Detecting Brain Tumors using Deep Learning

Image Classification

Joshua Furtado

6th May 2021

I. Definition

Project Overview

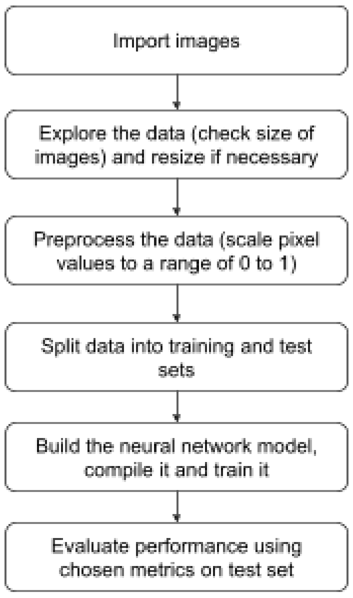
Over 700,000 Americans are living with a brain tumor today and studies show that more than 84,000 people will be diagnosed with a primary brain tumor in 2021[1]. Diagnosing a brain tumor usually begins with a Magnetic Resonance Imaging (MRI) scan. The results are then reviewed by a neurologist to see if there is a tumor in the brain.

Using artificial intelligence to detect a brain tumor from the MRI scan would save money and most importantly time. Not only that, this could also possibly reduce human error in the tumor detection. With today's ever-growing population, it is imperative that doctor's use technology to determine brain scan results in a timely fashion.

Problem Statement

The problem in hand is to accurately classify MRI scans of the brain to detect if a tumor is present or not. In this project, we build a machine learning model that can detect a tumor in a brain scan image. In order to do so, we need a dataset of brain images with and without tumor to solve this supervised machine learning problem. For the task in hand, we picked the Brain MRI Images for Brain Tumor Detection dataset from Kaggle.

The workflow to approach the problem by building a neural network model to classify brain scans is summarized in the flowchart below. Note, that the best parameters to configure the layers of the model will be obtained by hyperparameter tuning. Once we are satisfied with the neural network model, we will build a Support Vector Machine classifier to compare its performance.


Import images
Explore the data (check size of images) and resize if necessary
Preprocess the data (scale pixel values to a range of 0 to 1)
Split data into training and test sets
Build the neural network model, compile it and train it
Evaluate performance using chosen metrics on test set

Metrics

Accuracy will be the primary metric used to evaluate the models.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The secondary evaluation metric used will be recall. This is an important metric as it penalizes the false negative classifications. If a person has a tumor and our model predicts that the scan shows no tumor, that would be a terrible mistake!

$$Recall = \frac{TP}{TP + FN}$$

where TP: $True\ Positive,\ TN$: $True\ Negative,\ FP$: $False\ Positive,\ FN$: $False\ Negative$

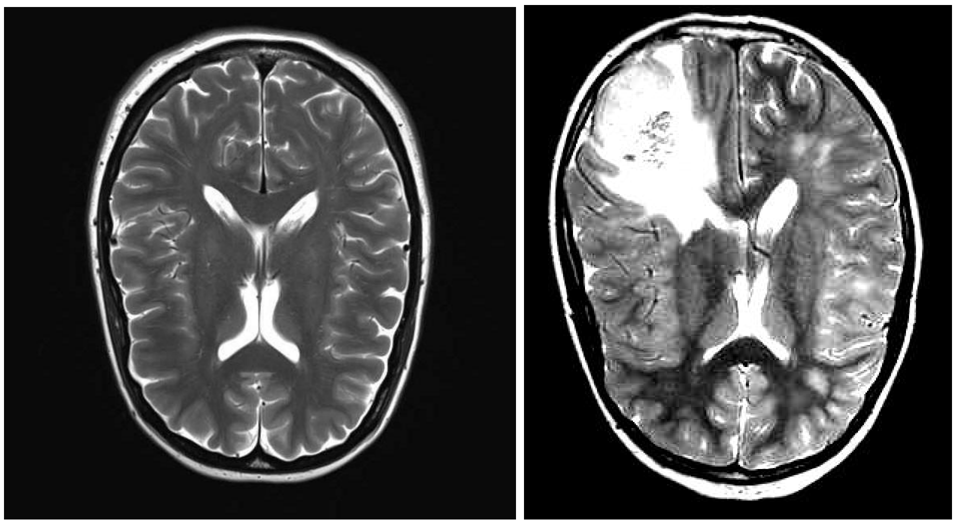
II. Analysis

Data Exploration

The Brain MRI Images for Brain Tumor Detection dataset from Kaggle contains 155 images of MRI scans with brain tumors and 98 without any tumor. So class imbalance is not an issue here. The brain tumor scans are in the folder labelled “yes” and healthy brain images are in the “no” folder. In total, there are 253 images. This is not a large sample size to use to train our model. We address this issue later in the data preprocessing section.

Exploratory Visualization

Below are sample scans showing a brain without tumor (left) and a brain with tumor(right).



All the images in the dataset are RGB images with each image of shape (image height, image width, 3). The R, G and B channels are identical for all images as they are grayscale images. Important point to note is that all images don't have the same dimensions and will have to be resized. Moreover, pixel values range from 0 to 255 and will have to be scaled before training the model.

Algorithms and Techniques

Convolution Neural Networks (CNNs) have become the state-of-the-art computer vision technique for image classification. Hence, CNN was opted as the algorithm to train for the task. We build a sequential neural network model from scratch consisting of three convolution blocks (16, 32 and 64 units resp.) with a max pool layer in each of them. These layers act as the feature extractors. The output of these layers after passing into a dropout layer is flattened and then passed through a fully connected layer with 128 units that is activated by a relu activation function. Finally, we have a dense layer with sigmoid activation that outputs the probability of detecting a tumor. Note that the input layer accepts images of default image height, image width and 3 channels. Hence, we preprocess the images by resizing and scaling them.

Benchmark

As a benchmark to compare the performance of the CNN model against, we use a Support Vector Machine (SVM) classifier. The shape of the features need to be tweaked a little before using SVM. The images of size (128, 128, 3) need to be flattened before fitting the SVM. The contributor to this dataset on Kaggle does not specify the data source and hence we will not be able to compare our model performance to a published benchmark.

III. Methodology

Data Preprocessing

Our dataset has 253 images in total. We use data augmentation to increase sample size. We start by defining a data augmentation layer which adds random flip, rotation and zoom to an image.

```
# define augmentation layer
data_augmentation = tf.keras.Sequential([
layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
     layers.experimental.preprocessing.RandomRotation(0.2),
     layers.experimental.preprocessing.RandomZoom(0.1)])
```

We then define an augment_image function that taken in an input image and returns a list of a specified number of augmented images.

```
def augment_image(image, n_augmented_images):
  '''
  Returns a list of augmented images for the given input image
  Arguments:
  image (array) - input image
  number_of_images (int) - number of augmented images to return
  Returns:
  images (list) - list of augmented images
  '''

  image = tf.expand_dims(image, 0)
  images = []

  for i in range(n_augmented_images):
    augmented_image = data_augmentation(image)
    images.append(np.array(augmented_image[0]))

  return images
```

We generate 12 augmented images for each image in our dataset, resulting in a total number of 3289 image samples. That's a lot more than we had in our original dataset. All images are grayscale, thus when imported will have the same value for R, G and B channels. We can use any one channel for our model. However, all images do not have the same dimensions and hence will have to be resized or padded before passing them into the neural network model. Additionally, we normalize images by scaling pixel values ranging from 0 to 255 to a range of 0 to 1.

The input data images will be used to train the model to classify MRI scan images as having a tumor or not. The images in this case will be the features for the algorithms and a label, either 0 or 1 will be used to classify those with no tumor and tumor respectively. We defined a preprocess_data function that takes in the path where the images are stored and a desired shape to resize the images to. After reading, resizing and scaling the images, it stores them in an array X (features) and their corresponding labels in an array y (labels). The augment_image function defined earlier is called for each image in the preprocess_data function.

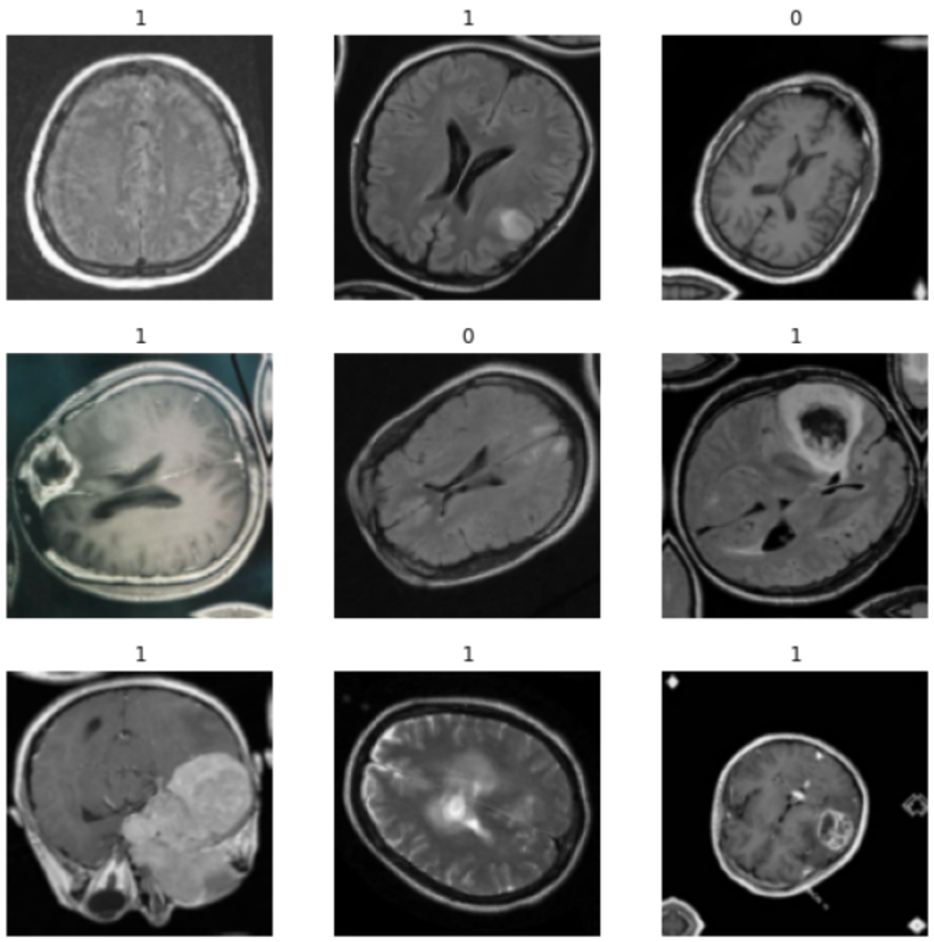
```
def preprocess_data(path, img_size, n_augmented_images):
  '''
  Reads in images classified into folders, resizes and scales them. Returns
  those processed images as features and their associated labels as well.
  Arguments:
    path (str) - path to classified image folders
    img_size (tuple) - tuple containing resized image height and width
  Returns:
    X (array) - features (brain scan images)
    y (array) - feature labels (0 - no tumor, 1 - tumor)
  '''

  unsuccessful_files = {}

  X = []
  y = []

  for folder_name in os.listdir(path):
    if folder_name == 'no':
      label = 0
    else:
      label = 1
    folder_path = os.path.join(path, folder_name)

    for fname in os.listdir(folder_path):
      fpath = os.path.join(folder_path, fname)
      try:
        img = cv2.imread(fpath)
        img = cv2.resize(img, img_size)
        img = img / 255.0
        X.append(img)
        y.append(label)
        X += augment_image(img, n_augmented_images)
        y += [label] * n_augmented_images

      except Exception as e:
        unsuccessful_files[fname] = e

  if unsuccessful_files:
    print(f'Error processing the following files:\n')
    for index, key in enumerate(unsuccessful_files, 1):
      print(f'{index}. {key} - {unsuccessful_files[key]}')
  else:
    print('Successfully processed all images.')

  X = np.array(X)
  y = np.array(y)

  return X, y
```

We use sklearn's train test split to split the data into training (75%), validation (12.5%) and test (12.5%) sets.

```
# split data into train, validation and test sets
from sklearn.model_selection import train_test_split
```

```
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=0.25,
random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5,
random_state=42)
```

Visualizing the first 9 images in our training set, we can see a blend of the original and augmented images.



Implementation

We build and train a Convolutional Neural Network (CNN) model to classify MRI brain scans as having a tumor or not. We build a sequential model consisting of three convolution blocks (16, 32 and 64 units resp.) with a max pool layer in each of them. These layers act as the feature extractors. The output of these layers after passing into a dropout layer is flattened and then passed through a fully connected layer with 128 units that is activated by a relu activation function. Finally, we have a dense layer with sigmoid activation that outputs the probability of detecting a tumor.

```
# create the model
model = Sequential([
  layers.Input((img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(1, activation='sigmoid')
])
```

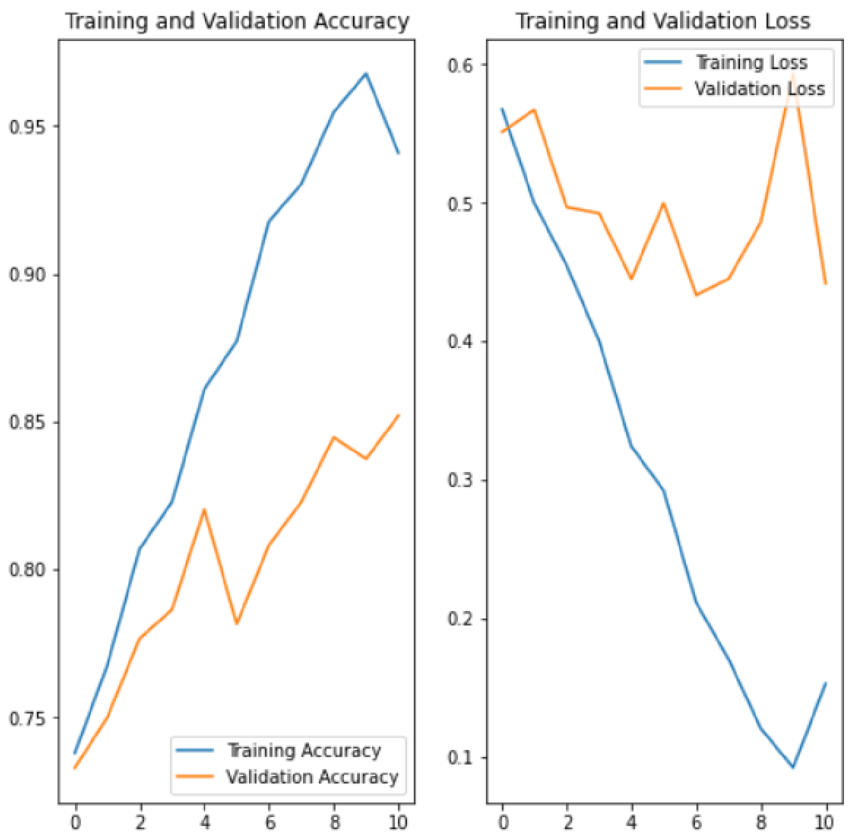
We then compile the model using adam optimizer, binary cross entropy loss and accuracy as metric.

```
# compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

Finally, we use batch size of 32 and 20 epochs to train the model. We add an early stopping callback to prevent overfitting the data. Additionally, the dropout layer in our sequential model should help with this as well.

```
# train the model
early_stop  =  tf.keras.callbacks.EarlyStopping(monitor='val_loss',  mode='min',  verbose=1,
patience=4)
history = model.fit(
  X_train,
  y_train,
  batch_size=32,
  validation_data=(X_val, y_val),
  epochs=20,
  callbacks=[early_stop]
)
```

Plotting training and validation accuracy as well as loss, we can clearly see the effect of early stopping in the figure below.



Refinement

To refine the model, we optimize some of the model parameters. We try to improve upon the base CNN model by evaluating a range of values in the hyperparameter space using cross validation. For this, we define a model with a range of different parameter values.

```
# create the model
def model_builder(hp):
  model = Sequential([
    layers.Input((img_height, img_width, 3)),
    layers.Conv2D(hp.Int('conv2d_1_units', min_value=8, max_value=16, step=8), 3,
padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(hp.Int('conv2d_2_units', min_value=16, max_value=32, step=16), 3,
padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(hp.Int('conv2d_3_units', min_value=32, max_value=64, step=32), 3,
padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(hp.Float('dropout', 0.1, 0.3, step=0.1)),
    layers.Flatten(),
    layers.Dense(hp.Int('dense_units', min_value=64, max_value=128, step=64),
activation='relu'),
    layers.Dense(1, activation='sigmoid')
  ])

  # compile the model
  model.compile(optimizer='adam',
                loss=tf.keras.losses.BinaryCrossentropy(),
                metrics=['accuracy'])
  return model
```

Then, a Keras hyperband tuner instance is defined.

```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=20,
                     factor=3,
                     directory=LOG_DIR)
```

And a hyperparameter search is done.

```
# perform hyperparameter search
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=4)
tuner.search(X_train,
             y_train,
             batch_size=32,
             validation_data=(X_val, y_val),
             epochs=20,
             callbacks=[early_stop])
```

We obtained the following best hyperparameters:

conv2d_1_units	16
conv2d_2_units	32
conv2d_3_units	64
dense_units	64
dropout	0.1

IV. Results

Model Evaluation and Validation

We test the model by making predictions on our test set, which has not been seen by the model before. Note that the model outputs a probability from 0 to 1, hence we round this value to obtain a classified label for each of the predictions.

```
# make predictions on the test set
y_pred = model.predict(X_test)
y_pred = np.squeeze(y_pred).round().astype(int)

# classification report
from sklearn.metrics import classification_report , confusion_matrix
print(classification_report(y_test, y_pred))
```

Testing our first trained model, without using data augmentation gave us 75% accuracy and 75% weighted average recall. This is mainly due to the fact that we do not have enough samples in our dataset. Our dataset has 253 images in total.

```
              precision    recall  f1-score   support

           0       0.75      0.64      0.69        14
           1       0.75      0.83      0.79        18

    accuracy                           0.75        32
   macro avg       0.75      0.74      0.74        32
weighted avg       0.75      0.75      0.75        32
```

We use data augmentation to increase sample size. We generate 12 augmented images for each image in our dataset, resulting in a total number of 3289 image samples.Testing the CNN trained with an augmented dataset, we get an accuracy of 86% and weighted average recall of 86%. That's a 15% increase in model performance.

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.85       173
           1       0.91      0.85      0.88       238

    accuracy                           0.86       411
   macro avg       0.86      0.87      0.86       411
weighted avg       0.87      0.86      0.86       411
```

After building, training and testing the model using the best hyperparameters, we obtained 91% accuracy and 91% recall. That's ~6% improvement compared to our previous model.

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.89       173
           1       0.91      0.94      0.93       238

    accuracy                           0.91       411
   macro avg       0.91      0.91      0.91       411
weighted avg       0.91      0.91      0.91       411
```

Justification

We compare our model performance to a Support Vector Machine (SVM) classifier as benchmark. We use the augmented dataset over here. The shape of the features are tweaked a little before using SVM. The images of size (128, 128, 3) are flattened before fitting the SVM. For the SVM classifier, we use sklearn's built-in SVC model. We create a SVM classifier and train it.

```
from sklearn import svm

# Create a classifier: a support vector classifier
clf = svm.SVC(gamma=0.001)

# Learn the digits on the train subset
clf.fit(X_train, y_train)
```

Testing the SVM classifies on our test set resulted in an accuracy and weighted average recall of 79%.

```
              precision    recall  f1-score   support

           0       0.83      0.64      0.73       352
           1       0.77      0.90      0.83       471

    accuracy                           0.79       823
   macro avg       0.80      0.77      0.78       823
weighted avg       0.80      0.79      0.79       823
```

Our tuned CNN model with 91% classification accuracy and 91% recall was 15% better than the SVC classifier used as a benchmark. If we compare the SVC classifier to the original CNN model prior to tuning, the original CNN's accuracy and recall was ~9% better than the SVC classifier.

V. Conclusion

Reflection

To summarize our work, we started with the original dataset and trained a CNN model to classify brain scans with and without tumors. We quickly realised that the model performance wasn't the greatest. This was attributed to the small sample size of the original dataset. We then used image augmentation to generate augmented images for each image in the original dataset resulting in 3289 samples as compared to the 253 images we had in our original dataset.

Without data augmentation		With data augmentation	
Accuracy	Recall	Accuracy	Recall
0.75	0.75	0.86	0.86

We then optimized our model performance further using hyperparameter tuning to find the best hyperparameters.

Without hyperparameter tuning		With hyperparameter tuning	
Accuracy	Recall	Accuracy	Recall
0.86	0.86	0.91	0.91

Finally, we built and trained a SVM classifier as a benchmark. Since we did not use grid search to tune the SVM classifier's hyperparameters, we will compare its performance to our CNN model without hyperparameter tuning. Note that data augmentation was used in both cases.

SVC		CNN	
Accuracy	Recall	Accuracy	Recall
0.79	0.79	0.86	0.86

Improvement

Although we did get a good accuracy and recall of 91% with our tuned CNN model, there's room for improvement. Here are a few suggestions to improve performance:

1. Crop augmented images to remove artifacts we don't want. We can use OpenCV to find the largest contour and crop the rest.
2. Use a better dataset with more brain scan images so that we don't have to rely heavily on data augmentation.

3. We only ran a search to find the optimal number of units in each layer. Obtain optimal number of epochs, batch size and number of convolution layers as well using hyperparameter tuning.

The code for this work can be found here.

References

1. Porter KR, McCarthy BJ, Freels S,Kim Y, Davis FG. Prevalence estimates for primary brain tumors in the United States by age, gender, behavior, and histology. Neuro-Oncology 12(6):520-527, 2010.