TKO_3120 Machine Learning and Pattern Recognition

Image recognition exercise

This is the template for the image recognition exercise. Some **general instructions**:

- write a clear *report*, understandable for an unspecialized reader: define shortly the concepts and explain the phases you use
  - use the Markdown feature of the notebook for larger explanations
- return your output as a working Jupyter notebook
- name your file as MLPR25_exercise_surname_firstname.jpynb
- write easily readable code with comments
  - if you exploit some code from web, provide a reference
  - avoid redundant code! Exploit the relevent parts and modify the code for your purposes to produce only what you need
- it is ok to discuss with a friend about the assignment. But it is not ok to copy someone's work. Everyone should submit their own implementation

Deadline 14th of March at 16:00

- No extension granted, unless you have an extremely justified reason. In such case, ask for extension well in advance!
- Start now, do not leave it to the last minute. This exercise will need some labour!
- If you encounter problems
  - Google first
  - ask for help at the discussion area at Moodle
  - email tmvaha@utu.fi

Grading

The exercise covers a part of the grading in this course. The course exam has 5 questions, 6 points of each. Exercise gives 6 points, i.e. the total score is 36 points. Two extra points can be acquired by completing the bonus task.

From the template below, you can see how many exercise points can be acquired from each task. Exam points are given according to the table below: 9-10 exercise points: 1 point 11-12 exercise points: 2 points 13-14 exercise points: 3 points 15-16 exercise points: 4 points 17-18 exercise points: 5 points 19-20 exercise points: 6 points   To pass the exercise, you need at least 8 exercise points, distributed somewhat evenly into tasks (you can't just implement Introduction, Data preparation and Feature extraction and leave the left undone!)

Introduction

Write an introductory chapter for your report **(1 p)** E.g.

- What is the purpose of this task?
- What kind of data were used? Where did it originate?
- Which methods did you use?

Three sets of image URLs provided as text files (in Moodle): grass, sand and stairs

Images are with different resolution and dimensions Images have been gathered from https://unsplash.com/

Purpose of the task is to explore different machine learning classifiers and see how well they could perform on a task where the point is to classify different kind of images (grass, sand and stairs). The images are from unsplash.com and they have different resolutions and dimensions, so they have to be resized so that every image is the same size.

The images need some preprocessing before training different ML models to classify them. Preprocessing methods that are used contains the resizing of the images like said before, but also grayscaling them and reducing quantization levels of the images. Then as a feature extraction method, RGB statistics and Gray-Level-Co-occurence Matrix (GLCM) are being used. To classify the images, the goal is to train three different ML-models, Ridge Classifier, Random Forest and Multi-Layer Perceptron (MLP), using different hyper parameters. The ML-models are trained and evaluated using Grid Search and Stratified Cross-validation, to see which hyper parameters work best for each model, and which model has the best overall performance.

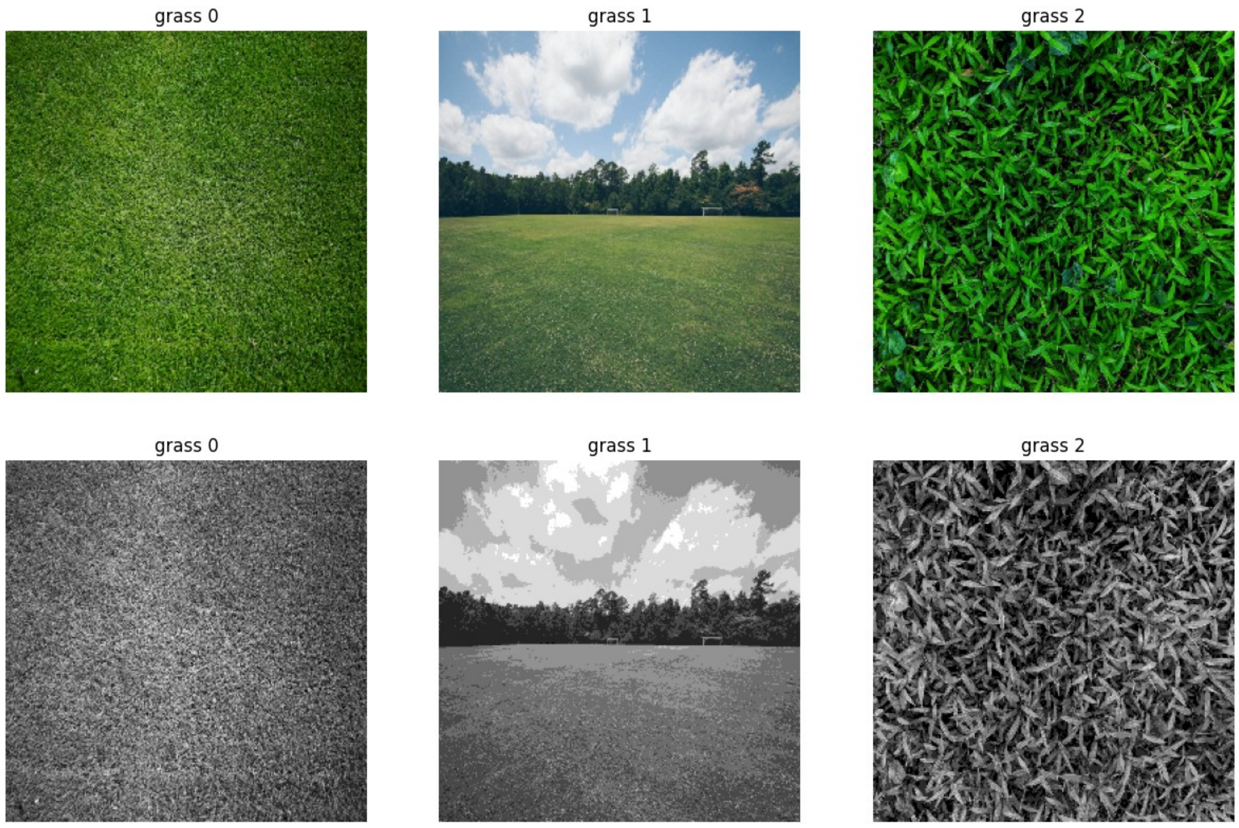
The code was mainly written with the help of documentation. There was a lot of trial and error, and some web searches where the main source for helpful information was geeksforgeeks.org and stackoverflow.com.

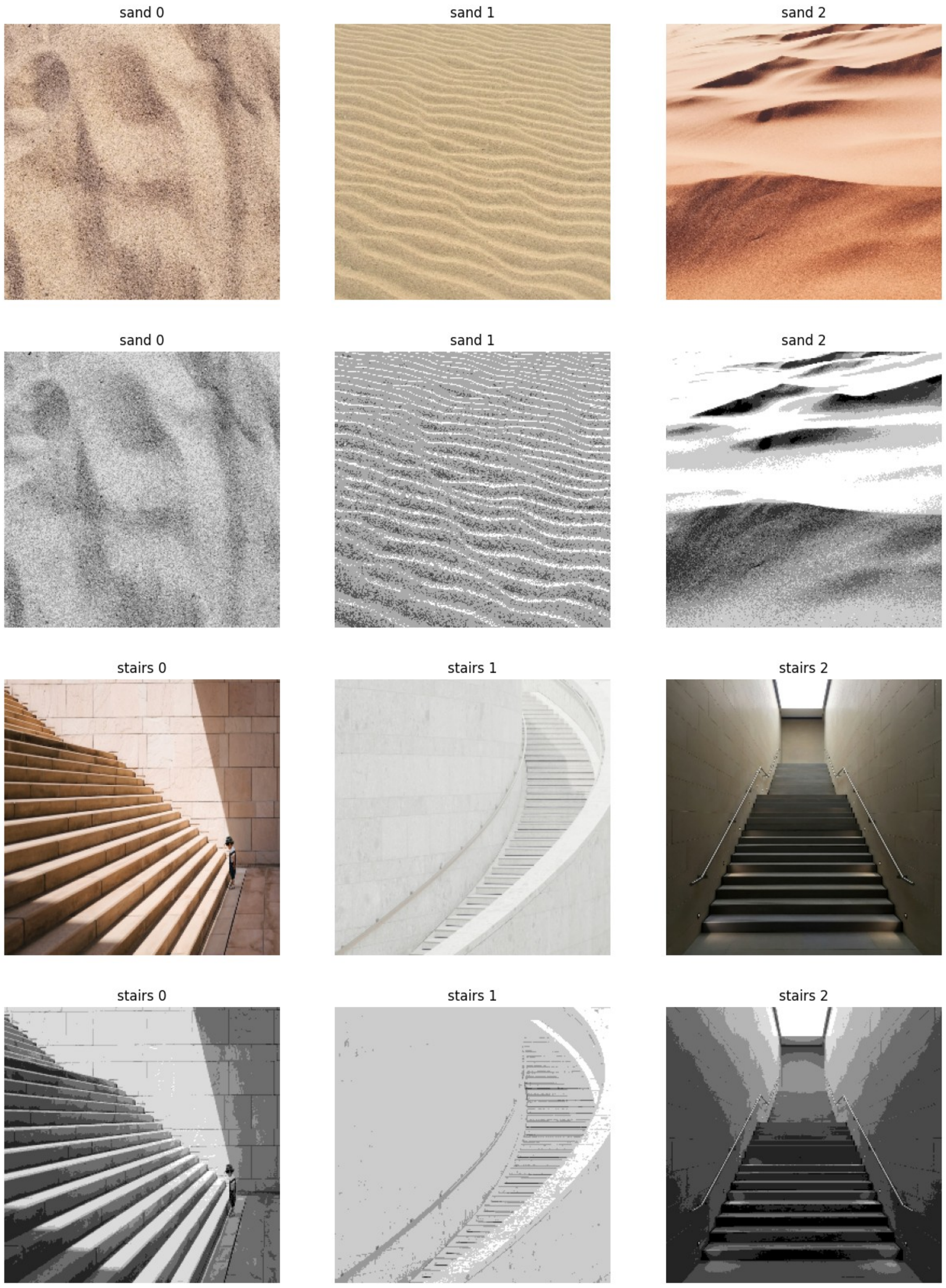
Data preparation

Perform preparations for the data **(3 p)**

- import all the packages needed for this notebook in one cell
- read the URL:s from the text files and import the images
- crop and/or resize the images into same size
- for GLCM and GLRLM, change the images into grayscale and reduce the quantization level, e.g. to 8 levels

```python
# Packages needed for the exercise
import os
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from skimage import color, feature
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import imageio.v2 as imageio

from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

def load_images(url_file, save_folder, resize_size=(256, 256),
quant_levels=8):
    """Loads images from URLs in a text file, preprocesses them, and
saves them."""

    os.makedirs(save_folder, exist_ok=True)
    urls = np.loadtxt(url_file, dtype='U150')  # Load URLs

    images = []
    processed_images = []

    for i, url in enumerate(urls):
        try:
            img = imageio.imread(url)
            img_resized = cv2.resize(img, resize_size) #resize
            img_grayscale = color.rgb2gray(img_resized) #grayscale
            img_quantized = np.round(img_grayscale * (quant_levels -
1)) / (quant_levels - 1) #quantize

            imageio.imwrite(os.path.join(save_folder,
f"image_{i}.jpg"), img_resized) #save resized image.

            images.append(img_resized)
            processed_images.append(img_quantized)

        except Exception as e:
            print(f"Error processing {url}: {e}")

    return images, processed_images #return both the color resized
images, and the processed grayscale images.

def visualize_images(images, titles):
    """Function that displays a few images to ensure correct
preprocessing"""
    fig, axes = plt.subplots(1, len(images), figsize=(15,5))
    for ax, img, title in zip(axes, images, titles):
        ax.imshow(img, cmap='gray')
        ax.set_title(title)
        ax.axis('off')
    plt.show()

# Define file paths and perform preprocessing for the images

categories = ['grass', 'sand', 'stairs']
all_images_resized = {}
all_images = {}

for cat in categories:
    url_file = f"{cat}.txt"
    save_folder = f"images/{cat}"
    resized_images, processed_images = load_images(url_file,
save_folder) #load and preprocess
    all_images_resized[cat] = resized_images
    all_images[cat] = processed_images

# Visualize a sample of images (showing both resized and fully
preprocessed versions)

for cat in categories:
    visualize_images(all_images_resized[cat][:3], [f"{cat} {i}" for i
in range(3)])
    visualize_images(all_images[cat][:3], [f"{cat} {i}" for i in
range(3)])
```




sand 0
sand 1
sand 2
sand 0
sand 1
sand 2
stairs 0
stairs 1
stairs 2
stairs 0
stairs 1
stairs 2

Feature extraction

First order texture measures (6 features)

- Calculate the below mentioned color features for each image **(1 p)**
  - Mean for each RGB color channel
  - Variance for each RGB color channel

```
def compute_rgb_stats(images):
    """Function that computes mean and variance for each RGB channel
for each image"""
    rgb_stats = []
    for img in images:
        means = np.mean(img, axis=(0, 1))
        variances = np.var(img, axis=(0, 1))
        rgb_stats.append((means, variances))
    return rgb_stats


# Running the function and printing the results
all_rgb_stats = {}

for cat in categories:
    all_rgb_stats[cat] = compute_rgb_stats(all_images_resized[cat])
#using resized color images

for category, stats in all_rgb_stats.items():
    print(f"Category: {category}")
    for i, (means, variances) in enumerate(stats):
        print(f" Image {i}: Mean={means}, Variance={variances}")
```

```
Category: grass
 Image 0: Mean=[ 65.17550659 102.69487     26.87080383],
Variance=[ 994.60558049 1159.78737358  617.09956506]
 Image 1: Mean=[120.70666504 140.40126038 124.32608032],
Variance=[4169.84086499 3309.53480615 5471.86825048]
 Image 2: Mean=[27.3782196  89.64558411 16.60505676],
Variance=[ 527.52948106 2890.56083651  419.10535712]
 Image 3: Mean=[101.56672668 118.17645264  22.62190247],
Variance=[1080.76364447 1236.52883151  560.88614077]
 Image 4: Mean=[108.63632202 152.2432251   39.01411438],
Variance=[2162.49768218 1542.96195971 1488.15273963]
 Image 5: Mean=[ 96.64616394 103.82754517  56.37017822],
Variance=[2403.57171471 2434.82630792  723.02748835]
 Image 6: Mean=[ 86.97601318 140.8813324  116.24606323],
Variance=[2058.55020588  331.12268253 7559.26940894]
 Image 7: Mean=[125.5919342  116.36338806  34.8469696 ],
Variance=[3617.15765528 3560.02088796  922.68204129]
 Image 8: Mean=[76.23397827 93.09941101 33.98083496],
Variance=[2032.44049343 2421.11818447  717.04226576]
```

```
 Image 9: Mean=[ 53.2088623  176.58050537  30.22225952],
Variance=[716.07585286 701.23564535 578.66892346]
 Image 10: Mean=[ 76.14782715 121.88156128  50.80026245],
Variance=[1107.31811295  877.90397154  852.42125603]
 Image 11: Mean=[123.74987793 154.61456299  92.39741516],
Variance=[3573.08197019 3019.72268464 7787.5551502 ]
 Image 12: Mean=[ 94.08889771 141.69421387  64.98841858],
Variance=[1824.26861697 1955.81878732  266.66113601]
 Image 13: Mean=[56.45988464 91.01847839 62.49740601],
Variance=[2048.68616541 2266.36802097 1877.52932067]
 Image 14: Mean=[148.78361511 163.00804138  81.85089111],
Variance=[3246.06198801 1729.1902277  6404.93778852]
 Image 15: Mean=[131.0774231  165.96540833  63.53993225],
Variance=[3089.676867   1894.06211213  450.29549404]
 Image 16: Mean=[ 46.18838501 121.66732788  72.11445618],
Variance=[ 877.66000669 1024.64329655  636.53757422]
 Image 17: Mean=[73.84268188 88.42097473 39.78643799],
Variance=[1869.1512764  2019.29008069 1408.42704752]
 Image 18: Mean=[102.21130371 137.74543762  39.37017822],
Variance=[2030.61044107 1720.56381433  254.47991144]
 Image 19: Mean=[138.11172485 146.96035767 129.95927429],
Variance=[4313.34262009 4028.4800569  7990.93234714]
 Image 20: Mean=[123.30905151 168.6885376   80.87565613],
Variance=[555.65216783 341.69943893 430.85027652]
 Image 21: Mean=[ 67.94949341 144.46072388  42.63983154],
Variance=[ 892.57713048 1184.3029923  1521.84659704]
 Image 22: Mean=[ 42.2520752  104.35881042  19.48117065],
Variance=[1574.79598568 1426.00685994 1012.68082832]
 Image 23: Mean=[20.92672729 36.06211853  8.38043213],
Variance=[ 567.13006812 1002.25253472  221.87892252]
 Image 24: Mean=[106.04910278 135.58796692  83.36714172],
Variance=[1956.86096172 2197.49528306 1552.7829163 ]
 Image 25: Mean=[ 72.7035675  103.63079834  46.39054871],
Variance=[1671.03653268 2262.07667231  728.3010087 ]
 Image 26: Mean=[ 77.99256897 109.7648468   32.07159424],
Variance=[3171.55742464 4470.64302145 1292.91583374]
 Image 27: Mean=[82.70617676 90.90867615 53.90632629],
Variance=[2946.16299651 2746.23892863 1578.31149196]
 Image 28: Mean=[ 8.87191772 50.06257629 14.69122314],
Variance=[ 330.2377148  2114.00852012  668.06347033]
 Image 29: Mean=[117.15226746 167.99388123  25.8710022 ],
Variance=[1019.74419438 1119.19931864  486.88320087]
 Image 30: Mean=[144.79928589 161.03482056  66.82658386],
Variance=[4018.08077708 3384.48835052 1984.48431588]
 Image 31: Mean=[ 78.3256073  111.76875305  20.03955078],
Variance=[476.73374734 576.64389729 373.32143378]
 Image 32: Mean=[60.76809692 71.45506287 21.45857239],
Variance=[2139.86888942 1959.34478241  410.59267462]
 Image 33: Mean=[137.51359558 149.63684082  54.03659058],
Variance=[4416.38070505 3524.18784807 3610.62491235]
 Image 34: Mean=[137.59558105 132.78437805  93.52784729],
Variance=[2415.01460693 2044.95654368 2726.05626189]
 Image 35: Mean=[160.62115479 163.31886292 138.51626587],
Variance=[2822.84335374 2978.45144229 8991.42652375]
 Image 36: Mean=[80.05635071 89.58171082 66.08270264],
Variance=[4644.74967799 3076.24616147 3262.92678454]
 Image 37: Mean=[93.24650574 86.23893738 53.73622131],
Variance=[2197.51331634 1867.94152038 2038.68559354]
 Image 38: Mean=[ 92.745224   112.32450867  49.58984375],
Variance=[4030.87324531 3861.05327872 4601.14228821]
 Image 39: Mean=[133.70562744 152.32659912 119.27418518],
Variance=[2608.49821418 3092.01152759 7629.84042611]
 Image 40: Mean=[125.96302795 109.48245239  70.34893799],
Variance=[2018.64134852 1233.66912567 1058.60166147]
 Image 41: Mean=[ 93.60733032 118.55802917  58.72695923],
Variance=[3598.38594114 2853.81419853 1327.09607252]
 Image 42: Mean=[90.95950317 96.53120422 41.91120911],
Variance=[2921.10443911 2913.33905315 1331.4156849 ]
 Image 43: Mean=[148.33659363 145.52159119 107.40974426],
Variance=[4977.09661988 4432.21749036 7839.22946377]
 Image 44: Mean=[71.07865906 74.3742218  48.74539185],
Variance=[873.8924181  845.83644425 237.13750623]
 Image 45: Mean=[93.54562378 96.1530304  13.15490723],
Variance=[1280.90957008 1047.53784574  325.71294223]
 Image 46: Mean=[87.86526489 98.49838257 70.84008789],
Variance=[2125.81650221 1767.16341902 1135.91314882]
 Image 47: Mean=[83.72543335 93.54856873 58.20845032],
Variance=[4366.58617321 4352.78636788 4073.60350587]
 Image 48: Mean=[116.16567993 127.87133789 100.42495728],
Variance=[2537.86769176 2819.74724001 5843.61631682]
 Image 49: Mean=[47.62254333 81.64761353 23.70121765],
Variance=[1384.58691184 2468.36169414  820.69855565]
 Image 50: Mean=[145.92681885 148.90214539 107.74934387],
Variance=[1002.4635471   720.69542631  796.44814258]
 Image 51: Mean=[108.38806152 134.47953796 114.30955505],
Variance=[2548.61170684 2983.21585938 7017.84495632]
 Image 52: Mean=[99.93748474 92.8449707  59.14376831],
Variance=[1325.23103142 1000.51963657 2319.01827111]
 Image 53: Mean=[171.81872559 148.50189209  83.90382385],
Variance=[332.14786468 384.23775887 563.68528445]
 Image 54: Mean=[ 99.9708252  118.09959412 111.95367432],
Variance=[2927.67200039 3405.05497664 7579.64272087]
 Image 55: Mean=[142.6100769  157.04911804 141.92012024],
Variance=[ 5939.45605324  7164.92841932 12813.22630051]
 Image 56: Mean=[ 76.81343079 104.64700317  85.47399902],
Variance=[2343.32241544 3466.99218401 2606.17653952]
 Image 57: Mean=[70.71156311 68.55354309 43.64897156],
Variance=[1865.10679134 1657.45080746 1174.46932359]
```

```
 Image 58: Mean=[119.03115845 115.80487061  95.25059509],
Variance=[3631.40048789 4059.40418526 5231.32299007]
 Image 59: Mean=[82.68525696 83.260849   56.1499176 ],
Variance=[2479.74985955 2484.31362589 1552.8209848 ]
 Image 60: Mean=[ 70.92622375 151.92733765 131.62538147],
Variance=[ 3460.16052692  2423.57498141 12011.38030608]
 Image 61: Mean=[148.64073181 128.8968811   87.3684082 ],
Variance=[3885.16796921 2667.90895267 3824.37455982]
Category: sand
 Image 0: Mean=[197.64067078 168.76818848 146.72042847],
Variance=[880.54707575 916.34619628 827.91839127]
 Image 1: Mean=[203.2212677  178.97331238 141.67863464],
Variance=[220.49432979 206.94809453 182.40705451]
 Image 2: Mean=[209.29985046 155.79611206 127.2434845 ],
Variance=[1740.21354124 2435.32085647 2264.09203671]
 Image 3: Mean=[179.01112366 130.53225708  82.89756775],
Variance=[1166.92881609  968.13204664  820.28877827]
 Image 4: Mean=[204.18753052 186.99484253 175.40589905],
Variance=[3572.38402176 3346.37491237 4138.21361815]
 Image 5: Mean=[179.62693787 175.6418457  167.01971436],
Variance=[968.23385626 869.40837711 843.56848952]
 Image 6: Mean=[141.18392944 112.02810669  98.49250793],
Variance=[1450.74760307 1279.65274395  888.66342654]
 Image 7: Mean=[164.87820435 153.73675537 128.57157898],
Variance=[2290.48037456 2112.90018468 1530.4140659 ]
 Image 8: Mean=[192.60292053 185.86630249 181.27357483],
Variance=[ 518.70330507  640.14554661 1101.7529919 ]
 Image 9: Mean=[180.36584473 156.3400116  135.12669373],
Variance=[ 911.3190385  2117.1028522  3293.60795688]
 Image 10: Mean=[170.8265686  122.760849    92.39538574],
Variance=[1437.61228605 1097.88764628  758.4938166 ]
 Image 11: Mean=[209.61230469 189.51223755 142.92541504],
Variance=[484.693717   483.23498818 656.58736311]
 Image 12: Mean=[102.19995117  97.53816223  94.48564148],
Variance=[2307.48452514 2059.68091669 1667.82172377]
 Image 13: Mean=[206.37413025 188.52276611 153.30577087],
Variance=[ 21.29067536  39.59655812 110.9424081 ]
 Image 14: Mean=[140.28546143 110.550354    80.58854675],
Variance=[1350.63628521 1021.31014758  511.53216802]
 Image 15: Mean=[164.86343384 169.30140686 174.58346558],
Variance=[ 794.65703938  769.24242112 1417.76525029]
 Image 16: Mean=[240.83924866 232.61247253 227.98545837],
Variance=[25.60411811 27.14970344 35.02787997]
 Image 17: Mean=[210.69203186 200.97344971 191.04875183],
Variance=[342.85442015 316.49517521 962.72258359]
 Image 18: Mean=[168.64073181 171.33656311 173.51933289],
Variance=[ 705.48458909  836.39729961 1060.37999733]
 Image 19: Mean=[138.43199158 124.81881714 121.991745  ],
Variance=[465.31849497 525.17991081 716.5106893 ]
```

```
 Image 20: Mean=[105.34742737 100.39781189  91.03889465],
Variance=[3254.7949589  2018.43508371 1446.37316677]
 Image 21: Mean=[149.6709137  127.05984497 106.13156128],
Variance=[593.29158929 596.73628674 605.89663206]
 Image 22: Mean=[148.32920837 127.90274048 104.52159119],
Variance=[1624.83551528 1552.82385357 1246.16295545]
 Image 23: Mean=[190.31480408 164.22714233 141.35786438],
Variance=[543.59076351 485.13596739 396.69680796]
 Image 24: Mean=[191.93711853 182.79118347 165.9370575 ],
Variance=[2199.73089395 1238.74611429 1249.57686405]
 Image 25: Mean=[ 90.77832031  92.09313965 105.92230225],
Variance=[2014.28215694 1072.21077691  826.34988957]
 Image 26: Mean=[163.01998901 163.83578491 157.18232727],
Variance=[2997.75298789 3516.12311886 3941.7168941 ]
 Image 27: Mean=[148.40005493 103.18133545  66.96788025],
Variance=[957.35036255 944.03904738 622.4472868 ]
 Image 28: Mean=[144.72566223 122.17008972 103.77934265],
Variance=[3920.5309247 3857.9499971 4745.8808605]
 Image 29: Mean=[141.05067444 137.48257446 132.74751282],
Variance=[3089.42499863 3525.53671418 4993.23533776]
 Image 30: Mean=[150.70574951 140.7381134  131.0151062 ],
Variance=[ 212.20940664  556.53726563 1362.78840706]
 Image 31: Mean=[204.41307068 189.09472656 167.83303833],
Variance=[ 604.97852728  464.7323904  1239.16789406]
 Image 32: Mean=[133.5993042  121.95803833 110.77824402],
Variance=[2561.4408833  2294.74198311 2430.41321137]
 Image 33: Mean=[159.37843323 165.36399841 154.78134155],
Variance=[4334.61806454 1903.44357022 1815.210886  ]
 Image 34: Mean=[85.47067261 83.26908875 98.0894928 ],
Variance=[ 678.18868458  547.75115387 1149.03378486]
 Image 35: Mean=[192.45915222 181.9234314  147.55827332],
Variance=[456.68580094 573.26473661 934.2469094 ]
 Image 36: Mean=[157.07003784 116.15129089  95.18304443],
Variance=[236.75855295 201.50608751 177.06143492]
 Image 37: Mean=[116.22605896 109.84883118 104.57411194],
Variance=[3358.73992824 3308.0540065  2863.55108701]
 Image 38: Mean=[167.40454102 134.04960632  91.82614136],
Variance=[344.82178968 306.04116409 212.27974021]
 Image 39: Mean=[114.76925659  78.63705444  67.26324463],
Variance=[2469.22776334 1796.95738185 3227.1696549 ]
 Image 40: Mean=[161.70343018 138.37788391 109.07772827],
Variance=[8670.65392867 6286.07145241 4107.20675739]
 Image 41: Mean=[165.49079895 186.43383789 198.93824768],
Variance=[3465.05387042  497.90349001  595.83690015]
 Image 42: Mean=[122.68699646  75.60864258  46.76083374],
Variance=[890.30212949 648.53034157 278.79790204]
 Image 43: Mean=[139.32754517 143.16079712 144.741745  ],
Variance=[2995.43379082 3913.46581299 5068.09991507]
 Image 44: Mean=[162.28530884 131.79670715  96.19549561],
Variance=[1503.76192162 3162.34339187 4704.31367966]
 Image 45: Mean=[171.68280029 159.42562866 138.3624115 ],
Variance=[193.38208088 315.58693717 549.21251472]
 Image 46: Mean=[185.73886108 175.08332825 174.53343201],
Variance=[2942.71272077 1696.8172599  1360.12397386]
 Image 47: Mean=[146.07331848 109.57104492  77.91680908],
Variance=[1934.34904029 1474.65565819  842.71979436]
 Image 48: Mean=[77.71893311 86.86607361 83.78042603],
Variance=[2467.13950726 1207.56511731  955.72028092]
 Image 49: Mean=[147.18106079 103.59765625  83.21980286],
Variance=[659.93047383 198.80845642  98.63959869]
 Image 50: Mean=[103.76994324 110.53735352 113.12628174],
Variance=[2500.73194022 1832.20032591 2043.63029315]
 Image 51: Mean=[61.33366394 69.02778625 82.05461121],
Variance=[1123.47816117  837.07889406  647.54390788]
 Image 52: Mean=[126.2190094 109.3059845  89.3167572],
Variance=[6488.19216244 5828.85844562 6165.83037471]
 Image 53: Mean=[142.33261108 106.87324524  99.83886719],
Variance=[984.4875144  383.89450147 368.35971737]
 Image 54: Mean=[119.41697693 119.26315308 119.05377197],
Variance=[1951.57321581 1386.98232516 1076.45182049]
 Image 55: Mean=[180.36734009 113.26086426  62.29112244],
Variance=[3087.29484032 1312.07809852  252.46235161]
 Image 56: Mean=[106.10816956  86.00219727  85.88169861],
Variance=[1980.91293619 1827.89705938 1761.58013625]
 Image 57: Mean=[125.99386597 147.97544861 143.07676697],
Variance=[9105.37807517 1686.68950648  630.33750588]
 Image 58: Mean=[218.29270935 210.17460632 193.37632751],
Variance=[ 296.60867243  606.37989776 1335.03325858]
 Image 59: Mean=[150.12765503 143.1751709  131.87776184],
Variance=[1824.73565732 1692.07402097 2207.24197007]
 Image 60: Mean=[162.95967102 186.39790344 217.27615356],
Variance=[5003.32929094 2126.06306262  830.25729634]
Category: stairs
 Image 0: Mean=[163.46815491 136.81117249 119.68351746],
Variance=[3877.45690215 4267.45691314 4869.31132256]
 Image 1: Mean=[213.73921204 213.85095215 211.08587646],
Variance=[164.16381642 167.21286164 171.80253124]
 Image 2: Mean=[90.98168945 86.65829468 75.74549866],
Variance=[2823.28790325 2671.80618059 2591.93451191]
 Image 3: Mean=[44.73443604 50.37136841 52.20704651],
Variance=[4599.15408515 3449.18813532 2785.17684305]
 Image 4: Mean=[94.96392822 98.17088318 95.58172607],
Variance=[4898.49393808 3503.14366576 3287.42602959]
 Image 5: Mean=[134.29216003 105.41482544  86.21325684],
Variance=[2406.27317828 1485.38395013  319.24004398]
 Image 6: Mean=[123.2822113  118.18960571 117.59585571],
Variance=[3132.63185886 3018.91055236 2881.5642675 ]
 Image 7: Mean=[233.33728027 203.17819214 156.53175354],
Variance=[  497.7501031   2285.11509815 11200.34515259]
 Image 8: Mean=[217.64761353 215.29472351 214.75518799],
Variance=[2203.50650005 2467.44251122 2859.31467035]
 Image 9: Mean=[249.02326965 145.96946716  63.71859741],
Variance=[ 201.12096426 2327.13174292 4218.79034994]
 Image 10: Mean=[182.27528381 176.99606323 175.82839966],
Variance=[3178.09772651 3887.93766761 4570.90582554]
 Image 11: Mean=[213.95581055 216.25614929 212.73794556],
Variance=[1346.03936809 1310.22810397 1495.68395198]
 Image 12: Mean=[96.02738953 95.52622986 96.37799072],
Variance=[2302.64162957 1837.06123216 1976.33655416]
 Image 13: Mean=[142.78971863 146.16908264 138.78720093],
Variance=[3759.91383533 3382.73262627 4176.16641699]
 Image 14: Mean=[178.74801636 182.66860962 182.88235474],
Variance=[4268.3523978  4219.53159888 4342.02900627]
 Image 15: Mean=[129.04789734 128.5002594  128.0793457 ],
Variance=[3794.47370377 3890.2266845  3781.68794864]
 Image 16: Mean=[152.53562927 159.21311951 157.53923035],
Variance=[4989.47956552 5115.99213196 5414.61570951]
 Image 17: Mean=[80.75592041 92.78955078 96.70349121],
Variance=[4018.98870396 2498.82643867 3111.63968263]
 Image 18: Mean=[92.40333557 85.07574463 73.58251953],
Variance=[1797.58376512 1974.91961675 2301.32308555]
 Image 19: Mean=[112.78735352 113.19784546 118.85113525],
Variance=[4106.62384886 4066.78214623 4107.90676384]
 Image 20: Mean=[131.86221313 124.29083252 119.25274658],
Variance=[4495.7942472  4532.4870412  5487.05681619]
 Image 21: Mean=[71.20906067 82.96147156 65.66021729],
Variance=[1441.31904083 1175.87313409  807.88637876]
 Image 22: Mean=[76.82322693 85.83618164 84.47624207],
Variance=[2272.34172186 2058.54359812 1941.91719069]
 Image 23: Mean=[149.20288086 153.23643494 151.7512207 ],
Variance=[5805.05204004 5956.23796753 5900.08645481]
 Image 24: Mean=[73.12843323 77.02780151 66.28996277],
Variance=[2721.14788784 2042.03075173 2146.29966488]
 Image 25: Mean=[162.01913452 162.590271   151.07469177],
Variance=[4435.24252327 4326.79025202 6032.29656043]
 Image 26: Mean=[100.46746826  82.91090393  81.65875244],
Variance=[2825.99241092  619.29041699   96.04468292]
 Image 27: Mean=[194.95988464 179.49404907 155.69764709],
Variance=[2502.48851832 2522.32189452 3024.54116634]
 Image 28: Mean=[159.45837402 156.55603027 148.26333618],
Variance=[2817.88037787 3041.37588892 3994.99693824]
 Image 29: Mean=[77.40879822 83.84153748 62.82044983],
Variance=[2165.65095348 2410.06337535 2102.7033727 ]
 Image 30: Mean=[137.58680725 122.94477844 109.34187317],
Variance=[2854.9669518  3104.1026177  3134.06444292]
 Image 31: Mean=[141.96217346 188.23669434 214.51139832],
Variance=[1998.80784955 1444.39386593 2356.6709211 ]
```

```
 Image 32: Mean=[56.98509216 54.61329651 51.26312256],
Variance=[1670.59241387 1496.04270589 1600.20698112]
 Image 33: Mean=[ 93.52233887 105.79701233 113.51382446],
Variance=[2421.98085473 2164.09580467 1961.80870781]
 Image 34: Mean=[35.82521057 35.82521057 35.82521057],
Variance=[2878.65559062 2878.65559062 2878.65559062]
 Image 35: Mean=[82.35855103 79.39906311 67.60639954],
Variance=[3895.45328931 3099.62930759 2896.53912469]
 Image 36: Mean=[98.48344421 92.69560242 73.53961182],
Variance=[2567.21972712 1544.58768574 1169.34181469]
 Image 37: Mean=[ 98.05860901 101.14811707 100.44831848],
Variance=[1515.75262272 1412.83736004 1840.98216178]
 Image 38: Mean=[144.15283203 144.15283203 144.15283203],
Variance=[3821.74299979 3821.74299979 3821.74299979]
 Image 39: Mean=[84.5574646  92.87744141 94.8160553 ],
Variance=[4244.81527082 4206.67553115 4000.68651652]
 Image 40: Mean=[140.30867004 124.08898926 104.24687195],
Variance=[3857.12266408 3989.9142916  4278.11161588]
 Image 41: Mean=[56.34840393 56.88166809 50.81637573],
Variance=[3687.66842489 3789.72963465 3418.81021523]
 Image 42: Mean=[82.77679443 80.13110352 77.35809326],
Variance=[4333.37864607 4496.74495786 4779.07071331]
 Image 43: Mean=[61.23774719 73.17965698 75.15061951],
Variance=[1738.09061677 1828.30356932 1805.91577507]
 Image 44: Mean=[ 98.77696228  98.45779419 105.49307251],
Variance=[4526.10288784 4492.71236052 4819.30430381]
 Image 45: Mean=[80.9041748  76.45503235 73.57928467],
Variance=[5930.05145595 5712.65557068 5451.23230038]
 Image 46: Mean=[43.92709351 59.12120056 57.23750305],
Variance=[4689.95009846 3961.58063208 2769.47467445]
 Image 47: Mean=[147.58534241 111.04490662  73.95672607],
Variance=[3361.64298523 3812.86970276 4490.44871941]
 Image 48: Mean=[141.84007263 108.87684631  83.71852112],
Variance=[1968.8471802  1630.92799418 1457.5697412 ]
 Image 49: Mean=[40.47967529 58.2953949  68.14456177],
Variance=[3915.7829121  4073.22940751 4003.33637118]
 Image 50: Mean=[104.98545837 100.51754761  86.61909485],
Variance=[7206.66679599 5847.89843476 4952.56931251]
 Image 51: Mean=[93.2794342  69.75875854 42.37174988],
Variance=[2762.64611392 2776.19039875 3020.58099453]
 Image 52: Mean=[188.73970032 192.21965027 196.89309692],
Variance=[816.4851463  765.6961843  741.02995357]
 Image 53: Mean=[130.3561554  155.12373352 151.91488647],
Variance=[3321.85128749 2762.63601143 3009.44047908]
 Image 54: Mean=[145.31744385 136.625       96.64712524],
Variance=[5091.59109341 4894.25350952 2452.88155241]
 Image 55: Mean=[51.52584839 53.85643005 55.34396362],
Variance=[1549.84079304 1781.09143296 1957.74512286]
 Image 56: Mean=[167.34333801 172.60073853 173.0670929 ],
Variance=[2931.76509936 2137.37071113 2901.38705433]
 Image 57: Mean=[141.56231689 146.32931519 151.76556396],
Variance=[ 9423.94408414 10141.79108765 10448.87616157]
 Image 58: Mean=[187.04212952 187.17404175 187.22917175],
Variance=[842.39234437 837.07291992 829.31092941]
 Image 59: Mean=[130.30703735 139.85528564 143.82792664],
Variance=[7014.63857719 5856.17211196 5536.54853102]
 Image 60: Mean=[202.45205688 193.45727539 145.40748596],
Variance=[1016.95750981 2086.07642168 9504.11351148]
 Image 61: Mean=[119.71203613 113.46075439 104.42323303],
Variance=[5466.54296181 3562.42509552 3060.03954751]
```

Second order texture measures (4 features)

- Gray-Level-Co-Occurrence (GLCM) features (4 features) **(2 p)**
  - For each image
    - calculate the GLC matrix
    - calculate the "correlation" feature using the GLC matrix that you acquired
      - in horizontal and vertical directions for two reference pixel distances (you can choose the distances)
    - explain your choise for the distances

Gather your features into an input array X, and the image classes into an output array y. Standardize the feature values in X.

```python
def compute_glcm_features(images, distances=[1, 3]):
    """Function that calculates the GLC matrix and then calculates the 
correlation feature"""
    features = []
    glcm = []
    for img in images:
        glcm = feature.graycomatrix((img * 255).astype(np.uint8), 
distances=distances, angles=[0, np.pi / 2], levels=256, 
symmetric=True, normed=True)
        corr_matrix = feature.graycoprops(glcm, 'correlation')
        corr_features = corr_matrix.flatten().tolist()
        features.append(corr_features)
    return features

# Perform GLCM correlation feature calculation, store into array X 
with class labels in array y
all_glcm_features = []
y = []


for cat in categories:
    glcm_features = compute_glcm_features(all_images[cat]) #Using the 
processed images
    all_glcm_features.extend(glcm_features)
    y.extend([cat] * len(glcm_features))

# Convert into NumPy array, standardize and store in X
X_not_standardized = np.array(all_glcm_features)
scaler = StandardScaler()
X = scaler.fit_transform(X_not_standardized)

print(X)
```

```
[[-2.87745775 -2.73971524 -2.39000028 -2.19509225]
 [ 0.63927832  0.63932968  0.84178148  0.88780036]
 [-1.13687065 -0.88091814 -2.35882595 -1.9756754 ]
 [-1.27232485 -0.49008672 -1.46689439 -0.9972423 ]
 [ 0.21528607  0.68875333  0.04083544  0.72021857]
 [-0.46375713  0.40190198 -0.12798951  0.19473963]
 [ 0.34755662  0.11776607  0.53733786  0.05217871]
 [-3.5466807  -3.76185133 -2.96196331 -2.86062334]
 [ 0.11032975  0.67460715 -0.73383463  0.53967069]
 [-0.15419951 -0.20803802  0.0176529  -0.19098505]
 [-1.38619    -0.86267065 -1.05502729 -0.92290862]
 [ 0.70426862  0.49819834  0.86186526  0.59809717]
 [ 0.67770214  0.67513021  0.91659531  0.96509342]
 [-1.61234021 -1.93884148 -1.90601458 -1.94061809]
 [ 0.48145931  0.44901505  0.38158099  0.50207907]
 [ 0.36445966  0.49178231 -0.03970349  0.27130374]
 [-0.16917933  0.76966643 -1.2277425   0.79711124]
 [ 0.03839057  0.12934997  0.06011518 -0.00789755]
 [-0.04731009  0.6715589  -0.40691287  0.65115964]
 [ 0.41325542  0.52341606  0.50744918  0.70408042]
 [-0.49040943  0.67330035 -1.2615448   0.62400134]
 [-1.98126912 -0.49620894 -1.92430037 -1.04579687]
 [-0.31860238  0.78702993 -1.49604465  0.87112232]
 [-0.18136775  0.06542071 -1.04762142 -0.51040295]
 [-0.59098806 -0.19553709 -0.98909413 -0.16986887]
 [ 0.08038169  0.61550487 -0.74317017  0.46892951]
 [ 0.06528669  0.86879463 -0.63907505  0.97523551]
 [-0.53438805  0.34220409 -0.1452904   0.34829122]
 [ 0.19565473  0.22518533 -0.29849596 -0.16338083]
 [-1.24203851 -1.77589709 -1.18334074 -1.27604317]
 [ 0.0744154   0.6760743  -1.02608576  0.35688567]
 [ 0.09821205  0.61688283 -0.05102062  0.59161198]
 [-2.67583232 -2.42775515 -2.12761509 -2.09471947]
 [-0.18190077 -0.04812324 -0.099999   -0.13668899]
 [-0.21246027 -0.11085053 -0.06078392  0.11354686]
 [ 0.63842796  0.56077079  0.72688916  0.66997611]
 [ 0.54008367  0.60039375  0.30960694  0.46122339]
 [ 0.52433804  0.14693158  0.29017365  0.05884319]
 [ 0.76627116  0.70807009  0.88266971  0.82066002]
 [ 0.4829977   0.46364615  0.6487384   0.70705448]
 [-0.6422023  -0.80942601 -0.93101628 -0.87453516]
 [-0.05353497  0.53972484  0.30235636  0.65202599]
 [-0.6580921  -0.54281361 -0.78545636 -0.87953072]
 [ 0.45859509  0.57939062  0.62501155  0.83740148]
 [ 0.27115288  0.44263673  0.14033186  0.4192658 ]
 [-0.90200622 -0.03120261 -0.96352466 -0.22343436]
 [ 0.35686607 -0.57730444 -0.25979897 -1.34276725]
 [-0.63232502 -0.38261486 -0.91980324 -0.57767076]
 [ 0.15544397  0.07343765  0.37222865  0.31091933]
 [ 0.25829614  0.59094041  0.11961462  0.63901457]
 [ 0.12213605  0.62784207 -0.24277246  0.56898094]
 [ 0.61919818  0.53629548  0.80981684  0.7574777 ]
 [-1.16397248  0.55950076 -1.47873668  0.3211994 ]
 [-2.27585448 -1.28658633 -1.74108889 -1.34576819]
 [-0.58061668 -0.33064684 -0.32539153 -0.15794627]
 [ 0.19746142  0.79483619  0.38056099  0.8687637 ]
 [ 0.43989267  0.61705191  0.49818027  0.60895796]
 [-0.93988418 -0.04008953 -1.81789437 -0.18761753]
 [ 0.12924715  0.13585969  0.35642115  0.3924562 ]
 [-2.87825997 -0.27564309 -2.4528239  -0.8833993 ]
 [ 0.45208795  0.50745564  0.55945761  0.60988946]
 [ 0.55999199  0.6267295   0.70910922  0.7442823 ]
 [-2.17732594 -2.08323696 -1.60367353 -1.3767997 ]
 [-1.84695284 -2.74314682 -1.65605448 -2.84384306]
 [ 0.61967271  0.50349852  0.86626806  0.4616401 ]
 [-2.46074206 -2.50270757 -2.07468189 -1.9088056 ]
 [ 0.7603763   0.75078506  0.94448195  0.98268141]
 [ 0.3194261   0.35447001 -0.35616334 -0.23648013]
 [ 0.8368567   0.79884075  1.0822203   1.0363337 ]
 [-0.0225837  -0.69387045  0.09036682 -1.93786611]
 [ 0.65047208  0.56105167  0.88267869  0.65597937]
 [ 0.87281096  0.78201741  1.04924382  0.90248907]
 [ 0.35630658 -0.75388332  0.56038771 -2.54258474]
 [-2.52560526 -2.90981666 -2.29525054 -2.62578562]
 [ 0.28070918  0.09217885  0.52921429  0.36066093]
 [-2.44075865 -3.47346298 -2.53972838 -1.81459885]
 [ 0.67438427  0.38331395  0.91014497 -0.12299702]
 [ 0.03893345  0.16938309  0.32972141  0.36609235]
 [-2.16871758 -1.82265486 -1.95709036 -1.7642539 ]
 [ 0.81452958  0.7060681   0.99245305  0.84407932]
 [-1.95683323 -2.16485097 -1.70576659 -1.68194832]
 [ 0.69865842  0.69388271  0.97162341  1.02637283]
 [-0.72458747 -0.72180359 -0.43213134 -0.37351925]
 [-2.99280081 -2.87679576 -2.28015874 -1.9945761 ]
 [-0.84321121 -1.49519029 -0.78319671 -2.19356597]
 [ 0.0353086   0.0627146   0.06188168  0.12759542]
 [ 0.36751846  0.21835327  0.33785713  0.19954709]
 [ 0.77212957  0.77036921  1.03309216  1.11027303]
 [ 0.64257585  0.65661802  0.91481779  0.96852791]
 [-4.21789156 -3.65193882 -2.86425973 -2.69150803]
 [ 0.86688434  0.58388021  1.09706405  0.62917982]
 [ 0.51224774  0.35333795  0.49845139  0.57102312]
 [ 0.79773189  0.69036537  0.99473019  0.85233395]
 [ 0.39517471  0.16098451  0.31076635 -0.07277078]
 [ 0.65566868  0.40723102  0.75059542  0.59766454]
 [ 0.84479103  0.73648927  1.0176461   0.94371121]
 [ 0.63740844  0.54697396  0.74630647  0.76396768]
 [-0.85309544 -1.14909755 -0.54735257 -0.87724459]
 [-2.28898564 -2.57211035 -1.81417739 -1.71193144]
 [ 0.81919053  0.76445179  1.01423651  0.89355445]
 [-2.1769306  -2.11381322 -1.61082446 -1.37615376]
 [ 0.91145802  0.88812849  1.1342802   1.13187256]
 [ 0.94457898  0.97277665  1.20694816  1.32700515]
 [ 0.50597365  0.07863983  0.55258691  0.03435697]
 [-1.52816974 -1.06658042 -0.94954264 -0.65086365]
 [ 0.72731645  0.62966661  0.82638618  0.83821012]
 [ 0.87063691  0.83270622  1.11830747  1.11754299]
 [-0.64412727 -1.15938672 -0.46865342 -1.30832113]
 [ 0.8413032   0.86262778  1.06363086  1.10373168]
 [ 0.6879572  -0.07722854  0.78042216 -1.5443069 ]
 [ 0.46546741 -0.05506825  0.08344826 -0.70728588]
 [-1.62664608 -1.24759899 -1.52851338 -1.08987097]
 [ 0.83638427  0.72318446  0.97123686  0.8692882 ]
 [ 0.44567372  0.56465461  0.45133018  0.75335349]
 [ 0.93156635  0.85687606  1.17940831  1.08278388]
 [ 0.01950414  0.00504927  0.14383722  0.091893  ]
 [-0.34233494 -0.20005812 -1.04007609 -0.281714  ]
 [ 0.70293426  0.5915935   0.94758245  0.86681074]
 [ 0.78060879  0.74261283  0.98246758  0.98729324]
 [ 0.88153162  0.85259562  1.13215856  1.10635908]
 [ 0.38050517  0.32231226  0.50908634  0.30738092]
 [ 0.46321246  0.28208651  0.44770694  0.39084002]
 [ 0.80392339  0.75382315  1.05049951  1.01346756]
 [ 0.72750709  0.52431747  0.62327395  0.24827417]
 [-0.04022776 -1.71698066 -0.15367786 -1.40672369]
 [ 0.73326307  0.65287375  0.90964025  0.72399621]
 [ 0.65510441  0.64551415  0.69904539  0.49446125]
 [ 0.78897977  0.76063643  0.94442032  1.00731563]
 [ 0.75177996  0.47804935  0.88516669  0.01188037]
 [ 0.45469431  0.64008628  0.11154675  0.56756133]
 [ 0.84208089  0.83563629  0.98926493  0.99996165]
 [ 0.59562546  0.66979148  0.52901833  0.74101793]
 [ 0.61982429  0.53026043  0.49282378  0.5367453 ]
 [ 0.69942403  0.05579953  0.81289423 -0.457571  ]
 [ 0.78661424  0.8310283   0.84098264  1.03326861]
 [-0.67441575 -0.33485797 -0.61043992 -0.3703951 ]
 [ 0.37388405 -1.65728619  0.30081793 -0.8935524 ]
 [ 0.62052797  0.62008523  0.67208587  0.81665965]
 [ 0.66108129  0.47436006  0.77544824  0.39902644]
 [ 0.21073255  0.01897531  0.01643463 -0.14643402]
 [ 0.77182762  0.41549056  0.99155907  0.75006066]
 [ 0.3391652   0.41857874  0.0383423   0.53481602]
 [ 0.36974628  0.76968655  0.36847522  0.88359152]
 [ 0.63981792  0.60730255  0.73875129  0.75993241]
 [-0.05320821 -0.02875596  0.30561935 -0.06685265]
 [ 0.4464028   0.70782393  0.26088319  0.88772251]
 [ 0.52213484  0.67170669  0.35152904  0.69361582]
 [ 0.74579284  0.75639078  0.88624079  0.8482684 ]
 [ 0.55732556  0.58935847  0.5394578   0.67774417]
 [-0.5133398  -1.34579072 -2.16715579 -2.3626072 ]
 [ 0.26719357  0.50936956  0.09032069  0.3290176 ]
 [ 0.75077796 -1.84009064  0.92938815 -2.10554752]
 [ 0.47886645  0.62511308  0.30297045  0.68576165]
 [ 0.36256365  0.10351794  0.35343023 -0.05807594]
 [ 0.86350687  0.35135469  1.10957223  0.07421951]
 [ 0.72965212  0.05418528  0.91318918 -0.91474501]
 [ 0.15503032  0.49178627 -0.09204352  0.45111325]
 [ 0.11947683  0.13040815  0.01140217  0.21473297]
 [ 0.87720435  0.43031379  1.11926847  0.45070366]
 [ 0.49520982  0.50020118  0.43961754  0.56217927]
 [ 0.4528222  -0.11779247  0.64462631 -0.56932457]
 [-0.01172802  0.43130591  0.11928462  0.51288968]
 [-0.02823101  0.80708739 -0.34594867  0.86940235]
 [ 0.29345152  0.3752812   0.25214841  0.50338379]
 [ 0.64626326  0.58299474  0.59103864  0.69475972]
 [ 0.4879933  -0.01742059  0.46084228 -1.00149868]
 [-0.19077901 -0.53752977 -0.37712227 -1.28805484]
 [ 0.60478854  0.61849726  0.45380604  0.67279894]
 [ 0.63554818  0.68562111  0.63149154  0.71356596]
 [ 0.74172186 -0.0377426   0.89210813 -0.52133368]
 [ 0.21662776  0.14567429 -0.15409391  0.01305418]
 [-0.55828937 -0.18837345 -1.12567917 -0.80923636]
 [ 0.41123102  0.75703304  0.36096939  0.85936601]
 [ 0.89424801  0.61546102  1.05314828  0.38550955]
 [ 0.42685492  0.39093417  0.29760301  0.29415408]
 [ 0.67832227  0.67144937  0.54890775  0.9312869 ]
 [ 0.59895484  0.44164987  0.62836072  0.56289819]
 [ 0.53012716 -0.02976048  0.68813745 -0.64610789]
 [ 0.50944225 -1.5855406   0.27265677 -1.44505186]
 [ 0.50934563  0.38137621  0.56529184  0.48145613]
 [ 0.50090741  0.22913436 -0.4655118  -1.15839963]
 [-0.47743345 -0.14220998 -0.79085521 -0.31472104]
 [ 0.67771322  0.83745429  0.66602441  1.0028655 ]
 [ 0.80639205  0.88273388  0.91480236  1.09858712]
 [ 0.60396661  0.18458436  0.61843847  0.24936985]]
```

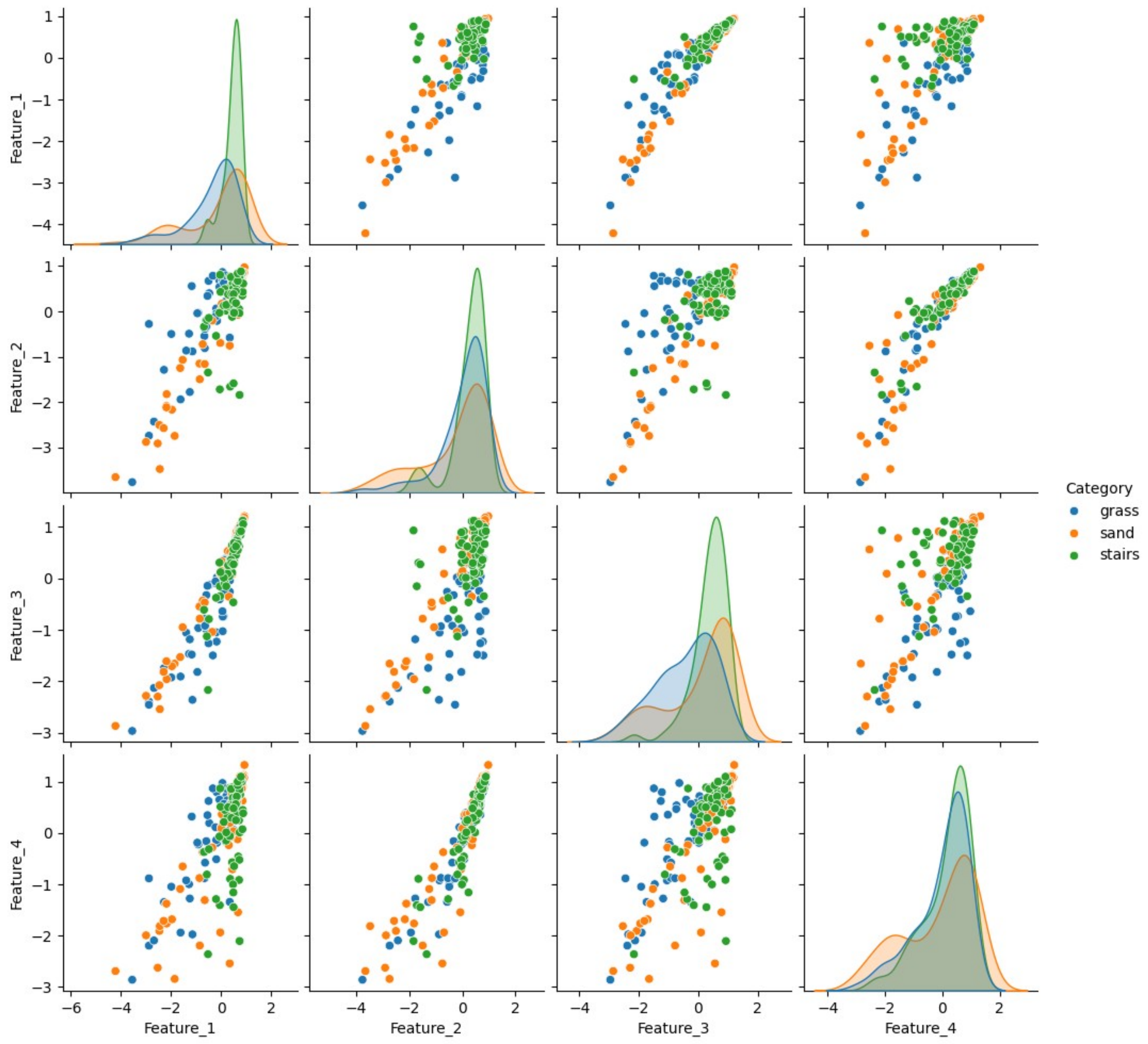
Feature relationships

Make illustrations of the feature relationships, and discuss the results

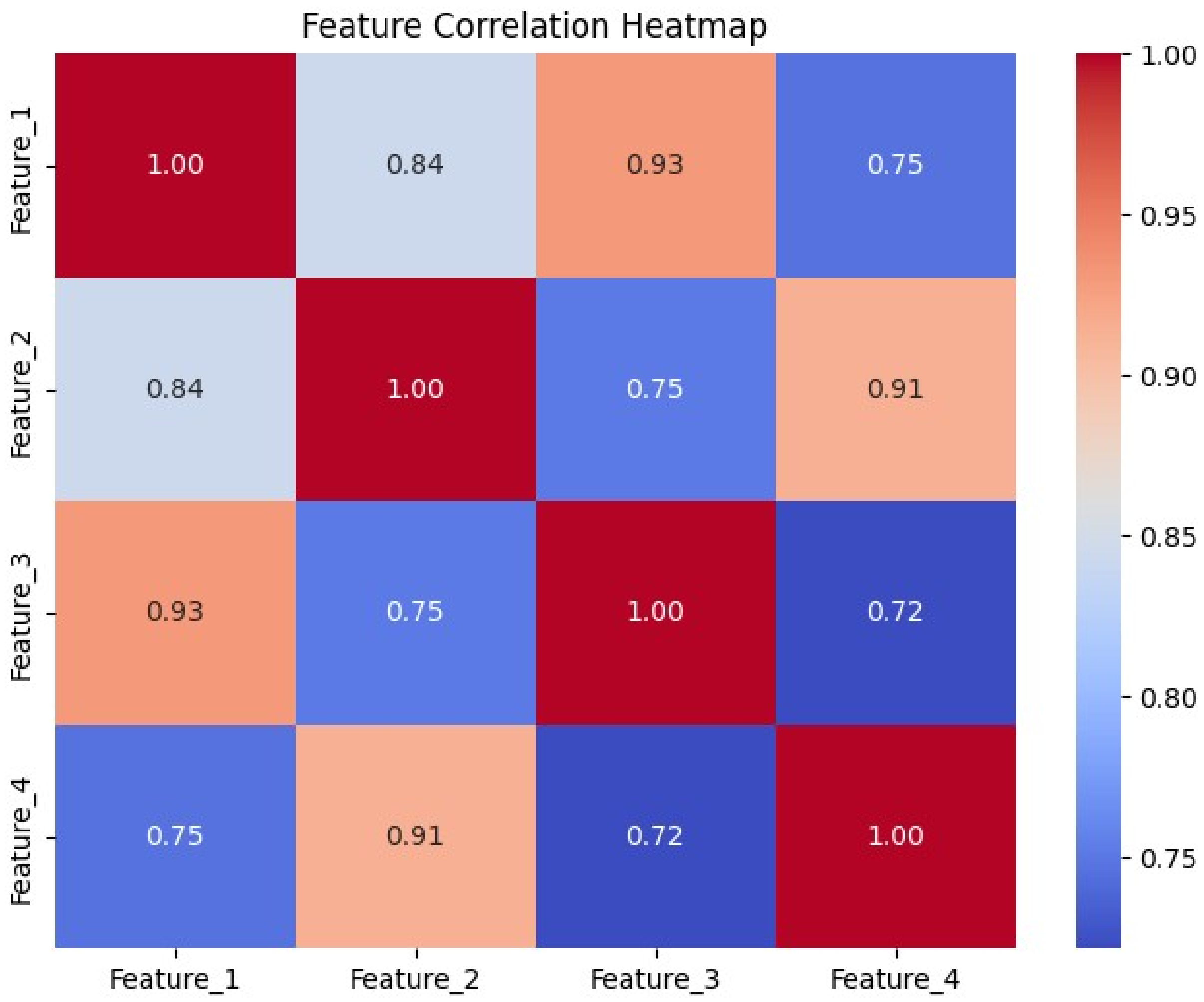
Pairplot

- Pairplot **(1 p)**
  - Which feature pairs possess roughly linear dependence?

```
# Convert array into a dataframe and print pairplot
df_X = pd.DataFrame(X, columns=[f'Feature_{i+1}' for i in range(4)])
df_X['Category'] = y
sns.pairplot(df_X, hue='Category')

<seaborn.axisgrid.PairGrid at 0x327cd3810>
```



```
# Print correlation matrix to check linear relationship between each
pair
df = pd.DataFrame(X, columns=[f'Feature_{i+1}' for i in range(4)])

correlation_matrix = df.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
#https://matplotlib.org/stable/users/explain/colors/colormaps.html
plt.title("Feature Correlation Heatmap")
plt.show()
```



	Feature_1	Feature_2	Feature_3	Feature_4
Feature_1	1.00	0.84	0.93	0.75
Feature_2	0.84	1.00	0.75	0.91
Feature_3	0.93	0.75	1.00	0.72
Feature_4	0.75	0.91	0.72	1.00

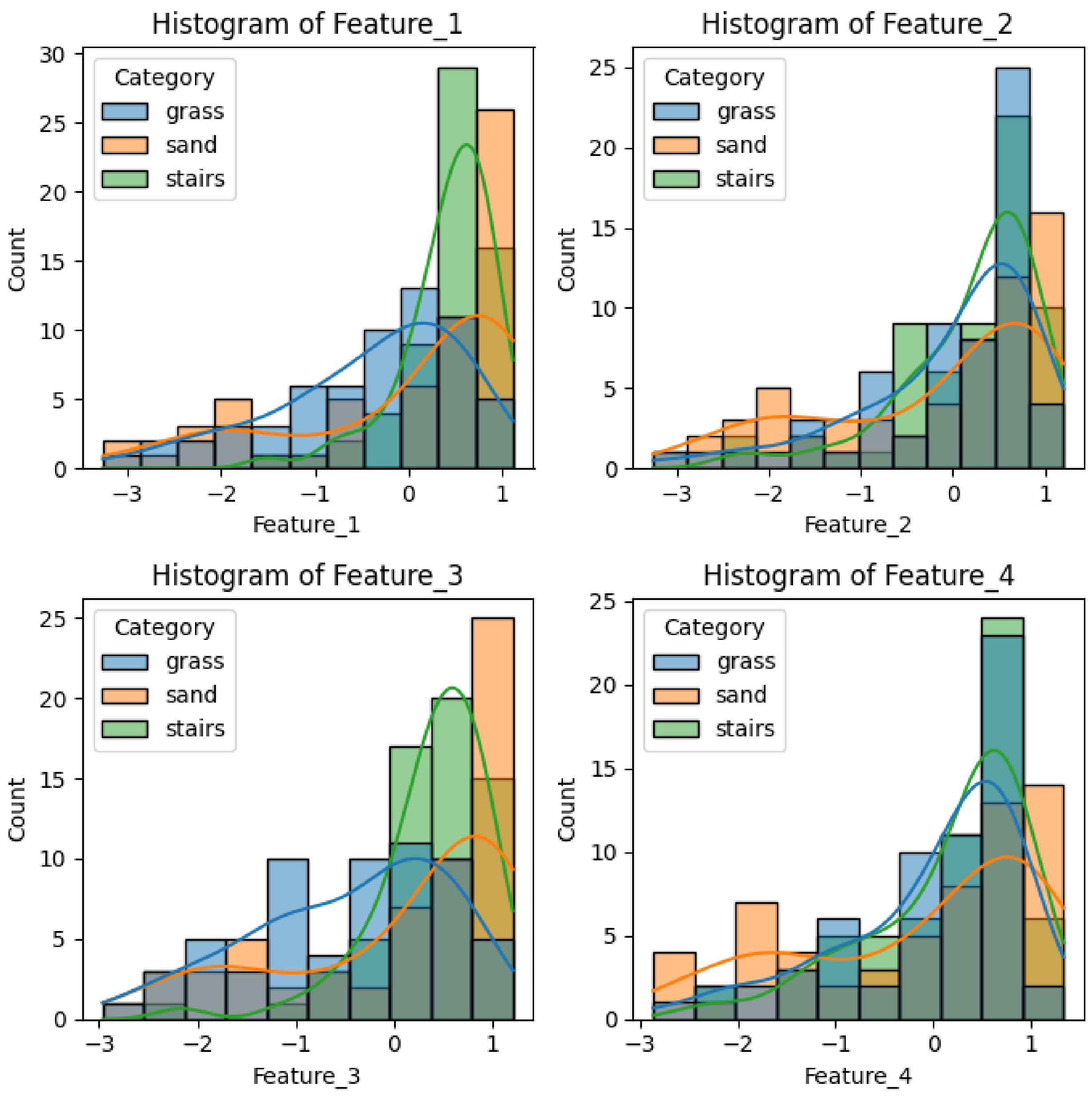
From the pairplot and the correlation matrix it's evident that feature pairs (1, 3) and (2, 4) have the clearest linear relationships. There is a certain level of correlation between every feature pair, which suggests that they all will have an impact when performing the classification.

Histograms

- Histograms **(1 p)**

- Plot a histogram for each Z-scored feature. Plot all the image classes in the same figure and use different color for each.
- Which features may have some discriminative power over image classes according to the histograms?

```python
# Standardized features into a DataFrame for plotting
df_features = pd.DataFrame(X, columns=[f'Feature_{i+1}' for i in
range(X.shape[1])])
df_features['Category'] = y

# Plot histograms
fig, axes = plt.subplots(2, 2, figsize=(7, 7))
axes = axes.flatten()

for i, feature in enumerate(df_features.columns[:-1]):  # Exclude last
column (Category column)
    sns.histplot(data=df_features, x=feature, hue='Category',
kde=True, ax=axes[i], alpha=0.5)
    axes[i].set_title(f'Histogram of {feature}')

plt.tight_layout()
plt.show()
```

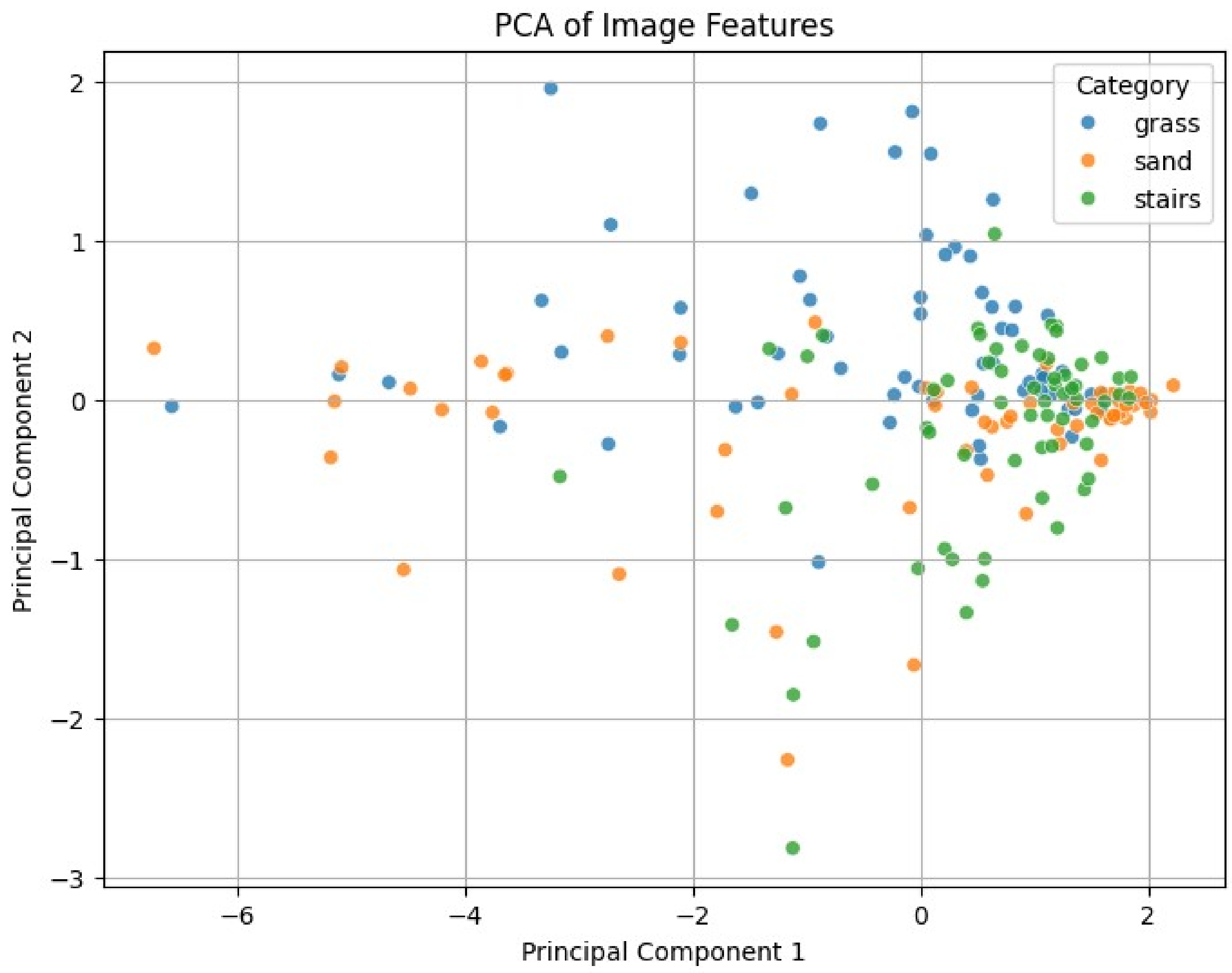


In histograms for features 1 and 3, there is a clear spike for category stairs, which would mean that those features have discriminative power over the other two image classes. Other than that, the distribution of values is pretty similar for all image classes. The histograms further validate previously made statement that every feature will have some importance, but features 1 and 3 possess the most discriminative power.

PCA

- PCA **(1 p)**
  - Plot the image glasses using different colors.
  - Can you see clusters in PCA?
  - Does this figure give you any clues, how well you will be able to classify the image types? Explain.

```
# Perform PCA with two components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Convert to DataFrame for plotting
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['Category'] = y

# Plot PCA results
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='Category',
palette='tab10', alpha=0.8)
plt.title('PCA of Image Features')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Category')
plt.grid(True)
plt.show()
```



There isn't really any clear clusters, but the grass class is more spread out with lower PC2 values while sand class is more in the middle and stairs class has more values with PC2 value under 0. PC1 values are very much concentrated around value 2, with some separation especially with grass and sand images. Based on the PCA it might be hard for something like ridge classifier to classify the images correctly, but a more robust model like MLP might do better.

Build classifiers and select the best hyperparameters with cross validation

Perform model selection for each classifier. Use 5-fold stratified cross validation (*StratifiedKFold* and *GridSearchCV* from sklearn). Use the following hyperparameters:

- Ridge Classifier (1 p)
  - strength of the regularization term: alpha = [0.001, 0.01, 0.1, 1.0]
- Random Forest **(1 p)**
  - n_estimators from 100 to 300 with 50 steps
  - max_features = ['sqrt', 'log2', None]
  - whether to use bootstrap or not
- MLP **(1 p)**
  - use one hidden layer
  - number of neurons in the hidden layer from 15 to 40 in 5 neuron steps
  - activation function: hyperbolic tanh function and rectified linear unit function
  - solver: stochastic gradient descent and adam
  - validation_fraction: 0.1 and 0.3
  - strength of the L2 regularization term: alpha = [0.01, 0.1, 1]

For each classifier:

- Report the selected combination of hyperparameters
- Report the accuracy value for each hyperparameter combination

For Random Forest model, report the feature importance for each feature. Which features seem to be the most important? Does this correspond with the observations you made in the data exploration? **(1 p)**

```
# Ridge classifier
# Hyperparameter grid
param_grid = {'alpha': [0.001, 0.01, 0.1, 1.0]}

# 5-fold stratified cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize the model
ridge = RidgeClassifier()

# Perform grid search
grid_search = GridSearchCV(ridge, param_grid, cv=cv,
scoring='accuracy')
```

```python
grid_search.fit(X, y)

# Print accuracy for each combination and the best score
print("Best parameter:", grid_search.best_params_)
print("Best accuracy:", grid_search.best_score_)

for alpha, mean_score in zip(param_grid['alpha'],
grid_search.cv_results_['mean_test_score']):
    print(f"Alpha {alpha}: {mean_score:.8f}")
```

```
Best parameter: {'alpha': 0.001}
Best accuracy: 0.5945945945945946
Alpha 0.001: 0.59459459
Alpha 0.01: 0.59459459
Alpha 0.1: 0.59459459
Alpha 1.0: 0.56756757
```

```python
# Random Forest
# Hyperparameter grid
param_grid = {
    'n_estimators': [100, 150, 200, 250, 300],
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}
features = ['feature_1', 'feature_2', 'feature_3', 'feature_4']

# Initialize the model
rf = RandomForestClassifier(random_state=42)

# 5-fold stratified cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform grid search
grid_search = GridSearchCV(rf, param_grid, cv=cv, scoring='accuracy',
n_jobs=-1, verbose=1)
grid_search.fit(X, y)

# Print accuracy for each combination and the best score
print("Best parameters:", grid_search.best_params_)
print("Best accuracy:", grid_search.best_score_)

results = grid_search.cv_results_
for mean_score, params in zip(results['mean_test_score'],
results['params']):
    print(f"{params}: {mean_score:.6f}")

# Train best model and get feature importances
best_rf = grid_search.best_estimator_
feature_importances = best_rf.feature_importances_

# Print feature importances
```

```
print("Feature Importances:")
for feature, importance in zip(features, feature_importances):
    print(f"{feature}: {importance:.4f}")
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best parameters: {'bootstrap': True, 'max_features': None,
'n_estimators': 100}
Best accuracy: 0.5567567567567567
{'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 100}:
0.540541
{'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 150}:
0.540541
{'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 200}:
0.535135
{'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 250}:
0.524324
{'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 300}:
0.524324
{'bootstrap': True, 'max_features': 'log2', 'n_estimators': 100}:
0.540541
{'bootstrap': True, 'max_features': 'log2', 'n_estimators': 150}:
0.540541
{'bootstrap': True, 'max_features': 'log2', 'n_estimators': 200}:
0.535135
{'bootstrap': True, 'max_features': 'log2', 'n_estimators': 250}:
0.524324
{'bootstrap': True, 'max_features': 'log2', 'n_estimators': 300}:
0.524324
{'bootstrap': True, 'max_features': None, 'n_estimators': 100}:
0.556757
{'bootstrap': True, 'max_features': None, 'n_estimators': 150}:
0.540541
{'bootstrap': True, 'max_features': None, 'n_estimators': 200}:
0.540541
{'bootstrap': True, 'max_features': None, 'n_estimators': 250}:
0.535135
{'bootstrap': True, 'max_features': None, 'n_estimators': 300}:
0.545946
{'bootstrap': False, 'max_features': 'sqrt', 'n_estimators': 100}:
0.551351
{'bootstrap': False, 'max_features': 'sqrt', 'n_estimators': 150}:
0.540541
{'bootstrap': False, 'max_features': 'sqrt', 'n_estimators': 200}:
0.545946
{'bootstrap': False, 'max_features': 'sqrt', 'n_estimators': 250}:
0.545946
{'bootstrap': False, 'max_features': 'sqrt', 'n_estimators': 300}:
0.540541
{'bootstrap': False, 'max_features': 'log2', 'n_estimators': 100}:
0.551351
```

```
{'bootstrap': False, 'max_features': 'log2', 'n_estimators': 150}:
0.540541
{'bootstrap': False, 'max_features': 'log2', 'n_estimators': 200}:
0.545946
{'bootstrap': False, 'max_features': 'log2', 'n_estimators': 250}:
0.545946
{'bootstrap': False, 'max_features': 'log2', 'n_estimators': 300}:
0.540541
{'bootstrap': False, 'max_features': None, 'n_estimators': 100}:
0.524324
{'bootstrap': False, 'max_features': None, 'n_estimators': 150}:
0.524324
{'bootstrap': False, 'max_features': None, 'n_estimators': 200}:
0.529730
{'bootstrap': False, 'max_features': None, 'n_estimators': 250}:
0.529730
{'bootstrap': False, 'max_features': None, 'n_estimators': 300}:
0.529730
Feature Importances:
feature_1: 0.2596
feature_2: 0.2510
feature_3: 0.2960
feature_4: 0.1934

# MLP
# Hyperparameter grid
param_grid = {
    'hidden_layer_sizes': [(n,) for n in range(15, 45, 5)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.01, 0.1, 1],
    'validation_fraction': [0.1, 0.3],
}

# Initialize the model
mlp = MLPClassifier(max_iter=5000, random_state=42)

# 5-fold stratified cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform grid search
grid_search = GridSearchCV(mlp, param_grid, cv=cv, scoring='accuracy',
n_jobs=-1, verbose=1)
grid_search.fit(X, y)

# Print accuracy for each combination and the best score
print("Best parameters:", grid_search.best_params_)
print("Best accuracy:", grid_search.best_score_)

results = grid_search.cv_results_
```

```
for mean_score, params in zip(results['mean_test_score'],
results['params']):
    print(f"{params}: {mean_score:.6f}")
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Best parameters: {'activation': 'tanh', 'alpha': 0.01,
'hidden_layer_sizes': (20,), 'solver': 'adam', 'validation_fraction':
0.1}
Best accuracy: 0.6324324324324324
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.491892
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.491892
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.589189
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.589189
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.464865
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.464865
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.632432
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.632432
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.475676
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.475676
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.551351
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.551351
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.513514
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.513514
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.551351
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.551351
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.491892
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.491892
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.572973
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.572973
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.481081
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.481081
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.540541
{'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.540541
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.491892
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.491892
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.600000
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.600000
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.464865
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.464865
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.621622
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.621622
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.481081
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.481081
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.583784
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.583784
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.508108
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.508108
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.616216
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.616216
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.497297
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.497297
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.616216
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.616216
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.481081
```

```
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.481081
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.632432
{'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.632432
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.486486
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.486486
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.589189
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.589189
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.454054
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.454054
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.545946
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.545946
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.475676
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.475676
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.572973
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.572973
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.497297
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.497297
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.572973
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.572973
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.491892
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.491892
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.583784
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.583784
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.475676
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.475676
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.578378
{'activation': 'tanh', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.578378
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.497297
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.497297
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.594595
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.594595
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.535135
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.535135
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.616216
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.616216
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.551351
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.551351
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.632432
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.632432
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.545946
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.545946
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.578378
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.578378
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.518919
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.518919
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.616216
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.616216
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.513514
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.513514
```

```
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.572973
{'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.572973
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.502703
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.502703
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.589189
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.589189
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.535135
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.535135
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.610811
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.610811
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.551351
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.551351
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.605405
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.605405
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.551351
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.551351
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.572973
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.572973
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.524324
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.524324
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.627027
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.627027
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.513514
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.513514
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.594595
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.491892
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.491892
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (15,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.589189
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (20,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.589189
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.540541
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.540541
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (25,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.535135
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.535135
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.583784
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (30,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.583784
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (35,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.594595
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.1}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'sgd', 'validation_fraction': 0.3}: 0.518919
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.1}: 0.589189
```

```
{'activation': 'relu', 'alpha': 1, 'hidden_layer_sizes': (40,),
'solver': 'adam', 'validation_fraction': 0.3}: 0.589189
```

Analyzing the results

Ridge Classifier

Best parameter: {'alpha': 0.001}

Best accuracy: 0.5945945945945946

Best results were attained by using regularization term alpha = 0.001, but same results were attained also if alpha was 0.1 or 0.01. Accuracy value 0.5946 means that the model was able to classify almost 60% of the images correctly.

Random Forest

Best parameters: {'bootstrap': True, 'max_features': None, 'n_estimators': 100}

Best accuracy: 0.5567567567567567

feature_1: 0.2596

feature_2: 0.2510

feature_3: 0.2960

feature_4: 0.1934

Random Forest was able to get an accuracy score of 0.5568 with the parameter values {'bootstrap': True, 'max_features': None, 'n_estimators': 100}. From feature importance values feature_3 had the highest score of 0.296, with feature_1 and feature_2 being pretty close to each other with scores 0.2596 and 0.2510 respectively. These feature importances align with the observations made in data exploration with histograms. All features are pretty close to each other, but there is still a clear difference between them.

MLP

Best parameters: {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,), 'solver': 'adam', 'validation_fraction': 0.1}

Best accuracy: 0.6324324324324324

MLP was able to classify 63.24% of the images correctly with parameter values of {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (20,), 'solver': 'adam', 'validation_fraction': 0.1}. This accuracy score is clearly the highest and corresponds to previously made statement that the task might need a more robust model to get better scores, since the features are pretty similar to each other.

Estimate the performance of the models with nested cross-validation

Estimate the performance of each model using nested cross validation. We'll use 4-fold Stratified Kfold cross-validation and the same parameter ranges as earlier for the inner loop. For the outer loop we'll use 5-fold Stratified Kfold cross-validation.

For each classifier:

- Ridge Classifier **(1 p)**
- Random Forest **(1 p)**
- MLP **(1 p)**

  Report the selected combination of the hyperparameters and the accuracy value for the best hyperparameter combination. Create a confusion matrix of the results. Calculate the mean accuracy of the outer rounds. What does it mean? **(1 p)**

```
# Ridge classifier
# Hyperparameter grid
param_grid_ridge = {'alpha': [0.001, 0.01, 0.1, 1.0]}

# Outer cross-validation
outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

ridge_accuracies = []
all_conf_matrices = []

y = np.array(y)

for train_idx, test_idx in outer_cv.split(X, y):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Inner CV
    inner_cv = StratifiedKFold(n_splits=4, shuffle=True,
random_state=42)
    ridge = RidgeClassifier()
    grid_search = GridSearchCV(ridge, param_grid_ridge, cv=inner_cv,
scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Evaluate best model on outer fold
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Store accuracy
    acc = accuracy_score(y_test, y_pred)
    ridge_accuracies.append(acc)
    # Store confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    all_conf_matrices.append(conf_matrix)

    print(f"Best parameters: {grid_search.best_params_}, Accuracy: 
{acc:.4f}")

# Mean accuracy
mean_accuracy_ridge = sum(ridge_accuracies) / len(ridge_accuracies)
print(f"Mean Accuracy (Ridge Classifier): {mean_accuracy_ridge:.4f}")

Best parameters: {'alpha': 0.001}, Accuracy: 0.5946
Best parameters: {'alpha': 0.001}, Accuracy: 0.7297
Best parameters: {'alpha': 0.001}, Accuracy: 0.5405
Best parameters: {'alpha': 1.0}, Accuracy: 0.5676
Best parameters: {'alpha': 0.1}, Accuracy: 0.4595
Mean Accuracy (Ridge Classifier): 0.5784

# Random Forest
# Hyperparameter grid
param_grid_rf = {
    'n_estimators': [100, 150, 200, 250, 300],
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}

# Outer cross-validation
outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

rf_accuracies = []
all_conf_matrices_rf = []

for train_idx, test_idx in outer_cv.split(X, y):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Inner CV
    inner_cv = StratifiedKFold(n_splits=4, shuffle=True, 
random_state=42)
    rf = RandomForestClassifier(random_state=42)
    grid_search = GridSearchCV(rf, param_grid_rf, cv=inner_cv, 
scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Evaluate best model on outer fold
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Store accuracy
    acc = accuracy_score(y_test, y_pred)
    rf_accuracies.append(acc)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    all_conf_matrices_rf.append(conf_matrix)

    print(f"Best parameters: {grid_search.best_params_}, Accuracy:
{acc:.4f}")

# Mean accuracy
mean_accuracy_rf = sum(rf_accuracies) / len(rf_accuracies)
print(f"Mean Accuracy (Random Forest): {mean_accuracy_rf:.4f}")
```

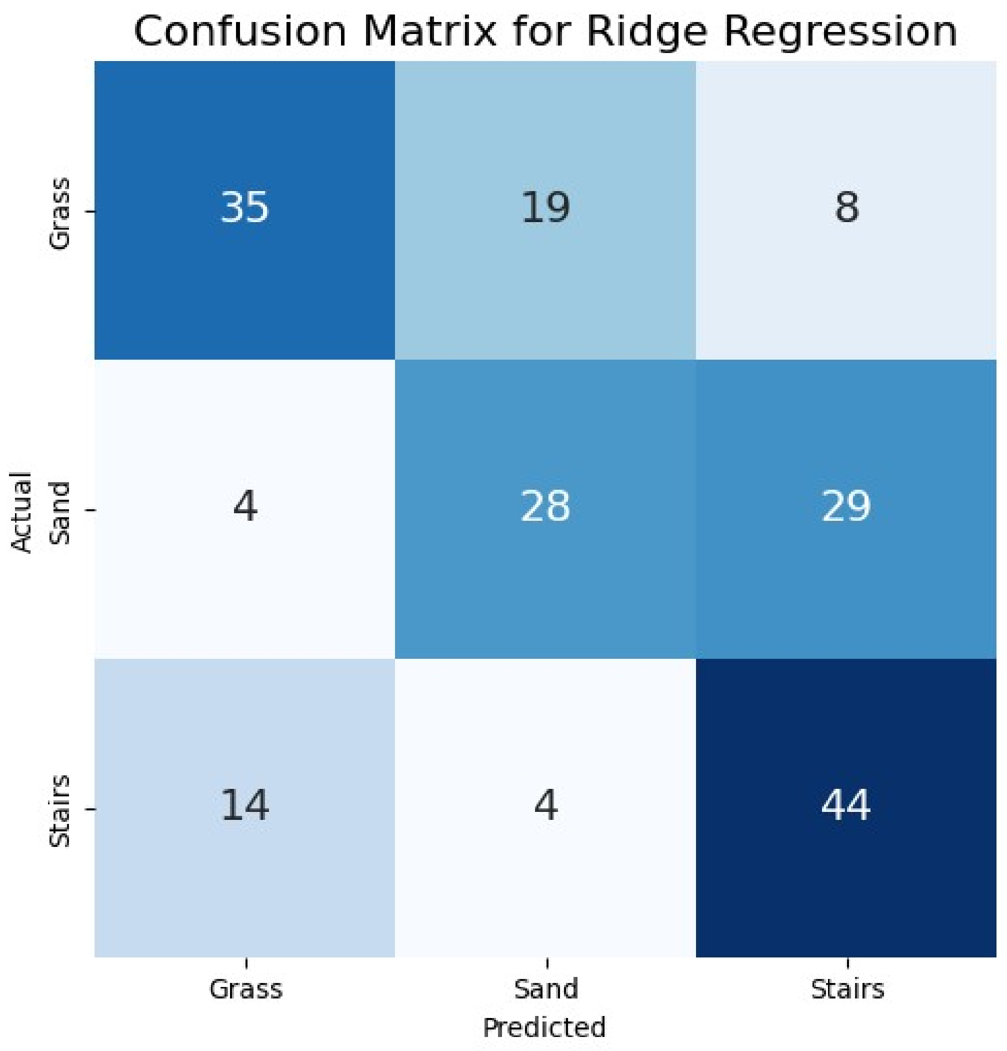
```
Best parameters: {'bootstrap': True, 'max_features': None,
'n_estimators': 100}, Accuracy: 0.5405
Best parameters: {'bootstrap': True, 'max_features': None,
'n_estimators': 250}, Accuracy: 0.4865
Best parameters: {'bootstrap': False, 'max_features': None,
'n_estimators': 100}, Accuracy: 0.4865
Best parameters: {'bootstrap': True, 'max_features': 'sqrt',
'n_estimators': 100}, Accuracy: 0.5676
Best parameters: {'bootstrap': True, 'max_features': None,
'n_estimators': 150}, Accuracy: 0.5676
Mean Accuracy (Random Forest): 0.5297
```

```
# MLP
# Hyperparameter grid
param_grid_mlp = {
    'hidden_layer_sizes': [(n,) for n in range(15, 45, 5)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.01, 0.1, 1],
    'validation_fraction': [0.1, 0.3]
}

# Outer cross-validation
outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

mlp_accuracies = []
all_conf_matrices_mlp = []

for train_idx, test_idx in outer_cv.split(X, y):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Inner CV
    inner_cv = StratifiedKFold(n_splits=4, shuffle=True,
random_state=42)
    mlp = MLPClassifier(max_iter=5000, random_state=42)
    grid_search = GridSearchCV(mlp, param_grid_mlp, cv=inner_cv,
scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Evaluate best model on outer fold
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Store accuracy
    acc = accuracy_score(y_test, y_pred)
    mlp_accuracies.append(acc)

    # Store confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    all_conf_matrices_mlp.append(conf_matrix)

    print(f"Best parameters: {grid_search.best_params_}, Accuracy:
{acc:.4f}")

# Mean accuracy
mean_accuracy_mlp = sum(mlp_accuracies) / len(mlp_accuracies)
print(f"Mean Accuracy (MLP): {mean_accuracy_mlp:.4f}")
```

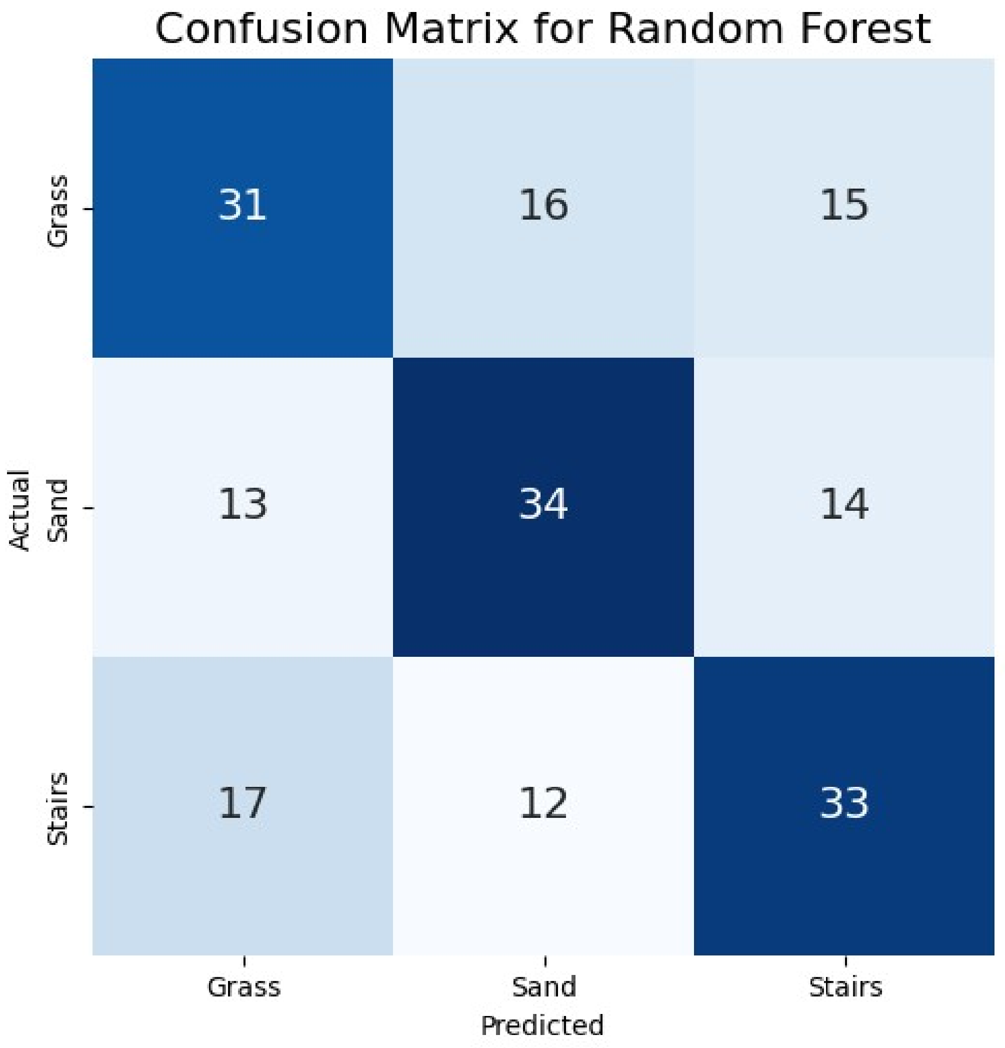
```
Best parameters: {'activation': 'tanh', 'alpha': 0.1,
'hidden_layer_sizes': (15,), 'solver': 'adam', 'validation_fraction':
0.1}, Accuracy: 0.5405
Best parameters: {'activation': 'relu', 'alpha': 0.01,
'hidden_layer_sizes': (30,), 'solver': 'adam', 'validation_fraction':
0.1}, Accuracy: 0.5135
Best parameters: {'activation': 'relu', 'alpha': 0.1,
'hidden_layer_sizes': (40,), 'solver': 'adam', 'validation_fraction':
0.1}, Accuracy: 0.5946
Best parameters: {'activation': 'tanh', 'alpha': 0.01,
'hidden_layer_sizes': (40,), 'solver': 'adam', 'validation_fraction':
0.1}, Accuracy: 0.5946
Best parameters: {'activation': 'relu', 'alpha': 0.1,
'hidden_layer_sizes': (20,), 'solver': 'adam', 'validation_fraction':
0.1}, Accuracy: 0.4865
Mean Accuracy (MLP): 0.5459
```

```
def plot_confusion_matrices(conf_matrices, model_name):
    # Sum up confusion matrices from different cross validation folds
    summed_matrix = np.sum(conf_matrices, axis=0)
    print(summed_matrix)
    # Plot the summed matrix as a heatmap
    plt.figure(figsize=(6, 6))
    sns.heatmap(summed_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False,
                xticklabels=['Grass', 'Sand', 'Stairs'],
yticklabels=['Grass', 'Sand', 'Stairs'],
                annot_kws={"size": 16})
    plt.title(f'Confusion Matrix for {model_name}', fontsize=16)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Plot confusion matrices
plot_confusion_matrices(all_conf_matrices, "Ridge Regression")
plot_confusion_matrices(all_conf_matrices_rf, "Random Forest")
plot_confusion_matrices(all_conf_matrices_mlp, "MLP")
```

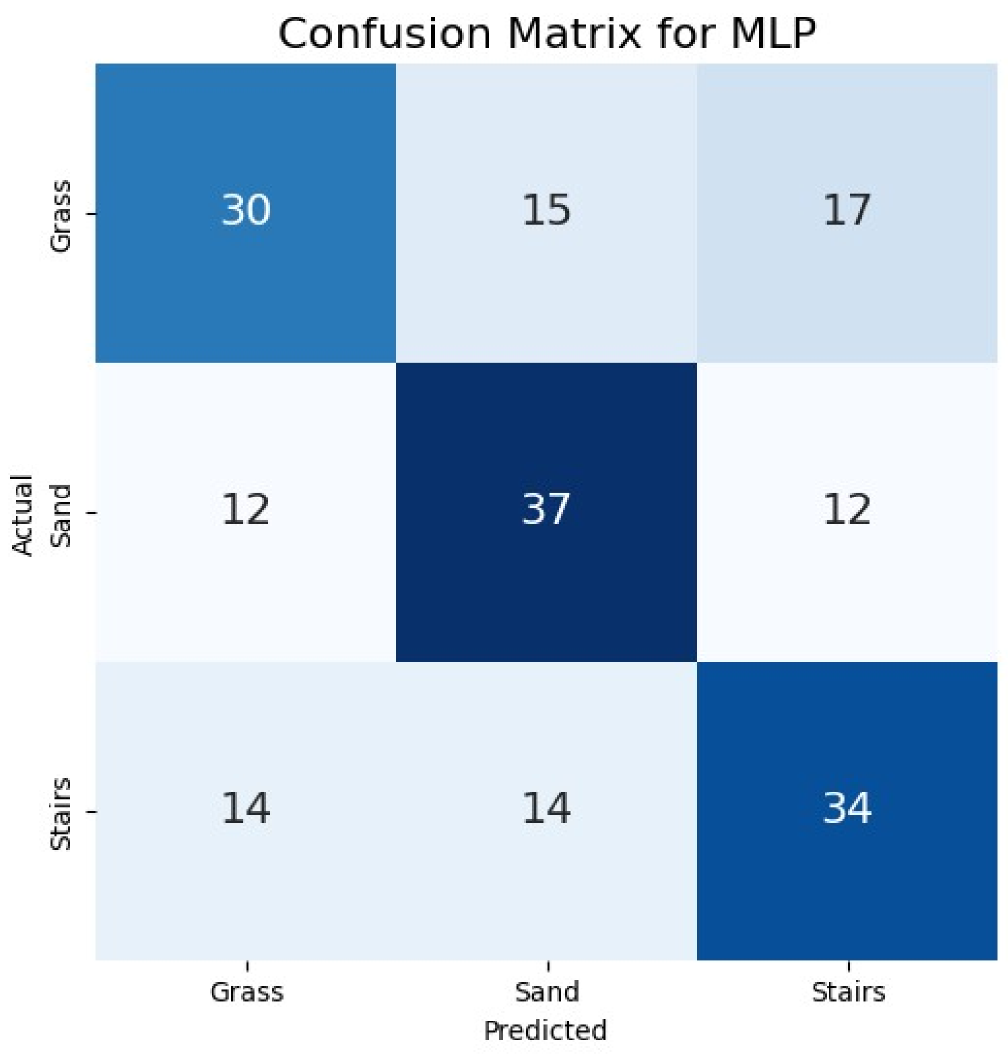
```
[[35 19  8]
 [ 4 28 29]
 [14  4 44]]
```



```
[[31 16 15]
 [13 34 14]
 [17 12 33]]
```



```
[[30 15 17]
 [12 37 12]
 [14 14 34]]
```



From the confusion matrices it can be seen that even though the ridge classifier has the best overall performance, it has hard time classifying the sand images, often confusing them to be stairs. This might also be a reason why it has so high true positive rate for classifying stairs correctly and increasing its overall accuracy value to be higher than other classifiers. Based on the confusion matrices, MLP might actually be the best of the three, since it has higher accuracy than random forest, and the true positive rate is more evenly distributed across different image classes, making it more consistent.

Analyzing the results

Ridge Classifier

Best parameters: {'alpha': 0.001}, Accuracy: 0.5946

Best parameters: {'alpha': 0.001}, Accuracy: 0.7297

Best parameters: {'alpha': 0.001}, Accuracy: 0.5405

Best parameters: {'alpha': 1.0}, Accuracy: 0.5676

Best parameters: {'alpha': 0.1}, Accuracy: 0.4595

Mean Accuracy (Ridge Classifier): 0.5784

Mean accuracy of the outer rounds for ridge regression was 0.5784. It is slightly better than what was achieved before in the model selection part, considerably better than what random selection would be (0.33)

Random Forest

Best parameters: {'bootstrap': True, 'max_features': None, 'n_estimators': 100}, Accuracy: 0.5405

Best parameters: {'bootstrap': True, 'max_features': None, 'n_estimators': 250}, Accuracy: 0.4865

Best parameters: {'bootstrap': False, 'max_features': None, 'n_estimators': 100}, Accuracy: 0.4865

Best parameters: {'bootstrap': True, 'max_features': 'sqrt', 'n_estimators': 100}, Accuracy: 0.5676

Best parameters: {'bootstrap': True, 'max_features': None, 'n_estimators': 150}, Accuracy: 0.5676

Mean Accuracy (Random Forest): 0.5297

Mean accuracy for Random Forest was 0.5297. It's a little bit worse than before in the model selection, but it is more likely to be closer to the models true performance on unseen data, because of the nested cross validation.

MLP

Best parameters: {'activation': 'tanh', 'alpha': 0.1, 'hidden_layer_sizes': (15,), 'solver': 'adam', 'validation_fraction': 0.1}, Accuracy: 0.5405

Best parameters: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (30,), 'solver': 'adam', 'validation_fraction': 0.1}, Accuracy: 0.5135

Best parameters: {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (40,), 'solver': 'adam', 'validation_fraction': 0.1}, Accuracy: 0.5946

Best parameters: {'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (40,), 'solver': 'adam', 'validation_fraction': 0.1}, Accuracy: 0.5946

Best parameters: {'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (20,), 'solver': 'adam', 'validation_fraction': 0.1}, Accuracy: 0.4865

Mean Accuracy (MLP): 0.5459

Mean accuracy for MLP was 0.5459. There is a significant drop in accuracy compared to before, but again it's a more realistic estimate of the models performance.

Discussion

- Discuss you results **(2 p)** E.g.

- Which model performs the best and why?
- What are the limitations?
- How could the results be improved?

The Ridge Classifier performs the best with a mean accuracy of 0.5784. This is a little bit surprising after the data exploration and model selection. This suggests that linear relationships captured by Ridge Regression are still effective, possibly because the GLCM features have captured some of the data's linear separability.

All models have relatively low mean accuracies, indicating limitations in the discriminative power of the features or potential class overlap. The models struggle to consistently classify across different data splits, as shown by the variance in accuracy between the outer folds.

Results could be improved by using different features, such as contrast, dissimilarity and homogeneity. Also using more complex models and exploring wider range of hyper parameters could improve the results to better capture the differences between different classes.