Image Classification for Brain Tumors

1. Introduction

Applying artificial intelligence (AI) to healthcare-related tasks, such as medical scan analysis, presents a possibility for accelerating disease detection, all while reducing the workload of healthcare professionals and minimizing human error in image interpretation. Given the abundance of medical imaging data, it is reasonable to apply machine learning (ML) algorithms to automate medical scan analysis.

The goal of this project is to detect abnormalities, in this case tumors, on MRI brain images. This project addresses a binary classification problem, where the goal is to classify images into two groups: healthy and tumor. A classification model can determine whether an MRI brain image belongs to a healthy individual or a patient with a tumor.

This report consists of five sections. In the problem formulation section, the machine learning classification problem is introduced, and data points, features and labels are explained. The methods section introduces the data source attributes and the preprosessing done on the data. That section also explains the two ML methods and loss function used in this project and the rationale behind choosing them. The results section presents the final results of the two ML models, compares them, and selects the better model. Finally, the conclusions section comprises the key insights and thoughts.

2. Problem Formulation

In this project, the task is to identify and distinguish MRI brain scans with tumors from those without, by using ML algorithms that can accurately classify images based on pixel intensity patterns. The dataset consists of MRI scans, which show either a healthy brain or one with a tumor. The dataset was obtained from Kaggle, and the owner of the data is Rajarshi Mandal. The original data source divided the tumors into three subtypes: glioma, meningioma, and pituitary, but for this project the subtypes are examined without distinction. Thus, this project is a binary classification problem.

Each MRI image represents a single data point. Since this project deals with image data, the features are naturally selected as pixel values. The features for each data point are the resized and rescaled pixels. The binary labels for the data points are '0', which represents a healthy brain and '1', which represents a brain with a tumor.

3. Methods

The dataset consists of 2000 images of healthy brains and 5023 images of tumors. The images are black and white, and the grayscale pixel intensities represent features. Features are numerical data ($0-255$), which are normalized between $0-1$. The pixels are normalized using TensorFlow so that they better fit the neural network model.

As the other ML method for this classification task, I selected convolutional neural network (CNN). I chose CNN because it is regarded as one of the most effective and commonly used architectures for image classification tasks (Chen *et al.*, 2021). CNN is a supervised machine learning method that, in this project, predicts the probability of an image belonging to each class. The model assigns probabilities to both classes, with one class receiving a higher probability based on the model's prediction.

For the second ML method, I selected support vector classification (SVC). SVC is a supervised ML algorithm for classification tasks. It was selected because it is well-regarded for its effectiveness in image classification tasks, especially when dealing with high-dimensional data (Thai, Hai and Thuy, 2012).

10% of the dataset was selected for testing. The testing data points consist of 702 images, which were moved to a separate directory. The remaining dataset was split into 80% for training and 20% for validation. The training set consists of 5056 data points and the validation set consists of 1265 data points. The training and validation set sizes are based on the recommendation in TensorFlow documentation.

For the loss function for CNN, I selected binary cross entropy (BCE). BCE is a typically used loss function for binary classification, and TensorFlow includes predefined functions for it. It measures the difference between the predicted probability p and the actual binary label y (0 or 1).

$$\text{BCE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{1}$$

For the loss function for SVC, I selected hinge loss. Hinge loss was selected because it is the standard loss function used in SVMs, including SVC, to maximize the margin between classes. Hinge loss works by penalizing misclassified examples and those that lie within the margin, which encourages the model to find a decision boundary that maximizes the distance between the closest points of the different classes (the support vectors).

$$\text{Hinge Loss} = \sum_{i=1}^{n}\max(0, 1 - y_i \cdot f(x_i))$$

4. Results

In this project, both CNNs and SVMs were applied to the brain MRI classification task, with the goal of differentiating between healthy brain images and those containing a tumor. The performance of each model was evaluated using training, validation, and test datasets, with a particular focus on comparing the accuracy and loss for both methods.

For the CNN method, the training accuracy after the tenth epoch reached nearly perfect levels, rounding up to 1.0 (100%). This suggests that the model was highly effective at learning the patterns within the training dataset. The validation accuracy of the CNN, though slightly lower, was still very high at 0.9881, which indicates that the model generalized well to the validation set. The test accuracy for CNN, which measures performance on a the separate set of test MRI images, was 0.9843 (98.43%).

In comparison, the SVC model achieved slightly lower training accuracy of 98.93% but still demonstrated strong performance. The validation accuracy for SVC was 96.52%, which is lower than that of CNN but still represents good generalization. The SVC model's test accuracy of 97.58% was slightly lower than the CNN model's, which achieved 98.43%.

When comparing the training and validation losses for both models, the CNN model demonstrated strong performance, with a training loss of 2.2261×10^{-4}, indicating that the model had almost no difficulty in fitting the training data. The validation loss for the CNN model was 0.1341, while the test loss was 0.1351. The SVC model had a higher training loss of 0.0432, but lower validation and test losses compared to CNN: 0.1030 and 0.0902, respectively. These metrics show that while SVC did not achieve as high training accuracy as CNN, it performed very efficiently on validation and test data, and thus, may have benefited from greater resilience to overfitting.

Ultimately, the final chosen method for this project was the CNN method. CNNs are generally better suited for image classification tasks due to their capacity to capture spatial hierarchies in images. They also have the advantage of being able to learn and extract features directly from raw pixel data without requiring manual feature engineering, making them a more scalable solution for future work involving larger and more complex datasets. Moreover, the difference in test accuracy between the two models was minimal (less than 1%), while the CNN model demonstrated higher training and validation accuracies. Thus, the CNN model was selected, with a test accuracy of 98.43% and a test loss of 0.1351. This performance indicates that the model is highly effective at classifying brain MRI images.

5. Conclusions

This report explored the use of ML methods to address a binary image classification problem that attempts to detect brain tumors from MRI scans. The main objective was to classify MRI images into two categories: healthy brains and brains with tumors. To accomplish this, two machine learning models, CNN and SVC, were implemented and evaluated using a dataset containing 7023 MRI images. The CNN model was chosen as the final model due to its superior performance and suitability for image-based tasks.

The results of the models show that both of them performed well on the classification task. The high levels of test accuracies indicate that the models can effectively distinguish between healthy and tumor-afflicted brains based on MRI images. However, CNN was ultimately selected due to its ability to automatically extract relevant features from the images, a capability that gives it an advantage in scalability and future applications.

Although the results suggest that the problem is solved satisfactorily, there is still room for improvement. The models, particularly CNN, showed signs of overfitting, as evidenced by the near-perfect training accuracy (100%) compared to the slightly lower validation accuracy. This indicates that the model may have learned the training data too well, possibly at the expense of generalization to unseen data. Further tuning of hyperparameters, the inclusion of regularization techniques, and training on a more diverse dataset may help to minimize overfitting.

References

Chen, L. *et al.* (2021). Review of Image Classification Algorithms Based on Convolutional Neural Networks. *Remote Sensing.* 13. doi: https://doi.org/10.3390/rs13224712

Thai, L.H., Hai, T.S. and Thuy, N.T. (2012). Image Classification using Support Vector Machine and Artificial Neural Network. *Information Technology and Computer Science.* 5. pp. 32-38. doi: https://doi.org/10.5815/ijitcs.2012.05.05

Data source: https://www.kaggle.com/datasets/rm1000/brain-tumor-mri-scans

Hige loss: https://www.geeksforgeeks.org/hinge-loss-relationship-with-support-vector-machines/

SVC: https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/

Tensorflow: https://www.tensorflow.org/tutorials/images/classification

Appendix

1. Convolutional Neural Network (CNN) model

```python
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras.layers as layers
from tensorflow.keras.models import Sequential


data_dir = 'archive'    # training and validation data directory
test_data_dir = 'test'  # test data directory

# Setting values for image and batch size
# These are standard values for many models
img_height, img_width = 256, 256
batch_size = 32

# Loading training data
train_dataset = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

# Loading validation data
val_dataset = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

# Loading test data
test_dataset = tf.keras.utils.image_dataset_from_directory(
  test_data_dir,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle=False
)

# Creating the sequential model
model = Sequential([
  # Rescaling pixel values and reshaping the image
  layers.Rescaling(1./255, input_shape=(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(),
  # Flattening into 1D
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  # Output layer with sigmoid activation
  layers.Dense(1, activation='sigmoid')
])

# Compiling using Adam optimizer and binary crossentropy for loss
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

# Training the model using 10 epochs
epochs=10
history = model.fit(
  train_dataset,
  validation_data=val_dataset,
  epochs=epochs
)


# Plotting:

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(epochs), acc, label='Training Accuracy')
plt.plot(range(epochs), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(epochs), loss, label='Training Loss')
plt.plot(range(epochs), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

predictions = model.predict(test_dataset)
predicted_classes = (predictions > 0.5).astype("int32")

test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")


```

```
Test Loss: 0.13505567610263824

Test Accuracy: 0.9843304753303528
```

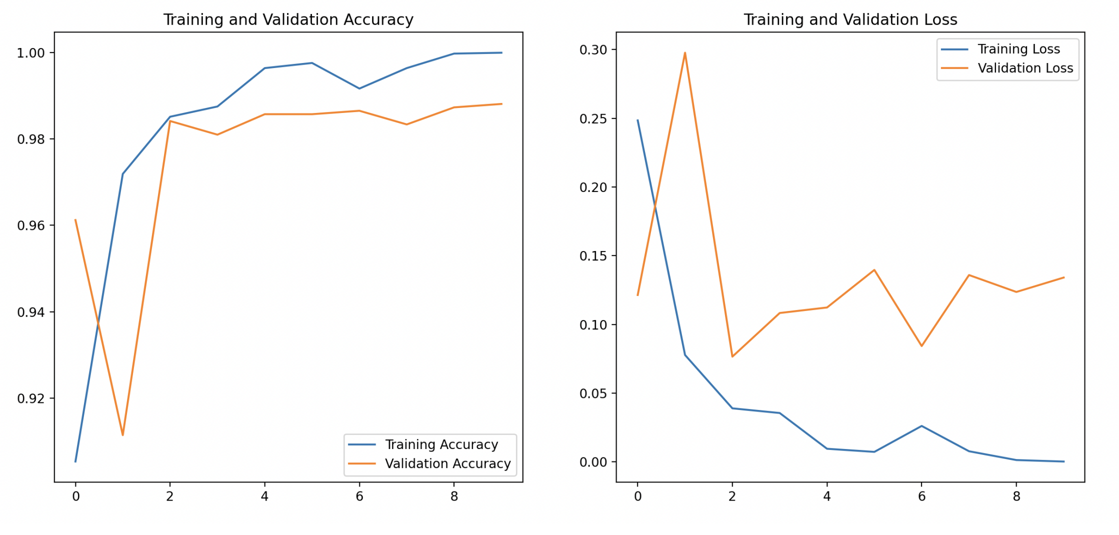

Figure 1: Training and validation accuracies and losses in the 10 epochs.

2. Support Vector Classification (SVC)

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from skimage.transform import resize
from skimage.io import imread
from sklearn.metrics import accuracy_score
from sklearn.metrics import hinge_loss


data_dir = 'archive'    # training and validation data directory
test_data_dir = 'test' # test data directory

# Setting image size and class names
img_height, img_width = 128, 128
classes = ['healthy','tumor']

# Function for loading data from a directory
def load_data(data_dir, classes, img_height, img_width):
    data_arr = []    # stores image data
    target_arr = [] # stores labels
    valid_extensions = ['.jpg', '.jpeg', '.png']

    # looping through classes ('healthy', 'tumor')
    for i in classes:
        path = os.path.join(data_dir, i)
        # looping through each image in the class
        for img in os.listdir(path):
            img_path = os.path.join(path, img)
            if os.path.splitext(img)[-1].lower() in valid_extensions:
                img_array = imread(img_path)
                img_resized = resize(img_array, (img_width, img_height))
                data_arr.append(img_resized.flatten())
                # Append index of the class (0: 'healthy', 1: 'tumor')
                target_arr.append(classes.index(i))

    # Converting to numpy arrays
    data = np.array(data_arr)
    target = np.array(target_arr)

    return data, target


# Load training and test data:
train_data, train_labels = load_data(data_dir,
                                     classes,
                                     img_height,
                                     img_width)
test_data, test_labels = load_data(test_data_dir,
                                   classes,
                                   img_height,
                                   img_width)



# Splitting the training data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(train_data,
                                                  train_labels,
                                                  test_size=0.2,
                                                  random_state=77,
                                                  stratify=train_labels)



# Initializing the SVC model
svc = SVC(kernel='rbf', class_weight='balanced')


# Training the model
svc.fit(x_train, y_train)

# Predicting labels for the training set
y_train_pred = svc.predict(x_train)

# Calculating training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")

y_train_decision = svc.decision_function(x_train)
y_train_mod = np.where(y_train == 1, 1, -1)
train_hinge_loss = hinge_loss(y_train_mod, y_train_decision)
print(f"Training Hinge Loss: {train_hinge_loss:.4f}")

# Predicting labels for the validation set
y_val_pred = svc.predict(x_val)
```

```python


# Calculating validation accuracy
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")


y_val_decision = svc.decision_function(x_val)
y_val_mod = np.where(y_val == 1, 1, -1)
val_hinge_loss = hinge_loss(y_val_mod, y_val_decision)
print(f"Validation Hinge Loss: {val_hinge_loss:.4f}")

# Predicting labels for the test set
y_test_pred = svc.predict(test_data)

# Calculating test accuracy
test_accuracy = accuracy_score(test_labels, y_test_pred)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")


y_test_decision = svc.decision_function(test_data)
y_test_mod = np.where(test_labels == 1, 1, -1)
test_hinge_loss = hinge_loss(y_test_mod, y_test_decision)
print(f"Test Hinge Loss: {test_hinge_loss:.4f}")




```

```
Training Accuracy: 98.93%

Training Hinge Loss: 0.0432

Validation Accuracy: 96.52%

Validation Hinge Loss: 0.1030

Test Accuracy: 97.58%

Test Hinge Loss: 0.0902
```