GROUP 8

• VARUN KHANNA	101415041
• AMAN GUPTA	101420524
• ALEX VECCHIETTINI	101439446
• MEHUL SHARMA	101414996
• MANASI RANE	101414495
• BELLE PENSAMIENTO	101398784
• KIRTI TAKRANI	101423492

INDEX

PART 1

IMAGE ENHANCER USING FOURIER TRANSFORM

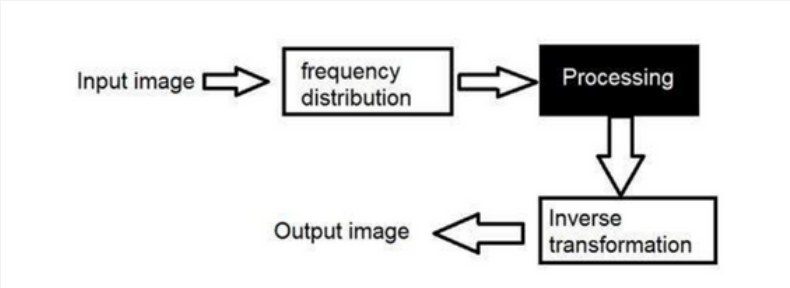
PROBLEM STATEMENT

Create software for image enhancement using Fourier transformation to obtain the image's frequency domain, in particular, implement the following:

- Blurring
- sharpening
- edge detection
- noise suppression

FREQUENCY DOMAIN

- Rate at which the pixel values are changing in spatial domain



FFT

- Each point represents a particular frequency contained in the spatial domain image.
- MAGNITUDE tells "how much" of a certain frequency component is present
- PHASE tells "where" the frequency component is in the image.

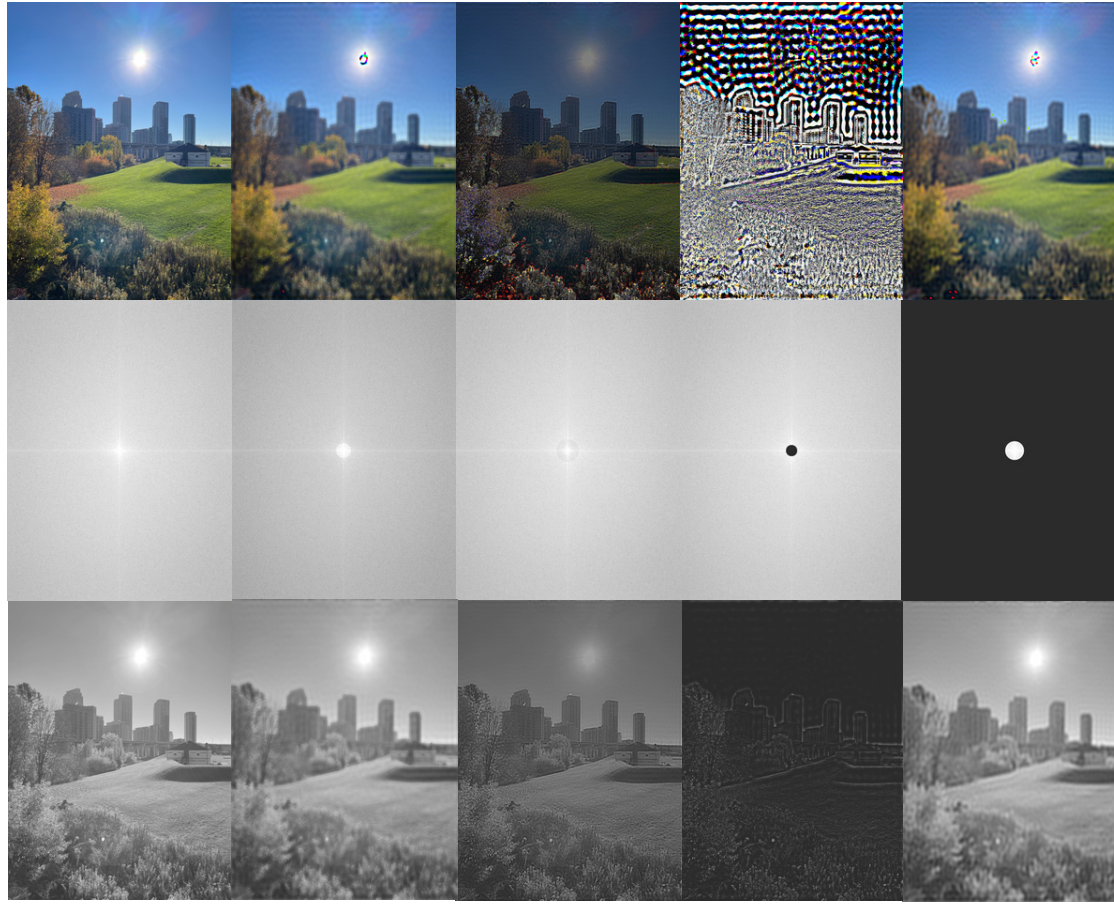
LPF AND HPF

LPF and HPF can be applied to the frequency domain to obtain blurring, sharpening, edge detection, and noise filtering. Different parameters will yield different results, in general, we can see that:

- Blurring is obtained with a LPF that dampens the high frequencies
- Sharpening can be obtained by applying an HPF to reduce the values of the low frequencies
- Edge detection is achieved by increasing the effect of the HPF and bringing the low frequencies to values close to 0
- Noise filtering, similar to blurring, is achieved through an LPF with a stronger effect on the high frequencies.

HOWEVER...

RESULTS



RESULTS EVALUATION

As we saw LPF and HPF are not ideal, in particular for color images where a color distortion becomes evident.

Furthermore, we can see how dampening values can result in a darker image.

While the latter can be resolved with image post-processing in the space domain to increase the overall brightness of the image and the first one can be resolved partially by fine-tuning the filter to the image, to obtain marginally better results there are better algorithms in the space domain to achieve the required image enhancements, such as:

- Sobel filter for edge detection
- Gaussian, Mean, Median, and Bilateral filters can be used to deal with different kind of noise.

Further improvements will be directed towards the development of a GUI to show in real time the result of different parameters in the LPF and HPF.

PART 2

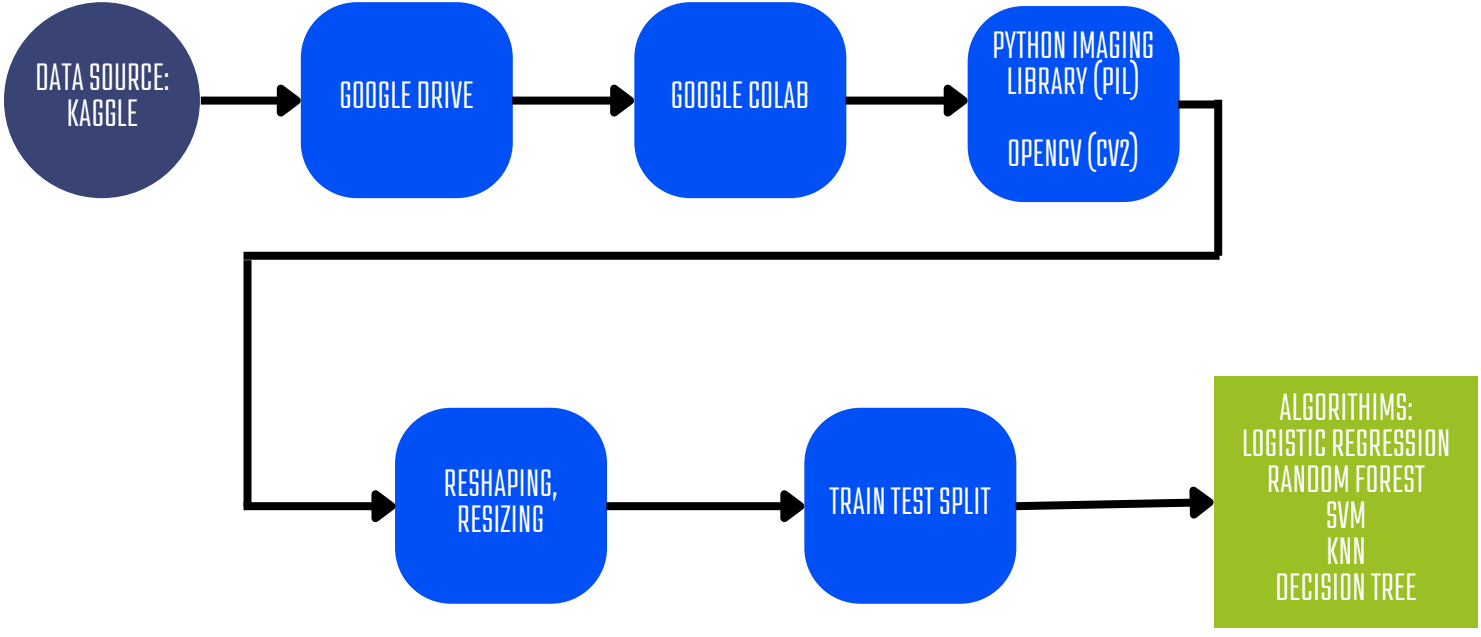
LANDSCAPE BASED IMAGE CLASSIFICATION

DATASET

- **https://www.kaggle.com/datasets/puneet6060/intel-image-classification**
- Our image dataset includes digital images curated for testing, training, and evaluating the performance of machine learning and artificial intelligence (AI) algorithms, commonly computer vision algorithms.
- This Data contains around 25 k images of size 150x150 distributed under 6 categories.

- {'BUILDINGS' -> 0,'FOREST' -> 1,'GLACIER' -> 2,'MOUNTAIN' -> 3,'SEA' -> 4,'STREET' -> 5 }

DATA PIPELINE



DATA PREPROCCESSING

```
def train_data():
    train_data_buildings = []
    train_data_forest = []
    for i in tqdm(os.listdir(train_buildings)):
        # load each image (i) and convert to grayscale
        path = os.path.join(train_buildings, i)
        img_i = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        # resizing the data
        img_i = cv2.resize(img_i, (image_size, image_size))
        # put each processed image into the empty list defined above
        train_data_buildings.append(img_i)
    # now do the same for each test forest image (j)
    for j in tqdm(os.listdir(train_forest)):
        path = os.path.join(train_forest, j)
        img_j = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img_j = cv2.resize(img_j, (image_size, image_size))
        train_data_forest.append(img_j)

    train_data = np.concatenate((np.asarray(train_data_buildings),np.asarray(train_data_forest)),axis=0)
    return train_data
```

Loading the Data into Grayscale

Train Test Split

```
[ ] # Splitting the data into test and train
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.15, random_state = 42)
    number_of_train = x_train.shape[0]
    number_of_test = x_test.shape[0]
    num_px = x_train.shape[2]
```

```
[ ] # Reshaping the training and test set
    x_train_flatten = x_train.reshape(number_of_train, x_train.shape[1]*x_train.shape[2])
    x_test_flatten = x_test.reshape(number_of_test, x_test.shape[1]*x_test.shape[2])
    print("X train flatten",x_train_flatten.shape)
    print("X test flatten",x_test_flatten.shape)
```

```
[ ] x_data = np.concatenate((train_data,test_data), axis = 0)
    x_data = (x_data-np.min(x_data))/(np.max(x_data)-np.min(x_data))
```

```
[ ] z1 = np.zeros(2191)
    o1 = np.ones(2271)
    Y_train = np.concatenate((o1, z1), axis=0)
    z = np.zeros(437)
    o = np.ones(474)
    Y_test = np.concatenate((o, z), axis=0)
```

creating target classes

(1 = building, 0= forest)

```
[ ] y_data = np.concatenate((Y_train,Y_test), axis = 0).reshape(x_data.shape[0],1)
```

```
# averages the pixels to normalize
x_train = x_train/255.0
x_test = x_test/255
```

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

Normalizes the pixel values, room for improvement

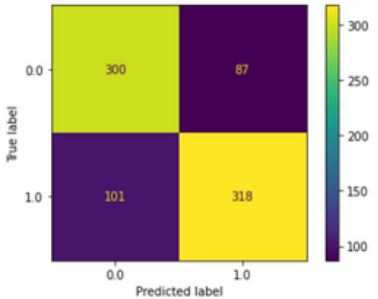
LOGISTIC REGRESSION

- Logistic Regression is one of the supervised machine learning algorithms that can work on both binary and multiclass classification very well.
- It operates basically through a sigmoidal function for values ranging between 0 and 1.
- In the context of image processing, this could mean identifying whether a given image belongs to a particular class (y=1) or not (y=0), e.g. "cat" or "not cat".
- Formulas used in logistic regression:
  - $z = w^T x + b$ Here z is the output variable, and x is the input variable. w and b will be initialized as zeros to start with and they will be modified by the number of iterations while training the model.

LOGISTIC REGRESSION

- This output z is passed through a sigmoid ($y_{predict} = a = sigmoid(z)$ that returns a value between 0 and 1.
- $a = \frac{1}{1 + e^{-z}}$ will be the final output that is the value in the 'y_train' or 'y_test'.

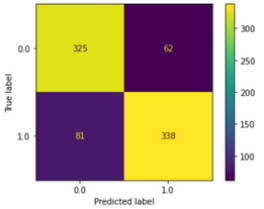
- Hyperparameters which gave the maximum accuracy: LogisticRegression(C=10,penalty="l2",random_state = 42)
- The accuracy we got from LR Model on the test dataset is **75%**
- Confusion Matrix: To see if its better at classifying forests or buildings



Buildings slightly more precise

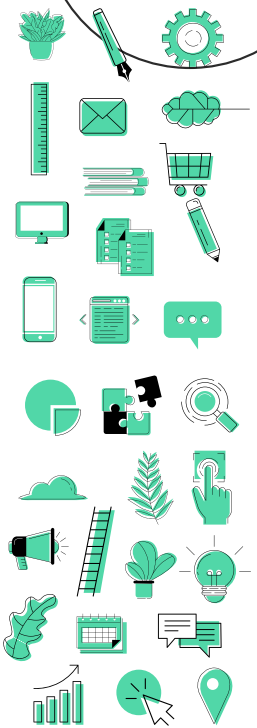
RANDOM FOREST

- A Random Forest is a classifier consisting of a collection of tree structured classifiers (...) independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x .
- $$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$
- This formula uses the class and probability to determine the Gini of each branch on a node, determining which of the branches is more likely to occur.
- Here, pi represents the relative frequency of the class you are observing in the dataset and c represents the number of classes.
- As expected random forest gives better performance than logistic regression with an accuracy of **82%** on the test dataset.
- More precise for Buildings



Conclusion

	Model	Accuracy Score
0	Logistic Regression	0.767
1	Random Forest	0.815
2	SVM	0.815
3	KNN	0.692
4	Decision Tree	0.711



Further Enhancements

Randomized SearchCV

GridsSearchCV

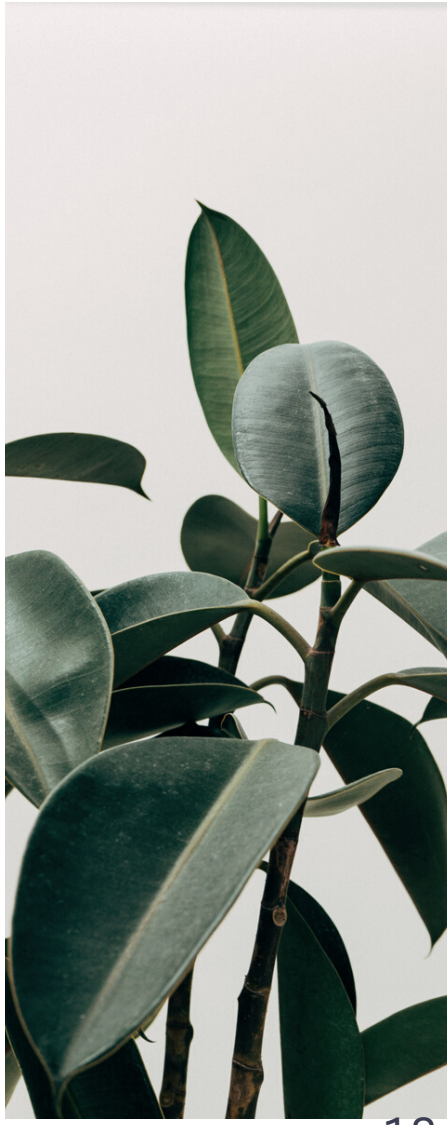
Ensemble Methods

Feature Selection

Confusion Matrices for other Algorithsm

THANK YOU !!!

Github repo: https://github.com/alexvecchiettini/image-processing.git



Question???