataset.shape, lab.shape)
```

OUTPUT

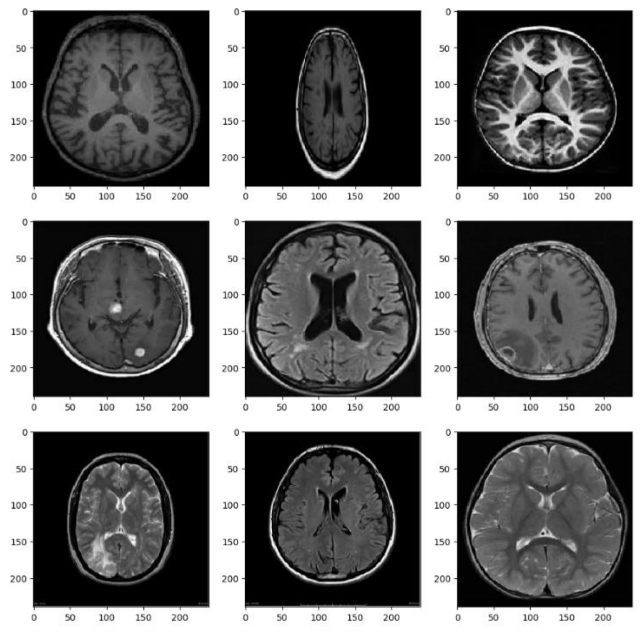
```
(253, 240, 240, 3) (253,)
```

```
x_train, x_test, y_train, y_test = train_test_split(dataset, lab, test_size=0.2, shuffle=True, random_state=42)
def plot_state(state):
    plt.figure(figsize= (12,12))
    for i in range(1, 10, 1):
        plt.subplot(3,3,i)
        img = load_img(folder_path + "/" + state + "/" + os.listdir(folder_path + "/" + state)[i], target_size=(pic_size, pic_size))
        plt.imshow(img)
    plt.show()
```

```
plot_state('yes')
```

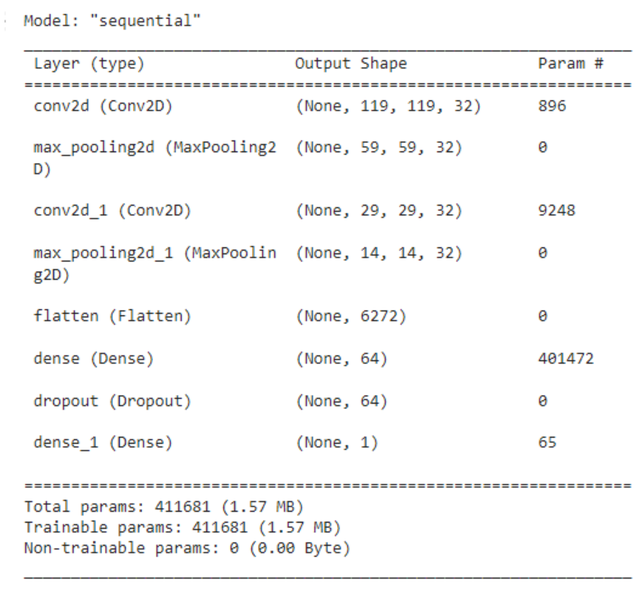


```
plot_state("no")
```



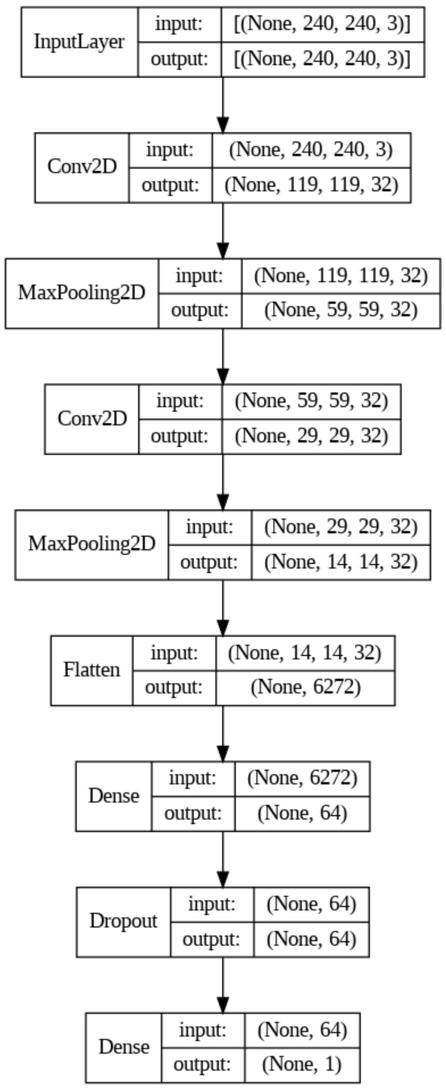
MODELING USING CNN

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),strides=(2,2), activation="relu", padding="valid",input_shape=(pic_size,pic_size,3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),strides=(2,2), activation="relu", padding="valid"),
    tf.keras.layers.MaxPooling2D((2, 2)),


    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=64, activation='relu',
                          kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3),
                          bias_regularizer=regularizers.L2(1e-2),
                          activity_regularizer=regularizers.L2(1e-3)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=1, activation='sigmoid'),
])


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```



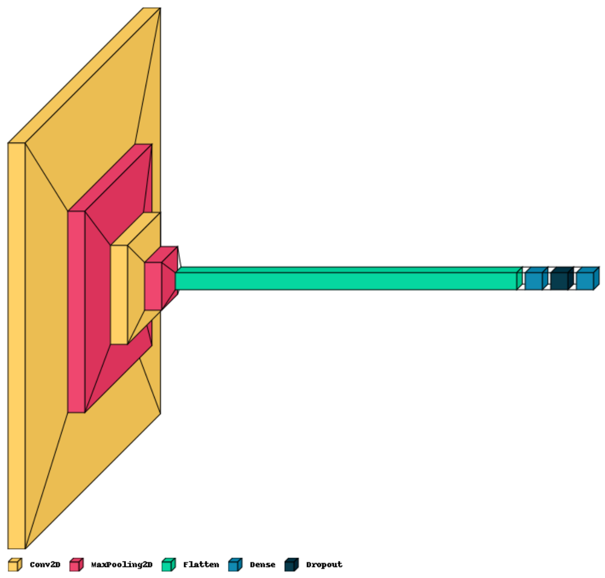
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 119, 119, 32)      896

 max_pooling2d (MaxPooling2  (None, 59, 59, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 32)        0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 64)                401472

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 411681 (1.57 MB)
Trainable params: 411681 (1.57 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

plot_model(model, show_shapes=True, show_layer_names=False)



InputLayer
input: [(None, 240, 240, 3)]
output: [(None, 240, 240, 3)]
Conv2D
input: (None, 240, 240, 3)
output: (None, 119, 119, 32)
MaxPooling2D
input: (None, 119, 119, 32)
output: (None, 59, 59, 32)
Conv2D
input: (None, 59, 59, 32)
output: (None, 29, 29, 32)
MaxPooling2D
input: (None, 29, 29, 32)
output: (None, 14, 14, 32)
Flatten
input: (None, 14, 14, 32)
output: (None, 6272)
Dense
input: (None, 6272)
output: (None, 64)
Dropout
input: (None, 64)
output: (None, 64)
Dense
input: (None, 64)
output: (None, 1)

visualkeras.layered_view(model, legend=True)



---

class_weights = class_weight.compute_class_weight(class_weight = "balanced", classes= np.unique(y_train), y= y_train)

class_weights = dict(zip(np.unique(y_train), class_weights))

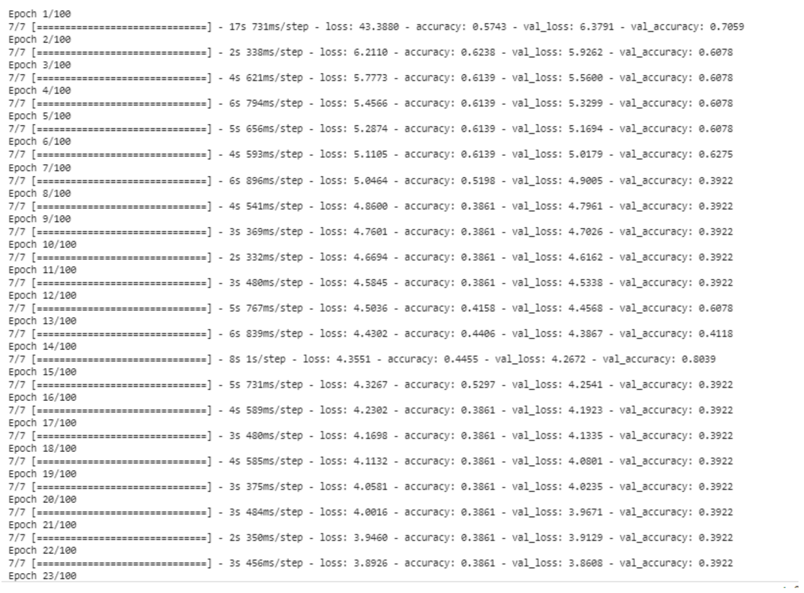
class_weights

OUTPUT

{0: 1.294871794871795, 1: 0.8145161290322581}

---

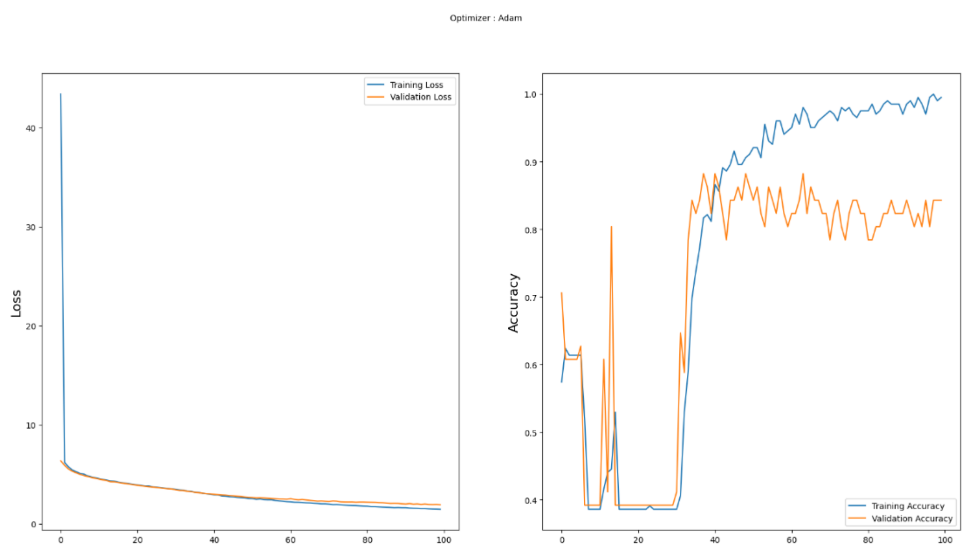
history = model.fit(x_train,y_train,epochs = 100, class_weight=class_weights, validation_data=(x_test, y_test),verbose=1)



```
Epoch 1/100
7/7 [==============================] - 17s 731ms/step - loss: 43.3880 - accuracy: 0.5743 - val_loss: 6.3791 - val_accuracy: 0.7059
Epoch 2/100
7/7 [==============================] - 2s 338ms/step - loss: 6.2110 - accuracy: 0.6238 - val_loss: 5.9262 - val_accuracy: 0.6078
Epoch 3/100
7/7 [==============================] - 4s 621ms/step - loss: 5.7773 - accuracy: 0.6139 - val_loss: 5.5600 - val_accuracy: 0.6078
Epoch 4/100
7/7 [==============================] - 6s 794ms/step - loss: 5.4566 - accuracy: 0.6139 - val_loss: 5.3299 - val_accuracy: 0.6078
Epoch 5/100
7/7 [==============================] - 5s 656ms/step - loss: 5.2874 - accuracy: 0.6139 - val_loss: 5.1694 - val_accuracy: 0.6078
Epoch 6/100
7/7 [==============================] - 4s 593ms/step - loss: 5.1105 - accuracy: 0.6139 - val_loss: 5.0179 - val_accuracy: 0.6275
Epoch 7/100
7/7 [==============================] - 6s 896ms/step - loss: 5.0464 - accuracy: 0.5198 - val_loss: 4.9005 - val_accuracy: 0.3922
Epoch 8/100
7/7 [==============================] - 4s 541ms/step - loss: 4.8600 - accuracy: 0.3861 - val_loss: 4.7961 - val_accuracy: 0.3922
Epoch 9/100
7/7 [==============================] - 3s 369ms/step - loss: 4.7601 - accuracy: 0.3861 - val_loss: 4.7026 - val_accuracy: 0.3922
Epoch 10/100
7/7 [==============================] - 2s 332ms/step - loss: 4.6694 - accuracy: 0.3861 - val_loss: 4.6162 - val_accuracy: 0.3922
Epoch 11/100
7/7 [==============================] - 3s 480ms/step - loss: 4.5845 - accuracy: 0.3861 - val_loss: 4.5338 - val_accuracy: 0.3922
Epoch 12/100
7/7 [==============================] - 5s 767ms/step - loss: 4.5036 - accuracy: 0.4158 - val_loss: 4.4568 - val_accuracy: 0.6078
Epoch 13/100
7/7 [==============================] - 6s 839ms/step - loss: 4.4302 - accuracy: 0.4406 - val_loss: 4.3867 - val_accuracy: 0.4118
Epoch 14/100
7/7 [==============================] - 8s 1s/step - loss: 4.3551 - accuracy: 0.4455 - val_loss: 4.2672 - val_accuracy: 0.8039
Epoch 15/100
7/7 [==============================] - 5s 731ms/step - loss: 4.3267 - accuracy: 0.5297 - val_loss: 4.2541 - val_accuracy: 0.3922
Epoch 16/100
7/7 [==============================] - 4s 589ms/step - loss: 4.2302 - accuracy: 0.3861 - val_loss: 4.1923 - val_accuracy: 0.3922
Epoch 17/100
7/7 [==============================] - 3s 480ms/step - loss: 4.1698 - accuracy: 0.3861 - val_loss: 4.1335 - val_accuracy: 0.3922
Epoch 18/100
7/7 [==============================] - 4s 585ms/step - loss: 4.1132 - accuracy: 0.3861 - val_loss: 4.0801 - val_accuracy: 0.3922
Epoch 19/100
7/7 [==============================] - 3s 375ms/step - loss: 4.0581 - accuracy: 0.3861 - val_loss: 4.0235 - val_accuracy: 0.3922
Epoch 20/100
7/7 [==============================] - 3s 484ms/step - loss: 4.0016 - accuracy: 0.3861 - val_loss: 3.9671 - val_accuracy: 0.3922
Epoch 21/100
7/7 [==============================] - 2s 350ms/step - loss: 3.9460 - accuracy: 0.3861 - val_loss: 3.9129 - val_accuracy: 0.3922
Epoch 22/100
7/7 [==============================] - 3s 456ms/step - loss: 3.8926 - accuracy: 0.3861 - val_loss: 3.8608 - val_accuracy: 0.3922
Epoch 23/100
```

CNN MODEL EVALUATION

```
plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)

plt.suptitle('Optimizer : Adam', fontsize=10)

plt.ylabel('Loss', fontsize=16)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.legend(loc='upper right')


plt.subplot(1, 2, 2)

plt.ylabel('Accuracy', fontsize=16)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.legend(loc='lower right')

plt.show()
```



```
results = model.evaluate(x_test, y_test)

print('The current model achieved an accuracy of {}%!'.format(round(results[1]*100,2)))
```

```
2/2 [==============================] - 1s 225ms/step - loss: 1.9464 - accuracy: 0.8431
The current model achieved an accuracy of 84.31%!
```

```
predictions = model.predict(x_test)

y_pred = []
```

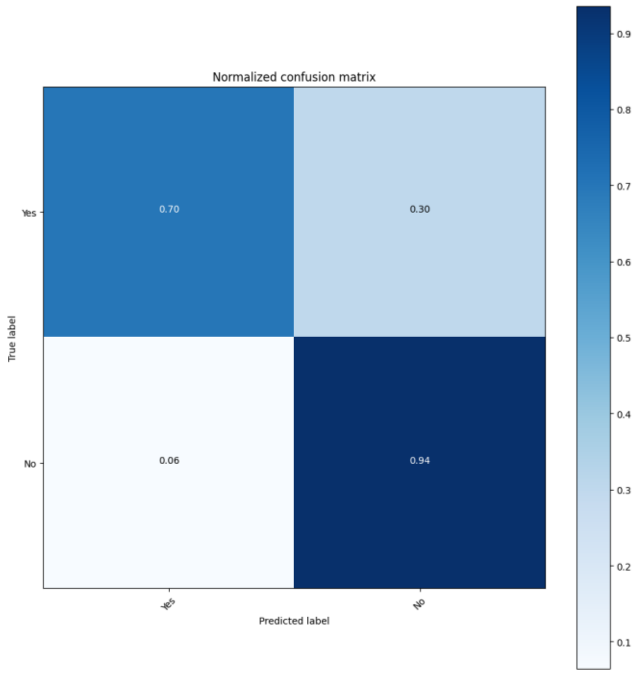
```
for i in predictions:
    if i >= 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

OUTPUT

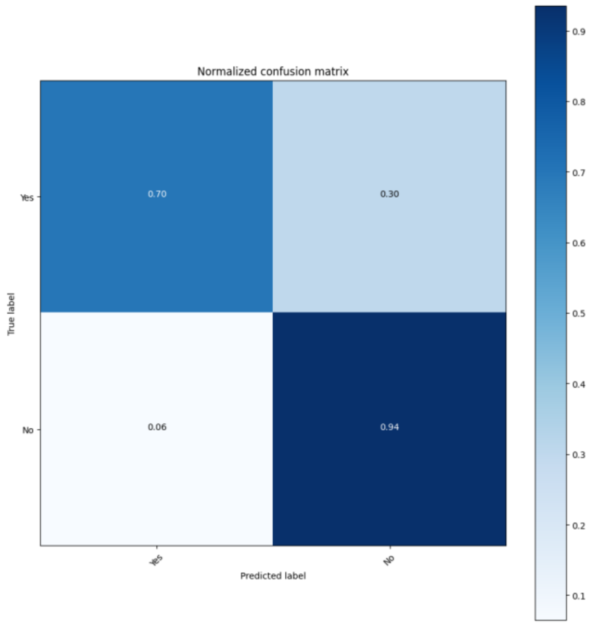
```
2/2 [==============================] - 1s 124ms/step
```

```
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Blues):
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
# plot normalized confusion matrix
```

```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=["Yes", "No"], title='Normalized confusion matrix')
plt.show()
```



```
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=["Yes", "No"], title='Normalized confusion matrix')
plt.show()
```



MODELING USING VISION TRANSFORMERS(VIT)

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 100
image_size = 240  # We'll resize input images to this size
patch_size = 20  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]  # Size of the dense layers of the final classifier
```

DATA AUGMENTATION

```
data_augmentation = tf.keras.Sequential
        layers.Normalization(),
        layers.Resizing(image_size, image_size),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(x_train)
```

MULTI-LAYER PERCEPTRON

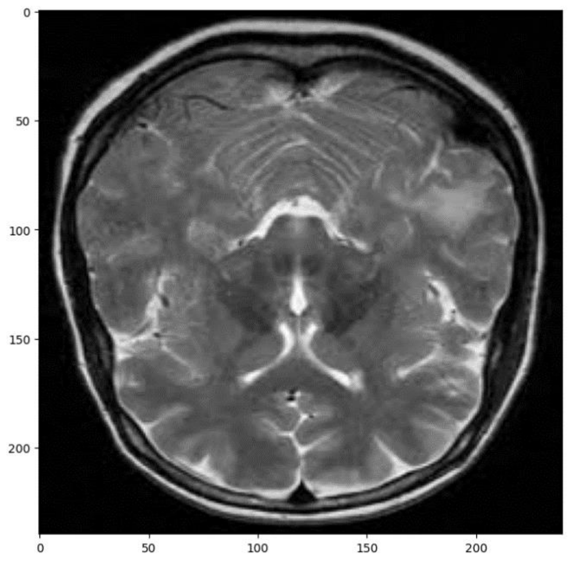
```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

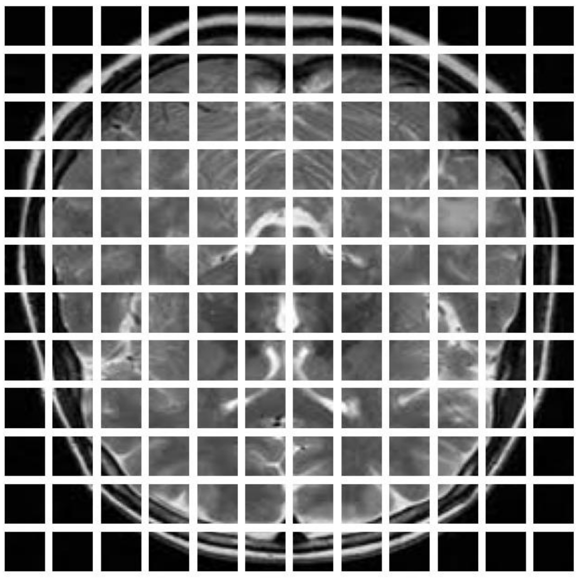
IMPLEMENT PATCH CREATION AS A LAYER

```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches

plt.figure(figsize=(8, 8))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(image_size, image_size)
)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")


n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(8, 8))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```

OUTPUT

```
Image size: 240 X 240
Patch size: 20 X 20
Patches per image: 144
Elements per patch: 1200
```





CREATING THE PATCH ENCODER

```
class PatchEncoder(tf.keras.layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim


    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

CONCLUSION

In conclusion, medical image segmentation for brain tumor detection is a vital application of deep learning in healthcare. Leveraging publicly available datasets and advanced algorithms like CNNs. This field has made significant strides in automating tumor region identification in MRI scans. Accurate segmentation remains a critical component in early diagnosis and treatment planning, enhancing the overall quality of patient care in the context of brain tumor detection and management.