By Dharun Kumar.

Pneumonia Detection Using U-Net for Medical Image Segmentation

Introduction

Pneumonia is a life-threatening respiratory condition that demands swift and accurate diagnosis. Machine learning, particularly semantic segmentation using U-Net architecture, offers a powerful tool for enhancing pneumonia detection in medical X-ray images. This project leverages advanced image analysis techniques to automatically segment and classify regions of interest, ultimately contributing to early and precise pneumonia diagnosis. The impact of this project is two-fold: it aids healthcare professionals in making faster and more accurate clinical decisions, potentially saving lives, while also reducing the burden on radiologists, thereby improving the efficiency of healthcare delivery.

```
import glob
import os
import cv2
from tensorflow.keras import metrics
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, Flatten, Dense
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Data collection and preprocessing

Dataset link - https://www.bing.com/ck/a?!&&p=111f89f21a8bb446JmltdHM9MTY5NTYwMDAwMCZpZ3VpZD0wNTEyMGQxMy1iNjM4LTYxZDEtMzFkMi0xZTg0YjdmYjYwNmImaW5zaWQ9NTI0OQ&ptn=3&hsh=3&fclid=05120d13-b638-61d1-31d2-1e84b7fb606b&psq=chest+x+ray+datset&u=a1aHR0cHM6Ly93d3cua2FnZ2xlLmNvbS9kYXRhc2V0cy9wYXVsdGltb3RoeW1vb25leS9jaGVzdC14cmF5LXBuZXVtb25pYQ&ntb=1

Data Augmentation

```
# Define the paths to your train, test, and validation directories
train_dir = '/content/drive/MyDrive/chest x-ray images/train'
test_dir = '/content/drive/MyDrive/chest x-ray images/test'
valid_dir = '/content/drive/MyDrive/chest x-ray images/val'

# Use an ImageDataGenerator to load and augment the data
datagen = ImageDataGenerator(rescale=1./255,   # Rescale pixel values to [0, 1]
                             shear_range=0.2,   # Shear transformations
                             zoom_range=0.2,    # Zoom transformations
                             horizontal_flip=True)  # Horizontal flips

# Load training data from directory and apply data augmentation
train_generator = datagen.flow_from_directory(train_dir,
                                              target_size=(224, 224),  # Adjust target size as needed
                                              batch_size=32,
                                              class_mode='categorical')  # Use 'binary' for binary classification

# Load validation data without data augmentation
valid_generator = datagen.flow_from_directory(valid_dir,
                                              target_size=(224, 224),
                                              batch_size=32,
                                              class_mode='categorical')

# Load testing data without data augmentation
test_generator = datagen.flow_from_directory(test_dir,
```

```
                                                target_size=(224, 224),
                                                batch_size=32,
                                                class_mode='categorical')

# Get class labels
class_labels = list(train_generator.class_indices.keys())

# Load the dataset into NumPy arrays
X_train, y_train = train_generator.next()
X_valid, y_valid = valid_generator.next()
X_test, y_test = test_generator.next()
```

```
Found 13768 images belonging to 2 classes.
Found 1057 images belonging to 2 classes.
Found 431 images belonging to 2 classes.
```

```
# Read the image
image = cv2.imread('/content/drive/MyDrive/chest x-ray images/test/normal/NORMAL2-IM-0331-0001.jpeg')

# Get the height and width of the image
shape = image.shape

# Print the size
print(f"image shape - {shape}")
```

```
image shape - (1064, 1674, 3)
```

Model selection and training

U-NET

Unet architecture: U-Net is a convolutional neural network architecture commonly used for semantic segmentation tasks, including medical image segmentation.

Input Layer: The architecture begins with an input layer (Input(shape=input_shape)), which defines the shape of the input images.

Encoder: The encoder part of the U-Net consists of multiple convolutional layers with increasing numbers of filters. In your code, two convolutional layers with 64 filters each (Conv2D(64, 3, activation='relu', padding='same')) are applied sequentially, followed by max-pooling (MaxPooling2D) to downsample the feature maps.

Middle: After the encoder, there is a middle section with two convolutional layers with 256 filters each (Conv2D(256, 3, activation='relu', padding='same')). This section captures high-level features.

Decoder: The decoder part of the U-Net consists of upsampling layers (UpSampling2D) followed by concatenation with feature maps from the corresponding encoder layers. This process helps to recover spatial information lost during downsampling. Two convolutional layers with 128 filters each (Conv2D(128, 3, activation='relu', padding='same')) are applied in the decoder.

Flatten and Classification: Finally, the output from the decoder is flattened (Flatten()) and connected to a dense layer with 2 units and a sigmoid activation function (Dense(2, activation='sigmoid')). This last layer is responsible for binary classification, where 2 represents the number of classes (in this case, likely representing pneumonia and non-pneumonia).

```
def unet(input_shape):
    inputs = Input(shape=input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    # Middle
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)

    # Decoder
    up1 = UpSampling2D(size=(2, 2))(conv3)
    up1 = concatenate([up1, conv2], axis=-1)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(up1)
    conv4 = Conv2D(128, 3, activation='relu', padding='same')(conv4)

    up2 = UpSampling2D(size=(2, 2))(conv4)
```

```
    up2 = concatenate([up2, conv1], axis=-1)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(up2)
    conv5 = Conv2D(64, 3, activation='relu', padding='same')(conv5)

    # Flatten the output and add a Dense layer for classification
    flatten = Flatten()(conv5)
    outputs = Dense(2, activation='sigmoid')(flatten)

    model = Model(inputs=inputs, outputs=outputs)
    return model

# Create the U-Net model with a specific input shape
model = unet(input_shape=(224,224,3))

# Model summary
model.summary()

# Model compilation
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        metrics.BinaryAccuracy(),  # Binary accuracy
        metrics.Precision(),       # Precision
        metrics.Recall(),          # Recall
        metrics.AUC(),             # AUC-ROC
    ]
)

model.summary()
# Assuming you have X_train and y_train for training data and labels
model.fit(X_train, y_train, epochs=12, batch_size=32, validation_split=0.2)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928

 max_pooling2d (MaxPooling2  (None, 112, 112, 64)      0
 D)

 conv2d_2 (Conv2D)           (None, 112, 112, 128)     73856

 conv2d_3 (Conv2D)           (None, 112, 112, 128)     147584

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 128)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 56, 56, 256)       295168

 conv2d_5 (Conv2D)           (None, 56, 56, 256)       590080

 up_sampling2d (UpSampling2  (None, 112, 112, 256)     0
 D)

 concatenate (Concatenate)   (None, 112, 112, 384)     0

 conv2d_6 (Conv2D)           (None, 112, 112, 128)     442496

 conv2d_7 (Conv2D)           (None, 112, 112, 128)     147584

 up_sampling2d_1 (UpSamplin  (None, 224, 224, 128)     0
 g2D)

 concatenate_1 (Concatenate  (None, 224, 224, 192)     0
 )

 conv2d_8 (Conv2D)           (None, 224, 224, 64)      110656

 conv2d_9 (Conv2D)           (None, 224, 224, 64)      36928

 flatten (Flatten)           (None, 3211264)           0

 dense (Dense)               (None, 2)                 6422530

=================================================================
Total params: 8305602 (31.68 MB)
Trainable params: 8305602 (31.68 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "model"
```

```
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

# Evaluate the model on the test dataset
test_loss = model.evaluate(X_test, y_test, verbose=1)

# Make predictions on the test dataset
predictions = model.predict(X_test)

# Threshold the predictions (e.g., using 0.5 as a threshold for binary segmentation)
threshold = 0.5
binary_predictions = (predictions > threshold).astype(np.uint8)

# Calculate confusion matrix and classification report
conf_matrix = confusion_matrix(y_test.ravel(), binary_predictions.ravel())
classification_rep = classification_report(y_test.ravel(), binary_predictions.ravel())

# Display confusion matrix and classification report
print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", classification_rep)

# Visualize segmentation results on sample test images
num_samples_to_visualize = 5  # Number of samples to visualize
sample_indices = np.random.choice(len(X_test), num_samples_to_visualize, replace=False)
```

```
1/1 [==============================] - 29s 29s/step - loss: 0.4112 - binary_accuracy: 0.8750 - precision: 0.8750 - recall: 0.8750 -
1/1 [==============================] - 31s 31s/step
Confusion Matrix:
 [[28  4]
 [ 4 28]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88        32
         1.0       0.88      0.88      0.88        32

    accuracy                           0.88        64
   macro avg       0.88      0.88      0.88        64
weighted avg       0.88      0.88      0.88        64
```

Conclusion

In conclusion, this project has demonstrated the potential of machine learning, specifically semantic segmentation using the U-Net architecture, in the critical domain of pneumonia detection from medical X-ray images. By automating the segmentation and classification of regions of interest, we have improved the accuracy and efficiency of pneumonia diagnosis, benefiting both patients and healthcare providers.

The developed model showcases promising results, but it's important to acknowledge its limitations and consider further improvements. However, this project is a significant step toward leveraging machine learning for enhanced healthcare solutions.

Future improvements

The future scope of this project is promising and includes the following directions for advancement:

Large-Scale Data: Expanding the dataset to include a broader range of pneumonia cases and diverse patient demographics can improve model generalization and robustness.

Multi-Class Segmentation: Extending the model to handle multiple classes of lung abnormalities, not just pneumonia, can provide a more comprehensive diagnostic tool.

Transfer Learning: Leveraging pre-trained models, particularly on medical imaging datasets, can boost model performance and reduce the need for extensive labeled data.

Real-Time Diagnosis: Implementing the model in a real-time or near-real-time diagnostic system for use in clinical settings.

Clinical Validation: Collaborating with medical professionals for thorough clinical validation and obtaining regulatory approvals for clinical deployment.

Enhanced User Interface: Developing a user-friendly interface for healthcare practitioners to interact with the model's results and improve the interpretability of its predictions.

Ensemble Learning: Combining multiple segmentation and classification models to enhance overall accuracy and reliability.

References

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv preprint arXiv:1505.04597.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical image analysis, 42, 60-88.

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Langlotz, C. (2018). CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning. arXiv preprint arXiv:1711.05225.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115-118.

Johnson, A. E., Pollard, T. J., Berkowitz, S. J., Greenbaum, N. R., Lungren, M. P., Deng, C. Y., ... & Mark, R. G. (2020). MIMIC-CXR: A large publicly available database of labeled chest radiographs. arXiv preprint arXiv:1901.07042.