```python
import time
import glob
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
from cnn_model import densenet121
import cv2
import os
from osgeo import gdal
import torch
import numpy as np
import torch.nn as nn
from torchvision import models
from torchvision.transforms import transforms

file_path = 'F:\Python\CJT.tif'
checkpoint_path = '../model/ResNet101/model.ckpt'
class Dataset:
    def __init__(self, file_path):
        self.file_path = file_path
        dataset = gdal.Open(self.file_path)
        self.XSize = dataset.RasterXSize
        self.YSize = dataset.RasterYSize
        self.img = dataset.ReadAsArray(0, 0, self.XSize, self.YSize)
        del dataset
        self.image = self.img[[2, 1, 0]].transpose(1, 2, 0)
        del self.img
        self.load_model()

    def load_model(self):
        #  Set up to use GPU devices
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        # Load model
        self.model = self.model = ResNet101()
        if torch.cuda.is_available():
            print("GPU")
            self.model.cuda()
        # Load weight parameter
        checkpoint = torch.load(checkpoint_path)
        self.model.load_state_dict(checkpoint)
        self.model.eval()

    def img_process(self, steps):
        # Setting the image pre-processing method
        transform = transforms.Compose([transforms.Resize(224),
                                        transforms.ToTensor(),
                                        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]), ])
        # Create a new diagram
        output_image = np.zeros((self.YSize, self.XSize), dtype=np.int8)
        # Calculate the rows and columns
        item_width = 280
        self.steps = steps
        rows = int((self.image.shape[0]-item_width) // self.steps)
        cols = int((self.image.shape[1]-item_width) // self.steps)
        print('Start:')
        print(str(self.steps) + '_img is mapping')
        num = 0
        k = 1
        self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)
        for i in range(0, rows):
            for j in range(0, cols):
                img = self.image[i * self.steps:(i * self.steps + item_width), j * self.steps:(j * self.steps + item_width), :]
                # if not (img == np.ones([item_width,item_width,3])*255).any():
                img = Image.fromarray(img)
                img = transform(img)
                img = torch.unsqueeze(img, 0)
                with torch.no_grad():
                    img = img.to(self.device)
                    outputs = self.model(img)
                    _, predicted = torch.max(outputs.data, 1)
                    if predicted == 0:
                        # Paste each image in order in the corresponding position
                        output_image[i * self.steps:(i * self.steps + item_width), j * self.steps:(j * self.steps + item_width)] += np.ones((item_width, item_width), dtype=np
                num += 1
                if int(num / (rows * cols) * 100) == k:
                    print('done --------{:.4f}%\n'.format(num / (rows * cols) * 100))
                    k += 1
                elif i == rows and j == cols:
                    print('done --------{}%'.format(100))
        print('--' * 20)
        print(str(self.steps)+'_img mapping is complete!')
        return output_image

steps = 140
# for i in range(len(steps)):
dataset = Dataset(file_path)
print(dataset.image.shape)
star_time = time.time()
output_image = dataset.img_process(steps=steps)
end_time = time.time()
# Computing time
t = np.round((end_time-star_time), 4)
output_image = output_image*255
cv2.imwrite(os.path.join('../output/mm.png'), output_image)
```