When training a DiRA model based on the MoCo-v2 architecture, the file “main_DiRA_moco.py” must be called along with the required arguments. In this file, we see that the train and validation datasets are defined as follows:

```
dataset_train =  ChestX_ray14(pathImageDirectory=args.data, pathDatasetFile=args.train_list,
                        augment=transformation.Transform(mode=args.mode))

dataset_valid = ChestX_ray14(pathImageDirectory=args.data, pathDatasetFile=args.val_list,
                        augment=transformation.Transform(mode=args.mode))
```

Here, pathImageDirectory is equal to the user-specified path to the images folder, pathDatasetFile is set to be equal to the list of images to be used in training or validation, and augment is set to be equal to an object of class “Transform” from their included file, “transformation.py”. The overall call to ChestX_ray14 is a call to a class by the same name in their included file, “data_loader.py”, which retrieves all images and applies augmentations per the “augment” object.

In the above snippets of code, an object called “augment” of type “Transform” is created. This object then performs transformations on images in the “data_loader.py” file as follows:

```
return self.augment(imageData)
```

To understand exactly what augmentations are performed, we must examine the contents of the “transformation.py” file:

First, the image that is passed through is copied twice – let us call these copies image_1 and image_2. Both of these images are cropped according to the following:

```
RandomResizedCrop(224, scale=(0.2, 1.))
```

This rescales the images to size 224x224, while ignoring the lower 20% of the image (likely due to the fact that it is not useful in the prediction task – it is just the lower ribcage area). Directly after this, image_1 is copied again – let us call this image_3 – which is converted to greyscale, and normalized (with respect to pixel numeric values).

From here, image_1 and image_2 go through the following transformations:

- Colour Jitter is performed at a probability of 0.8 with the following parameters:
    - Brightness = 0.4
    - Contrast = 0.4
    - Saturation = 0.4

    - Hue = 0.1
- A custom Gaussian Blur is performed at a probability of 0.5 – this is functionally identical to a regular Gaussian Blur, but the kernel size is chosen automatically based on the value of sigma, which is between 0.1 and 2
- A horizontal flip is performed at a probability of 0.5

After these transformations, the program checks to see if the training mode is “Di”, indicating just a discriminative approach.

If the mode is “Di”, then image_1 and image_2 are turned into tensors and returned by themselves.

If the mode is anything else, then we perform the following custom transformations after turning the images into numpy arrays and selecting a random number between 0 and 1, which we will call rand_num:

- If rand_num is less than or equal to 0.3, cut-out is performed in a loop that iterates up to 10 times, or until another random number between 0 and 1 generated within the loop is less than 0.1:
    - The x and y dimensions of the region that will be cutout are chosen randomly to be between 10 and 70 – this is done separately, so that regions are rectangular
    - The starting x and y positions for the cutout are selected, ensuring that the cut-out does not exceed the dimensions of the image
    - The region defined by (starting x + size of x dimension cut-out, starting y + size of y dimension cut-out) is set to be equal to 0
- If rand_num is greater than 0.3 and less than equal to 0.35, a sort of reverse cutout is performed with the same loop structure:
    - A copy of the image is made, and the entire original image is set to be 0
    - The x and y dimensions of the regions that will be restored are chosen randomly to be between 50 and 70 – again, chosen separately
    - The starting x and y positions for the restoration are selected, ensuring that the restoration does not attempt to restore outside the bounds of the image
    - The original image that was set to be blank then has the defined region set to be equal to the region of the copied image (restoring that section)
- If rand_num is greater than 0.35 and less than or equal to 0.65, shuffling is performed 10 times:
    - A copy of the image is made
    - The x and y dimensions of the regions that will be shuffled are chosen randomly to be between 10 and 15 separately
    - Two starting x and y positions are chosen randomly, ensuring that they will not overlap when the shuffle is attempted
    - The original image then has the two calculated regions swapped by setting them equal to the opposite regions in the copied image
- If rand_num is greater than 0.65, no transformations take place

From here, image_1 and image_2 are set to be tensors, and they are returned along with image_3.

These are all transformations that are applied to an image in the “data_loader.py” file