```python
import numpy as np
import cv2
from skimage.feature import local_binary_pattern
from sklearn.decomposition import PCA, KernelPCA

from tqdm import tqdm


class ImagePreprocessor:
    def __init__(self, normalize, hog_tf, pca_tf, lbp_tf, win_size=None, block_size=None,
                 block_stride=None, cell_size=None, nbins=None):
        self.normalize = normalize
        self.hog_tf = hog_tf
        self.pca_tf = pca_tf
        self.lbp_tf = lbp_tf
        self.win_size = win_size
        self.block_size = block_size
        self.block_stride = block_stride
        self.cell_size = cell_size
        self.nbins = nbins

    def scale(self, x_train, x_test):
        # normalize
        if self.normalize:
            x_train = x_train / 255
            x_test = x_test / 255

        # hog
        if self.hog_tf:
            # reshape as 28 x 28
            pix = self.win_size[0]
            x_train = x_train.reshape((x_train.shape[0], pix, pix, -1))
            x_test = x_test.reshape((x_test.shape[0], pix, pix, -1))

            x_train = self.hog_transform(x_train)
            x_test = self.hog_transform(x_test)

            # flatten
            x_train = x_train.reshape((x_train.shape[0], -1))
            x_test = x_test.reshape((x_test.shape[0], -1))

        # pca
        if self.pca_tf:
            x_train = self.pca_transform(x_train)
            x_test = self.pca_transform(x_test)

        # lbp
        if self.lbp_tf:
            x_train = self.lbp_transform(x_train)
            x_test = self.lbp_transform(x_test)

        return x_train, x_test

    def hog_transform(self, imgs):
        hog = cv2.HOGDescriptor(self.win_size, self.block_size, self.block_stride,
                                self.cell_size, self.nbins)
        features = []
        for img in imgs:
```

```
59              features.append(hog.compute(img))
60          return np.array(features)
61
62      def pca_transform(self, imgs):
63          pca = PCA(n_components=200, whiten=True)
64          pc = pca.fit_transform(imgs)
65
66          return pc
67
68      def lbp_transform(self, imgs):
69          n_imgs = imgs.shape[0]
70          imgs = imgs.reshape(n_imgs, 32, 32, 3)
71          features = []
72          for img in tqdm(imgs):
73              gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
74              feature = local_binary_pattern(gray_img, 22, 9, method='uniform')
75              features.append(feature)
76          features = np.array(features).reshape(n_imgs, -1)
77          return features
78
79
80  class TextPreprocessor:
81      def __init__(self):
82          super().__init__()
83
84      def scale(self, x_train, x_test):
85          return x_train, x_test
86
```