**SRGAN ASSIGNMENT**

**Introduction**

This assignment will be focused on using the concept of transfer learning on a pre-trained CNN to build a dog/cat classifier. The document is the manual of the work done on then SRGAN. The system project can be run on the collab. The more you train the more picture is clear.

**STEPS.**

The code defines two custom PyTorch Datasets, mydata and testOnly\_data, as well as two data augmentation classes, crop and augmentation. These datasets and transformations are often used in deep learning projects for image processing, such as super-resolution tasks.

Here is the breakdown of the codemydata Dataset:

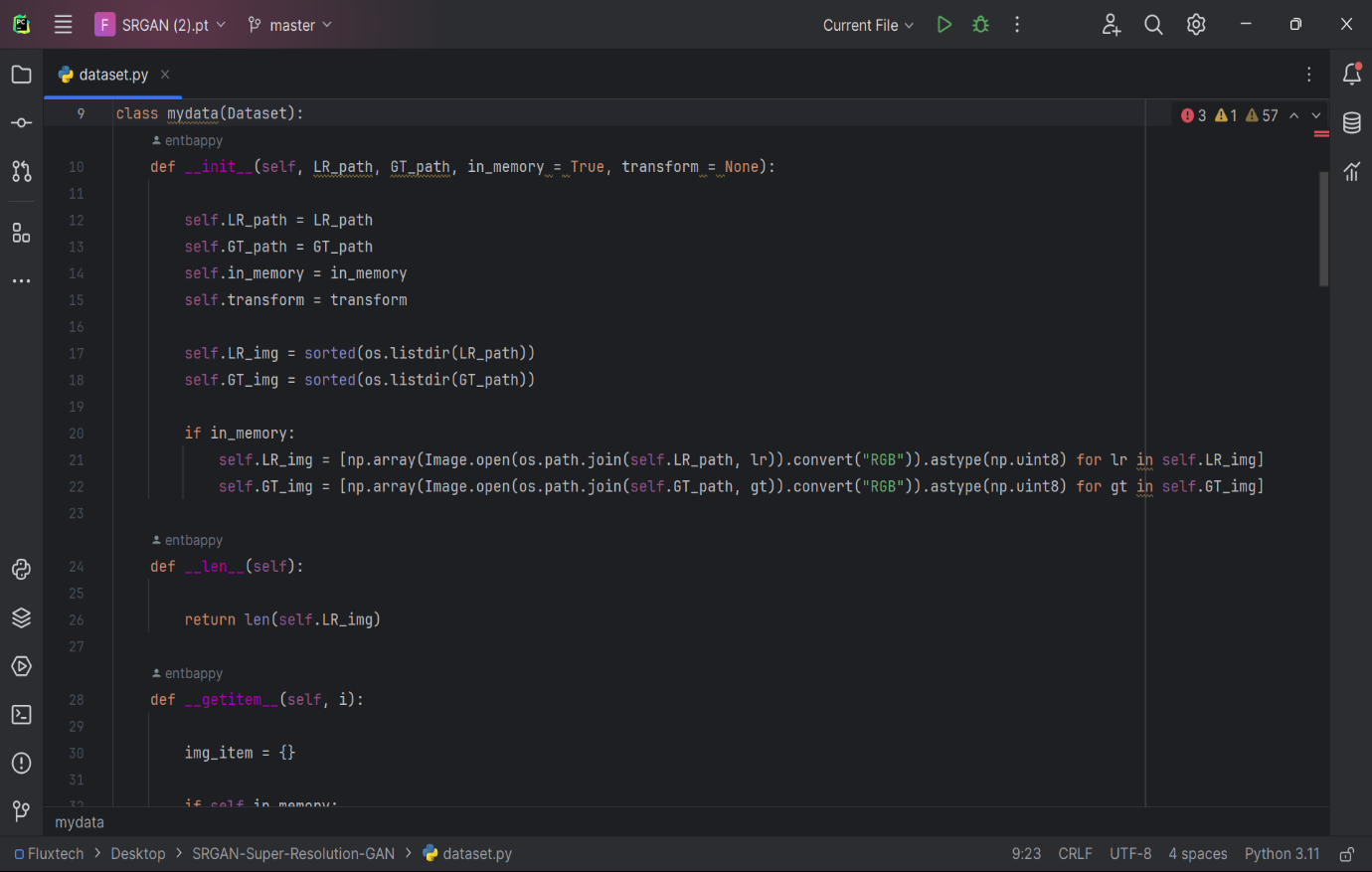
This dataset loads paired low-resolution (LR) and ground truth (GT) images.

It can optionally load images into memory as numpy arrays for faster access.

The transform parameter allows for data augmentation during training.

The **\_\_len\_\_** method returns the number of LR images.

The **\_\_getitem\_\_** method returns a dictionary containing the LR and GT images after preprocessing. The LR and GT images are normalized to the range [-1, 1] and their dimensions are transposed to (C, H, W) format.



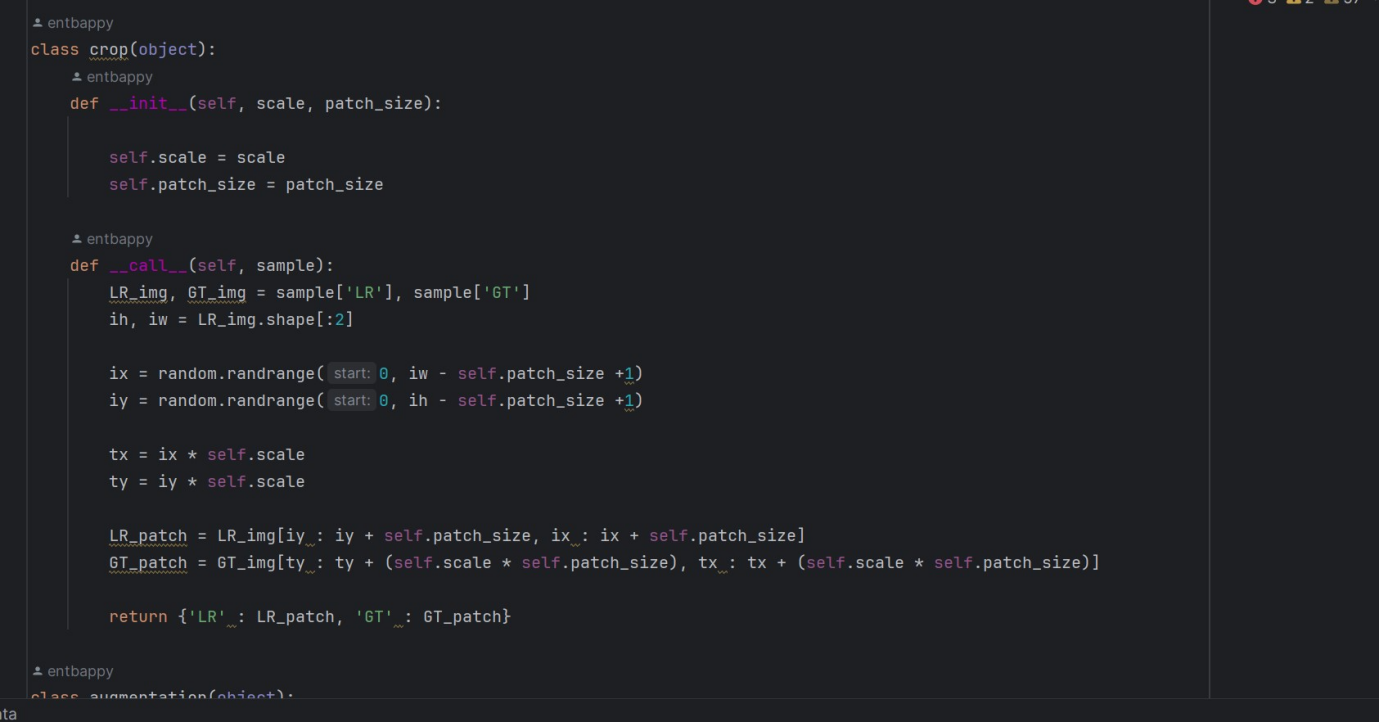
**testOnly\_data Dataset:**

This dataset is similar to mydata, but it is designed for testing and only loads LR images.



**Crop Transformation:**

This transformation takes an LR and GT image and randomly crops a patch of a specified size.

It is useful for data augmentation during training to provide the network with different image regions to learn from. 

**Augmentation Transformation:**

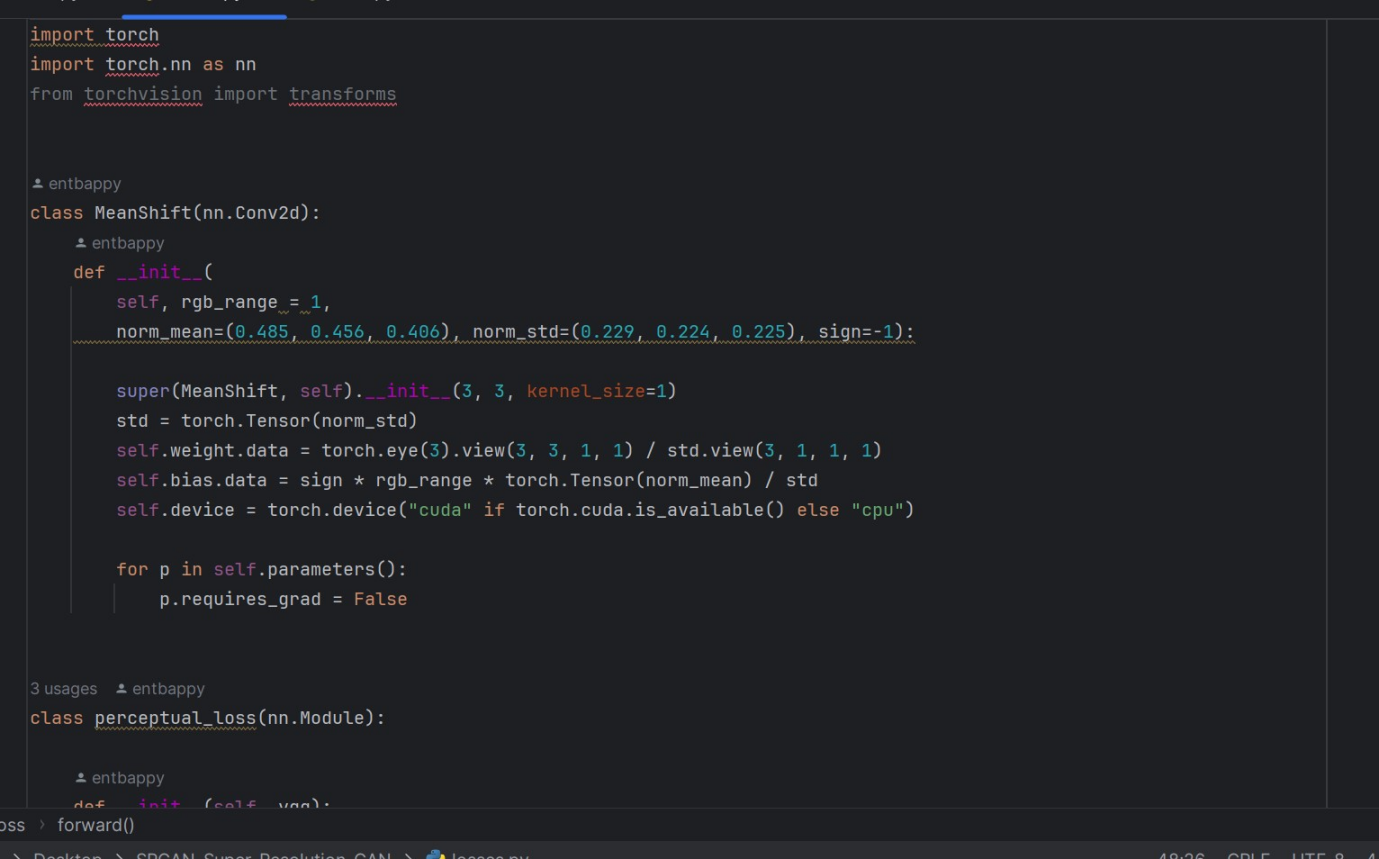
This transformation performs random horizontal flipping, vertical flipping, and rotation on both LR and GT images.

It's also used for data augmentation during training to increase the diversity of the training data. 

**Looses.py file**

This code snippet defines several custom PyTorch modules for perceptual loss and total variation loss. Let me break down what each of these modules does:

MeanShift Module:



This module performs mean shift on the input image tensor. It shifts the mean of the input tensor to a specific value.

It inherits from nn.Conv2d and is essentially a 1x1 convolutional layer with fixed weights and biases.

Mean and standard deviation normalization is applied to the input tensor using these fixed weights and biases.

This normalization is often used as a preprocessing step in neural networks, especially when working with pre-trained models like VGG.

**perceptual\_loss Module:**



This module computes the perceptual loss between a high-resolution (HR) image and its super-resolved (SR) version.

It first normalizes the HR and SR images using the MeanShift module.

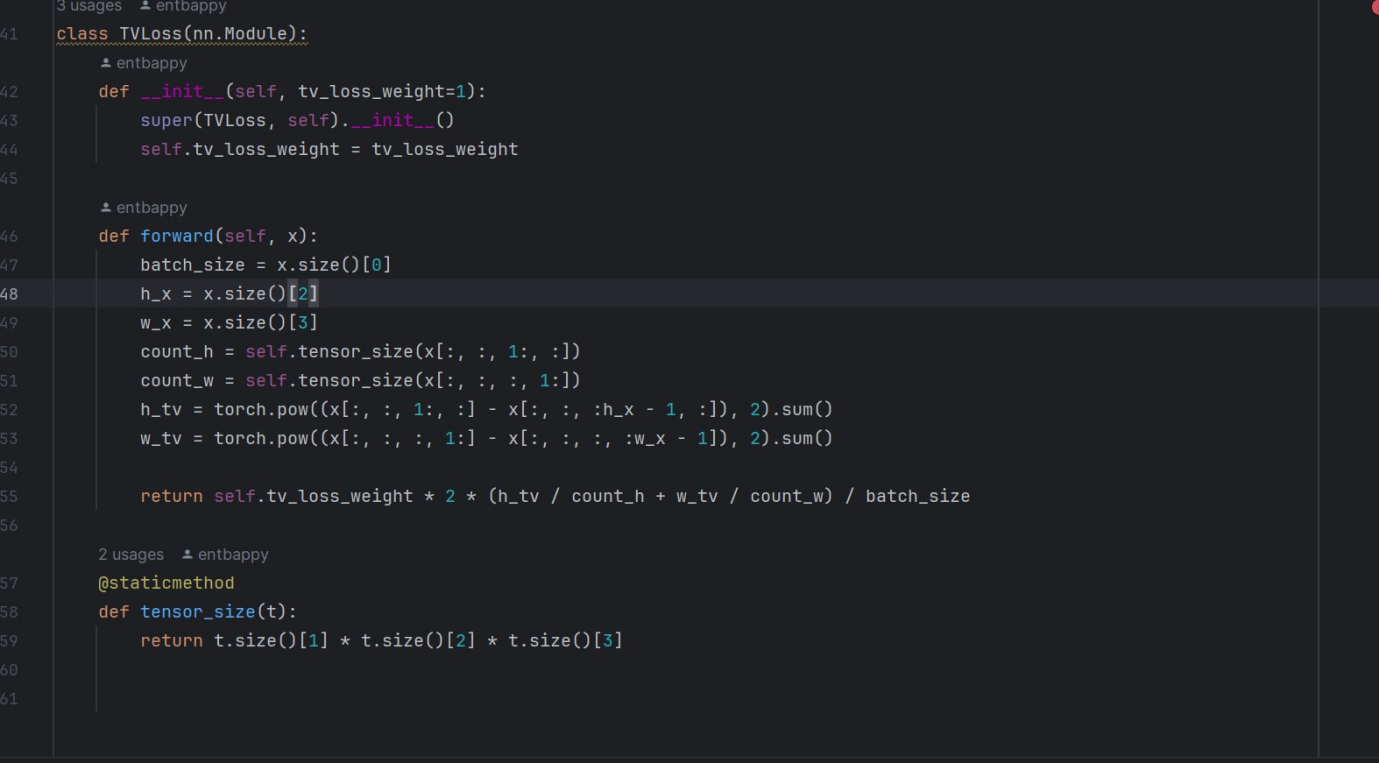
Then, it passes these normalized images through a pre-trained VGG model (self.vgg) and extracts feature maps at a specified layer (layer parameter).

The mean squared error (MSE) loss is computed between the feature maps of HR and SR images at the specified layer.

The method also returns the feature maps of HR and SR images.

This type of loss encourages the super-resolved image to have similar feature representations as the high-resolution image.

TVLoss Module:



This module computes the total variation (TV) loss on the input tensor.

TV loss measures the total variation of pixel intensities in an image and encourages spatial smoothness.

It calculates the differences between neighboring pixels along the height and width dimensions of the input tensor and sums them up.

The loss is normalized by the number of pixels in the input tensor.

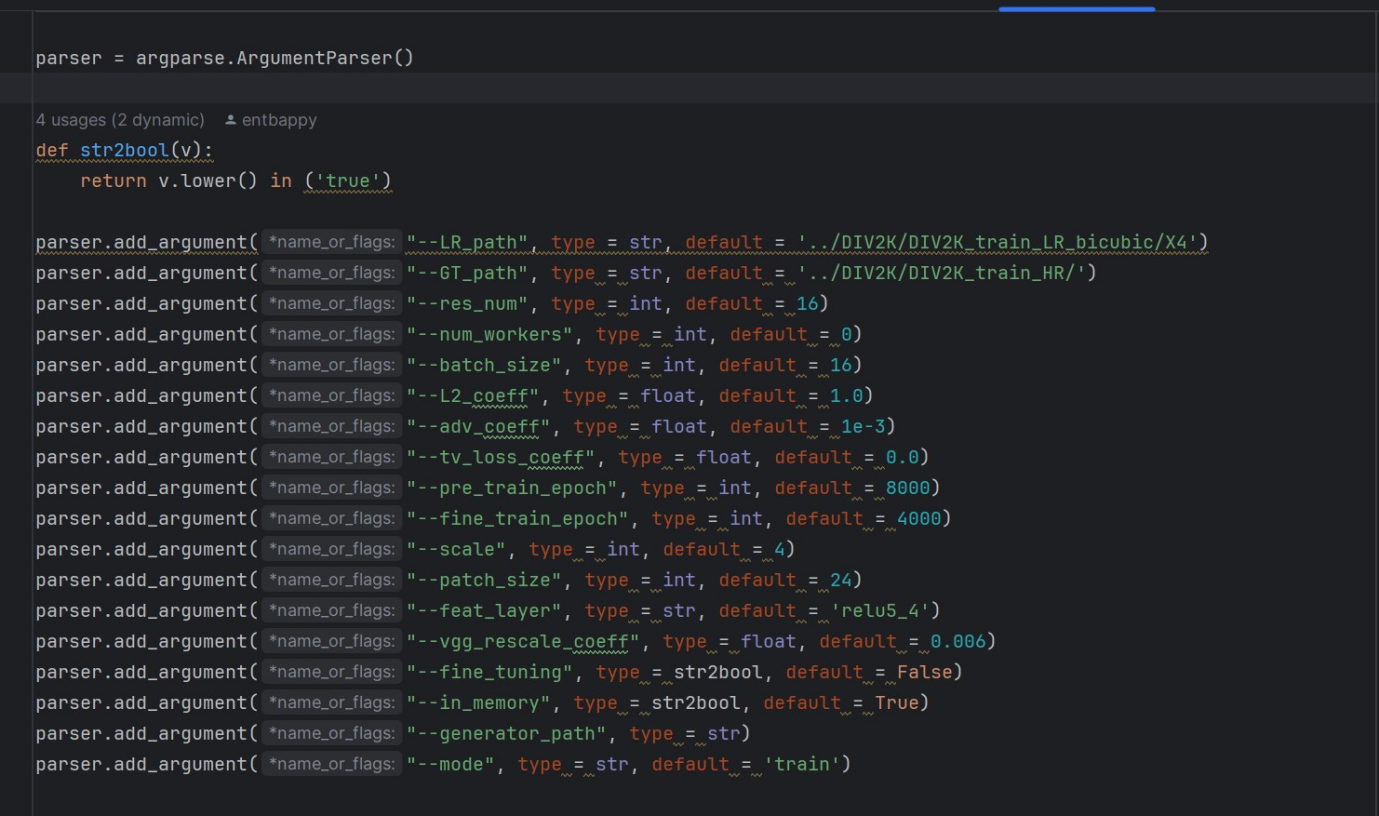
This type of loss is useful for removing noise and encouraging smoothness in the generated images.

**Main.py**

The main.py Python script defines a command-line interface using argparse. The script imports functions (train, test, and test\_only) from a module called mode. Based on the args.mode value provided in the command line, the script calls the appropriate function.

Here's a summary of what this script does:

Argument Parsing:



It uses argparse to parse command-line arguments. These arguments include file paths for low-resolution (LR) and ground truth (GT) images, training parameters (such as number of epochs, batch size, coefficients for different loss components), scaling factor, patch size, feature layer for perceptual loss, and other configurations.

Function Dispatching:



Depending on the value of args.mode, the script calls different functions from the mode module:

If args.mode is 'train', it calls the train(args) function.

If args.mode is 'test', it calls the test(args) function.

If args.mode is 'test\_only', it calls the test\_only(args) function.

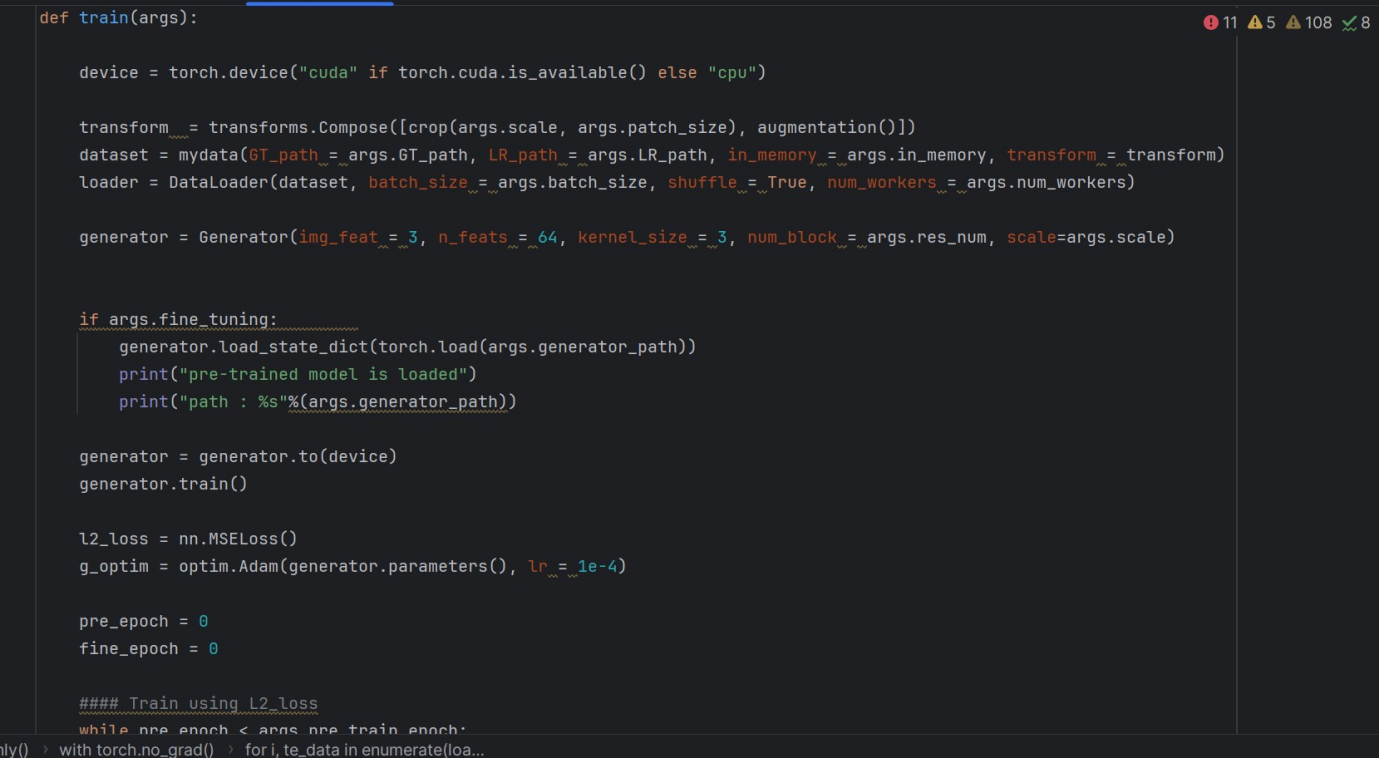
Dynamic Import:

The actual implementations of train, test, and test\_only functions are assumed to be defined in the mode module. The script imports these functions dynamically based on the args.mode value.

**Model.py**

This section includes the **train, test, and test\_only** functions, which are used for training the model, evaluating it, and performing inference on test images. Here's a brief overview of these functions:

**train(args):** This function is for training the SRGAN model. It has the following major steps:



Data loading and preprocessing: Loads the training data and applies transformations.

Model setup: Creates the SRGAN generator and discriminator models.

Pre-training: Trains the generator using L2 loss for a specified number of epochs.

Fine-tuning: Further trains the generator using perceptual and adversarial loss, along with training the discriminator to distinguish between real and generated images.

Loss functions: Defines various loss functions, including MSE loss, BCE loss, perceptual loss, adversarial loss, and total variance loss.

Training loop: Alternates between training the discriminator and the generator while updating their parameters.

Learning rate scheduling: Adjusts the learning rate during training.

Model checkpointing: Periodically saves the model weights.

test(args): This function is for evaluating the trained SRGAN model on a test dataset. It performs the following steps:

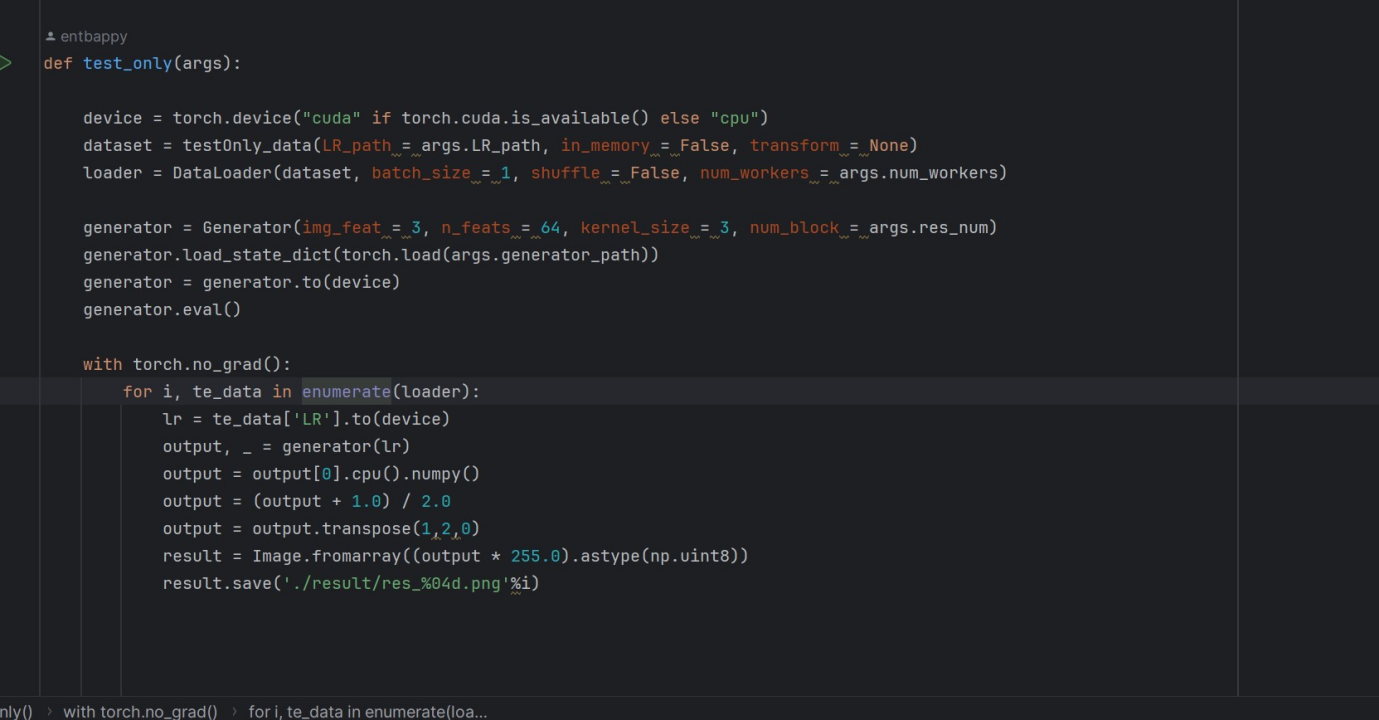
Data loading: Loads the test dataset.

Model setup: Loads the pre-trained SRGAN generator.

Testing loop: Iterates through the test data, generates high-resolution images from low-resolution inputs, and computes PSNR (Peak Signal-to-Noise Ratio) to evaluate the image quality.

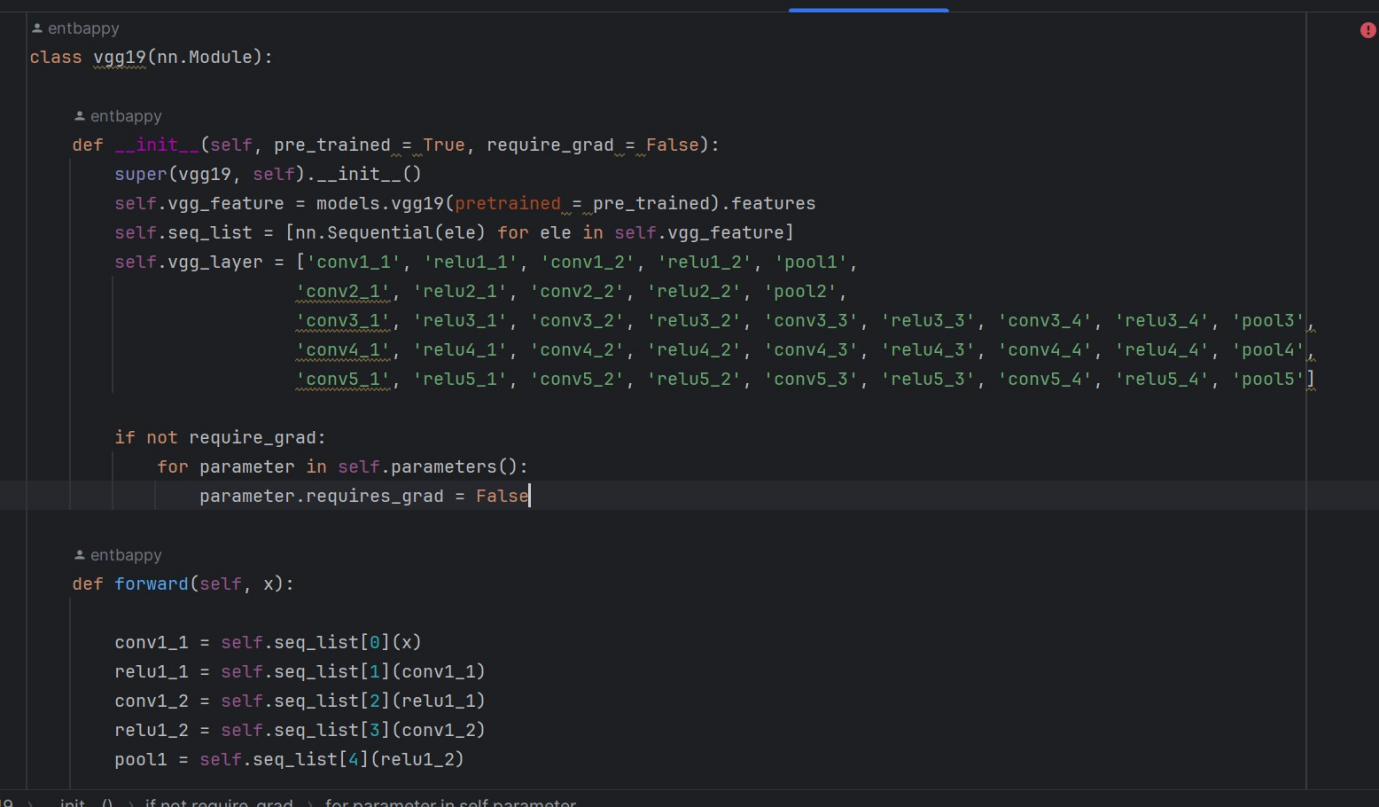
Result saving: Saves the generated images and PSNR values to a file.

test\_only(args): This function is for performing inference with the trained SRGAN model on test images. It has a similar structure to the test function but doesn't compute PSNR or save PSNR values to a file. Instead, it saves the generated images.



**VGG9.PY**

The provided code defines a custom VGG-19 network for feature extraction. The vgg19 class inherits from nn. Module and constructs the VGG-19 architecture using pre-trained layers from the torchvision library. The network extracts features from different layers and provides the output as a namedtuple.





Here's a summary of what the code does:

Initialization: The class takes two optional parameters - pre\_trained and require\_grad. If pre\_trained is True, the model loads pre-trained weights from torchvision. If require\_grad is False, the model parameters are set to not require gradients, effectively freezing the layers during training.

Forward Method: The forward method defines the forward pass of the network. It passes the input through various convolutional and pooling layers, storing intermediate outputs at specific layers. The intermediate outputs are then packed into a namedtuple and returned.

Named tuple Output: The output of the forward pass is a namedtuple named vgg\_output, containing the intermediate feature maps from specific layers of the VGG-19 network. This allows easy access to different layers' outputs.9