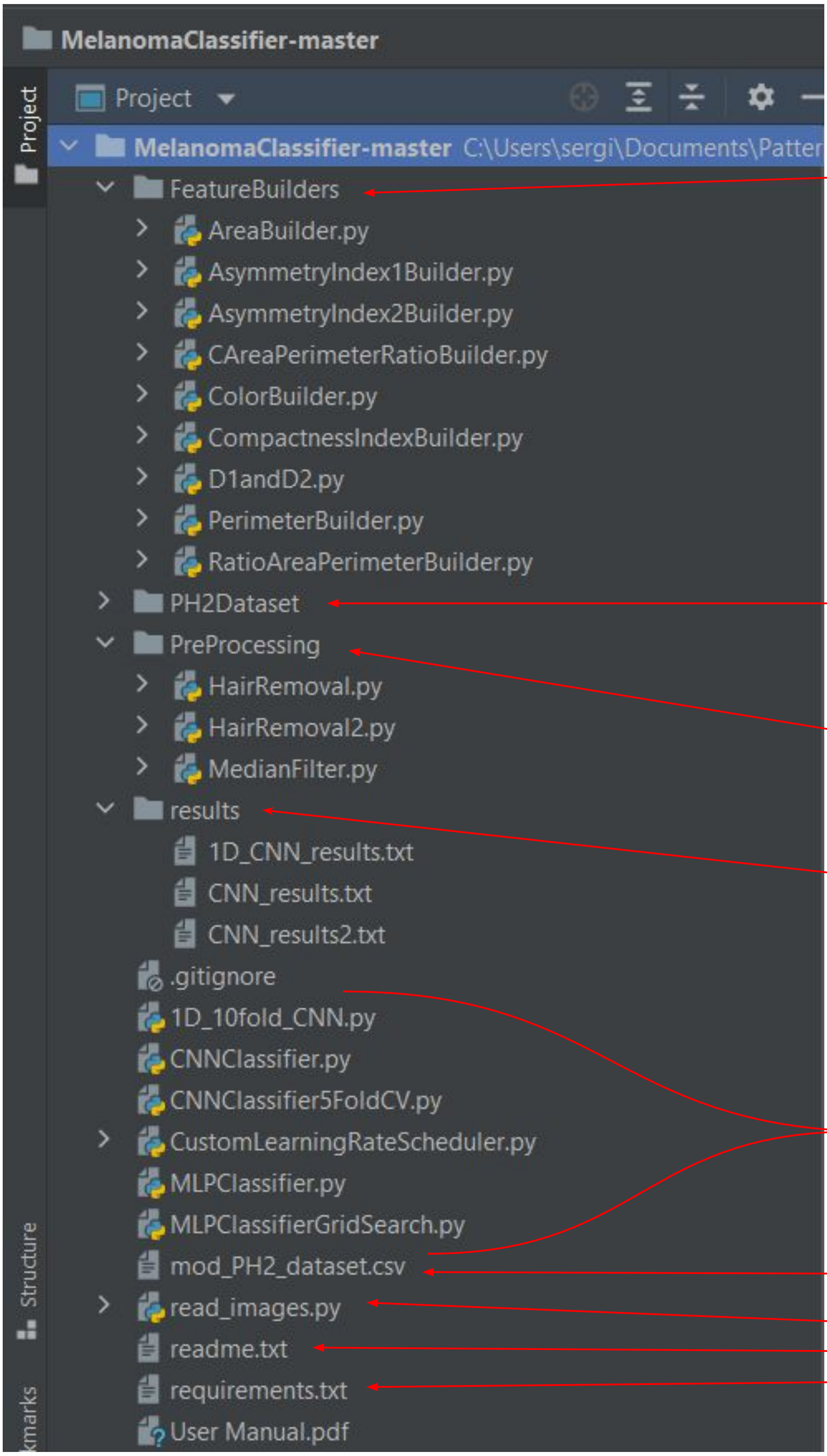
Structure of the Program



```
MelanomaClassifier-master
Project
MelanomaClassifier-master  C:\Users\sergi\Documents\Patter
    FeatureBuilders
        AreaBuilder.py
        AsymmetryIndex1Builder.py
        AsymmetryIndex2Builder.py
        CAreaPerimeterRatioBuilder.py
        ColorBuilder.py
        CompactnessIndexBuilder.py
        D1andD2.py
        PerimeterBuilder.py
        RatioAreaPerimeterBuilder.py
    PH2Dataset
    PreProcessing
        HairRemoval.py
        HairRemoval2.py
        MedianFilter.py
    results
        1D_CNN_results.txt
        CNN_results.txt
        CNN_results2.txt
    .gitignore
    1D_10fold_CNN.py
    CNNClassifier.py
    CNNClassifier5FoldCV.py
    CustomLearningRateScheduler.py
    MLPClassifier.py
    MLPClassifierGridSearch.py
    mod_PH2_dataset.csv
    read_images.py
    readme.txt
    requirements.txt
    User Manual.pdf
```

Folder containing all the files that compute and extract the features

Raw dataset

Folder that contains the different preprocessing steps

Loss, Accuracy, Recall of our models

Different models, old versions, final versions and experimental models

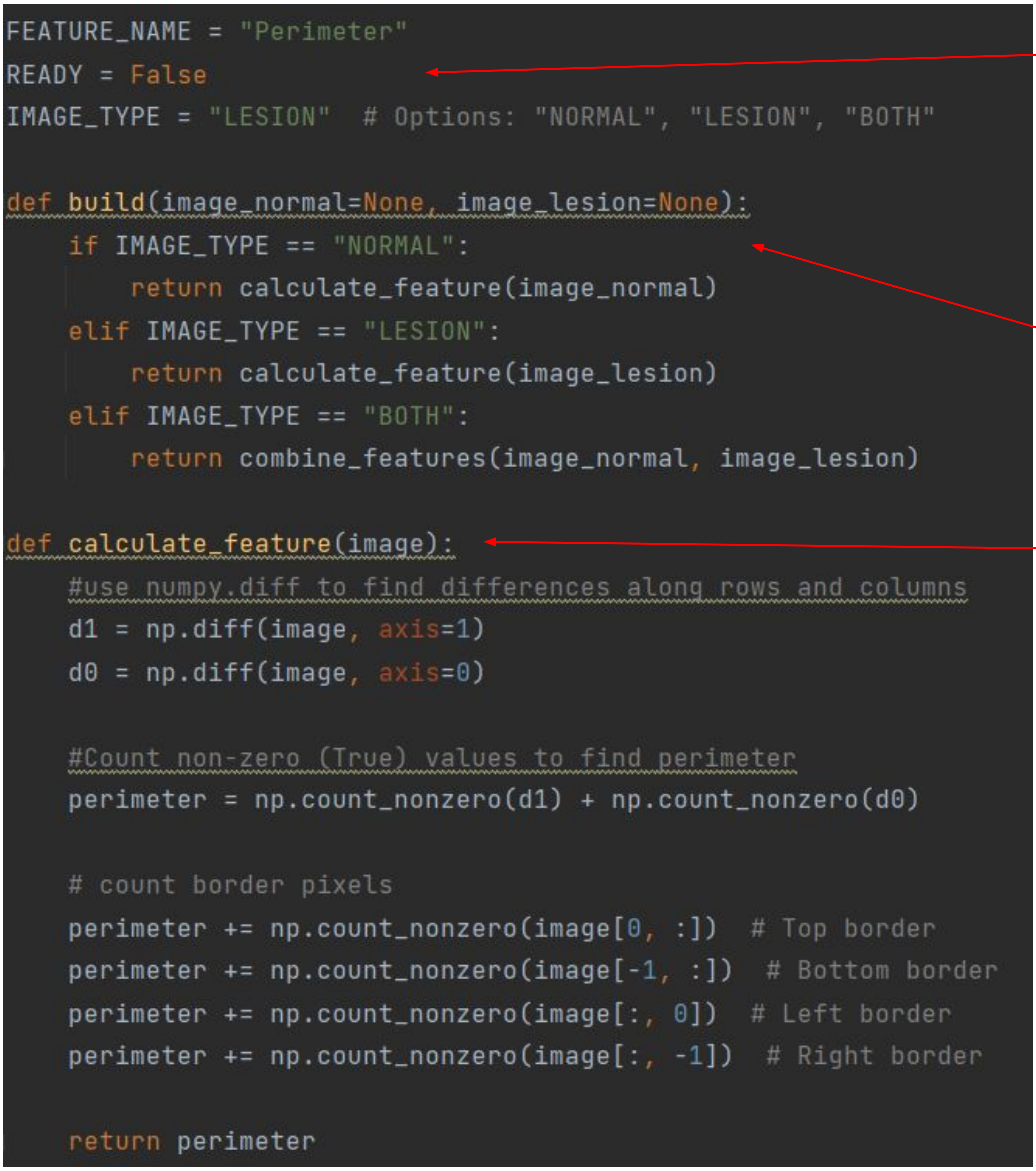
Used dataset (only labels and names)

Image Loader class

Instructions

Libraries

Structure of a Builder



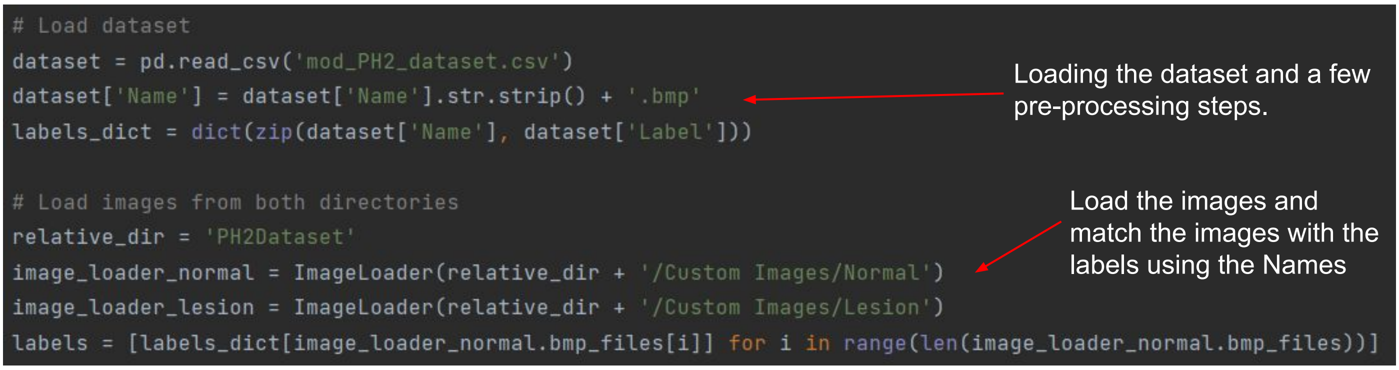
```python
FEATURE_NAME = "Perimeter"
READY = False
IMAGE_TYPE = "LESION"  # Options: "NORMAL", "LESION", "BOTH"

def build(image_normal=None, image_lesion=None):
    if IMAGE_TYPE == "NORMAL":
        return calculate_feature(image_normal)
    elif IMAGE_TYPE == "LESION":
        return calculate_feature(image_lesion)
    elif IMAGE_TYPE == "BOTH":
        return combine_features(image_normal, image_lesion)

def calculate_feature(image):
    #use numpy.diff to find differences along rows and columns
    d1 = np.diff(image, axis=1)
    d0 = np.diff(image, axis=0)

    #Count non-zero (True) values to find perimeter
    perimeter = np.count_nonzero(d1) + np.count_nonzero(d0)

    # count border pixels
    perimeter += np.count_nonzero(image[0, :])  # Top border
    perimeter += np.count_nonzero(image[-1, :])  # Bottom border
    perimeter += np.count_nonzero(image[:, 0])  # Left border
    perimeter += np.count_nonzero(image[:, -1])  # Right border

    return perimeter
```

Attributes of the Builder: Name, Ready (True if you want to include as a feature for the models) and Image_Type (kind of image that will receive as input)

build function that select the image input

Function that extracts and computes the specific feature of the Builder. This is different in every builder

First steps of every model

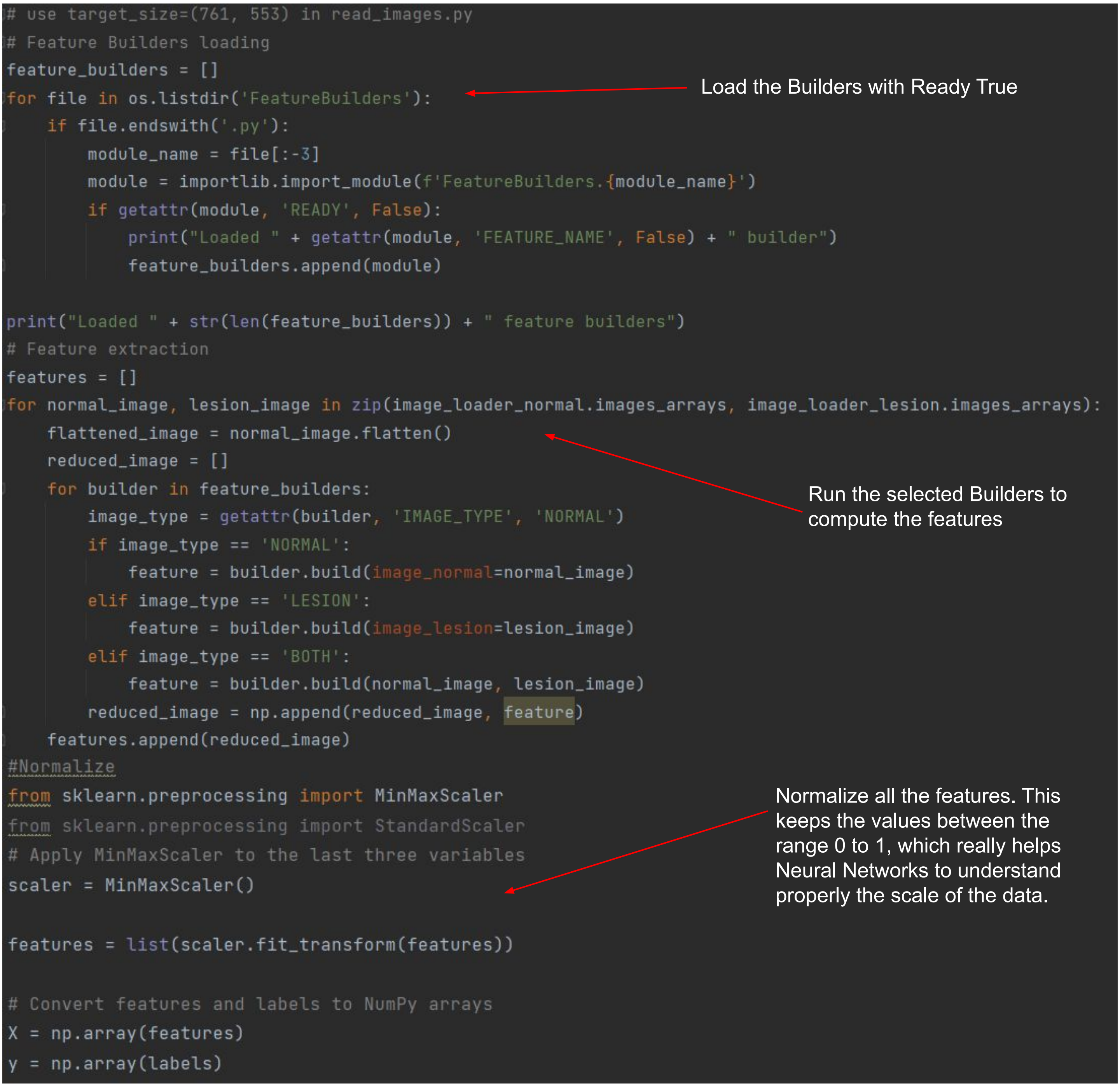


```
# Load dataset
dataset = pd.read_csv('mod_PH2_dataset.csv')
dataset['Name'] = dataset['Name'].str.strip() + '.bmp'
labels_dict = dict(zip(dataset['Name'], dataset['Label']))

# Load images from both directories
relative_dir = 'PH2Dataset'
image_loader_normal = ImageLoader(relative_dir + '/Custom Images/Normal')
image_loader_lesion = ImageLoader(relative_dir + '/Custom Images/Lesion')
labels = [labels_dict[image_loader_normal.bmp_files[i]] for i in range(len(image_loader_normal.bmp_files))]
```

Loading the dataset and a few pre-processing steps.

Load the images and match the images with the labels using the Names

Next steps of models that use features



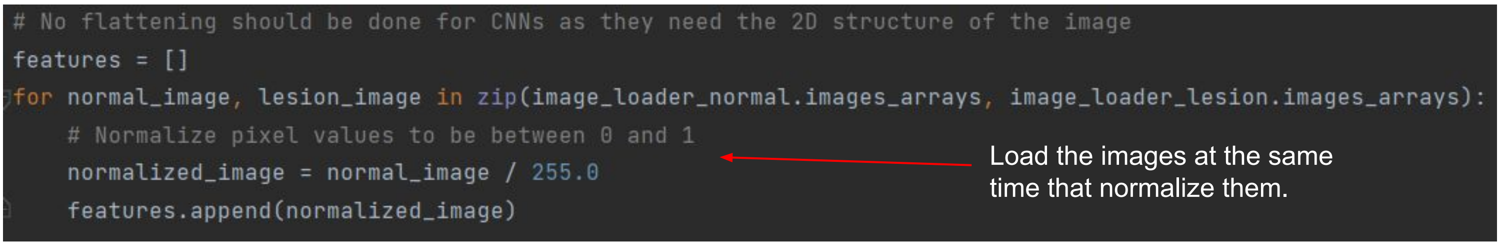
```
# use target_size=(761, 553) in read_images.py
# Feature Builders loading
feature_builders = []
for file in os.listdir('FeatureBuilders'):
    if file.endswith('.py'):
        module_name = file[:-3]
        module = importlib.import_module(f'FeatureBuilders.{module_name}')
        if getattr(module, 'READY', False):
            print("Loaded " + getattr(module, 'FEATURE_NAME', False) + " builder")
            feature_builders.append(module)

print("Loaded " + str(len(feature_builders)) + " feature builders")
# Feature extraction
features = []
for normal_image, lesion_image in zip(image_loader_normal.images_arrays, image_loader_lesion.images_arrays):
    flattened_image = normal_image.flatten()
    reduced_image = []
    for builder in feature_builders:
        image_type = getattr(builder, 'IMAGE_TYPE', 'NORMAL')
        if image_type == 'NORMAL':
            feature = builder.build(image_normal=normal_image)
        elif image_type == 'LESION':
            feature = builder.build(image_lesion=lesion_image)
        elif image_type == 'BOTH':
            feature = builder.build(normal_image, lesion_image)
        reduced_image = np.append(reduced_image, feature)
    features.append(reduced_image)
#Normalize
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# Apply MinMaxScaler to the last three variables
scaler = MinMaxScaler()

features = list(scaler.fit_transform(features))

# Convert features and labels to NumPy arrays
X = np.array(features)
y = np.array(labels)
```

Load the Builders with Ready True

Run the selected Builders to compute the features

Normalize all the features. This keeps the values between the range 0 to 1, which really helps Neural Networks to understand properly the scale of the data.

Next steps of models that use the images



```
# No flattening should be done for CNNs as they need the 2D structure of the image
features = []
for normal_image, lesion_image in zip(image_loader_normal.images_arrays, image_loader_lesion.images_arrays):
    # Normalize pixel values to be between 0 and 1
    normalized_image = normal_image / 255.0
    features.append(normalized_image)
```

Load the images at the same
time that normalize them.

Model’s steps

After the first steps, each model: MLP, MLP gridSearch, 1D and 2D CNN, execute their individual code lines.

Outcome of the Program

While a model is running, the epochs are printed and their metrics too. Once a model is finished, some insights are printed, such as metrics like loss, accuracy and recall. In some cases, additional plots are showed: ROC curves or confusion matrices.