Okay, let's analyze your data augmentation code and how it might relate to the issues you're seeing.

Understanding Your Data Augmentation

You're using tf.keras.preprocessing.image_dataset_from_directory to load your training and validation data. Here's what it does:

- **Loads images from directories:** It reads images from the directory structure you provide (train_copy_path, test_copy_path).
- **Resizes images:** It resizes the images to the specified image_size (img_rows, img_cols).
- **Creates batches:** It groups the images into batches of the given batch_size.
- **One-hot encodes labels:** It converts the class labels into a categorical (one-hot encoded) format, which is necessary for training a neural network.
- **Shuffles training data:** It shuffles the training data (but not the validation data).
- **Repeats training data:** You are repeating the training data indefinitely with train_datagen.repeat().

How This Relates to Your Problem

While image_dataset_from_directory is a convenient way to load data, it doesn't, by itself, perform any *data augmentation* in the sense of modifying the images (e.g., rotating, flipping, zooming). It primarily handles loading, resizing, batching, and labeling.

Potential Issues and How to Address Them

Here are the key areas to focus on, given your data loading and the problems you're encountering:

1. **Lack of Explicit Augmentation:**
   - **Problem:** You're not actually applying any data augmentation transformations to your images. This could limit the model's ability to generalize, but it's *unlikely* to be the primary cause of the *near-zero* performance you're seeing. A model with *no* augmentation should still perform *better* than random guessing, unless there is a more fundamental problem.
   - **Solution:** To apply data augmentation, you should use tf.keras.preprocessing.image.ImageDataGenerator. This class can perform a wide range of augmentations, such as rotations, flips, zooms, and shifts. You would typically use it in conjunction with model.fit.
2. **Image Size Consistency:**

- **Check:** Double-check that img_rows and img_cols are consistent with the image size you used during model training. If there is a mismatch, this *will* cause very poor performance. It's likely that you're using the correct size, but it's worth verifying.
- **Verify:** Add print statements to your code to confirm the image sizes:
  - Print img_rows and img_cols where they are defined.
  - Print img.shape of a sample image *before* it goes into the model in your prediction script.

3. **Class Order Consistency:**
   - This is still the MOST LIKELY PROBLEM. image_dataset_from_directory infers the class labels from the directory structure. It's crucial that the order in which it does this is the same as the order the model was trained on.
   - **Action:**
     - **Explicitly define the class names:** Instead of relying on the directory structure, provide an explicit list of class names to image_dataset_from_directory using the class_names argument. This will ensure that the classes are in the correct order. For example:

       ```
       train_datagen = tf.keras.preprocessing.image_dataset_from_directory(
           train_copy_path,
           image_size=(img_rows, img_cols),
           batch_size=batch_size,
           label_mode='categorical',
           shuffle=True,
           class_names = ['apple', 'banana', 'cherry', 'orange'] # Add ALL your classes in the correct order
       )
       ```

     - Ensure that the order you provide in class_names is *exactly* the same as the order you used during training.
4. **Preprocessing Consistency:**
   - image_dataset_from_directory does *not* apply the preprocessing that your VGG16 model expects (i.e., tf.keras.applications.vgg16.preprocess_input). You must apply this *before* passing images to your model for prediction.
   - **Action:**
     - In your prediction script, ensure that you are applying tf.keras.applications.vgg16.preprocess_input to the images *before* making predictions. This is ABSOLUTELY CRUCIAL.
5. **Batching in Prediction:**
   - Ensure that you are handling batching correctly in your prediction script.

VGG16 expects a batch of images as input, even if you are only predicting on a single image. You should expand the dimensions of your image array to create a batch of size 1.