Project Documentation: Deep Learning Model for Radio Burst Classification

1. Importing Modules

This section imports the necessary Python libraries and modules for the project. These include:

- Libraries for data manipulation (numpy, pandas), image processing (cv2), and visualization (matplotlib, seaborn).
- Machine learning tools (sklearn, tensorflow, tensorflow_hub).
- Utilities for multi-threading and progress tracking (ThreadPoolExecutor, tqdm).

2. Constants Definition

Defines key constants for the project:

- DATA_PATH: Directory containing the dataset.
- IMG_SHAPE: Image dimensions for resizing.
- TEST_SPLIT: Fraction of data reserved for testing.
- NUM_CLASSES: Number of classes in the dataset.
- SEED: Seed for reproducibility.

3. Utility Functions

read_image(full_img_path)

Reads an image from a file path, converting it to RGB format. Handles errors during file reading.

read_data(path, verbose=False, shape=IMG_SHAPE)

Reads and processes all images from the dataset, categorizing them by labels (folder names). Uses multi-threading for faster processing.

preprocess_spectrogram(data)

Applies preprocessing to spectrogram data by subtracting the median value along each axis to normalize the images.

4. Load and Preprocess the Dataset

- Reads images and their corresponding labels using read_data.
- Preprocesses the data using preprocess_spectrogram.
- Visualizes sample spectrograms from the dataset to understand their distribution.

5. Data Preparation

- Splits the data into training and testing sets.
- Encodes class labels into numeric format.
- Normalizes the image data to a range of 0 to 1.

6. Deep Learning Models

6.1 Custom CNN Model

create_cnn_model(input_shape, num_classes)

Defines a Convolutional Neural Network (CNN) with:

Three convolutional layers with ReLU activation, followed by max-pooling and dropout.
A fully connected dense layer, followed by dropout.
An output layer with a softmax activation for classification.

Training the Model: Uses the Adam optimizer and sparse categorical cross-entropy loss. The model checkpoint callback saves the best-performing model during training.

Evaluation: Prints the training and testing accuracies. Saves the model weights for later use.

6.2 Pretrained Models

Pretrained Models Setup

- Defines a dictionary of pretrained models (VGG16, DenseNet121, MobileNet, MobileNetV2) from TensorFlow's Keras applications.

create_pretrained_model(model_class, input_shape, num_classes)

- Creates a transfer learning model using a pretrained base (frozen layers) and a custom classification head.

Cross-Validation

- Performs K-Fold Cross-Validation (5 folds) to evaluate each pretrained model.
- Saves the cross-validation results to CSV files.

6.3 Ensemble Methods

Soft Voting

- Combines predictions from pretrained models using the average of their predicted probabilities.
- Evaluates the ensemble on the test set and displays a confusion matrix.

Stacking

- Uses predictions from pretrained models as input features for a meta-model (logistic regression).
- Trains the meta-model and evaluates its performance.

6.4 Vision Transformers (ViT)

FlexibleViTModel

Implements a Vision Transformer (ViT) using TensorFlow Hub.

Includes a resizing layer, a ViT feature extractor, and a dense classification head.

Training: Uses early stopping to avoid overfitting and trains the model on the preprocessed dataset.

Evaluation: Reports train and test accuracies, and saves the model weights.

6.5 Swin Transformer

SwinTransformerModel

Implements a Swin Transformer using TensorFlow Hub.

Includes a resizing layer, a Swin feature extractor, and a dense classification head.

Training: Uses early stopping and evaluates the model on the preprocessed dataset.

Evaluation: Reports train and test accuracies, and saves the model weights.

6.6 ConvNext

ConvNextModel

Implements a ConvNext architecture using TensorFlow Hub.

Includes a resizing layer, a ConvNext feature extractor, and a dense classification head.

Training: Uses early stopping and evaluates the model on the preprocessed dataset.

Evaluation: Reports train and test accuracies, and saves the model weights.

7. Results Comparison

- Compares the performance of all models (CNN, Ensemble, Stacking, Vision Transformers, Swin, ConvNext).
- Saves the results to a CSV file for further analysis.

8. Conclusion

This project demonstrates the application of deep learning techniques, including custom CNNs, pretrained models, and state-of-the-art architectures like Vision Transformers, for classifying radio burst spectrograms. Results from various methods are compared to identify the best-performing model.