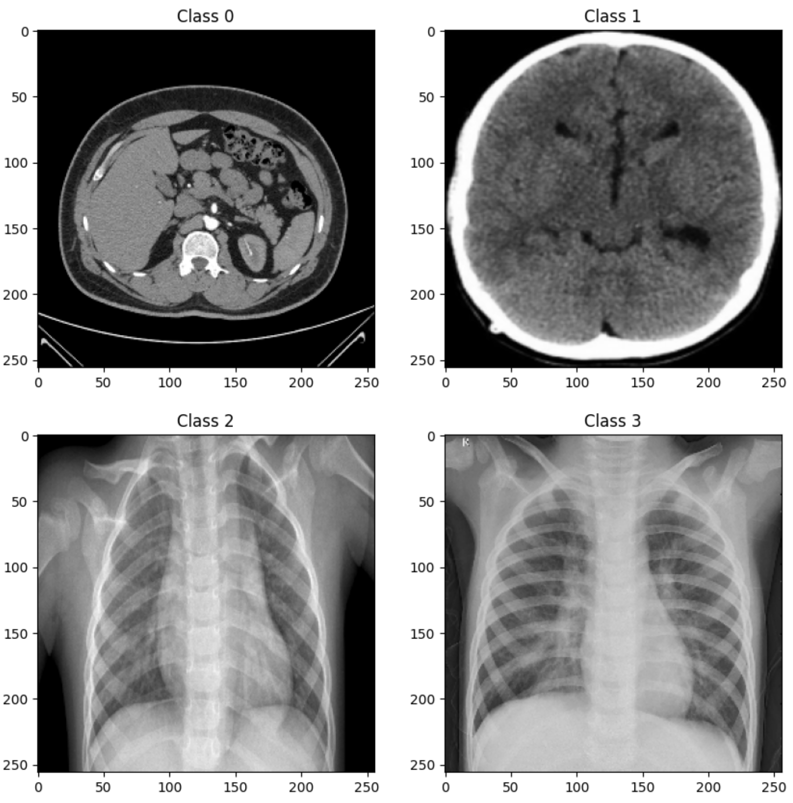
CodeLab – 4
Feature engineering and classification of medical images

Feature engineering is the art and science of selecting, transforming, and creating meaningful input features from raw data. Machine learning algorithms can extract valuable patterns and make accurate predictions. This exercise focuses on biomedical images derived from three distinct modalities: chest X-rays, abdomen and head CT scans. [1]–[3].



Feature extraction tasks heavily rely on the type of data and the objective. For instance, object detection tasks rely on edge detection, blob detection, and shape descriptors. For more complex tasks, e.g., ML aided diagnosis of pneumonia, convolutional neural networks (CNN) are often employed due to their high performance in image processing. The multi-layer structure remains identical as in neural network, with the main difference being in the learned parameters, which are the kernel weights, as opposed to the neuron weights. Kernels are small windows (filters), which overlap to the input image/data. The element wise products between the kernel weights and the overlapping portion of the input image are added together to produce a new feature map. After each summation step, the filter slides to the next region of the image.

In this lab, you will use a set of selected biomedical images to classify and categorize the dataset. Meanwhile, the effect of the feature selection on the image dataset will be discussed. You will also tell the differences before and after feature engineering. Hyperparameter tuning is also an important factor in the training process of modern ML workflow. A grid search method will be deployed to find the optimal model parameters for this specific task.

Provide your answers in the Jupyter Notebook either as a comment or insert a markdown cell. Please print out your Jupyter Notebook in PDF format (or HTML) and upload it to Brightspace. You will get points for each part separately, and you will get the grades based on your overall performance. Table 1 illustrates the points of each part, and Table 2 shows the grading scheme of CodeLab 4.

Table 1: Points of CodeLab4

Part	Points
Task 1	1
Task 2	3
Task 3	3
Task 4	3
Bonus	2
Total	12

Table 2: Grading scheme of CodeLab4

Grade	Points
0	≤5
6	6-7
8	8-9
10	≥10

Task 1 – Data Loading & Preparation

A1) Complete the given "*load_data*" function. It loads the data from the dataset folder. Four different classes are present in the dataset, and labels will be assigned based on their filenames. Make sure all three different formats of images are imported.

The function "*fnmatch*" is used to search for a specific string in the filenames. Labels should be assigned as follows:

- Abdomen CT scans -> Class
- Brain CT scans -> Class 1
- Chest X-Rays normal -> Class 2
- Chest X-rays with pneumonia -> Class 3

A2) Load the data using the "*load_data()*" function and verify the dataset sizes. (400 samples and 400 labels)

A3) Plot a random image for each class in the same figure. Hint: you can use the function `random.choice()` to pick from the indexes that correspond to a specific class (`np.where(condition)`)

The images have different characteristics as RGB or Black and White, and all have varying shapes. For this reason, you're going to resize them and convert them in grayscale.

The "*convert_to_grayscale*" function takes an image as input and converts it to grayscale using the `*cv2.cvtColor*` function from the OpenCV library. The `*cv2.COLOR_BGR2GRAY*` parameter specifies the conversion from BGR color space to grayscale. The function returns the grayscale image.

Only the RGB images need to be converted to grayscale. Otherwise cv2 will raise an error, so check for this verifying the shape of each image. (RGB images can have three or four channels, one for each colour + transparency in the case of .png files.)

The `*resize_images_in_list*` function takes a list of images and resizes each image to a specified target size. If the image has three or four channels (indicating a color image), first convert the image to grayscale using the `*convert_to_grayscale*` function. Then, resize the image using the `*cv2.resize*` function and append the resized image to a python list. Final output is the resized images as a *numpy* array. The default target size is set to [128,128] (if no target size is specified).

A4) Resize the Image into (128,128) using the "*resize_images_in_list() function*"

A5) Plot one random image for each class, after resizing and grayscale conversion, in the same figure.

Task 2 – Kernels, Gradients and Convolution

Edge detection is a technique used in image processing to identify the boundaries between different objects in an image. It works by detecting sharp changes in brightness or color within an image. These changes are called edges and can be used to segment an image into different regions. Edge detection is often used as a preprocessing step for other computer vision tasks, such as object detection and image segmentation.

Padding is a technique to increase the spatial dimensions of an input matrix by adding extra rows and columns of zeros around the edges of the input matrix. Padding can prevents loss of information as after a convolution operations the output matrix is smaller than the original one. It can also help to prevent the edges of the input matrix from being ignored by convolution operations. Manually perform the padding operation by adding a row and a column of zeros on all sides of the original matrix.

Look at the given function called `*manual_padding*` that takes a matrix as an input and returns the padded matrix form.

B1) Using the given 4x4 matrix of integers to perform zero padding. Padding is made by adding a row and a column on all sides of the original matrix. Print the padded matrix.

Pooling operations are used to reduce the spatial dimensions of an input matrix while retaining the most important features. The most common types of pooling operations are max pooling and average pooling. Max pooling takes the maximum value within a window of a given size and moves the window across the input matrix. This reduces the spatial dimensions of the input matrix while retaining the most important features. Average pooling takes the average value within a window of a given size and moves the window across the input matrix. This also reduces the spatial dimensions of the input matrix while retaining the most important features. Pooling operations can help to reduce the number of features used by a model, which can help to prevent overfitting and improve generalization.

B2) Using the padded matrix you gained from the last step, perform max pooling with a 2x2 window using the given *"max_pooling" function.*

The given code defines two Sobel kernels for horizontal and vertical edge detection. These kernels are used in image processing to detect edges in an image. The horizontal Sobel kernel is used to detect horizontal edges, while the vertical Sobel kernel is used to detect vertical edges. These kernels are applied to an image using convolution to highlight the edges in the image.

B3) Sobel filter is a commonly used filter in image processing. The purpose of using a Sobel filter is to extract the image's vertical and horizontal edges. The fundamental operation of the Sobel filter is convolution. Given both the vertical and horizontal Sobel kernel, Use the convolution function to apply the two kernels to detect the edges. *"gradient_x"* is the result of the convolution of Sobel horizontal kernel, while "*gradient_y*" is the result of the Sobel vertical kernel.

B4) Calculate the absolute value of the gradient and its orientation (HINT: Think of them as trigonometric components. `*np.arctan2()*` returns a value in radians, convert it to degrees.).

B5) Plot the original image and the results obtained in the step before in a single figure (1x5 subfigures, you can use any colormap for the gradients; for the angles plot, use `*cmap='twilight'*` and plot the color bar.)

Using the *"mpimg.imread()"* function, load the "*airplane.tiff*" image file and:

- B7) Compute the horizontal and vertical gradients.
- B8) Compute the magnitude of the gradient and its orientation.
- B9) Perform max pooling on the image and repeat the previous steps (gradients, magnitudes, orientations).
- B10) Plot all the results in a single plot containing all the above results (2x5 subfigures)

Questions

1. What is the difference between convolution with a kernel and pooling? Hint: Look at the size of the kernels and the resulting images.

2. Why two different kernels are used for horizontal and vertical edge detection? Could a single kernel be used for both?

3. What differences can you spot between the pooled and the original image? and in the resulting gradients?

4. Although the image is very simple and our brains can infer the edges of the object very simply, why are the resulting edges very noisy? What could be implemented to mitigate the noise?

Task 3 – Feature Extraction: Histogram of Gradients and PCA

In the context of feature extraction, HOG (Histogram of Oriented Gradients) and PCA (Principal Component Analysis) are two commonly used techniques.

Histogram of Oriented Gradients (HOG) is a feature extraction technique used to detect objects in images. It operates by dividing an image into small cells and then grouping the gradients in each cell according to a specified number of orientations. These grouped gradients, known as histograms, are concatenated to create a feature vector suitable for object detection. The HOG algorithm calculates gradient orientations in localized regions of an image, providing a comprehensive view of the image's gradient structure. To apply HOG, you can use the 'hog' function from the *'sklearn.feature'* library, which allows you to extract both the HOG image and its corresponding HOG features. You can also print one of the processed images generated using the HOG technique.

The `*get_hog*` function is a function that calculates the Histogram of Oriented Gradients (HOG) features and images for a given set of images.

C1) Extract the HOG feature from the dataset using all four combinations of *size_1=[16,32]* and *orientations=[4, 8].* Computing the HOG features can take up to three minutes.

C2) Plot the 250th image of each variant of HOG images, and the shapes of the respective feature vectors.

Principal Component Analysis (PCA) is a fundamental method in machine learning used for constructing features and dimensionality reduction. In this section, you will utilize the PCA function from *'sklearn.decomposition'* to extract features from the provided dataset. PCA identifies a dataset's principal components, representing the directions in which the data exhibits the most variation. By projecting a feature vector onto a lower-dimensional subspace defined by these principal components, PCA effectively reduces the dimensionality of the data while retaining the essential information capturing the primary data variations.

C3) Apply `*PCA*` to the resized dataset. Hint: `*PCA*` works in one dimension only, so reshape the images accordingly.

C4) Plot (scatter) the first two components of the PCA transformed features.

C5) Plot the explained variance ratio and cumulative explained variance ratio of the first 20 components.

Questions

1. What did you notice between the HOG images extracted using 16 or 32 pixels/cell? Relate to feature vector(s) sizes and compare it with the original flattened images.

2. How does the model performance vary between generated HOG images?

3. In the PCA plot, what do you observe in the two-component plot?

4. Can you relate the description of Classes 2 and 3 in the dataset, with the behavior highlighted by the previous question?

5. What does the explained variance of each component represent?

Task 4 – Model Training

In this task you are going to classify images and the related extracted features. We will define a function "*Classifier*" that fits the classification model, compute Accuracy, Recall, Precision, F1, and AUC and plot the confusion matrix.

The "*Classifier*" function takes the following inputs: sklearn classification model *"clf"*, training features "*X_train*", training labels "y_*train*", test features "*X_test*", and test labels "*y_test*".

D1) Complete the `Classifier` function so that:

- The input model is fitted to the given data, and predictions are made on the test data.
- The following metrics are computed with respect to the test data and printed:
  - Accuracy
  - Precision
  - Recall
  - F1 Score.
  - **Note**: for these last 3 metrics, when computing include the argument `average='macro'`.
- The confusion matrix is plotted and displayed.

D2) Run "*Classifier*" with the original (resized and then flattened) image dataset:

- **K-Nearest Neighbours**: using as *"n_neighbors=[3,4,5]"*
- **Support Vector Classifier:** changing between *"kernels=['linear','poly','rbf']",* when using poly use *"degree=3"*. For all kernels use the following hyperparameters *"gamma='auto', random_state=42, C=1.0, coef0=0.0, tol=1e-3]".*
- **Multi Layer Perceptron:** changing the "*hidden_layer_sizes=[(100),(100,100),(100,100,100)]*". For the remaining parameters use the following hyperparameters *[activation='relu', solver='adam', alpha=0.0001, max_iter=200, shuffle=True, random_state=42]*

D3) Use the four sets of HOG features obtained in the previous tasks, repeat the training process on the following classifiers with the given hyperparameters and settings.

- **K-Nearest Neighbours**: using as *"n_neighbors=4"*
- **Support Vector Classifier:** changing between *"kernel=poly"*. Use the following hyperparameters *"gamma='auto', random_state=42, C=1.0, coef0=0.0, tol=1e-3]".* (Iteration limit!)
- **Multi Layer Perceptron:** changing the "*hidden_layer_size=100)*". For the remaining parameters use the following hyperparameters *[activation='relu', solver='adam', alpha=0.0001, max_iter=200, shuffle=True, random_state=42]*

D4) Run *"Classifier"* with the computed PCA features:

- **K-Nearest Neighbours**: using as *"n_neighbors=4"*
- **Support Vector Classifier:** changing between *"kernel=linear",*.
- **Multi Layer Perceptron:** changing the "*hidden_layer_size=100)*". For the remaining parameters use the following hyperparameters *[activation='relu', solver='adam', alpha=0.0001, max_iter=200, shuffle=True, random_state=42]*

Questions

1. What is the best performing model trained on the flattened images? Why?

2. What is the best performing model using the HOG features? With which HOG set? Why?

3. What is the best performing model using PCA features? With how many components? Why?

4. Regarding the models trained with PCA features, analyze the SVC performance.

5. For the SVC with PCA features, try the RBF and poly kernels in a new cell, using the three variations of the PCA sets. Compare the performances of these kernels? (Iteration limits might be required for poly kernels)

6. Based on the confusion matrices, can you spot a recurring pattern in the classification results? Relate this pattern with the component plot in task 3 and the original dataset.

7. Explain why higher number of layers/neurons in the MLP does not translate in better accuracy?

Bonus Task

For this bonus task you will have to discard the samples belonging to classes 0 and 1, and assign the 0 and 1 values to classes 2 and 3. Then you will train a KNN classifier, a SVC and an MLP classifier using the grid search algorithm to identify the best hyperparameters for the new task.

Use the following parameter grids:

- *knn_param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15]}*
- *svm_param_grid = {'kernel': ['linear', 'rbf','poly'], 'C': [0.1, 1, 10], 'max_iter':[300]}*
- *mlp_param_grid = {'hidden_layer_sizes': [(100,), (100, 100), (100, 100, 100)], 'alpha': [0.0001, 0.001, 0.01], 'max_iter':[300]}*

Use the identified parameters for the following tasks.

- Train with normal flattened images
- Train with HOG
- Train with PCA

Compare and analyze the best-performing hyperparameter set for each model and training data. Identify the most suitable model and corresponding feature set for this CodeLab. Can you propose another feature engineering method to improve the model performance? Explain.

References

[1] F. Kitamura, 'Head CT - hemorrhage'. https://www.kaggle.com/datasets/felipekitamura/head-ct-hemorrhage (accessed Sep. 05, 2023).

[2] N. Islam and K. M. Humaion, 'CT KIDNEY DATASET: Normal-Cyst-Tumor and Stone'. https://www.kaggle.com/datasets/nazmul0087/ct-kidney-dataset-normal-cyst-tumor-and-stone (accessed Sep. 05, 2023).

[3] P. Mooney, ‘Chest X-Ray Images (Pneumonia)’. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (accessed Sep. 05, 2023).