You're asking to define the augmentation function, which is a great way to encapsulate the ImageDataGenerator setup for clarity and reusability.

As previously discussed, here's how you can define functions to create ImageDataGenerator instances for both training (with augmentation) and validation/test (without augmentation, usually just for consistent preprocessing if needed):

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def get_train_augmentation_generator():
    """
    Returns an ImageDataGenerator configured for training data
augmentation.
    This generator will apply random transformations to images during
training
    to increase dataset variability and prevent overfitting.
    """
    return ImageDataGenerator(
        rotation_range=20,          # Randomly rotate images by up to
20 degrees
        width_shift_range=0.2,      # Randomly shift images
horizontally by up to 20% of total width
        height_shift_range=0.2,     # Randomly shift images vertically
by up to 20% of total height
        shear_range=0.2,            # Apply random shear
transformation
        zoom_range=0.2,             # Randomly zoom in/out on images
        horizontal_flip=True,       # Randomly flip images
horizontally
        fill_mode='nearest'         # Strategy to fill in newly
created pixels after transformations
                                    # ('nearest' fills with the
nearest pixel value)
    )

def get_validation_test_generator():
    """
    Returns an ImageDataGenerator for validation and test data.
    This generator typically does NOT apply random augmentations,
    as we want to evaluate the model on consistent, un-augmented data.
    It can still be used for consistent preprocessing if needed (e.g.,
rescaling),
    but if your data is already normalized (like X_val, X_test in our
full script),
    you can often directly use the NumPy arrays.
    """
    return ImageDataGenerator(
        # No augmentation parameters here
        # If you were loading images directly from directories for
validation/test,
        # and needed a preprocessing step like rescaling that wasn't
done beforehand,
        # you might add 'rescale=1./255' here if the images weren't
already normalized.
    )

# How you would typically use these functions in your main script:
# train_datagen = get_train_augmentation_generator()
# # If you are feeding numpy arrays (X_val, y_val) directly to
model.fit's validation_data,
# # you might not explicitly need val_datagen for this purpose, as the
arrays are already processed.
# # However, if you were using .flow_from_directory for validation,
you would use it.
# # val_datagen = get_validation_test_generator()
```

Explanation of the Augmentation Parameters:
Each parameter within the ImageDataGenerator controls a specific type of random transformation:

- rotation_range=20: Images will be randomly rotated by an angle between -20 and +20 degrees.
- width_shift_range=0.2: Images will be randomly shifted horizontally by up to 20\% of their total width.
- height_shift_range=0.2: Images will be randomly shifted vertically by up to 20\% of their total height.
- shear_range=0.2: A shearing transformation is applied. This slants the image along an axis. A value of 0.2 means the shear intensity is 20\%.
- zoom_range=0.2: Images will be randomly zoomed in or out. A value of 0.2 means a zoom range of [0.8, 1.2] (i.e., 1 - 0.2 to 1 + 0.2).
- horizontal_flip=True: Images are randomly flipped horizontally. This is common for many image classification tasks, as patterns can often be symmetric.
- fill_mode='nearest': When transformations like rotation or shifting create new pixels (empty spaces), this parameter dictates how those spaces are filled. 'nearest' fills them with the value of the nearest existing pixel. Other options include 'reflect', 'wrap', or 'constant'.

These functions allow you to easily configure your data augmentation pipeline for training and ensure consistency for evaluation data.