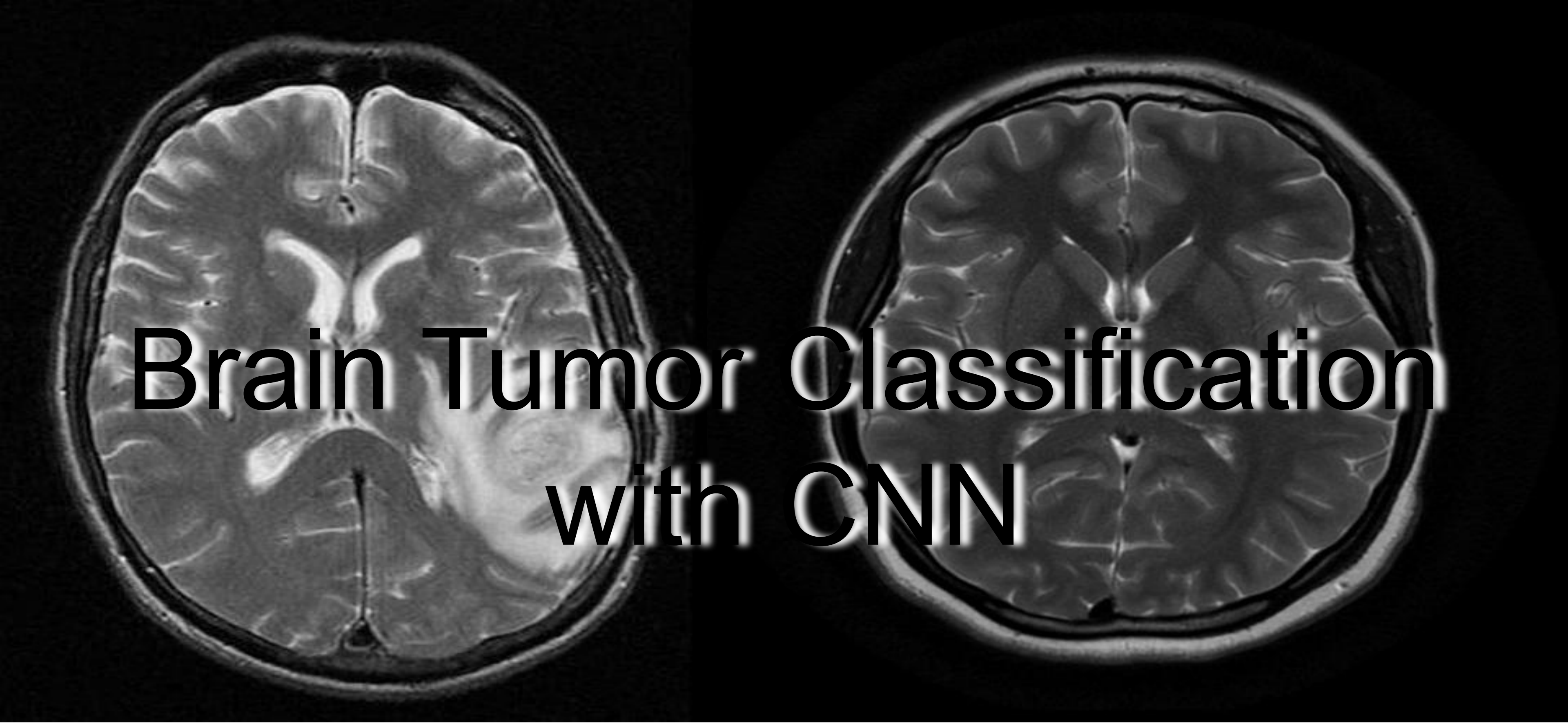

Brain Tumor Classification with CNN

Kimaya Havle, Sifat Naseem and Ifunanya Ezeumeh

SAT 5114 Final Project

Introduction

A brain tumor occurs when abnormal cells form within the brain. There are more than 100 distinct types of primary brain tumors.

There are two main types of tumors: cancerous (malignant) tumors and benign tumors. All types of brain tumors may produce symptoms that vary depending on the part of the brain involved.

These symptoms may include headaches, seizures, problems with vision, vomiting and mental changes.

Brain tumors can have lasting and life-altering physical, cognitive, and psychological impacts on a patient’s life.

Long Term Goal and Major Application

The long-term goal and major application of this project is to aid in the **early detection of brain tumors from MRI scans**.

Early detection will, in many cases, reduce the incidence of tumor progression from benign to malignant which would then lead to higher survival rates and better health outcomes.

Data Preprocessing

```
[26] random.shuffle(dataset)
```

```
TRAINING_SAMPLES_SIZE = 0.7
VALIDATION_SAMPLES_SIZE = 0.15
TESTING_SAMPLES_SIZE = 0.15
```

```
[28] if (TRAINING_SAMPLES_SIZE + VALIDATION_SAMPLES_SIZE + TESTING_SAMPLES_SIZE) > 100:
         raise ValueError
```

```
[29] total_dataset_size = len(dataset)

     training_start_index = 0
     training_end_index = training_start_index + math.floor(total_dataset_size * TRAINING_SAMPLES_SIZE)

     validation_start_index = training_end_index
     validation_end_index = validation_start_index + math.floor(total_dataset_size * VALIDATION_SAMPLES_SIZE)

     testing_start_index = validation_end_index
     testing_end_index = testing_start_index + math.floor(total_dataset_size * TESTING_SAMPLES_SIZE)
```

```
[30] training_dataset = dataset[training_start_index:training_end_index]
     validation_dataset = dataset[validation_start_index:validation_end_index]
     testing_dataset = dataset[testing_start_index:testing_end_index]
```

```
[31] def count_class_labels(dataset, dataset_type):
         yes_count = 0
         no_count = 0
         for data in dataset:
```

Data Preprocessing

```
def count_class_labels(dataset, dataset_type):
    yes_count = 0
    no_count = 0
    for data in dataset:
        label = data[1]
        if label == 'yes':
            yes_count = yes_count + 1
        else:
            no_count = no_count + 1
    print("Number of YES labels in the {0} dataset are {1}".format(dataset_type, yes_count))
    print("Number of NO labels in the {0} dataset are {1}".format(dataset_type, no_count))
    print("---")
```

```
[32] count_class_labels(training_dataset, "training")
     count_class_labels(validation_dataset, "validation")
     count_class_labels(testing_dataset, "testing")
```

```
Number of YES labels in the training dataset are 110
Number of NO labels in the training dataset are 67
---
Number of YES labels in the validation dataset are 24
Number of NO labels in the validation dataset are 13
---
Number of YES labels in the testing dataset are 19
Number of NO labels in the testing dataset are 18
---
```

```
[33] def reshape_image_array(image):
         '''
         Reshapes the image numpy array to make it four dimension since ImageDataGenerator requires a four dimensioned array
```

Data Preprocessing

[33]
```
def reshape_image_array(image):
    '''
    Reshapes the image numpy array to make it four dimension since ImageDataGenerator requires a four dimensioned array
    '''
    return image.reshape((1,) + image.shape)
```

[34]
```
def resize_image(image, target_image_size=TARGET_IMAGE_SIZE):
    resized = cv2.resize(image, dsize=TARGET_IMAGE_SIZE, interpolation=cv2.INTER_CUBIC)
    return resized
```

```
def split_yes_no_dataset(dataset):
    yes = []
    no = []
    for data in tqdm(dataset):
        label = data[1]
        image = data[0]
        resized = resize_image(image)
        reshaped = reshape_image_array(resized)
        if label == "yes":
            yes.append(reshaped)
        elif label == "no":
            no.append(reshaped)
    return yes, no
```

[36]
```
training_yes_dataset, training_no_dataset = split_yes_no_dataset(training_dataset)
```

```
100%|██████████| 177/177 [00:00<00:00, 852.57it/s]
```

[37]
```
validation_yes_dataset, validation_no_dataset = split_yes_no_dataset(validation_dataset)
```

Data Augmentation

In [ ]:

```
def augment_images(dataset, output_path):
    if not os.path.exists(output_path):
          os.makedirs(output_path)

    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,
        preprocessing_function=preprocess_input
    )
    image_count = 0
    for image in tqdm(dataset):
        image_count = image_count + 1
        generator = datagen.flow(
            image,
            save_to_dir=output_path
            )
        iteration = 0
        for batch in generator:
            iteration = iteration + 1
            if iteration == 5:
              break
```

In [ ]:

```
augment_images(dataset=training_yes_dataset, output_path=AUGMENTED_TRAIN_PATH_YES)
```

```
100%|██████████| 107/107 [00:23<00:00,  4.65it/s]
```

In [ ]:

```
augment_images(dataset=training_no_dataset, output_path=AUGMENTED_TRAIN_PATH_NO)
```

```
100%|██████████| 70/70 [00:14<00:00,  4.95it/s]
```

In [ ]:

```
augment_images(dataset=validation_yes_dataset, output_path=AUGMENTED_VALIDATION_PATH_YES)
```

```
100%|██████████| 25/25 [00:05<00:00,  4.83it/s]
```

In [ ]:

```
augment_images(dataset=validation_no_dataset, output_path=AUGMENTED_VALIDATION_PATH_NO)
```

```
100%|██████████| 12/12 [00:02<00:00,  5.32it/s]
```

In [ ]:

```
augment_images(dataset=testing_yes_dataset, output_path=AUGMENTED_TEST_PATH_YES)
```

CNN Architecture

In [39]:
```
KERNEL_SIZE = (2, 2)
STRIDES = (2, 2)
ACTIVATION_FUNCTION_RELU = 'relu'
PADDING_SAME = 'same'
DROPOUT = 0.25

model = Sequential()

# Block 1
model.add(Conv2D(32, kernel_size = KERNEL_SIZE, padding = PADDING_SAME, input_shape = (224, 224, 3)))
model.add(Conv2D(32, kernel_size = KERNEL_SIZE,  activation = ACTIVATION_FUNCTION_RELU, padding = PADDING_SAME))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(DROPOUT))

# Block 2
model.add(Conv2D(64, kernel_size = KERNEL_SIZE, activation = ACTIVATION_FUNCTION_RELU, padding = PADDING_SAME))
model.add(Conv2D(64, kernel_size = KERNEL_SIZE, activation = ACTIVATION_FUNCTION_RELU, padding = PADDING_SAME))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = KERNEL_SIZE, strides = STRIDES))
model.add(Dropout(DROPOUT))

# Block 3
model.add(Conv2D(128, kernel_size = KERNEL_SIZE, activation = ACTIVATION_FUNCTION_RELU, padding = PADDING_SAME))
model.add(Conv2D(128, kernel_size = KERNEL_SIZE, activation = ACTIVATION_FUNCTION_RELU, padding = PADDING_SAME))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = KERNEL_SIZE, strides = STRIDES))
model.add(Dropout(DROPOUT))

model.add(Flatten())

model.add(Dense(512, activation = ACTIVATION_FUNCTION_RELU))
model.add(Dropout(0.5))
model.add(Dense(2, activation = 'softmax'))

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

In [40]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

CNN Architecture

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      416

 conv2d_1 (Conv2D)           (None, 224, 224, 32)      4128

 batch_normalization (BatchN  (None, 224, 224, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 dropout (Dropout)           (None, 112, 112, 32)      0

 conv2d_2 (Conv2D)           (None, 112, 112, 64)      8256

 conv2d_3 (Conv2D)           (None, 112, 112, 64)      16448

 batch_normalization_1 (Batc  (None, 112, 112, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 56, 56, 64)        0

 conv2d_4 (Conv2D)           (None, 56, 56, 128)       32896

 conv2d_5 (Conv2D)           (None, 56, 56, 128)       65664

 batch_normalization_2 (Batc  (None, 56, 56, 128)      512
 hNormalization)
```

CNN Architecture

```
)

dropout (Dropout)           (None, 112, 112, 32)      0

conv2d_2 (Conv2D)           (None, 112, 112, 64)      8256

conv2d_3 (Conv2D)           (None, 112, 112, 64)      16448

batch_normalization_1 (Batc  (None, 112, 112, 64)     256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
2D)

dropout_1 (Dropout)         (None, 56, 56, 64)        0

conv2d_4 (Conv2D)           (None, 56, 56, 128)       32896

conv2d_5 (Conv2D)           (None, 56, 56, 128)       65664

batch_normalization_2 (Batc  (None, 56, 56, 128)      512
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
2D)

dropout_2 (Dropout)         (None, 28, 28, 128)       0

flatten (Flatten)           (None, 100352)            0

dense (Dense)               (None, 512)               51380736

dropout_3 (Dropout)         (None, 512)               0

dense_1 (Dense)             (None, 2)                 1026
```

CNN Only Result

```
 dropout_17 (Dropout)        (None, 56, 56, 64)        0

 conv2d_28 (Conv2D)          (None, 56, 56, 128)       32896

 conv2d_29 (Conv2D)          (None, 56, 56, 128)       65664

 batch_normalization_14 (Bat  (None, 56, 56, 128)      512
 chNormalization)

 max_pooling2d_14 (MaxPoolin  (None, 28, 28, 128)      0
 g2D)

 dropout_18 (Dropout)        (None, 28, 28, 128)       0

 flatten_4 (Flatten)         (None, 100352)            0

 dense_8 (Dense)             (None, 512)               51380736

 dropout_19 (Dropout)        (None, 512)               0

 dense_9 (Dense)             (None, 2)                 1026

=================================================================
Total params: 51,510,466
Trainable params: 51,510,018
Non-trainable params: 448
_________________________________________________________________
```

In [41]:

```
result = model.fit(X_train, y_train, epochs = 5, batch_size = 50, verbose = 1, validation_data = (X_validation, y_validation))
```

```
Epoch 1/5
18/18 [==============================] - 240s 13s/step - loss: 24.1879 - accuracy: 0.6377 - val_loss: 10.7073 - val_accuracy: 0.6793
Epoch 2/5
18/18 [==============================] - 238s 13s/step - loss: 5.3883 - accuracy: 0.6539 - val_loss: 4.6722 - val_accuracy: 0.7446
Epoch 3/5
18/18 [==============================] - 238s 13s/step - loss: 2.2838 - accuracy: 0.7118 - val_loss: 5.3780 - val_accuracy: 0.7011
Epoch 4/5
18/18 [==============================] - 237s 13s/step - loss: 0.8764 - accuracy: 0.7465 - val_loss: 3.8675 - val_accuracy: 0.6793
Epoch 5/5
18/18 [==============================] - 238s 13s/step - loss: 0.5018 - accuracy: 0.7616 - val_loss: 2.9331 - val_accuracy: 0.6848
```

In [42]:

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
6/6 [==============================] - 12s 2s/step - loss: 1.3595 - accuracy: 0.7814
```

Transfer Learning with VGG16

```python
# Set hyperparameters
img_width, img_height = 224, 224
batch_size = 32
num_epochs = 10
num_classes = 3
train_dir = '/content/augmented dataset/train'
val_dir = '/content/augmented dataset/validation'
test_dir = '/content/augmented dataset/test'

# Load pre-trained VGG16 model
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

# Add custom top layer
x = vgg_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Create the final model
model = Model(inputs=vgg_base.input, outputs=predictions)

# Freeze the pre-trained layers
for layer in vgg_base.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Preprocess the images using data generators
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')
val_generator = val_datagen.flow_from_directory(val_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')
test_generator = test_datagen.flow_from_directory(test_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')

# Train the model
model.fit(train_generator, steps_per_epoch=train_generator.samples // batch_size, epochs=num_epochs, validation_data=val_generator, validation_steps=val_generator.samples // batch_size
```

Transfer Learning Result

```python
# Preprocess the images using data generators
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')
val_generator = val_datagen.flow_from_directory(val_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')
test_generator = test_datagen.flow_from_directory(test_dir, target_size=(img_width, img_height), batch_size=batch_size, class_mode='categorical')

# Train the model
model.fit(train_generator, steps_per_epoch=train_generator.samples // batch_size, epochs=num_epochs, validation_data=val_generator, validation_steps=val_generator.samples // batch_size

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_generator, steps=test_generator.samples // batch_size)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

[5] Python

```
Found 864 images belonging to 3 classes.
Found 184 images belonging to 3 classes.
Found 183 images belonging to 3 classes.
Epoch 1/10
27/27 [==============================] - 13s 452ms/step - loss: 0.7439 - accuracy: 0.5741 - val_loss: 0.6470 - val_accuracy: 0.6938
Epoch 2/10
27/27 [==============================] - 12s 443ms/step - loss: 0.6170 - accuracy: 0.6713 - val_loss: 0.5851 - val_accuracy: 0.7312
Epoch 3/10
27/27 [==============================] - 12s 437ms/step - loss: 0.5699 - accuracy: 0.7488 - val_loss: 0.5223 - val_accuracy: 0.7750
Epoch 4/10
27/27 [==============================] - 12s 461ms/step - loss: 0.5421 - accuracy: 0.7535 - val_loss: 0.5184 - val_accuracy: 0.7437
Epoch 5/10
27/27 [==============================] - 13s 492ms/step - loss: 0.5304 - accuracy: 0.7627 - val_loss: 0.4801 - val_accuracy: 0.7625
Epoch 6/10
27/27 [==============================] - 12s 459ms/step - loss: 0.5161 - accuracy: 0.7650 - val_loss: 0.4877 - val_accuracy: 0.7500
Epoch 7/10
27/27 [==============================] - 12s 457ms/step - loss: 0.4862 - accuracy: 0.7824 - val_loss: 0.4819 - val_accuracy: 0.7750
Epoch 8/10
27/27 [==============================] - 12s 431ms/step - loss: 0.4611 - accuracy: 0.7986 - val_loss: 0.5533 - val_accuracy: 0.7625
Epoch 9/10
27/27 [==============================] - 12s 453ms/step - loss: 0.4413 - accuracy: 0.8299 - val_loss: 0.4841 - val_accuracy: 0.7375
Epoch 10/10
```

Conclusion

The model accuracy after training and testing dataset with the CNN is 78%. After applying transfer learning with VGG16, the model accuracy is 81%