CNN for Brain Tumor MRI Classification

-- Author: Kris Li

The model

I will use a simple CNN in this project to see how useful it actually is with complicated datasets such as non-uniform MRI scans.

The dataset

We will use an image dataset from Kaggle (Chakrabarty, 2008). This dataset contains two folders, `\No` and `\Yes`, which contain 98 healthy MRI images and 155 tumor MRI images correspondingly.

Import Packages

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import os
import scipy
```

In [2]:
```
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools
```

In [3]:
```
import cv2, imutils
```

In [4]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, load_model
from keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D, Flatt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping, Callba
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Add, Activation
```

In [5]:
```
# For reproducibility
np.random.seed(11)
tf.random.set_seed(11)
```

Data Preprocessing

We can use a simple tool *keras/image_dataset_from_directory* here to organize the images (TensorFlow, 2023). Most importantly, we seperate training/testing sets by the 8/2 rule, and resize all images into 256x256.

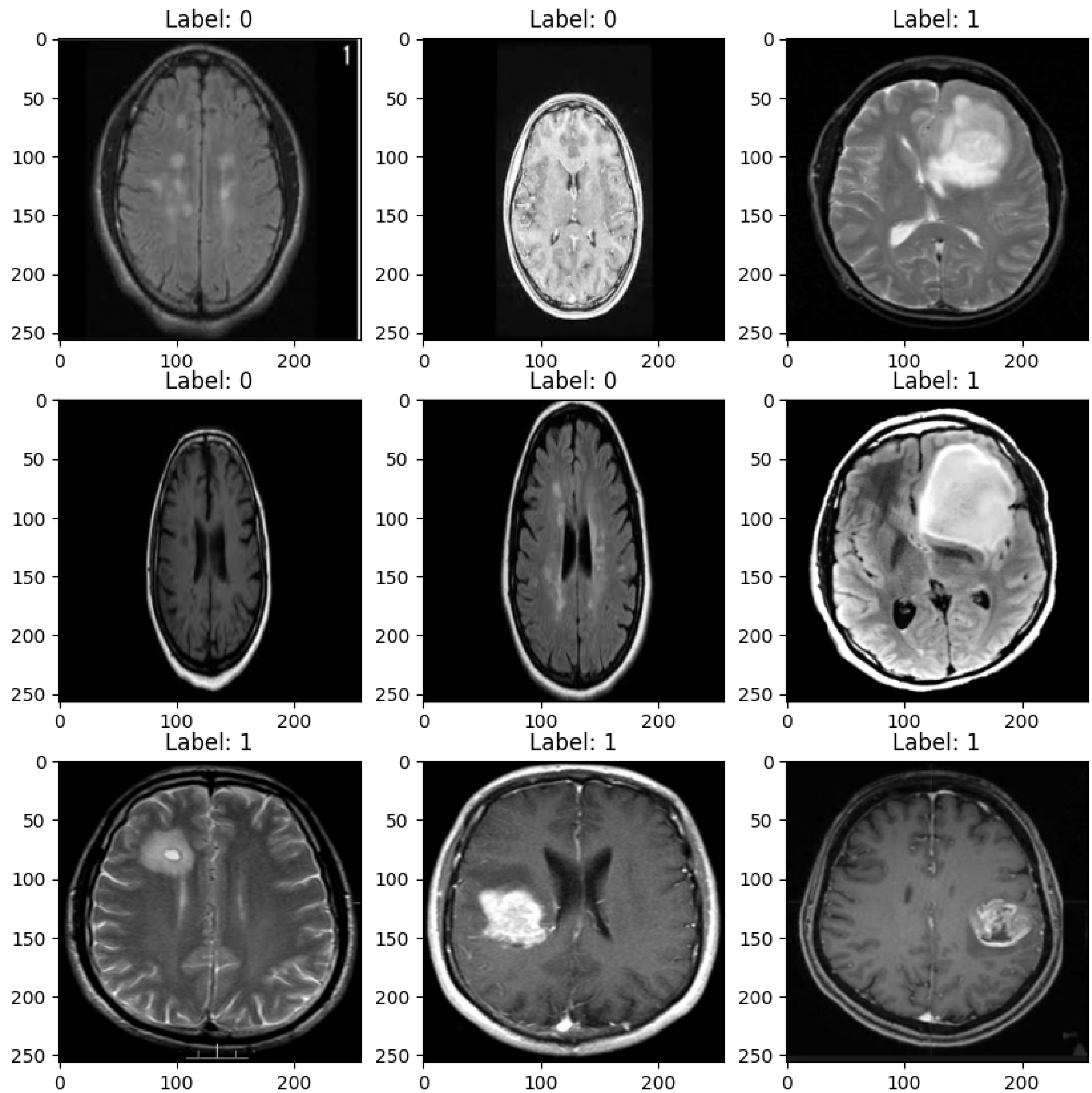
In [6]:
```
train_dataset = keras.utils.image_dataset_from_directory(
    "./MRI_folder/original",
    validation_split=0.2,
    subset="training",
    seed=11,
    image_size=(256, 256),
    color_mode='grayscale',
    labels='inferred'
)

val_dataset = keras.utils.image_dataset_from_directory(
    "./MRI_folder/original",
    validation_split=0.2,
    subset="validation",
    seed=11,
    image_size=(256, 256),
    color_mode='grayscale',
    labels='inferred'
)
```

```
Found 253 files belonging to 2 classes.
Using 203 files for training.
Found 253 files belonging to 2 classes.
Using 50 files for validation.
```

(`labels='inferred'` labels tumor MRIs as 1 and healthy MRIs as 0.)

In [7]:
```
for image_batch, labels_batch in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image_batch[i].numpy(), cmap = 'gray')
        plt.title(f'Label: {labels_batch[i].numpy()}')
    plt.show()
```



We can see that there are only 253 files in total, and the images aren't uniform in terms of composition. So lets do some data augmentation and crop these images.

In [53]:
```
# Partially credited to https://www.kaggle.com/code/ruslankl/brain-tumor-detection-

def crop_aug_imgs(input_folder, output_folder, add_pixels_value=0, AUG_COUNT = 9):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Initialize image data generator
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
```

```
        vertical_flip=True)

    for filename in os.listdir(input_folder):
        # Read the original image
        img_path = os.path.join(input_folder, filename)
        img = cv2.imread(img_path)

        # Augment and save
        basename, ext = os.path.splitext(filename)
        for i in range(AUG_COUNT):
            aug_img = datagen.random_transform(img)

            # Find the contours and get the bounding box for the augmented image
            gray = cv2.cvtColor(aug_img, cv2.COLOR_BGR2GRAY)
            gray = cv2.GaussianBlur(gray, (5, 5), 0)
            thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
            thresh = cv2.erode(thresh, None, iterations=2)
            thresh = cv2.dilate(thresh, None, iterations=2)
            cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APP
            cnts = imutils.grab_contours(cnts)
            c = max(cnts, key=cv2.contourArea)
            x, y, w, h = cv2.boundingRect(c)

            # Add padding around the bounding box
            ADD_PIXELS = add_pixels_value
            x -= ADD_PIXELS
            y -= ADD_PIXELS
            w += ADD_PIXELS * 2
            h += ADD_PIXELS * 2

            # Make sure padded coordinates are not outside the image
            x = max(0, x)
            y = max(0, y)
            w = min(aug_img.shape[1], x + w) - x
            h = min(aug_img.shape[0], y + h) - y

            # Crop the augmented image
            new_img = aug_img[y:y + h, x:x + w]

            # Save the augmented and cropped image
            aug_filename = basename + "_aug" + str(i + 1) + ext
            output_path = os.path.join(output_folder, aug_filename)
            cv2.imwrite(output_path, new_img)
```

In [10]:

```
input_folder = "./MRI_folder/original/no/"
output_folder = "./MRI_folder/aligned/no/"
crop_aug_imgs(input_folder, output_folder, add_pixels_value=10, AUG_COUNT = 9)
input_folder = "./MRI_folder/original/yes/"
output_folder = "./MRI_folder/aligned/yes/"
crop_aug_imgs(input_folder, output_folder, add_pixels_value=10, AUG_COUNT = 9)
```

Now let's re-import the images.

In [27]:

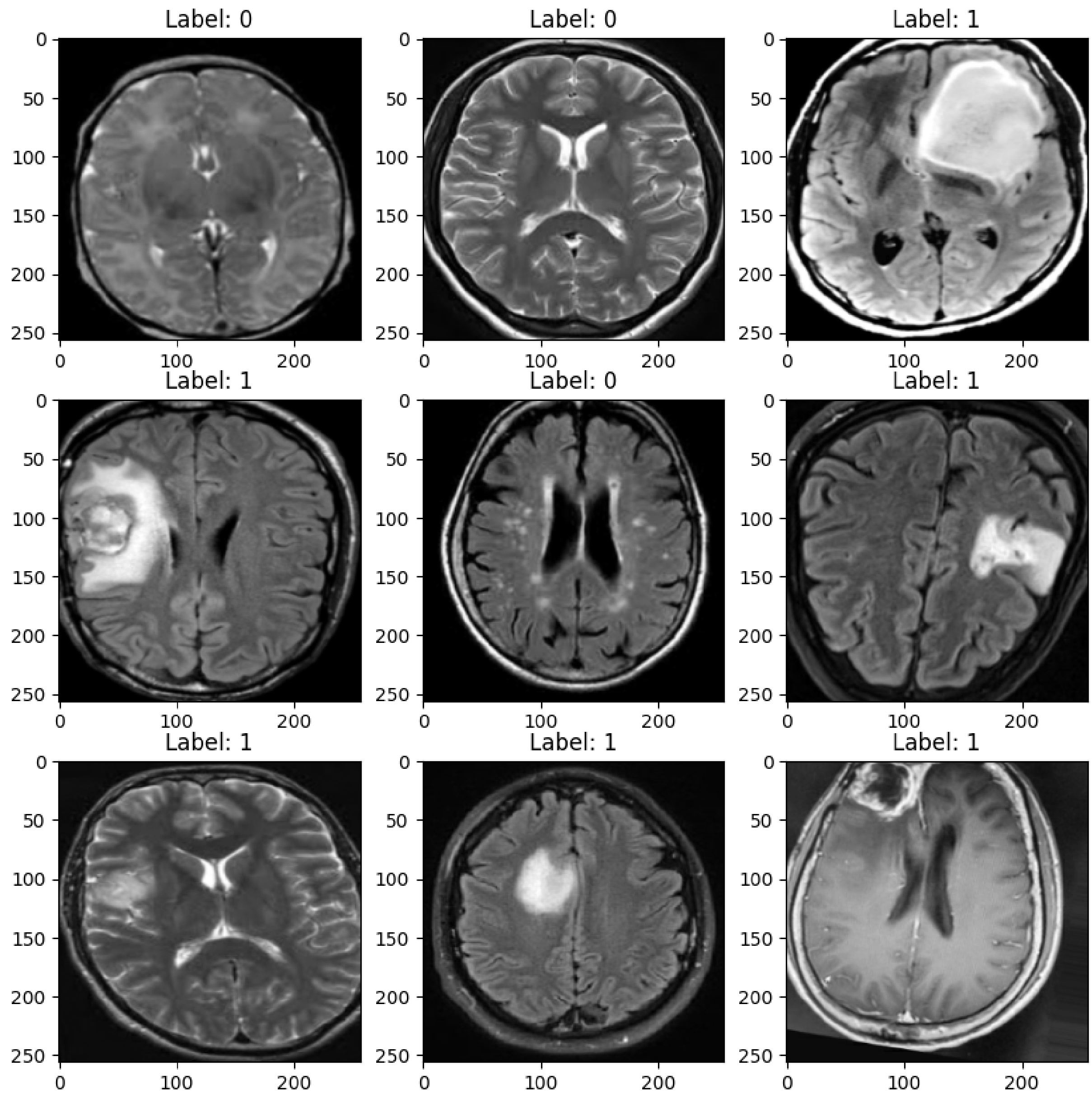
```
train_dataset = keras.utils.image_dataset_from_directory(
    "./MRI_folder/aligned",
```

```
    validation_split=0.3,
    subset="training",
    seed=11,
    image_size=(256, 256),
    color_mode='grayscale',
    labels='inferred',
    batch_size=25
)

val_dataset = keras.utils.image_dataset_from_directory(
    "./MRI_folder/aligned",
    validation_split=0.3,
    subset="validation",
    seed=11,
    image_size=(256, 256),
    color_mode='grayscale',
    labels='inferred',
    batch_size=25
)

for image_batch, labels_batch in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image_batch[i].numpy(), cmap = 'gray')
        plt.title(f'Label: {labels_batch[i].numpy()}')
    plt.show()
```

```
Found 2530 files belonging to 2 classes.
Using 1771 files for training.
Found 2530 files belonging to 2 classes.
Using 759 files for validation.
```


Label: 0
Label: 0
Label: 1
Label: 1
Label: 0
Label: 1
Label: 1
Label: 1
Label: 1

Model Construction

In [37]:
```
def lr_schedule(epoch):
    """Learning Rate Schedule"""
    lr = 0.001
    if epoch > 20:
        lr *= 0.5
    elif epoch > 50:
        lr *= 0.1
    return lr

# Learning rate scheduler callback
lr_scheduler = LearningRateScheduler(lr_schedule)
```

In [42]:
```
CNN_model = Sequential()

CNN_model.add(Conv2D(32, 3, padding="same", activation='relu', input_shape=(256, 25
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D())
CNN_model.add(Dropout(0.2))

CNN_model.add(Conv2D(64, 3, padding="same", activation='relu', kernel_regularizer=l
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D())
CNN_model.add(Dropout(0.2))

CNN_model.add(Conv2D(128, 3, padding='same', activation='relu', kernel_regularizer=
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D())
CNN_model.add(Dropout(0.2))

CNN_model.add(Conv2D(256, 3, padding='same', activation='relu', kernel_regularizer=
CNN_model.add(BatchNormalization())
CNN_model.add(MaxPooling2D())
CNN_model.add(Dropout(0.2))

CNN_model.add(Flatten())
CNN_model.add(Dense(128, activation='relu'))
CNN_model.add(Dropout(0.2))
CNN_model.add(Dense(64, activation='relu'))
CNN_model.add(Dropout(0.2))
CNN_model.add(Dense(1, activation='sigmoid'))

opt = Adam(learning_rate=0.001)
CNN_model.compile(optimizer=opt, loss='binary_crossentropy', metrics=["accuracy"])

CNN_model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_47 (Conv2D)          (None, 256, 256, 32)      320

 batch_normalization_47 (Bat  (None, 256, 256, 32)     128
 chNormalization)

 max_pooling2d_47 (MaxPoolin  (None, 128, 128, 32)     0
 g2D)

 dropout_59 (Dropout)        (None, 128, 128, 32)      0

 conv2d_48 (Conv2D)          (None, 128, 128, 64)      18496

 batch_normalization_48 (Bat  (None, 128, 128, 64)     256
 chNormalization)

 max_pooling2d_48 (MaxPoolin  (None, 64, 64, 64)       0
 g2D)

 dropout_60 (Dropout)        (None, 64, 64, 64)        0

 conv2d_49 (Conv2D)          (None, 64, 64, 128)       73856

 batch_normalization_49 (Bat  (None, 64, 64, 128)      512
 chNormalization)

 max_pooling2d_49 (MaxPoolin  (None, 32, 32, 128)      0
 g2D)

 dropout_61 (Dropout)        (None, 32, 32, 128)       0

 conv2d_50 (Conv2D)          (None, 32, 32, 256)       295168

 batch_normalization_50 (Bat  (None, 32, 32, 256)      1024
 chNormalization)

 max_pooling2d_50 (MaxPoolin  (None, 16, 16, 256)      0
 g2D)

 dropout_62 (Dropout)        (None, 16, 16, 256)       0

 flatten_12 (Flatten)        (None, 65536)             0

 dense_24 (Dense)            (None, 128)               8388736

 dropout_63 (Dropout)        (None, 128)               0

 dense_25 (Dense)            (None, 64)                8256

 dropout_64 (Dropout)        (None, 64)                0

 dense_26 (Dense)            (None, 1)                 65
```

```
=================================================================
Total params: 8,786,817
Trainable params: 8,785,857
Non-trainable params: 960
_________________________________________________________________
```

Model Fitting

In [43]:
```
CNN_history = CNN_model.fit(train_dataset, epochs=75, validation_data=val_dataset,
```

```
Epoch 1/75
71/71 [==============================] - 8s 101ms/step - loss: 3.6187 - accuracy: 0.
6222 - val_loss: 2.5138 - val_accuracy: 0.5968 - lr: 0.0010
Epoch 2/75
71/71 [==============================] - 7s 99ms/step - loss: 1.3996 - accuracy: 0.7
053 - val_loss: 1.4480 - val_accuracy: 0.6851 - lr: 0.0010
Epoch 3/75
71/71 [==============================] - 7s 99ms/step - loss: 0.9455 - accuracy: 0.7
578 - val_loss: 0.7461 - val_accuracy: 0.7365 - lr: 0.0010
Epoch 4/75
71/71 [==============================] - 7s 99ms/step - loss: 0.7751 - accuracy: 0.7
634 - val_loss: 0.5541 - val_accuracy: 0.7549 - lr: 0.0010
Epoch 5/75
71/71 [==============================] - 7s 99ms/step - loss: 0.5539 - accuracy: 0.7
420 - val_loss: 0.4549 - val_accuracy: 0.7589 - lr: 0.0010
Epoch 6/75
71/71 [==============================] - 7s 99ms/step - loss: 0.4836 - accuracy: 0.7
899 - val_loss: 0.4545 - val_accuracy: 0.8169 - lr: 0.0010
Epoch 7/75
71/71 [==============================] - 7s 99ms/step - loss: 0.4386 - accuracy: 0.8
340 - val_loss: 0.4688 - val_accuracy: 0.8063 - lr: 0.0010
Epoch 8/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3966 - accuracy: 0.8
379 - val_loss: 0.3910 - val_accuracy: 0.8669 - lr: 0.0010
Epoch 9/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3472 - accuracy: 0.8
780 - val_loss: 0.3746 - val_accuracy: 0.8696 - lr: 0.0010
Epoch 10/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3165 - accuracy: 0.8
679 - val_loss: 0.3199 - val_accuracy: 0.8722 - lr: 0.0010
Epoch 11/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3240 - accuracy: 0.8
724 - val_loss: 0.3133 - val_accuracy: 0.8854 - lr: 0.0010
Epoch 12/75
71/71 [==============================] - 7s 99ms/step - loss: 0.2864 - accuracy: 0.8
763 - val_loss: 0.3954 - val_accuracy: 0.8643 - lr: 0.0010
Epoch 13/75
71/71 [==============================] - 7s 98ms/step - loss: 0.2714 - accuracy: 0.8
707 - val_loss: 0.8532 - val_accuracy: 0.7905 - lr: 0.0010
Epoch 14/75
71/71 [==============================] - 7s 98ms/step - loss: 0.3459 - accuracy: 0.8
696 - val_loss: 0.4620 - val_accuracy: 0.8379 - lr: 0.0010
Epoch 15/75
71/71 [==============================] - 7s 98ms/step - loss: 0.2934 - accuracy: 0.8
730 - val_loss: 0.4582 - val_accuracy: 0.8366 - lr: 0.0010
Epoch 16/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3615 - accuracy: 0.8
690 - val_loss: 0.3798 - val_accuracy: 0.8682 - lr: 0.0010
Epoch 17/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3810 - accuracy: 0.8
746 - val_loss: 0.5763 - val_accuracy: 0.8458 - lr: 0.0010
Epoch 18/75
71/71 [==============================] - 7s 99ms/step - loss: 0.3187 - accuracy: 0.8
741 - val_loss: 1.0009 - val_accuracy: 0.7787 - lr: 0.0010
Epoch 19/75
71/71 [==============================] - 7s 101ms/step - loss: 0.2651 - accuracy: 0.
```

```
8842 - val_loss: 0.2641 - val_accuracy: 0.9051 - lr: 0.0010
Epoch 20/75
71/71 [==============================] - 7s 100ms/step - loss: 0.2250 - accuracy: 0.
9108 - val_loss: 0.3152 - val_accuracy: 0.8933 - lr: 0.0010
Epoch 21/75
71/71 [==============================] - 7s 101ms/step - loss: 0.2412 - accuracy: 0.
8978 - val_loss: 0.3103 - val_accuracy: 0.8920 - lr: 0.0010
Epoch 22/75
71/71 [==============================] - 7s 100ms/step - loss: 0.1959 - accuracy: 0.
9187 - val_loss: 0.3286 - val_accuracy: 0.9038 - lr: 5.0000e-04
Epoch 23/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1751 - accuracy: 0.9
255 - val_loss: 0.2732 - val_accuracy: 0.9104 - lr: 5.0000e-04
Epoch 24/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1426 - accuracy: 0.9
385 - val_loss: 0.2723 - val_accuracy: 0.9262 - lr: 5.0000e-04
Epoch 25/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1176 - accuracy: 0.9
526 - val_loss: 0.2943 - val_accuracy: 0.9275 - lr: 5.0000e-04
Epoch 26/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1364 - accuracy: 0.9
497 - val_loss: 0.3187 - val_accuracy: 0.9262 - lr: 5.0000e-04
Epoch 27/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1487 - accuracy: 0.9
447 - val_loss: 0.3303 - val_accuracy: 0.9078 - lr: 5.0000e-04
Epoch 28/75
71/71 [==============================] - 7s 100ms/step - loss: 0.1264 - accuracy: 0.
9543 - val_loss: 0.2634 - val_accuracy: 0.9209 - lr: 5.0000e-04
Epoch 29/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1063 - accuracy: 0.9
644 - val_loss: 0.3311 - val_accuracy: 0.9249 - lr: 5.0000e-04
Epoch 30/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1356 - accuracy: 0.9
554 - val_loss: 0.2717 - val_accuracy: 0.9315 - lr: 5.0000e-04
Epoch 31/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1143 - accuracy: 0.9
548 - val_loss: 0.2528 - val_accuracy: 0.9381 - lr: 5.0000e-04
Epoch 32/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1094 - accuracy: 0.9
627 - val_loss: 0.2388 - val_accuracy: 0.9368 - lr: 5.0000e-04
Epoch 33/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0984 - accuracy: 0.9
656 - val_loss: 0.3052 - val_accuracy: 0.9196 - lr: 5.0000e-04
Epoch 34/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1023 - accuracy: 0.9
644 - val_loss: 0.2321 - val_accuracy: 0.9354 - lr: 5.0000e-04
Epoch 35/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1151 - accuracy: 0.9
644 - val_loss: 0.2890 - val_accuracy: 0.9407 - lr: 5.0000e-04
Epoch 36/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1035 - accuracy: 0.9
712 - val_loss: 0.2869 - val_accuracy: 0.9420 - lr: 5.0000e-04
Epoch 37/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0912 - accuracy: 0.9
718 - val_loss: 0.2883 - val_accuracy: 0.9381 - lr: 5.0000e-04
Epoch 38/75
```

```
71/71 [==============================] - 7s 99ms/step - loss: 0.1035 - accuracy: 0.9
622 - val_loss: 0.2682 - val_accuracy: 0.9328 - lr: 5.0000e-04
Epoch 39/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0911 - accuracy: 0.9
678 - val_loss: 0.2416 - val_accuracy: 0.9433 - lr: 5.0000e-04
Epoch 40/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0865 - accuracy: 0.9
763 - val_loss: 0.3525 - val_accuracy: 0.9144 - lr: 5.0000e-04
Epoch 41/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0707 - accuracy: 0.9
814 - val_loss: 0.2516 - val_accuracy: 0.9420 - lr: 5.0000e-04
Epoch 42/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1132 - accuracy: 0.9
735 - val_loss: 0.3138 - val_accuracy: 0.9170 - lr: 5.0000e-04
Epoch 43/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0977 - accuracy: 0.9
768 - val_loss: 0.2643 - val_accuracy: 0.9433 - lr: 5.0000e-04
Epoch 44/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0897 - accuracy: 0.9
689 - val_loss: 0.3087 - val_accuracy: 0.9262 - lr: 5.0000e-04
Epoch 45/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0645 - accuracy: 0.9
825 - val_loss: 0.2605 - val_accuracy: 0.9447 - lr: 5.0000e-04
Epoch 46/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0941 - accuracy: 0.9
723 - val_loss: 0.2971 - val_accuracy: 0.9460 - lr: 5.0000e-04
Epoch 47/75
71/71 [==============================] - 7s 100ms/step - loss: 0.0704 - accuracy: 0.
9808 - val_loss: 0.2438 - val_accuracy: 0.9539 - lr: 5.0000e-04
Epoch 48/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0711 - accuracy: 0.9
842 - val_loss: 0.2335 - val_accuracy: 0.9513 - lr: 5.0000e-04
Epoch 49/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0651 - accuracy: 0.9
831 - val_loss: 0.2825 - val_accuracy: 0.9394 - lr: 5.0000e-04
Epoch 50/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0616 - accuracy: 0.9
825 - val_loss: 0.2905 - val_accuracy: 0.9407 - lr: 5.0000e-04
Epoch 51/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0676 - accuracy: 0.9
825 - val_loss: 0.2325 - val_accuracy: 0.9394 - lr: 5.0000e-04
Epoch 52/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0918 - accuracy: 0.9
774 - val_loss: 0.6580 - val_accuracy: 0.8340 - lr: 5.0000e-04
Epoch 53/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0844 - accuracy: 0.9
768 - val_loss: 0.2530 - val_accuracy: 0.9539 - lr: 5.0000e-04
Epoch 54/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0875 - accuracy: 0.9
695 - val_loss: 0.2498 - val_accuracy: 0.9433 - lr: 5.0000e-04
Epoch 55/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0668 - accuracy: 0.9
802 - val_loss: 0.2890 - val_accuracy: 0.9420 - lr: 5.0000e-04
Epoch 56/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0960 - accuracy: 0.9
723 - val_loss: 0.2583 - val_accuracy: 0.9447 - lr: 5.0000e-04
```

```
Epoch 57/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0701 - accuracy: 0.9
814 - val_loss: 0.2927 - val_accuracy: 0.9433 - lr: 5.0000e-04
Epoch 58/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0834 - accuracy: 0.9
780 - val_loss: 0.4702 - val_accuracy: 0.9209 - lr: 5.0000e-04
Epoch 59/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0780 - accuracy: 0.9
785 - val_loss: 0.2436 - val_accuracy: 0.9486 - lr: 5.0000e-04
Epoch 60/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0696 - accuracy: 0.9
831 - val_loss: 0.2816 - val_accuracy: 0.9341 - lr: 5.0000e-04
Epoch 61/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0718 - accuracy: 0.9
785 - val_loss: 0.5649 - val_accuracy: 0.9249 - lr: 5.0000e-04
Epoch 62/75
71/71 [==============================] - 7s 99ms/step - loss: 0.1071 - accuracy: 0.9
723 - val_loss: 0.3810 - val_accuracy: 0.9104 - lr: 5.0000e-04
Epoch 63/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0868 - accuracy: 0.9
729 - val_loss: 0.3607 - val_accuracy: 0.9157 - lr: 5.0000e-04
Epoch 64/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0843 - accuracy: 0.9
785 - val_loss: 0.3168 - val_accuracy: 0.9354 - lr: 5.0000e-04
Epoch 65/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0662 - accuracy: 0.9
802 - val_loss: 0.3729 - val_accuracy: 0.9289 - lr: 5.0000e-04
Epoch 66/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0545 - accuracy: 0.9
819 - val_loss: 0.4300 - val_accuracy: 0.9223 - lr: 5.0000e-04
Epoch 67/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0625 - accuracy: 0.9
859 - val_loss: 0.3083 - val_accuracy: 0.9499 - lr: 5.0000e-04
Epoch 68/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0611 - accuracy: 0.9
842 - val_loss: 0.3171 - val_accuracy: 0.9223 - lr: 5.0000e-04
Epoch 69/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0628 - accuracy: 0.9
831 - val_loss: 0.6732 - val_accuracy: 0.9183 - lr: 5.0000e-04
Epoch 70/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0568 - accuracy: 0.9
836 - val_loss: 0.3746 - val_accuracy: 0.9315 - lr: 5.0000e-04
Epoch 71/75
71/71 [==============================] - 7s 98ms/step - loss: 0.0689 - accuracy: 0.9
831 - val_loss: 0.2285 - val_accuracy: 0.9578 - lr: 5.0000e-04
Epoch 72/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0693 - accuracy: 0.9
814 - val_loss: 0.2726 - val_accuracy: 0.9526 - lr: 5.0000e-04
Epoch 73/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0512 - accuracy: 0.9
904 - val_loss: 0.2701 - val_accuracy: 0.9447 - lr: 5.0000e-04
Epoch 74/75
71/71 [==============================] - 7s 99ms/step - loss: 0.0831 - accuracy: 0.9
819 - val_loss: 0.3148 - val_accuracy: 0.9368 - lr: 5.0000e-04
Epoch 75/75
```

```
71/71 [==============================] - 7s 99ms/step - loss: 0.0662 - accuracy: 0.9
836 - val_loss: 0.2668 - val_accuracy: 0.9473 - lr: 5.0000e-04
```

In [45]:
```
# Save the model in the HDF5 format
CNN_model.save('./MRI_folder/model/CNN_model.h5')
```

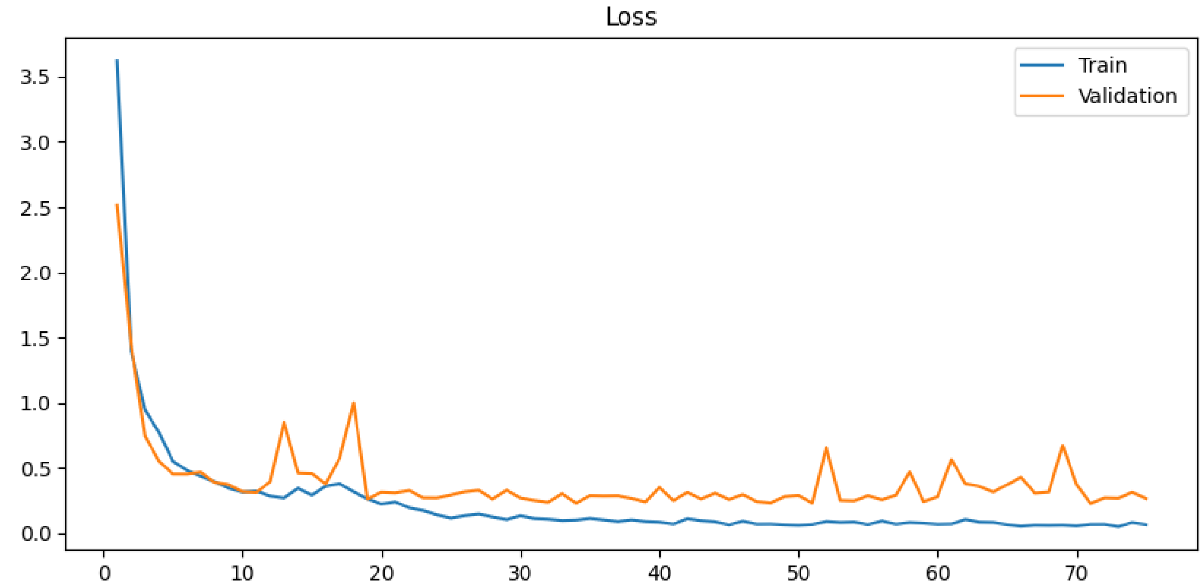
In [47]:
```
# Load model
CNN_model = load_model('./MRI_folder/model/CNN_model.h5')
```

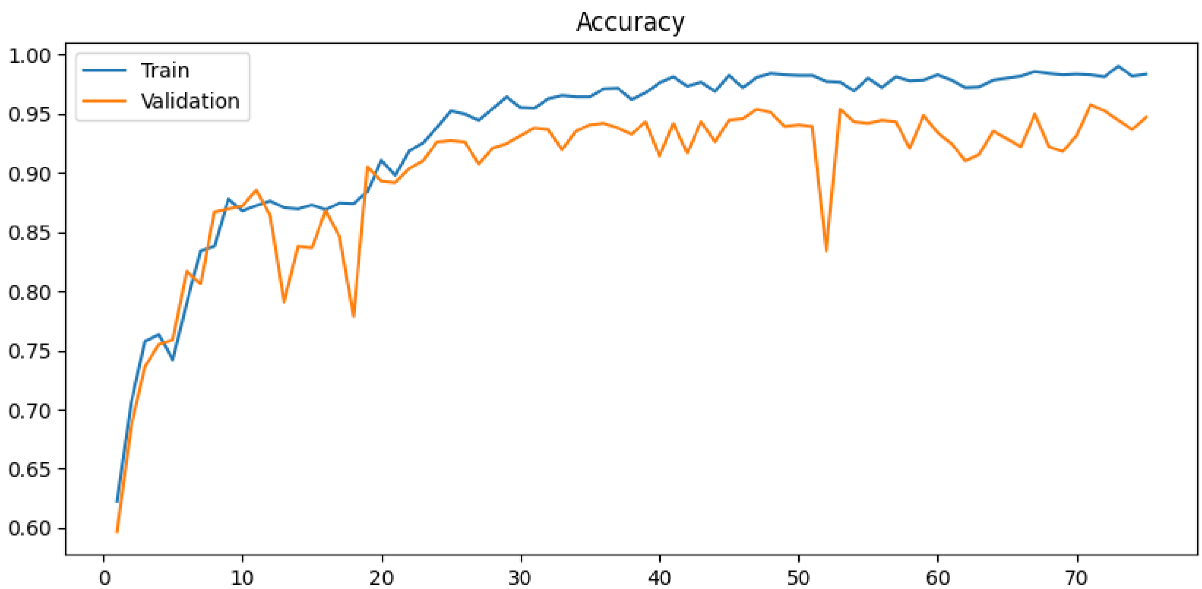
In [44]:
```
fig, ax = plt.subplots(2, 1, figsize=(10, 10))

# Loss
sns.lineplot(x=range(1, len(CNN_history.history['loss']) + 1), y=CNN_history.histor
sns.lineplot(x=range(1, len(CNN_history.history['val_loss']) + 1), y=CNN_history.hi
ax[0].set_title('Loss')
ax[0].legend()

# Accuracy
sns.lineplot(x=range(1, len(CNN_history.history['accuracy']) + 1), y=CNN_history.hi
sns.lineplot(x=range(1, len(CNN_history.history['val_accuracy']) + 1), y=CNN_histor
ax[1].set_title('Accuracy')
ax[1].legend()

plt.show()
```


Loss
Train
Validation
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50
60
70


Accuracy
Train
Validation
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
10
20
30
40
50
60
70

Model Testing

Now we load a brand new MRI imaging dataset for testing. (Nickparvar, 2008)

In [65]:
```
# Should've seperate the crop and aug functions...

def crop_imgs(input_folder, output_folder, add_pixels_value=0):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    for filename in os.listdir(input_folder):
        # Read the original image
        img_path = os.path.join(input_folder, filename)
        img = cv2.imread(img_path)

        # Find the contours and get the bounding box for the image
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
        cnts = imutils.grab_contours(cnts)
        if cnts:
            c = max(cnts, key=cv2.contourArea)
            x, y, w, h = cv2.boundingRect(c)

            # Add padding around the bounding box
            ADD_PIXELS = add_pixels_value
            x -= ADD_PIXELS
            y -= ADD_PIXELS
            w += ADD_PIXELS * 2
            h += ADD_PIXELS * 2

            # Make sure padded coordinates are not outside the image
            x = max(0, x)
            y = max(0, y)
            w = min(img.shape[1], x + w) - x
            h = min(img.shape[0], y + h) - y

            # Crop the image
            new_img = img[y:y + h, x:x + w]

            # Save the cropped image
            output_path = os.path.join(output_folder, filename)
            cv2.imwrite(output_path, new_img)
```

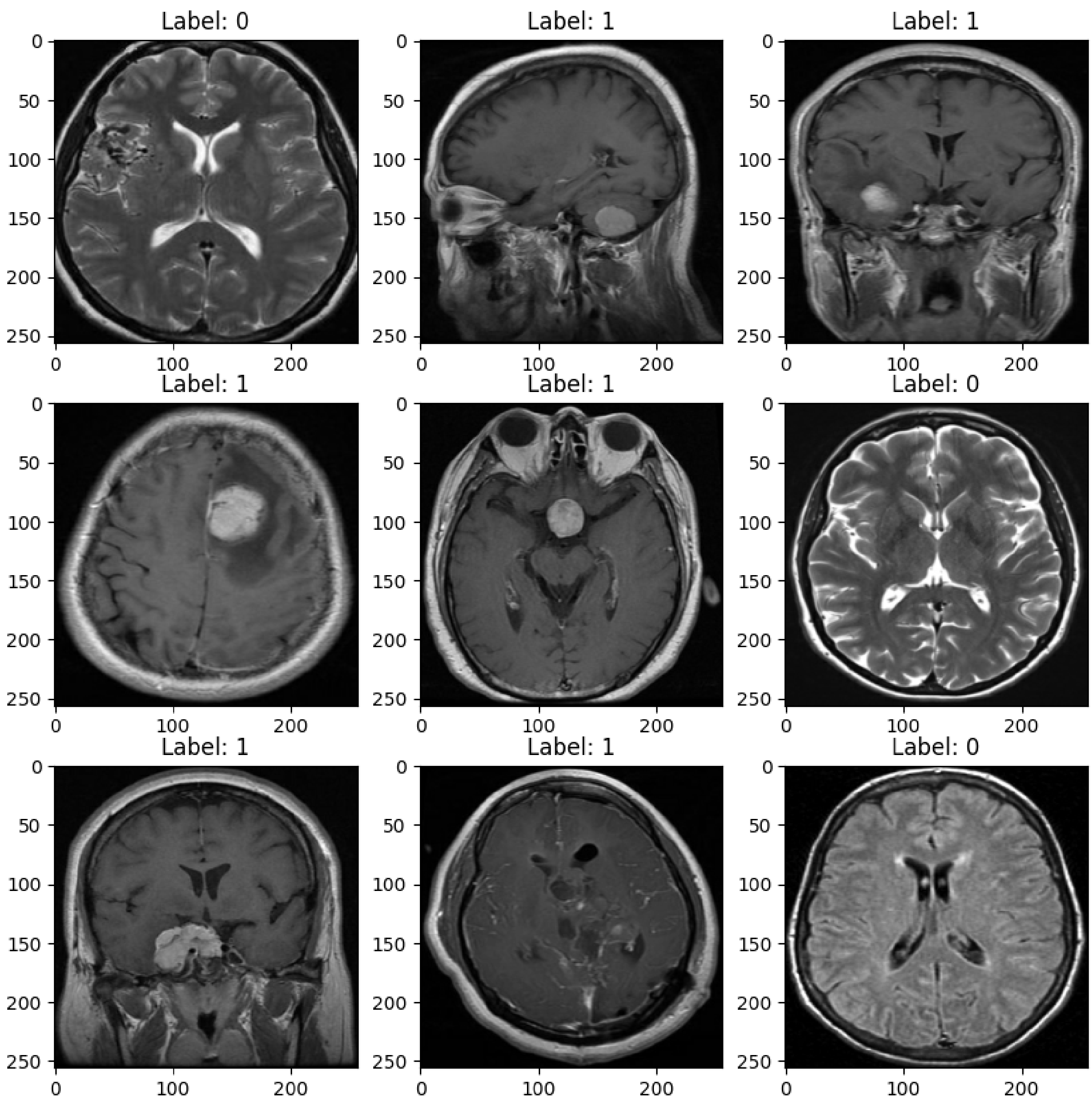
In [66]:
```
input_folder = "./MRI_folder/testing/no/"
output_folder = "./MRI_folder/test_aligned/no/"
crop_imgs(input_folder, output_folder, add_pixels_value = 5)
input_folder = "./MRI_folder/testing/yes/"
```

```python
output_folder = "./MRI_folder/test_aligned/yes/"
crop_imgs(input_folder, output_folder, add_pixels_value = 5)
```

In [67]:
```python
test = keras.utils.image_dataset_from_directory(
    "./MRI_folder/test_aligned",
    image_size=(256, 256),
    color_mode='grayscale',
    labels='inferred'
)

for image_batch, labels_batch in test.take(1):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image_batch[i].numpy(), cmap = 'gray')
        plt.title(f'Label: {labels_batch[i].numpy()}')
    plt.show()
```

```
Found 4245 files belonging to 2 classes.
```



There are 4245 files, 2000 negative and 2245 positive.

```
In [68]: CNN_model.evaluate(test)
```

```
133/133 [==============================] - 4s 27ms/step - loss: 0.9260 - accuracy:
0.8801
```

```
Out[68]: [0.9259923696517944, 0.8800942301750183]
```

Concluion

We reached an 88.01% accuracy with 0.9260 loss, which makes this model pretty valid usually. However, in a medical perspective, this isn't a acceptable accuracy. There are many ways to improve this model. Such as seperating all kinds of tumor and scanning angles for better classification; using higher quality scans for training; tweaking the CNN ti fit the characteristics of these scans more... This only serves as a presentation for the usage of CNN and should not be used in real life.

References

1. Chakrabarty, N. (2008). *Brain MRI Images for Brain Tumor Detection*. Kaggle [Image files]. Retrieved from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection/data

2. TensorFlow. (2023). *Tf.keras.utils.image_dataset_from_directory: tensorflow V2.14.0*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory

3. Nickparvar, M (2008). *Brain Tumor MRI Dataset.* Kaggle [Image files]. Retrieved from https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset