Prostate Classification

Introduction

In this document, I will provide a comprehensive overview of the methodology employed for prostate classification. The objective of the prostate classification challenge is to tackle a binary classification problem, specifically discerning between low-level and high-level risks in prostate categorization.

To address this challenge, I have chosen to implement the 3D wavelet transform method. This technique is utilized to extract essential approximation coefficients, which play a crucial role in capturing relevant features for effective classification. Subsequently, these extracted coefficients serve as the input data for training a sophisticated 3D convolutional neural network (CNN) model.

Dataset

The organizers provide image dataset and clinical dataset:

- Image dataset: T2w MRI → Harmonization to guarantee unification in the image contrast across scans.
- Clinical dataset: Patient age (age) and total prostate specific antigen level (PSA) at diagnosis (psa)
- Labels: String with the risk of the patient, two values “Low” or “High”.

In this dataset, we have 210 cases at a low risk and 85 cases at a high risk.

Problematic

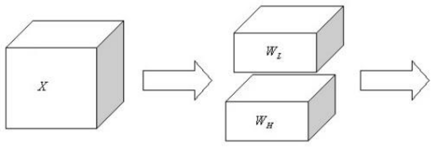
In the scenario involving 210 low-risk patients and 85 high-risk patients, we observed class imbalance. This imbalance may impact the training of classification models, leading the model to be biased towards a larger number of instances. To address this, I employ an ensemble analysis: randomly selecting 90 low-risk cases and 80 high-risk cases for training a model. This process is repeated three times, where 5 low-risk and 5 high-risk cases are combined to create a validation dataset. Three predictions are generated by the models, and the mean prediction serves as the final result.

Pre-processing

The first step is always to implement a preprocessing method. Given that our CT image volumes have different depths, after normalizing the data by mean and standard deviation values, we resize all the image volumes to the same dimensions: (32, 256, 256) with a length of 256, height of 256, and depth of 32.

Feature extraction

After the preprocessing process, I utilize the 3D wavelet transform method to extract the approximation coefficients.



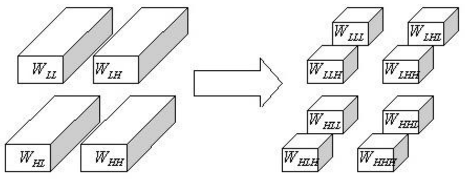


Fig. 1 The resolution of a 3-D signal is reduced in each dimension

The approximation coefficients capture the essential structural information of the signal or image, while discarding fine details. This is valuable in scenarios where the main trends and features of the data are more critical than specific details. For example, in medical imaging, the general shape and structure of an organ might be more relevant than individual pixel-level details.

3D convolutional neural network

For binary classification, we use a 3d CNN model and the architecture of the 3D CNN:

```
def get_model(depth=16, height=128, width=128):

    """Build a 3D convolutional neural network model."""

    inputs = keras.Input((depth, height,width, 1))

    x = layers.Conv3D(filters=16, kernel_size=3, 
activation="relu")(inputs)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv3D(filters=32, kernel_size=3, activation="relu")(x)
    x = layers.MaxPool3D(pool_size=2)(x)
    x = layers.BatchNormalization()(x)

    x = layers.GlobalAveragePooling3D()(x)
    x = layers.Dense(units=256, activation="relu")(x)
    x = layers.Dropout(0.4)(x)
    x = layers.Dense(units=128, activation="relu")(x)
    x = layers.Dropout(0.4)(x)



    outputs = layers.Dense(units=1, activation="sigmoid")(x)

    # Define the model.
    model = keras.Model(inputs, outputs, name="3dcnn")
    return model
```

Train model

The parameters for train the model shown below, I employ an ensemble analysis: randomly selecting 90 low-risk cases and 80 high-risk cases for training a model. This process is repeated three times, where 5 low-risk and 5 high-risk cases are combined to create a validation dataset. Three predictions are generated by the models, and the mean prediction serves as the final result.

```
   # Compile model.
    initial_learning_rate = 0.0001
    lr_schedule = keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate, decay_steps=100000, decay_rate=0.96,
staircase=True
            )

    model = get_model(depth=16, height=128, width=128)
    model.compile(
            loss="binary_crossentropy",
            optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
            metrics=["acc"],
                  )
# Define callbacks.
    checkpoint_cb = keras.callbacks.ModelCheckpoint(
            "Wavelet_3d_cnn_version1.h5", save_best_only=True
    )
    early_stopping_cb =
keras.callbacks.EarlyStopping(monitor="val_loss", patience=15)
    epochs = 100
    model.fit(
            train_dataset,
            validation_data=validation_dataset,
            epochs=epochs,
            batch_size = 32,
            shuffle=True,
            verbose=1,
            callbacks=[checkpoint_cb, early_stopping_cb])
```