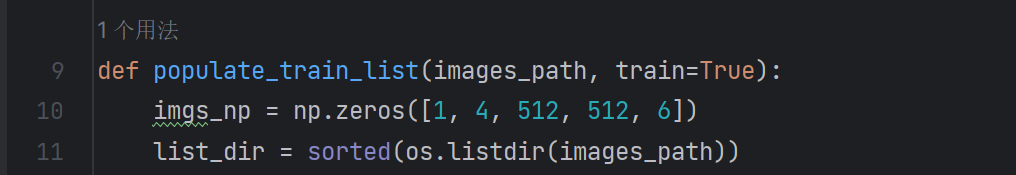
# Create\_dataset.py

函数定义部分

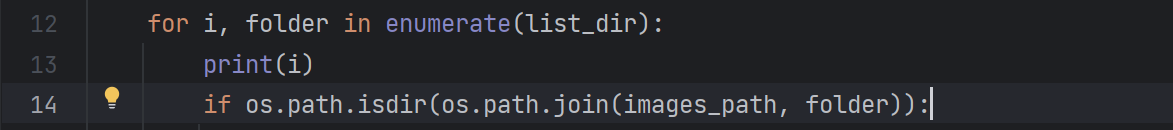
定义函数 populate\_train\_list，用于处理图像数据。

images\_path 是图像文件的路径。

train 是一个布尔值，表示是否用于训练数据。

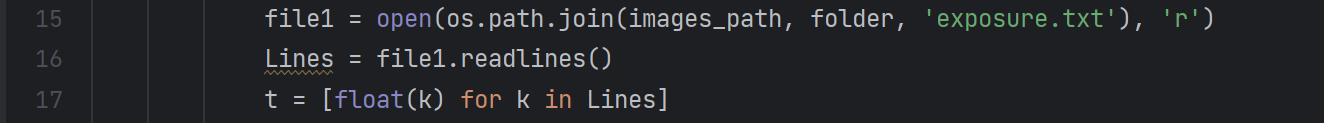
初始化一个空的 NumPy 数组 imgs\_np，形状为 [1, 4, 512, 512, 6]。

列出 images\_path 目录下的所有文件和文件夹，并进行排序。

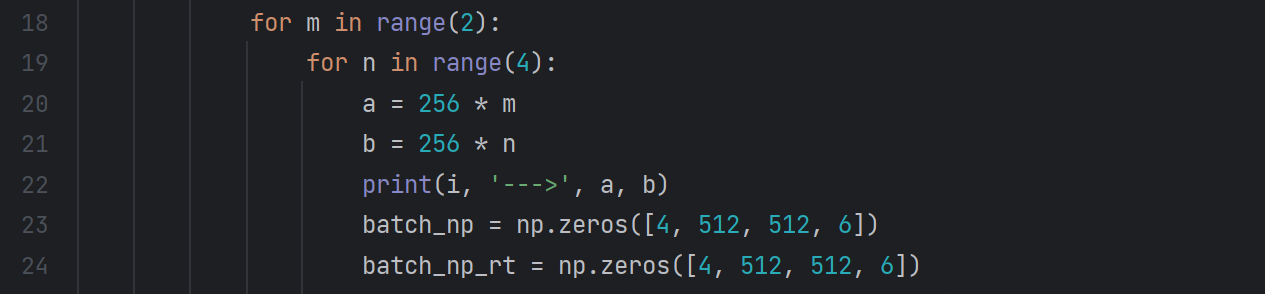
遍历目录列表 list\_dir，获取每个文件夹的索引 i 和名称 folder。

打印当前文件夹的索引 i。

检查当前文件夹是否是目录。

打开当前文件夹中的 exposure.txt 文件并读取其内容。

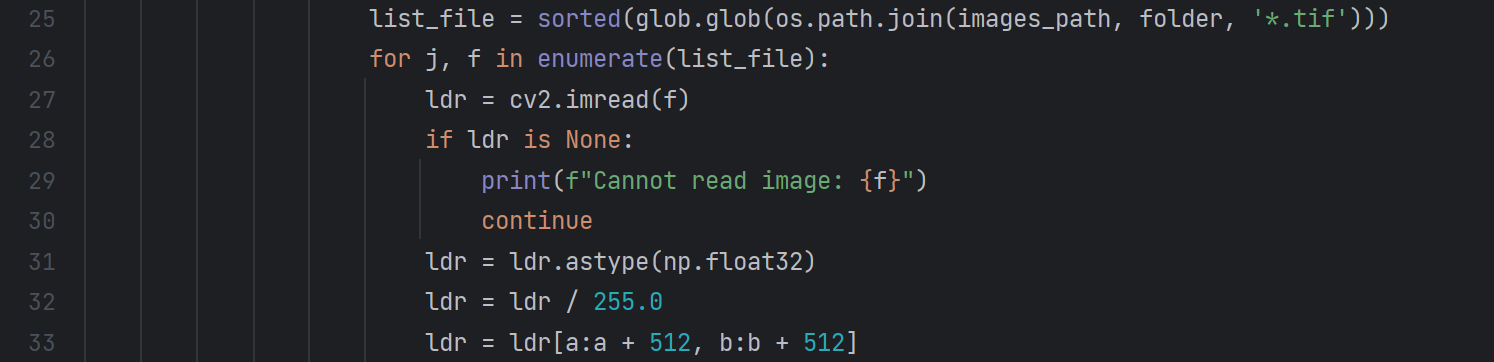
将文件中的每一行转换为浮点数并存储在列表 t 中。

双重循环用于处理图像的不同部分。

计算当前块的起始坐标 a 和 b。

打印当前处理的块的坐标。

初始化两个批量数组 batch\_np 和 batch\_np\_rt，形状为 [4, 512, 512, 6]。

列出当前文件夹中的所有 .tif 文件并排序。

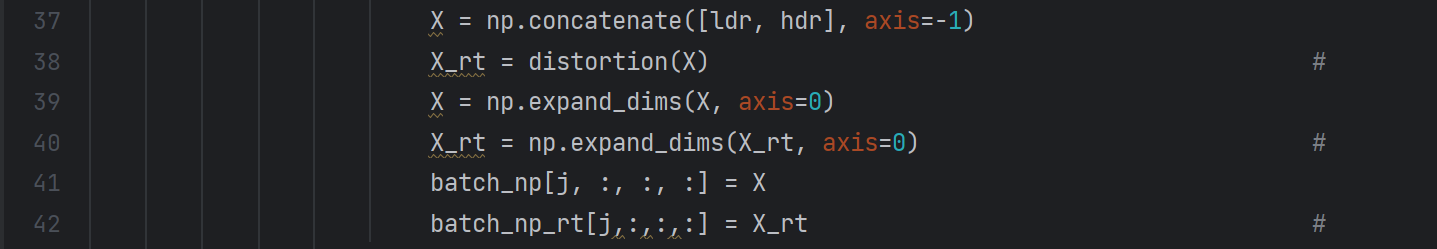
遍历这些文件，读取每个文件 f。

如果读取失败，打印错误消息并继续下一个文件。

将图像转换为 float32 类型并归一化。

截取图像的指定块。

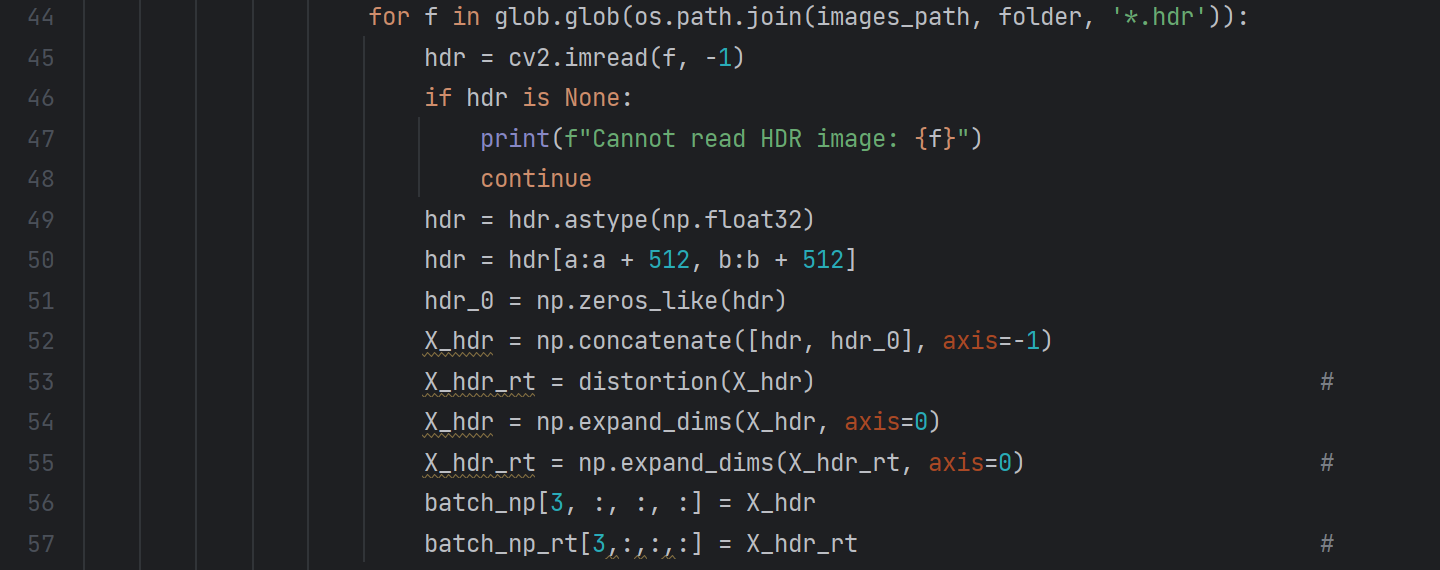
计算对应的 HDR 图像。

将 LDR 和 HDR 图像沿最后一个轴拼接。

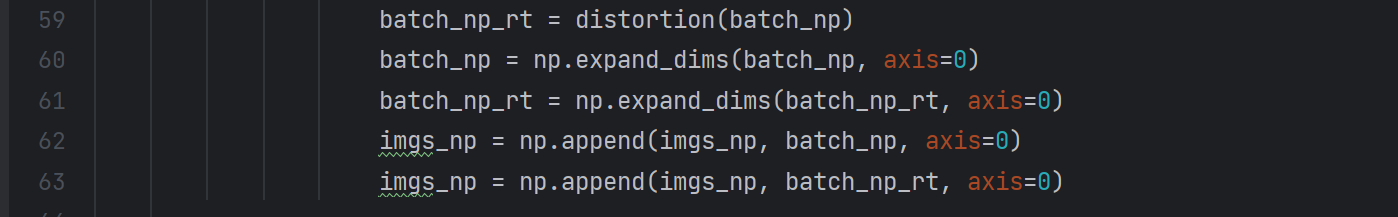
对拼接的图像进行数据增强（distortion 函数）。

扩展维度以匹配批处理大小。

将处理后的图像存入批处理数组。

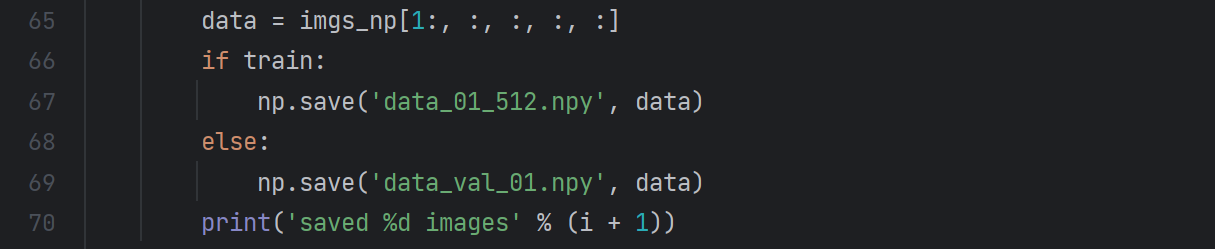
对 HDR 图像进行类似处理（读取、转换、截取、拼接、数据增强）。

将处理后的 HDR 图像存入批处理数组的最后一个位置。

对整个批处理数组进行数据增强。

扩展批处理数组的维度。

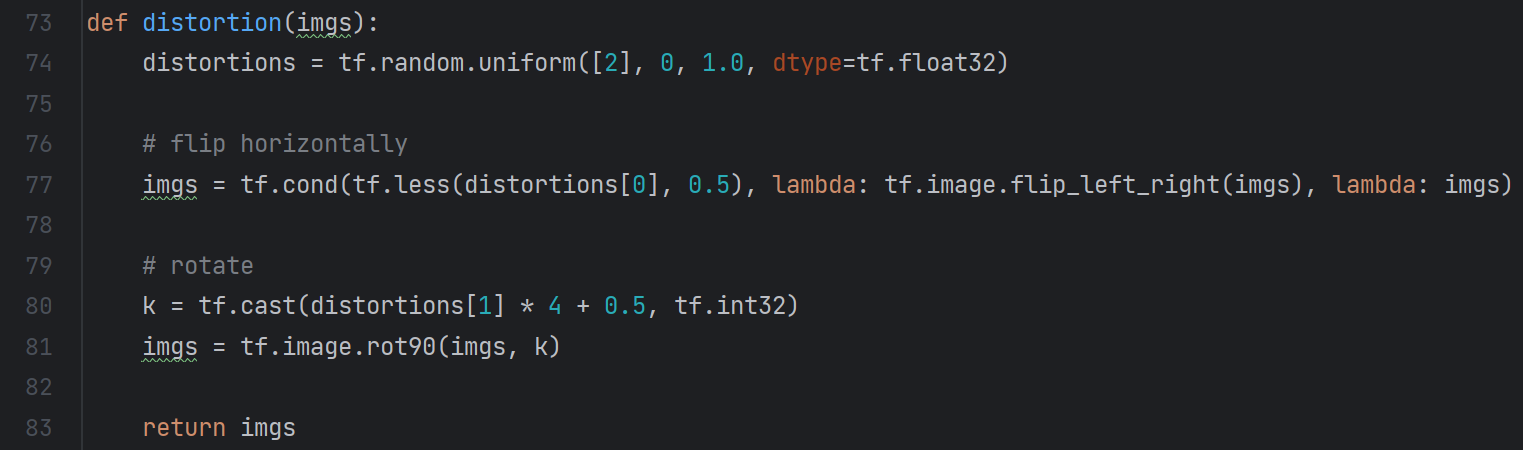
将批处理数组追加到 imgs\_np 中。

去掉初始的空数组。

根据 train 参数保存处理后的数据为 .npy 文件。

打印保存的图像数量。

数据增强函数

定义数据增强函数 distortion。

生成两个随机数用于控制数据增强操作。

根据第一个随机数决定是否水平翻转图像。

根据第二个随机数决定旋转图像的次数。

返回增强后的图像。

调用函数

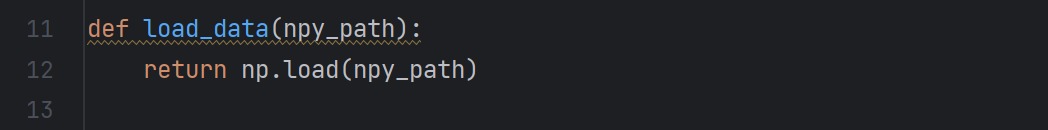


# Generate\_HDR\_dataset

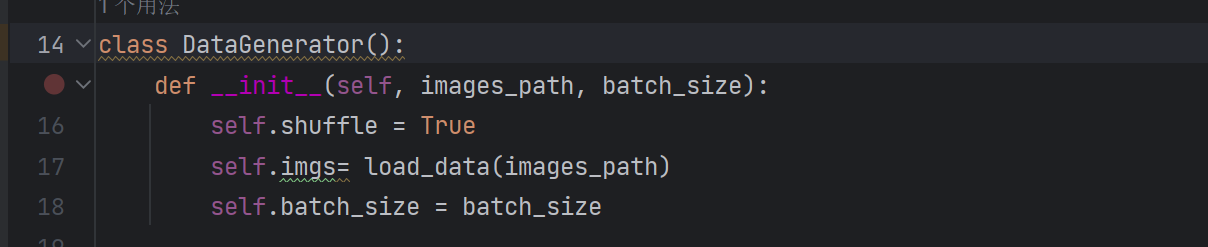
调用 populate\_train\_list 函数，处理指定路径下的图像数据，并保存为训练数据。

这段代码主要定义了一个数据生成器类 DataGenerator，用于加载和生成批量数据。

函数定义部分

定义函数 load\_data，用于从指定路径加载 .npy 文件并返回其内容（一个 NumPy 数组）。

类定义部分

定义一个数据生成器类 DataGenerator。

\_\_init\_\_ 是类的初始化方法。

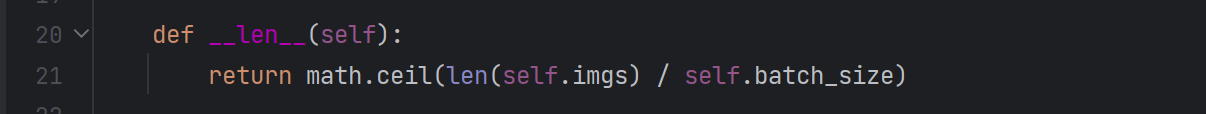
images\_path 是存储图像数据的 .npy 文件的路径。

batch\_size 是每个批次的大小。

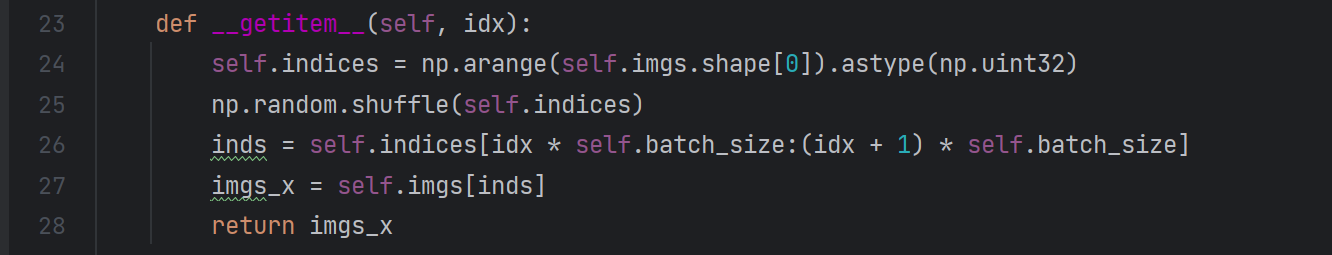
初始化一个属性 self.shuffle 为 True，表示是否在每个epoch结束时打乱数据。

调用 load\_data 函数加载图像数据，并将其存储在 self.imgs 属性中。

将 batch\_size 存储在 self.batch\_size 属性中。

定义 \_\_len\_\_ 方法，返回数据生成器的长度，即总批次数。

使用 math.ceil 计算数据生成器的长度，确保任何剩余的图像也会被包含在最后一个批次中。

定义 \_\_getitem\_\_ 方法，用于获取指定批次的图像数据。

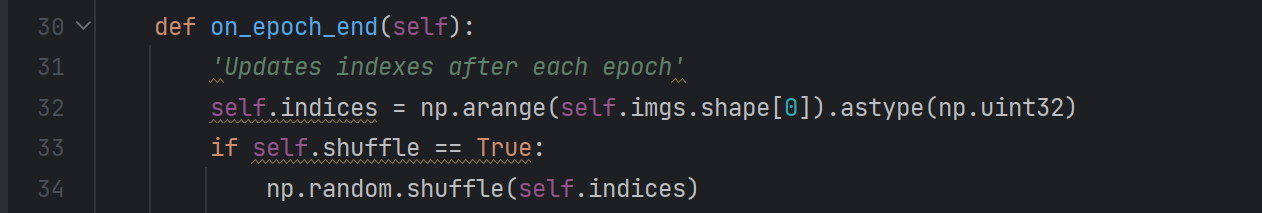
生成一个数组 self.indices，包含图像数据的索引，并转换为无符号32位整数类型。

使用 np.random.shuffle 随机打乱索引数组。

计算当前批次的索引范围 inds。

从图像数据中选择对应的图像 imgs\_x。

返回当前批次的图像数据 imgs\_x。

定义 on\_epoch\_end 方法，在每个epoch结束时调用。

生成一个新的索引数组 self.indices，包含所有图像数据的索引，并转换为无符号32位整数类型。

如果 self.shuffle 为 True，使用 np.random.shuffle 随机打乱索引数组。

load\_data 函数用于加载图像数据。

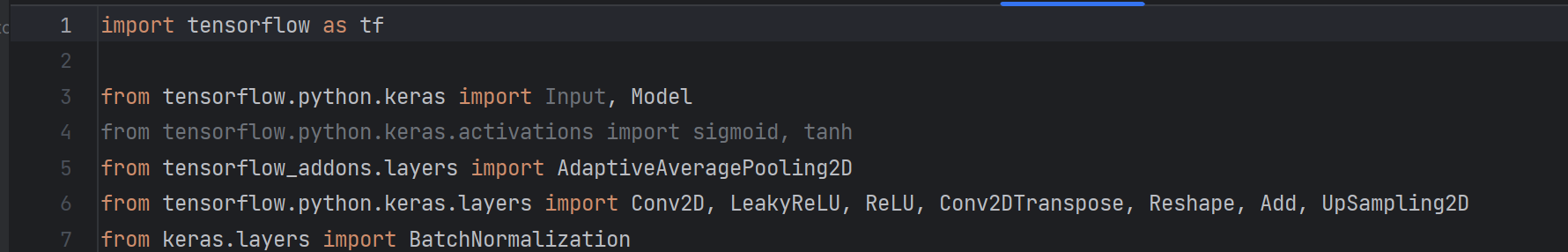
DataGenerator 类用于创建一个数据生成器对象，能够按批次加载和返回图像数据，并在每个epoch结束时随机打乱数据。

该生成器类实现了 \_\_len\_\_ 和 \_\_getitem\_\_ 方法，使其能够与 Keras 等深度学习框架兼容。

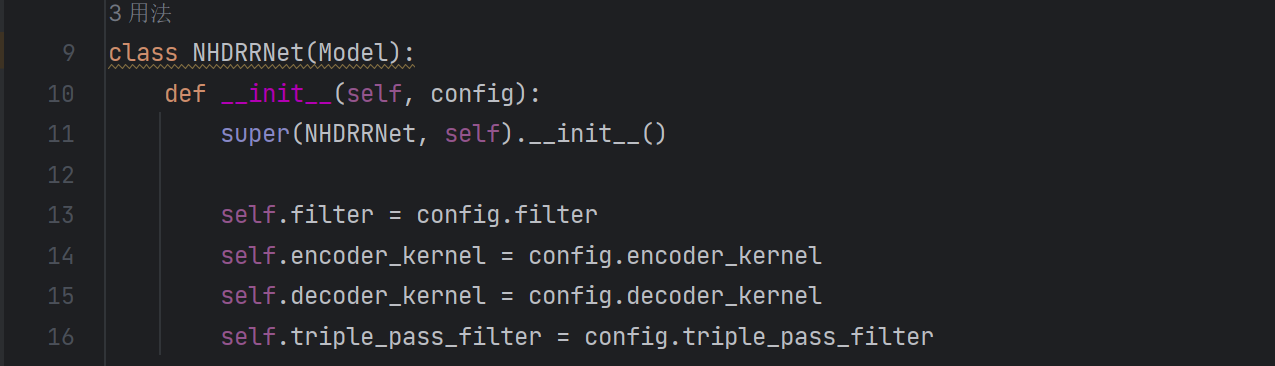
# HDR

以下是逐行解释 NHDRRNet 类的代码，该类继承了 TensorFlow 的 Model 类：

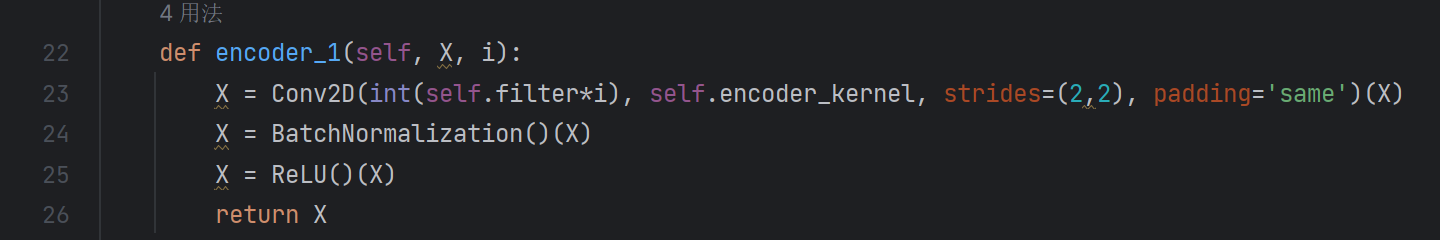
导入所需的库

这些导入了TensorFlow和Keras中常用的模块和层，包括卷积层、反卷积层、激活函数和批标准化层。

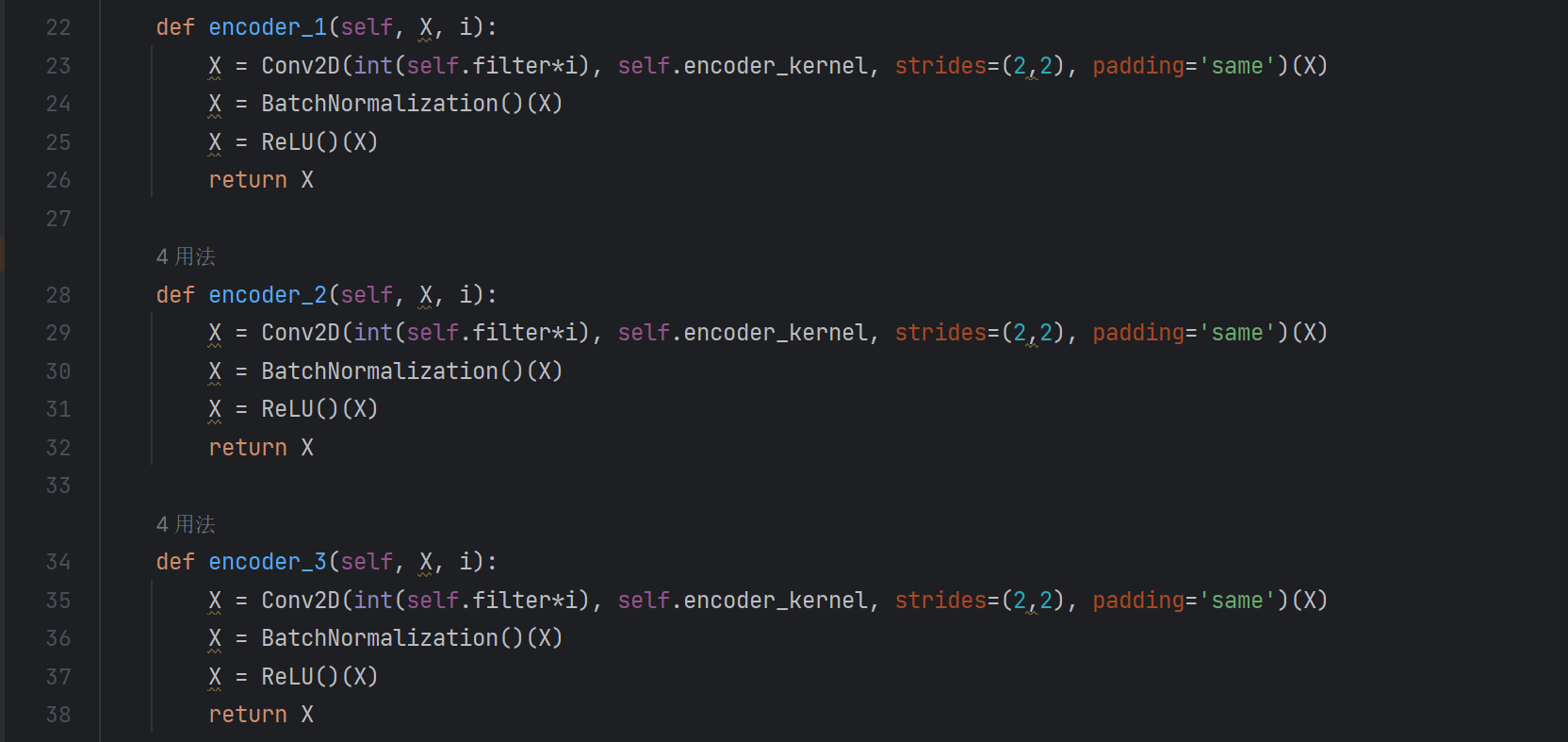
NHDRRNet 类定义

构造函数初始化了网络的配置参数，例如滤波器数量、编码器和解码器的卷积核大小、三重通过滤波器的数量。

自适应插值方法

该方法用于自适应地插值图像，将其放大到所需尺寸。

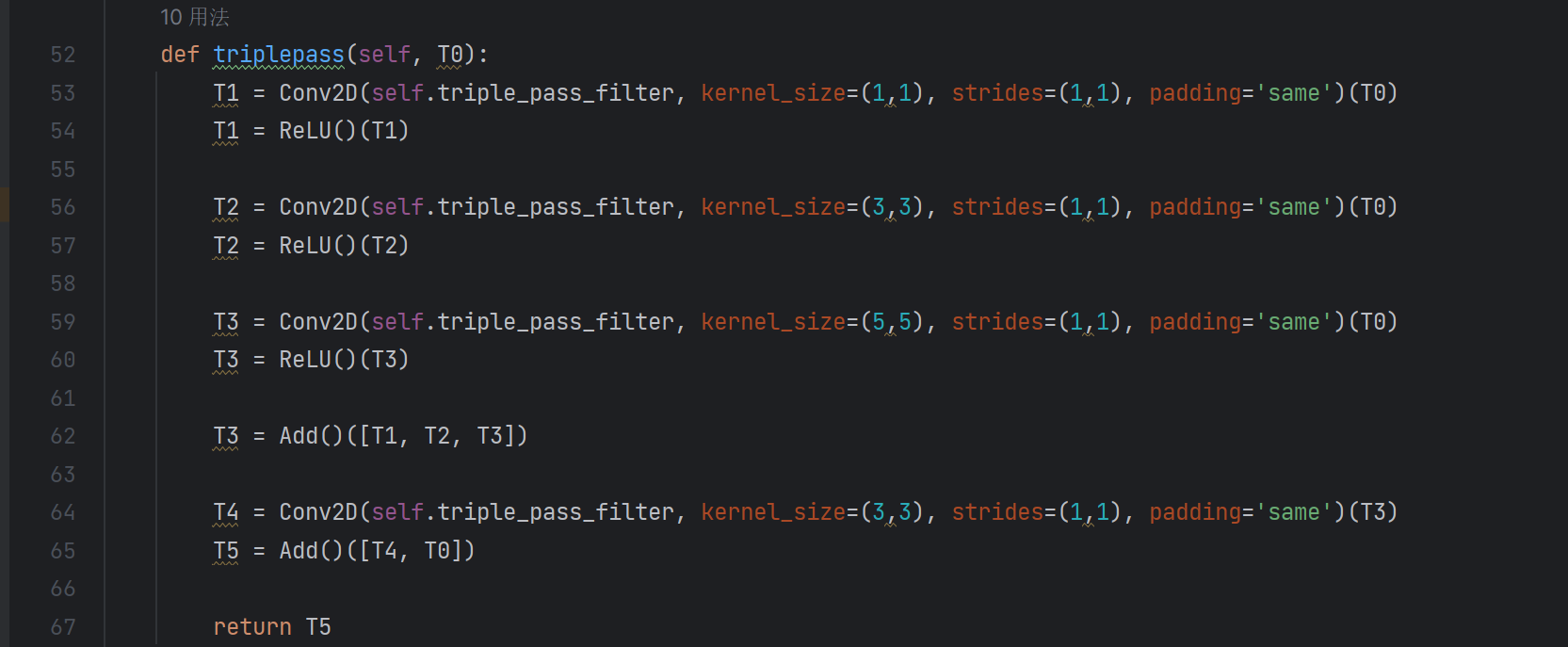
编码器部分

编码器函数 encoder\_1 将输入 X 进行卷积、批标准化和 ReLU 激活。其他的 encoder\_2 和 encoder\_3 与此类似。

解码器部分

解码器函数 decoder 和 decoder\_last 用于对特征图进行反卷积操作，将其还原到原始图像的尺寸。不同的是，decoder\_last 输出3个通道，而 decoder 的输出通道数根据 i 动态变化。

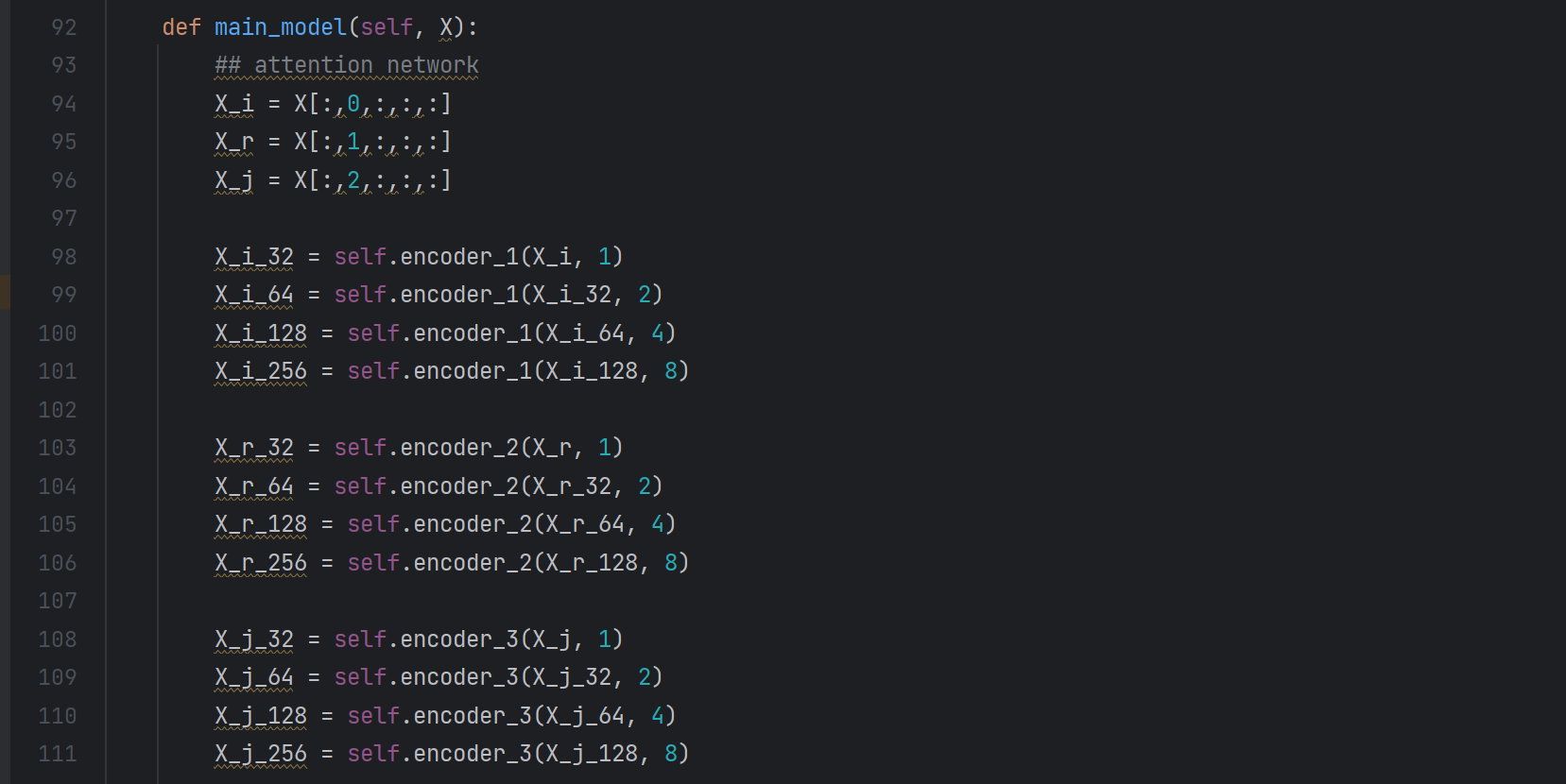
三重通过模块

三重通过模块将输入 T0 通过三个不同尺寸的卷积核（1x1, 3x3, 5x5）进行卷积操作，并将结果相加。然后进行一次3x3卷积并与原始输入相加。

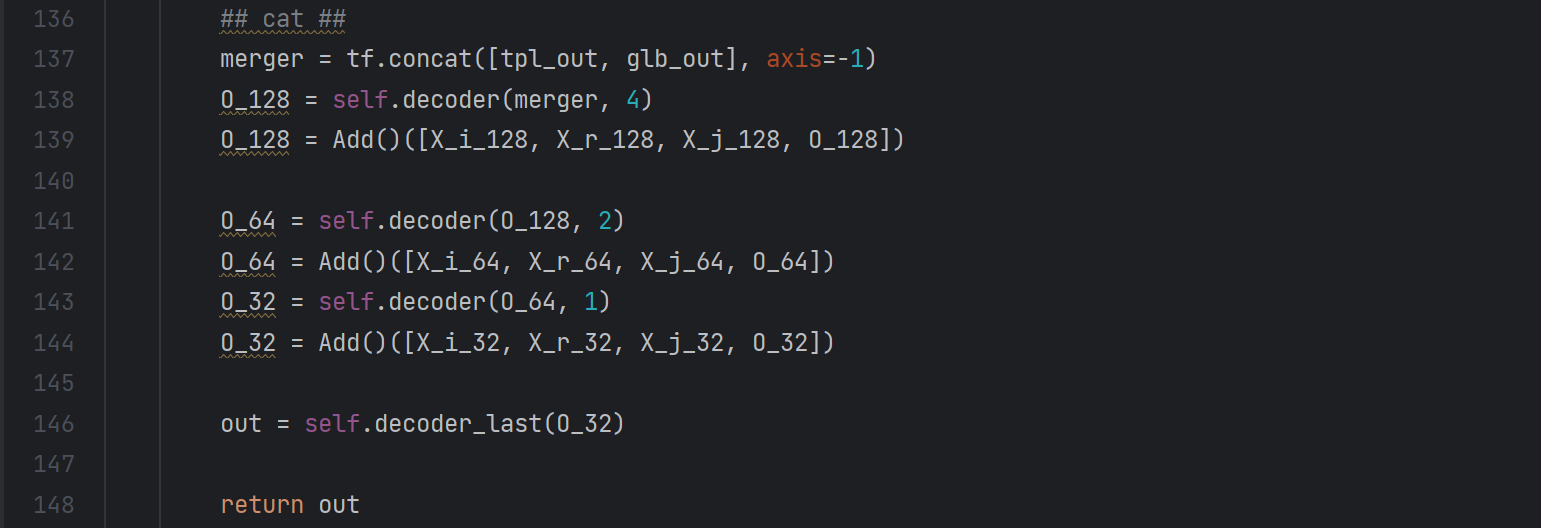
全局非局部模块

全局非局部模块通过计算特征图的相似性来捕获长距离依赖关系。首先将输入 X 通过1x1卷积，然后重塑为适合矩阵乘法的形状。计算相似性矩阵并与特征图相乘，最后通过1x1卷积恢复维度并与输入相加。

主模型定义





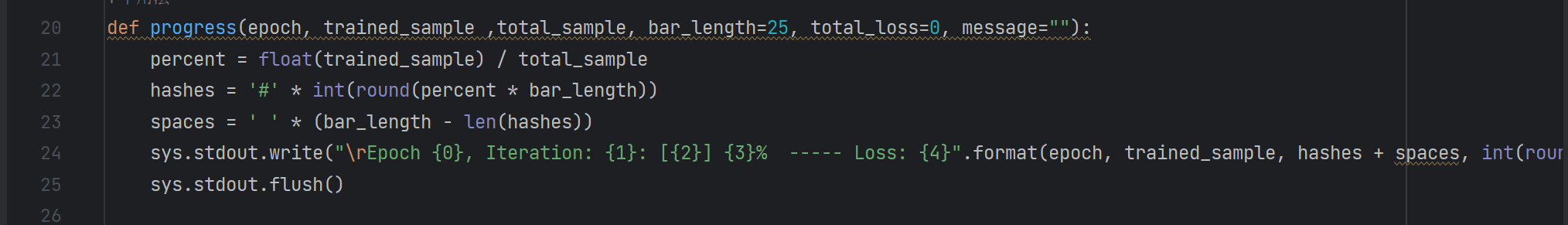


main\_model 方法是整个网络的主架构。首先对输入 X 的三个通道分别进行编码，然后将编码结果拼接，经过一个1x1卷积得到 encoder\_last。接着，通过三重通过模块和全局非局部模块进行特征提取，并将两者结果拼接。最后通过解码器逐步恢复到原始图像尺寸，并得到最终输出。

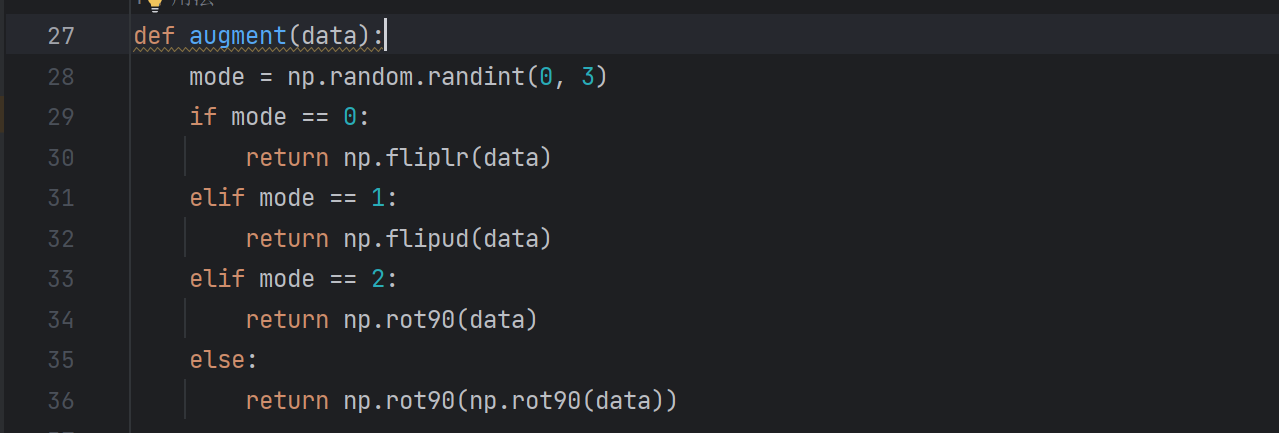
# Main

以下是 train.py 文件的逐行解释：

进度条函数

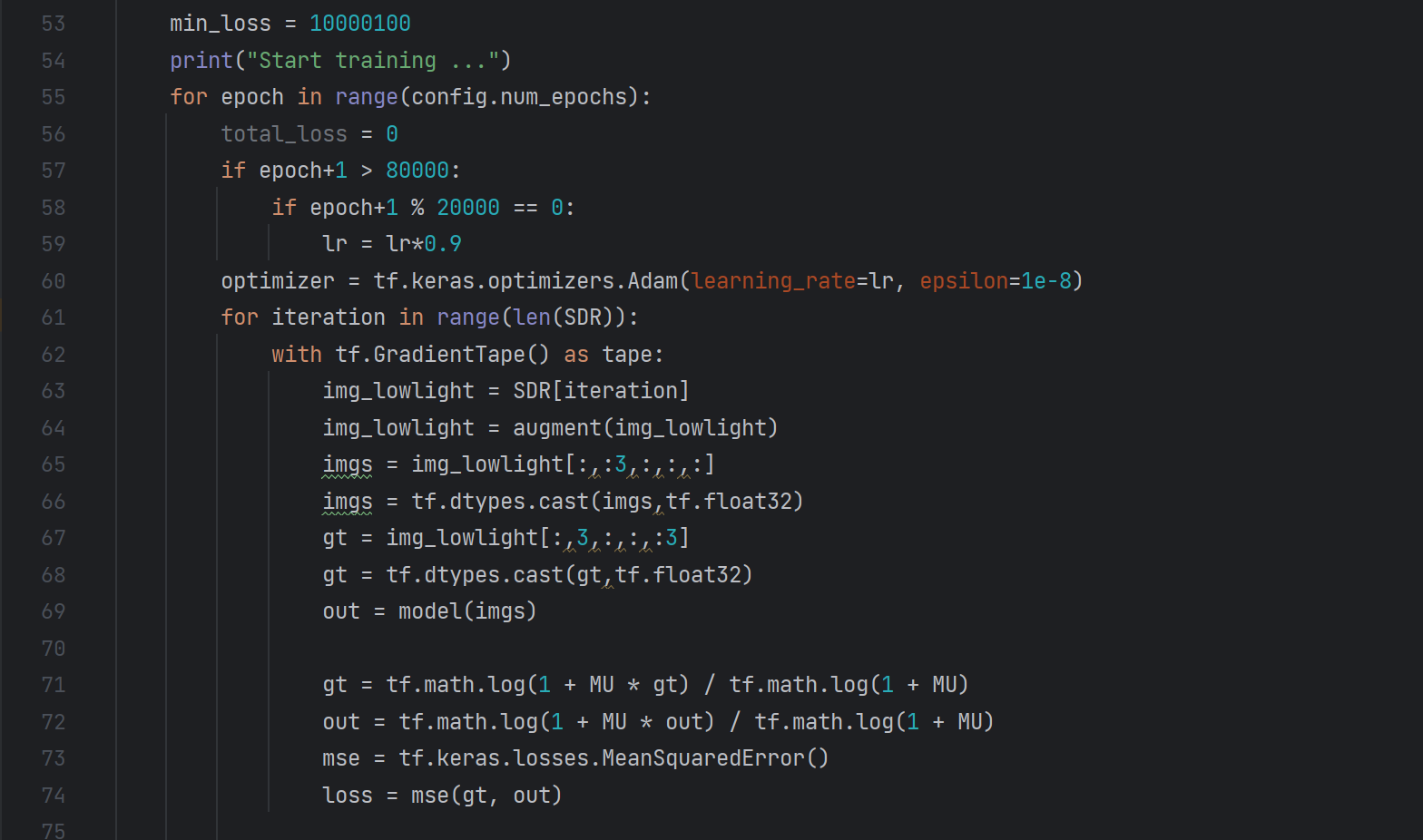
该函数用于显示训练进度条。根据训练样本的百分比，显示当前训练的进度和损失值。

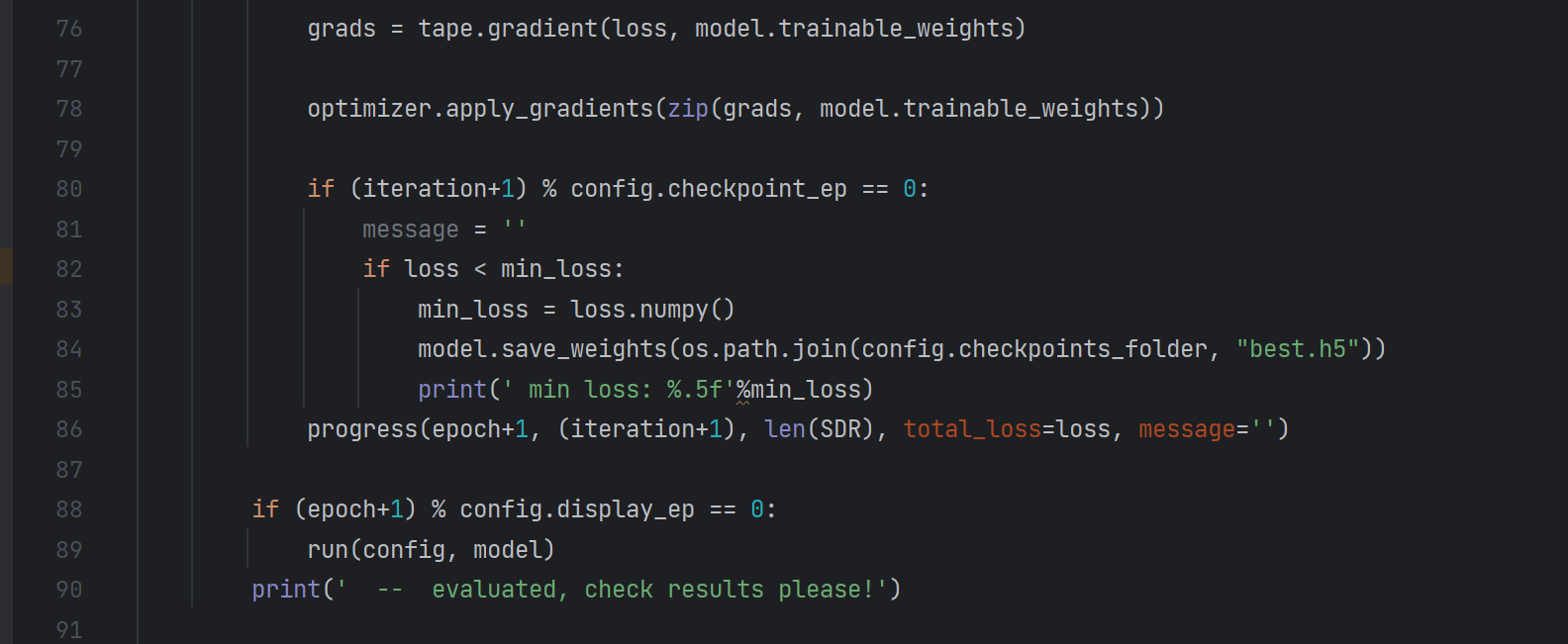
数据增强函数

该函数用于对输入数据进行随机增强，包括水平翻转、垂直翻转和旋转90度，以提高模型的泛化能力。

训练函数







设置训练使用的GPU设备。

初始化常数 MU 和数据生成器 SDR。

初始化学习率 lr。

创建 NHDRRNet 模型并定义输入和输出。

如果配置中设置了加载预训练模型，则加载预训练权重。

开始训练，遍历所有epoch：

动态调整学习率。

使用 tf.GradientTape 记录前向传播，以计算梯度。

从数据生成器中获取低光图像并进行数据增强。

将前3个通道作为输入，将第4个通道作为GT（Ground Truth）。

将输入和GT转换为浮点型。

通过模型前向传播，计算输出。

使用对数变换和常数 MU 进行处理。

计算均方误差（MSE）损失。

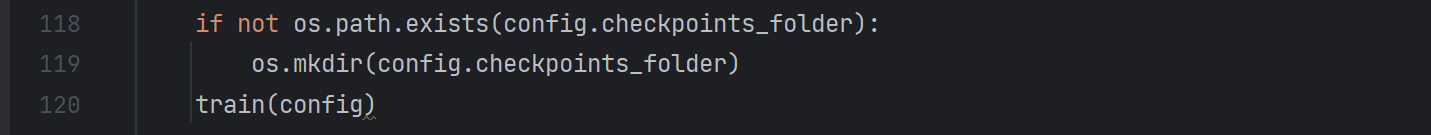
计算梯度并更新模型权重。

定期保存最优模型权重并更新进度条。

定期调用 run 函数进行模型评估。

参数解析和主函数



****

使用 argparse 模块解析命令行参数，设置默认值。

检查并创建检查点文件夹。

调用 train 函数开始训练模型。

# Test

以下是更新后的 run.py 文件的逐行解释：

读取测试数据函数

该函数用于读取和预处理测试数据：

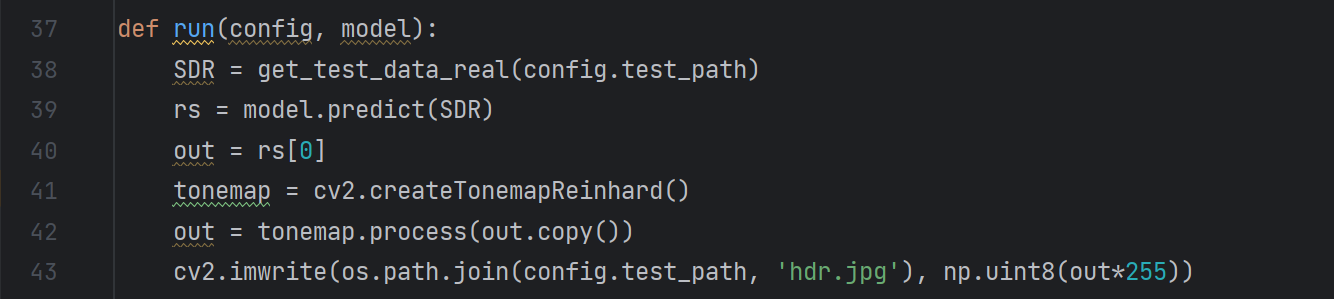
创建一个空的NumPy数组来存储图像数据。

读取曝光时间文件 exposure.txt，获取曝光时间列表。

遍历所有 .tif 格式的图像文件，读取图像并进行预处理（归一化、调整大小、颜色转换）。

根据曝光时间计算HDR图像，将LDR和HDR图像拼接在一起，存入NumPy数组。

运行模型并保存结果

该函数用于运行模型并保存生成的HDR图像：

调用 get\_test\_data\_real 函数读取测试数据。

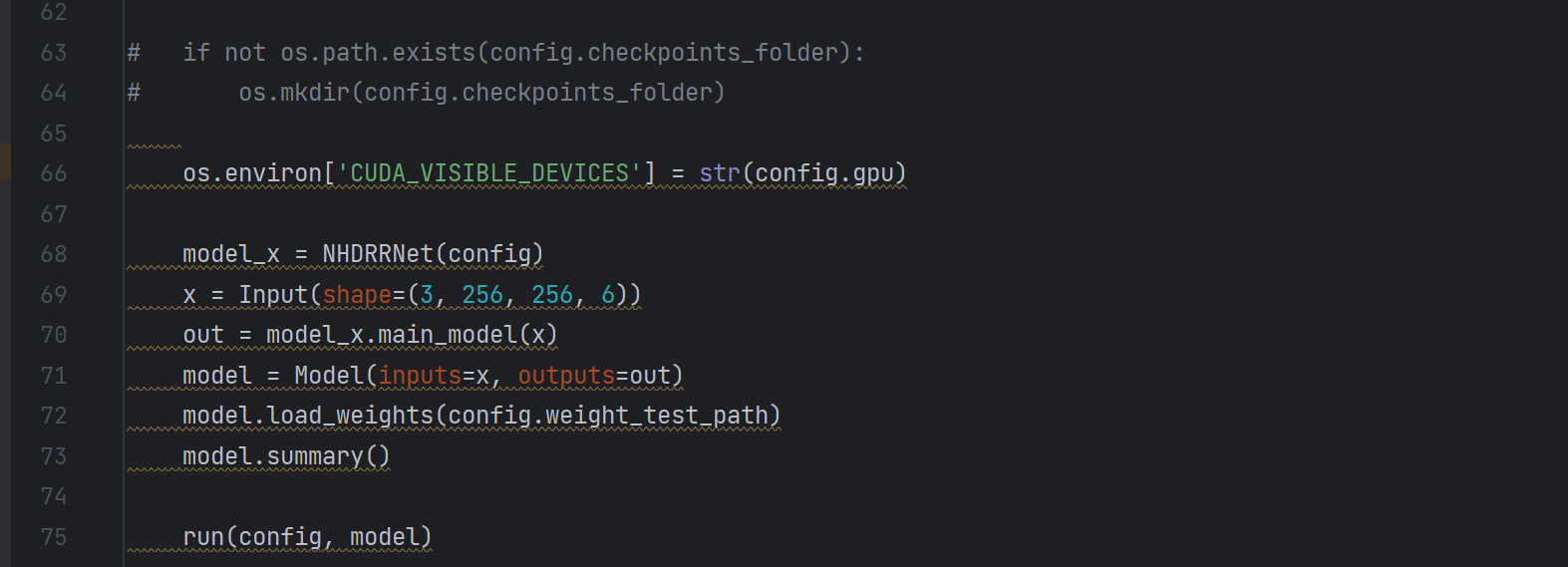
使用模型进行预测，获取输出。

使用Reinhard色调映射处理输出。

将处理后的图像保存为 hdr.jpg。

主函数





使用 argparse 模块解析命令行参数，设置默认值。

设置CUDA设备，以指定使用的GPU。

创建 NHDRRNet 模型实例并定义输入和输出。

加载预训练的模型权重。

打印模型结构。

调用 run 函数运行模型并保存结果。

# Val

以下是上述代码的逐行解释：

定义测试数据获取函数

该函数用于获取测试数据：

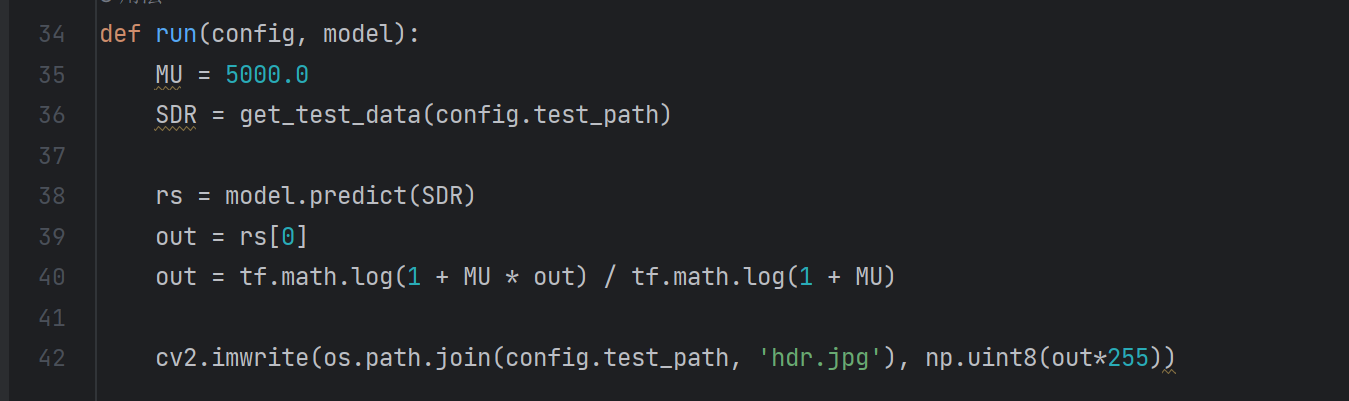
初始化一个空的NumPy数组 imgs\_np 来存储图像数据。

打开并读取曝光时间文件 exposure.txt，将曝光时间保存到列表 t 中。

遍历所有 .tif 格式的图像文件，读取图像并进行预处理（归一化、调整大小）。

根据曝光时间计算HDR图像，将LDR和HDR图像拼接在一起，存入NumPy数组 imgs\_np。

定义运行模型并保存结果的函数

该函数用于运行模型并保存生成的HDR图像：

设置一个常数 MU。

调用 get\_test\_data 函数读取测试数据 SDR。

使用模型进行预测，获取输出 rs。

对输出进行处理，将其转换为HDR图像。

将处理后的图像保存为 hdr.jpg。