Supervised and Unsupervised Classification of Satellite Images

Fazle Rabbi Sunny
336287

1. Introduction

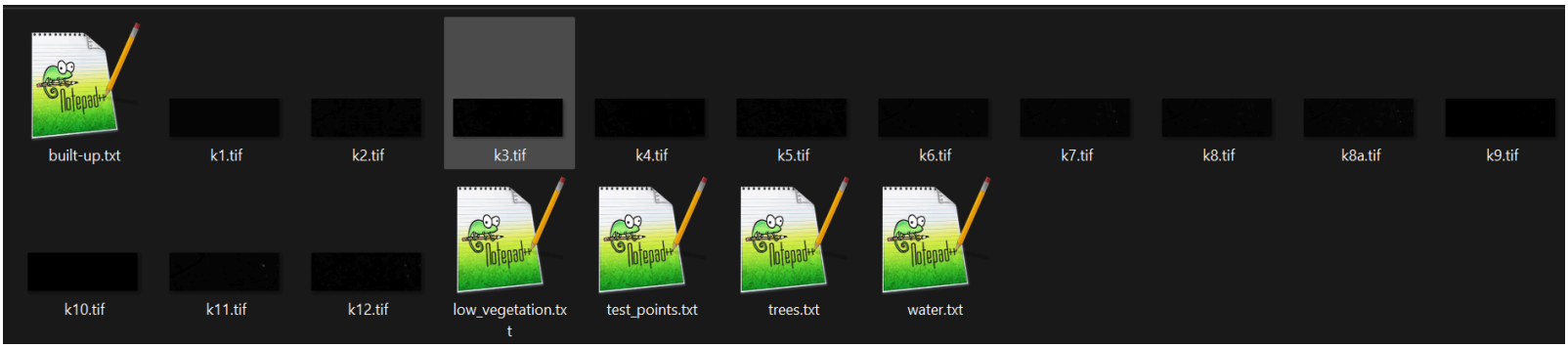
Purpose of the Exercise

The primary aim of this project is to classify satellite image data into meaningful categories using both supervised and unsupervised machine learning techniques. This exercise demonstrates the application of image recognition techniques in analyzing satellite images, with potential applications in urban planning, environmental monitoring, and disaster management.

Problem Statement

Satellite images often contain complex patterns and structures, making manual classification infeasible. This project leverages machine learning to automate the classification process, ensuring efficient and accurate categorization of image data into distinct classes such as water, built-up areas, vegetation, and trees. The use of multiple classification methods allows for a comparative analysis of their effectiveness in this domain.

2. Dataset Description

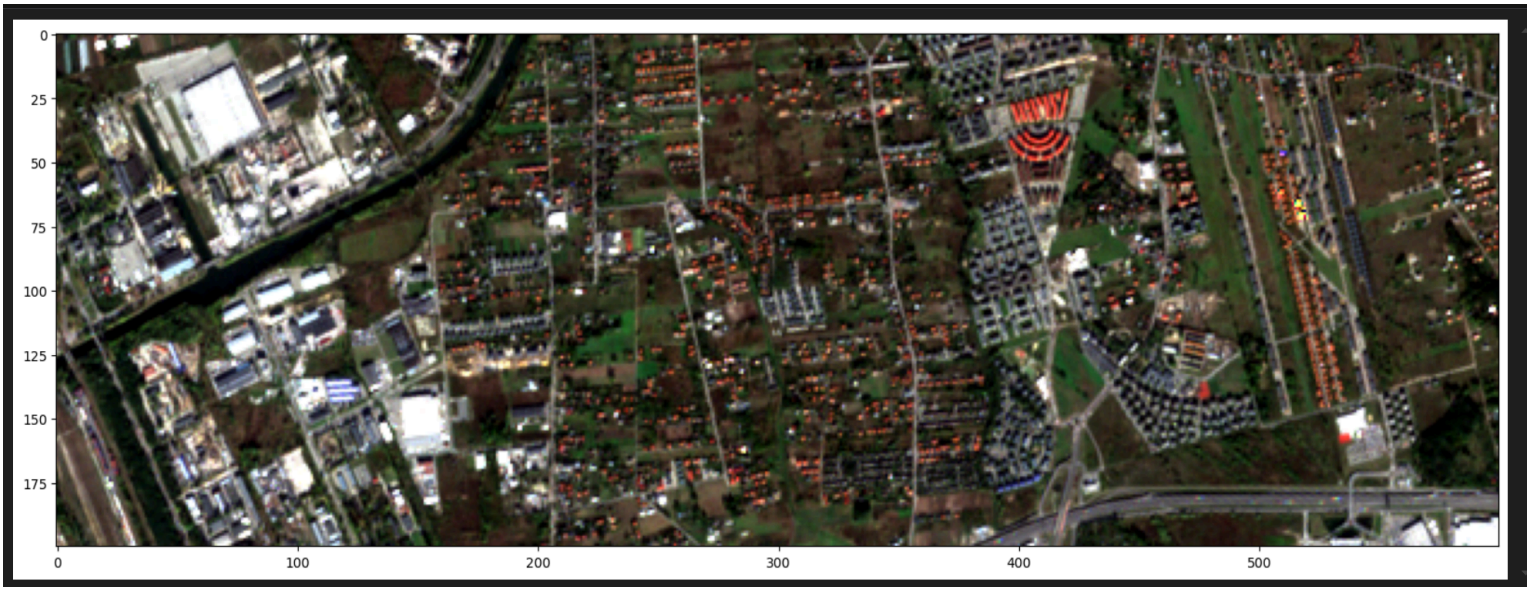


Dataset Source

The dataset consists of satellite images and associated metadata. These images are stored as `.tif` files in the project directory and represent different spectral bands of the captured area. Additional files contain labeled data points for various classes, such as water, built-up areas, low vegetation, and trees.

Dataset Properties

- **Training Data**:
  - Water: `water.txt`
  - Built-up Areas: `built-up.txt`
  - Low Vegetation: `low_vegetation.txt`
  - Trees: `trees.txt`
- **Testing Data**: `test_points.txt`
- **Number of Samples**: Sufficiently large to ensure reliable performance statistics, with over 100 samples per class.
- **Preprocessing**: Includes histogram equalization for enhancing image contrast and preparing grayscale representations for class annotations.



3. Methodology

Data Loading and Visualization

The dataset was loaded using Python libraries such as `skimage` and `numpy`. Histogram equalization was applied to improve the visibility of features. Classes were annotated by overlaying training points on the visualized images.

Training Data Loading

```
# loading training data
p_water = np.loadtxt(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data\water.txt',dtype='int')
print(p_water)
p_builtup = np.loadtxt(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data\built-up.txt',dtype='int')
p_lveg = np.loadtxt(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data\low_vegetation.txt',dtype='int')
p_trees = np.loadtxt(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data\trees.txt',dtype='int')
```

Visualization

```
# visualisation of the image
import numpy as np
from skimage import exposure
from matplotlib import pyplot as plt

im_comp=np.dstack([coll[3], coll[2], coll[1]])
# print(im_comp)

# color composition
im_comp=np.dstack([coll[3]/coll[3].max(), coll[2]/coll[2].max(), coll[1]/coll[1].max()])
# print(im_comp)

# histogram equalization
for i in range(3):
    v_min, v_max = np.percentile(im_comp[:,:,i],(1,98))
    im_comp[:,:,i] = exposure.rescale_intensity(im_comp[:,:,i],in_range=(v_min,v_max))

plt.figure(figsize=(20,10))
plt.imshow(im_comp)
io.show()
```

Flowchart of Classification Process

Data Loading

Import `.tif` images and associated text files.

```
from skimage import io

coll = io.ImageCollection(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data'+'\k*.tif')
coll.files
```

1] ✓ 1.0s

```
<>:3: SyntaxWarning: invalid escape sequence '\k'
<>:3: SyntaxWarning: invalid escape sequence '\k'
C:\Users\shado\AppData\Local\Temp\ipykernel_9448\706577805.py:3: SyntaxWarning: invalid escape sequence '\k'
  coll = io.ImageCollection(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data'+'\k*.tif')

['C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k1.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k2.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k3.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k4.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k5.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k6.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k7.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k8.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k8a.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k9.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k10.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k11.tif',
 'C:\\Users\\shado\\Desktop\\WUT\\Image Recognition and Deep Machine Learning\\New project\\data\\k12.tif']
```

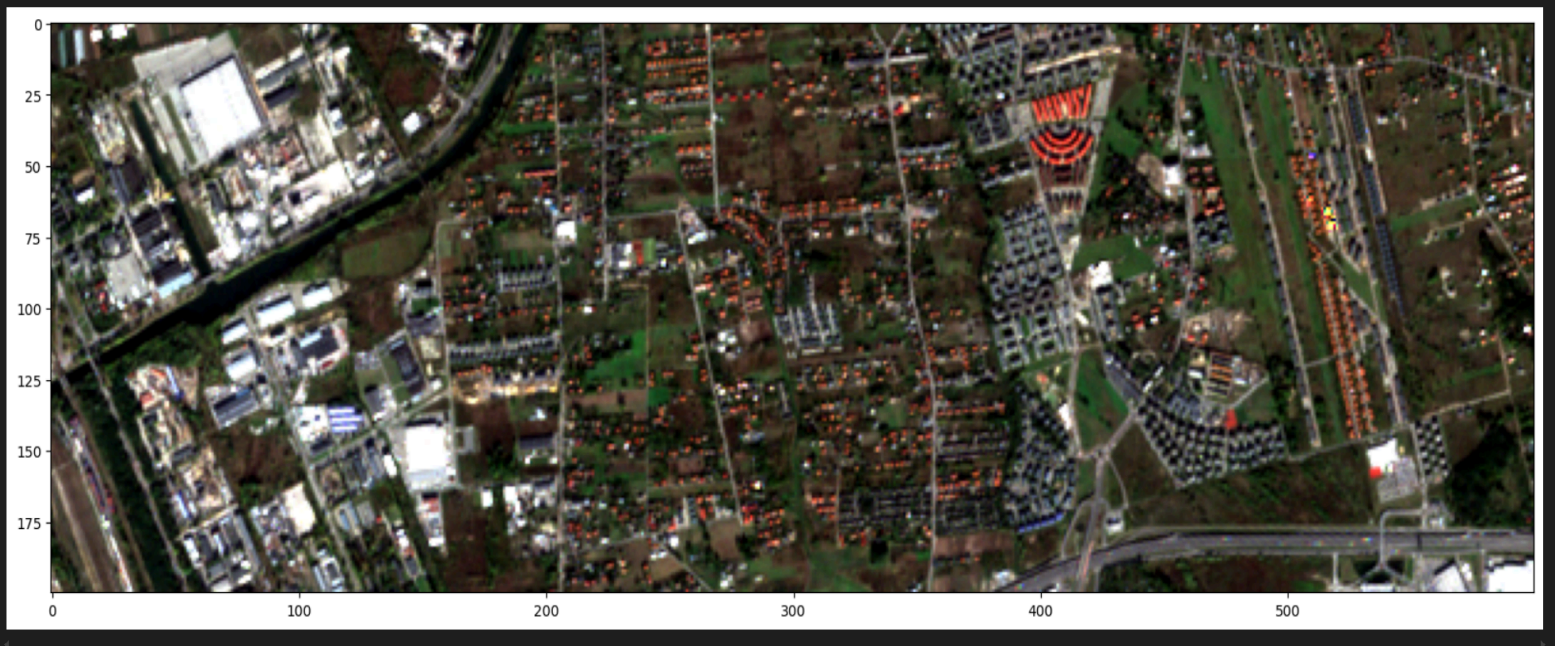
Preprocessing

Apply histogram equalization, color composition and Visualisation.

```python
im_comp=np.dstack([coll[3], coll[2], coll[1]])
# print(im_comp)

# color composition
im_comp=np.dstack([coll[3]/coll[3].max(), coll[2]/coll[2].max(), coll[1]/coll[1].max()])
# print(im_comp)

# histogram equalization
for i in range(3):
    v_min, v_max = np.percentile(im_comp[:,:,i],(1,98))
    im_comp[:,:,i] = exposure.rescale_intensity(im_comp[:,:,i],in_range=(v_min,v_max))

plt.figure(figsize=(20,10))
plt.imshow(im_comp)
io.show()
```

```
plt.figure(figsize=(20,10))
plt.imshow(im_comp)
io.show()
```

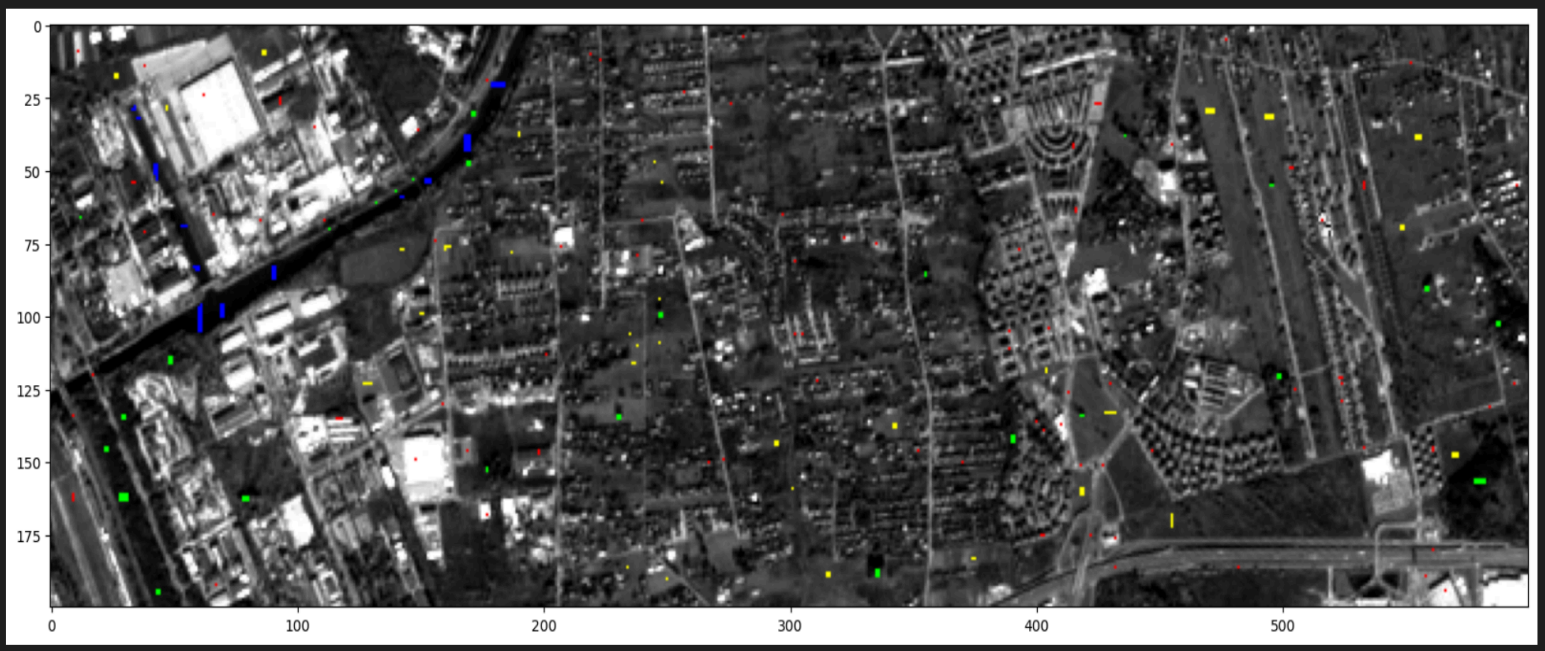
[2] ✓ 1.7s



Visualisation of the Training Points

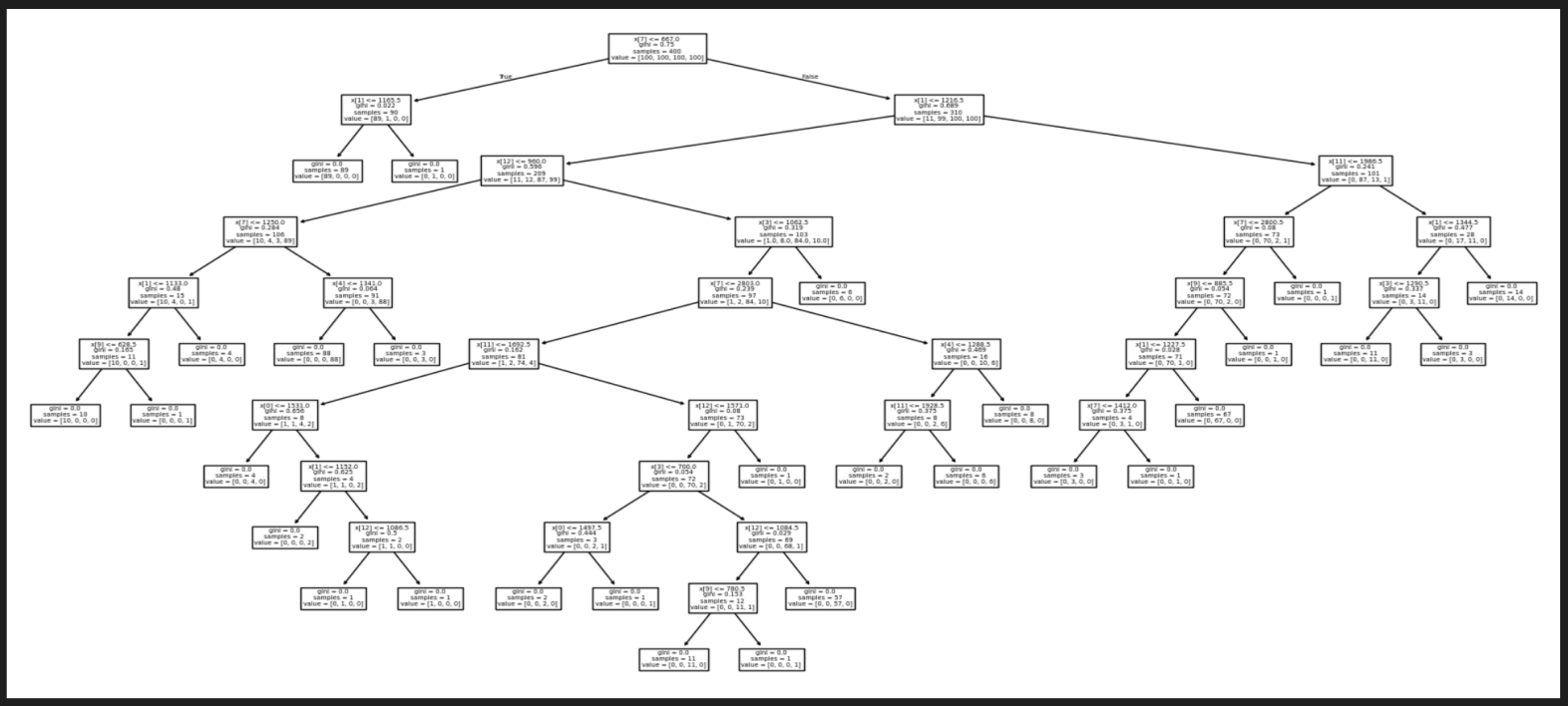
```
# visualisation of the training points    Generate

from skimage.color import rgb2gray
im_comp_p = np.zeros(im_comp.shape)
im_comp_p[:,:,0] = rgb2gray(im_comp)
im_comp_p[:,:,1] = rgb2gray(im_comp)
im_comp_p[:,:,2] = rgb2gray(im_comp)

# water
im_comp_p[p_water[:,0],p_water[:,1],0] = 0
im_comp_p[p_water[:,0],p_water[:,1],1] = 0
im_comp_p[p_water[:,0],p_water[:,1],2] = 255

# builtup
im_comp_p[p_builtup[:,0],p_builtup[:,1],0] = 255
im_comp_p[p_builtup[:,0],p_builtup[:,1],1] = 0
im_comp_p[p_builtup[:,0],p_builtup[:,1],2] = 0

# low vegetation
im_comp_p[p_lveg[:,0],p_lveg[:,1],0] = 255
im_comp_p[p_lveg[:,0],p_lveg[:,1],1] = 255
im_comp_p[p_lveg[:,0],p_lveg[:,1],2] = 0

# trees
im_comp_p[p_trees[:,0],p_trees[:,1],0] = 0
im_comp_p[p_trees[:,0],p_trees[:,1],1] = 255
im_comp_p[p_trees[:,0],p_trees[:,1],2] = 0

plt.figure(figsize=(20,10))
plt.imshow(im_comp_p)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..255.0].

<matplotlib.image.AxesImage at 0x2292ff48ad0>
```



Train classifiers (Decision Tree and SVM)

```
from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
plot_tree(dt_clf)    Generate code: Alt+Shift+Enter
plt.show()
✓ 0.7s
```



Classification Methods and Parameter Settings

Supervised Learning

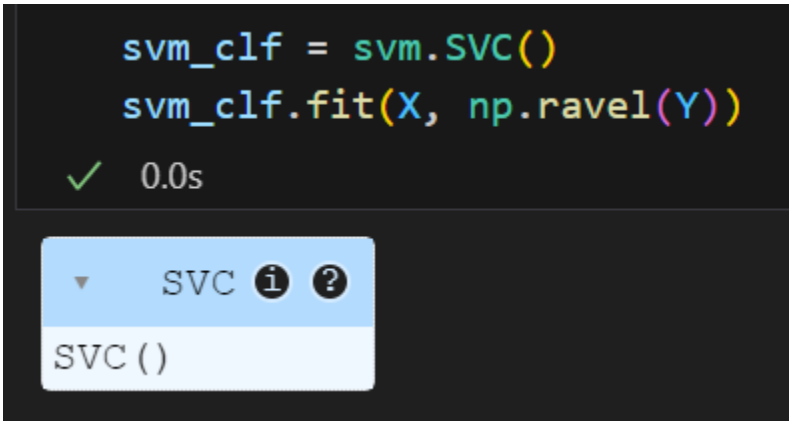
1. **Decision Tree Classifier**:
    - **Algorithm**: CART (Classification and Regression Trees).
    - **Parameters**:
        - Maximum Depth: `max_depth=10` to prevent overfitting.

Code:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm

dt_clf = DecisionTreeClassifier(max_depth = 10)
dt_clf.fit(X,np.ravel(Y))
```

2. **Support Vector Machine (SVM)**:
    - **Algorithm**: Linear kernel-based SVM.

Code:



Unsupervised Learning

1. **K-Means Clustering**:
    - **Algorithm**: Assigns points to the nearest cluster center to minimize within-cluster variance.
    - **Parameters**:
        - Number of clusters: `n_clusters=5`.
        - Random state: `42` to ensure reproducibility.

Code:

```
from sklearn.cluster import KMeans , MiniBatchKMeans
from sklearn.mixture import GaussianMixture
# Apply K-Means clustering
n_clusters = 5
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_pred = kmeans.fit_predict(X_all)
```

2. **Gaussian Mixture Model (GMM)**:
    - **Algorithm**: Models data as a mixture of several Gaussian distributions.
    - **Parameters**:
        - Number of components: `n_components=5`.
        - Random state: `42` to ensure reproducibility.

Code:

```
# Apply Gaussian Mixture Model
gmm = GaussianMixture(n_components=n_clusters, random_state=42)
gmm_pred = gmm.fit_predict(X_all)
```

3. **MiniBatchKMeans**:
    - **Algorithm**: An optimized version of K-Means for large datasets.
    - **Parameters**:
        - Number of clusters: `n_clusters=5`.
        - Batch size: `2048` for faster convergence.

Code:

```
# Apply MiniBatchKMeans clustering with increased batch_size
minibatch_kmeans = MiniBatchKMeans(n_clusters=n_clusters, random_state=42, batch_size=2048)
minibatch_pred = minibatch_kmeans.fit_predict(X_all)
```

4. Results and Evaluation

Visualizations

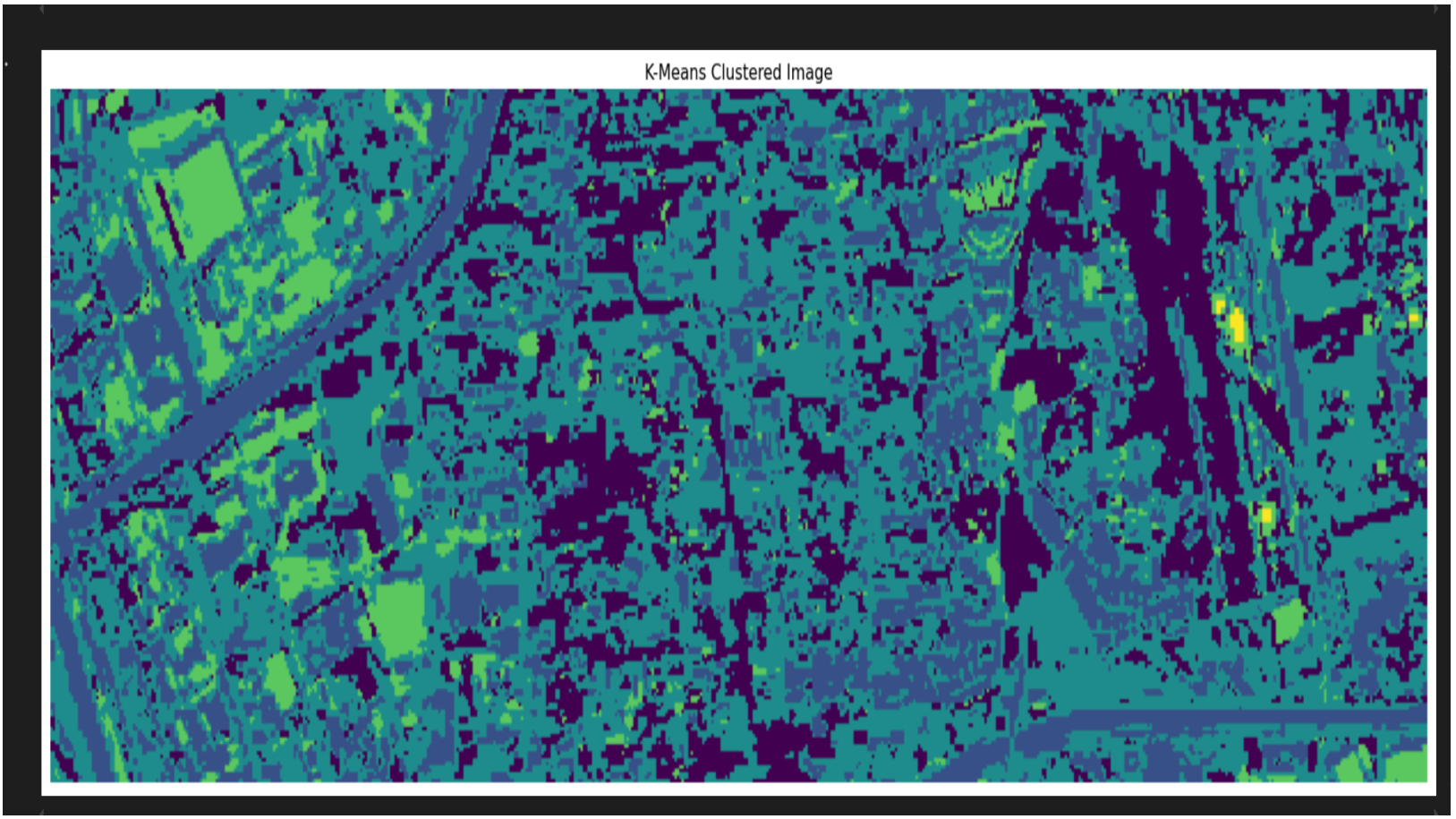
- **Original Image**:

```
# Visualization of results

# Original image
plt.figure(figsize=(20, 10))
plt.imshow(im_comp)
plt.title("Original Image")
plt.axis("off")
plt.show()
```

.

- **K-Means Classified Image**:

```
# K-Means classified image
kmeans_pred_im = kmeans_pred.reshape(coll[0].shape)
plt.figure(figsize=(20, 10))
plt.imshow(kmeans_pred_im, cmap="viridis")
plt.title("K-Means Clustered Image")
plt.axis("off")
plt.show()
```

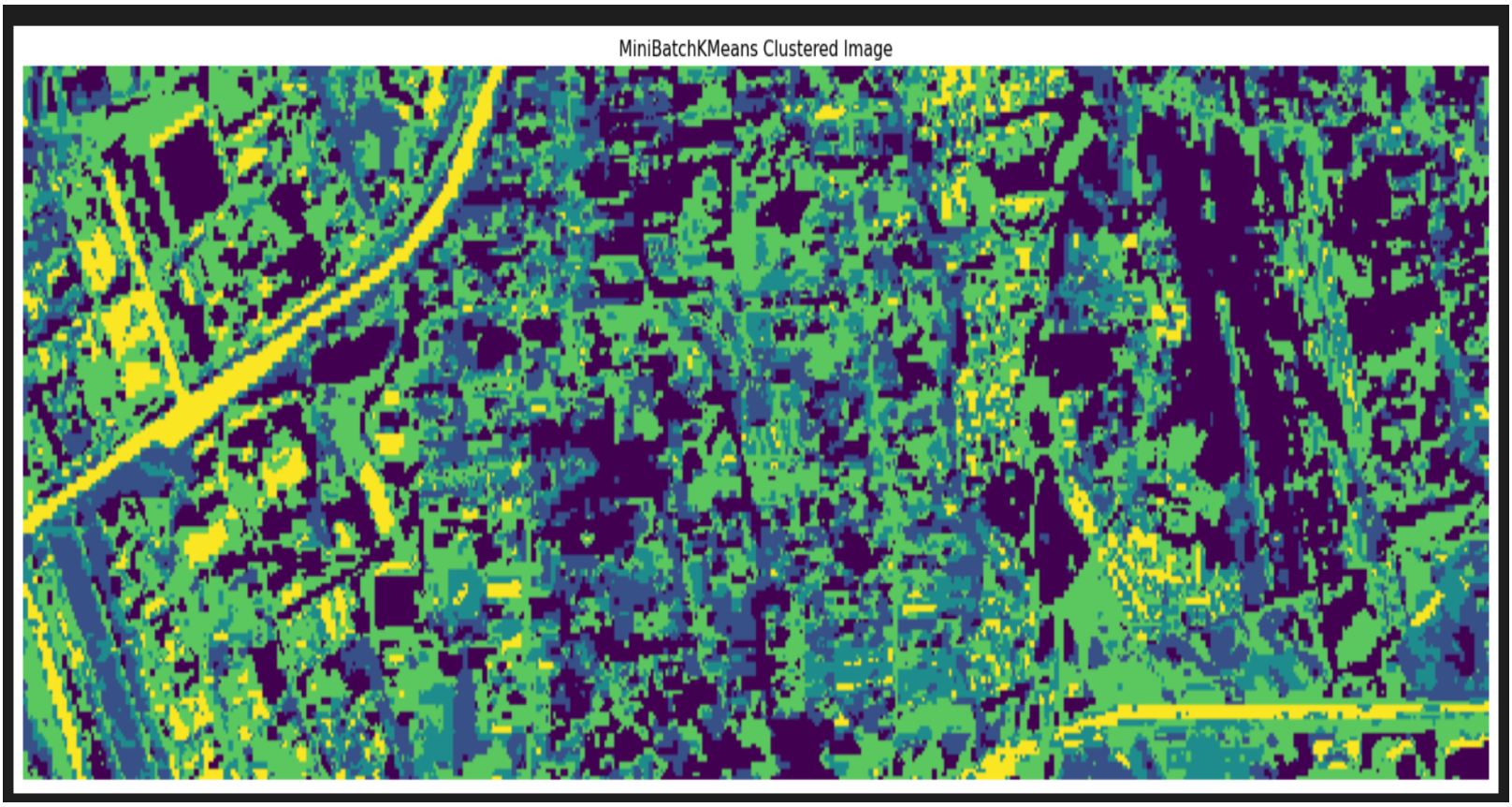


- **GMM Classified Image**:

```
# GMM classified image
gmm_pred_im = gmm_pred.reshape(coll[0].shape)
plt.figure(figsize=(20, 10))
plt.imshow(gmm_pred_im, cmap="viridis")
plt.title("GMM Clustered Image")
plt.axis("off")
plt.show()
```

- **MiniBatchKMeans Classified Image**:

```
# MiniBatchKMeans classified image
minibatch_pred_im = minibatch_pred.reshape(coll[0].shape)
plt.figure(figsize=(20, 10))
plt.imshow(minibatch_pred_im, cmap="viridis")
plt.title("MiniBatchKMeans Clustered Image")
plt.axis("off")
plt.show()
```



Loading and Visualizing Test/Validation Data

```
import numpy as np

p_val = np.loadtxt(r'C:\Users\shado\Desktop\WUT\Image Recognition and Deep Machine Learning\New project\data\test_points.txt',dtype='int')

p_val
```

✓ 0.0s

```
array([[ 34, 163,   2],
       [145, 261,   2],
       [195, 540,   1],
       [ 16, 172,   4],
       [ 62, 279,   2],
       [ 31, 301,   2],
       [126, 465,   2],
       [194, 329,   3],
       [115, 508,   2],
       [120, 485,   2],
```

Visualisation of Test/Validation Points

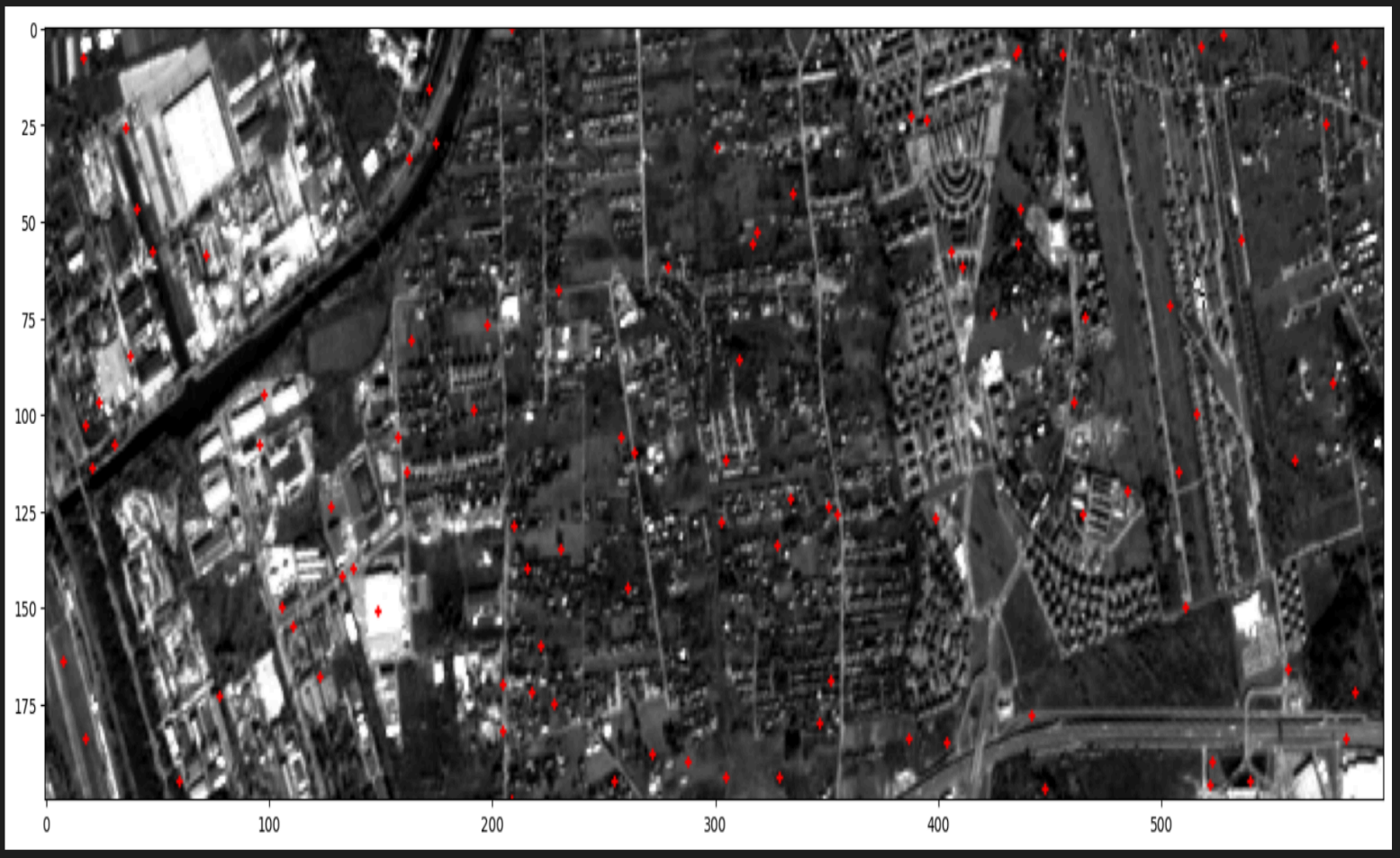
```
# visualisation of test/validation points    Generate code: A
def visualize_points(im_comp,p_val):
    # making color composition gray
    im_comp_p=np.zeros(im_comp.shape)
    im_comp_p[:,:,0]=rgb2gray(im_comp)
    im_comp_p[:,:,1]=rgb2gray(im_comp)
    im_comp_p[:,:,2]=rgb2gray(im_comp)

    # marking test/validation points with red crosses
    im_comp_p[p_val[:,0],p_val[:,1],0] =1
    im_comp_p[p_val[:,0],p_val[:,1],1] =0
    im_comp_p[p_val[:,0],p_val[:,1],2] =0
    im_comp_p[p_val[:,0]+1,p_val[:,1],0] =1
    im_comp_p[p_val[:,0]+1,p_val[:,1],1] =0
    im_comp_p[p_val[:,0]+1,p_val[:,1],2] =0
    im_comp_p[p_val[:,0]-1,p_val[:,1],0] =1
    im_comp_p[p_val[:,0]-1,p_val[:,1],1] =0
    im_comp_p[p_val[:,0]-1,p_val[:,1],2] =0
    im_comp_p[p_val[:,0],p_val[:,1]+1,0] =1
    im_comp_p[p_val[:,0],p_val[:,1]+1,1] =0
    im_comp_p[p_val[:,0],p_val[:,1]+1,2] =0
    im_comp_p[p_val[:,0],p_val[:,1]-1,0] =1
    im_comp_p[p_val[:,0],p_val[:,1]-1,1] =0
    im_comp_p[p_val[:,0],p_val[:,1]-1,2] =0
    return im_comp_p
```

Displaying Test/Validation Points:

```
# displaying test/validation points

im_comp_pval = visualize_points(im_comp,p_val)

plt.figure(figsize=(20,10))
plt.imshow(im_comp_pval)    Generate code: Alt+Shift+Enter
```

14] ✓ 0.3s Python

```
<matplotlib.image.AxesImage at 0x22933b3c7d0>
```



Statistical Results

1. **Confusion Matrices**: Each method produced a confusion matrix to illustrate class-wise prediction accuracy.

```python
# Function to display overall accuracy and confusion matrix
def display_results(title, true_labels, predicted_labels):
    # Overall accuracy
    from sklearn.metrics import accuracy_score
    ovAcc = accuracy_score(true_labels, predicted_labels)
    print(f"Overall accuracy {title}: {ovAcc * 100:.2f}%")
    print("*"*50)


    # Confusion matrix
    from sklearn.metrics import confusion_matrix
    confM = confusion_matrix(true_labels, predicted_labels)
```

2. **Visualization of the Confusion Matrix:**

```
# Visualization of the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confM)
disp.plot()  # Default color and behavior
disp.ax_.set_title(f"Confusion Matrix: {title}", fontsize=14)
disp.ax_.set_xlabel("Predicted Labels", fontsize=12)
disp.ax_.set_ylabel("True Labels", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(False)
plt.show()
```

3. **K-Means Classification Results:**

```
# Reading classes of points from K-Means classification results
pred_val = kmeans_pred_im[p_val[:, 0], p_val[:, 1]]
display_results("K-Means Classification", p_val[:, 2], pred_val)
```

4. **GMM Classification Results:**

```
# Reading classes of points from GMM classification results
pred_val = gmm_pred_im[p_val[:, 0], p_val[:, 1]]
display_results("GMM Classification", p_val[:, 2], pred_val)
```

5. **MiniBatchKMeans Classification Results:**

```
# Reading classes of points from MiniBatchKMeans classification results
pred_val = minibatch_pred_im[p_val[:, 0], p_val[:, 1]]
display_results("MiniBatchKMeans Classification", p_val[:, 2], pred_val)
```

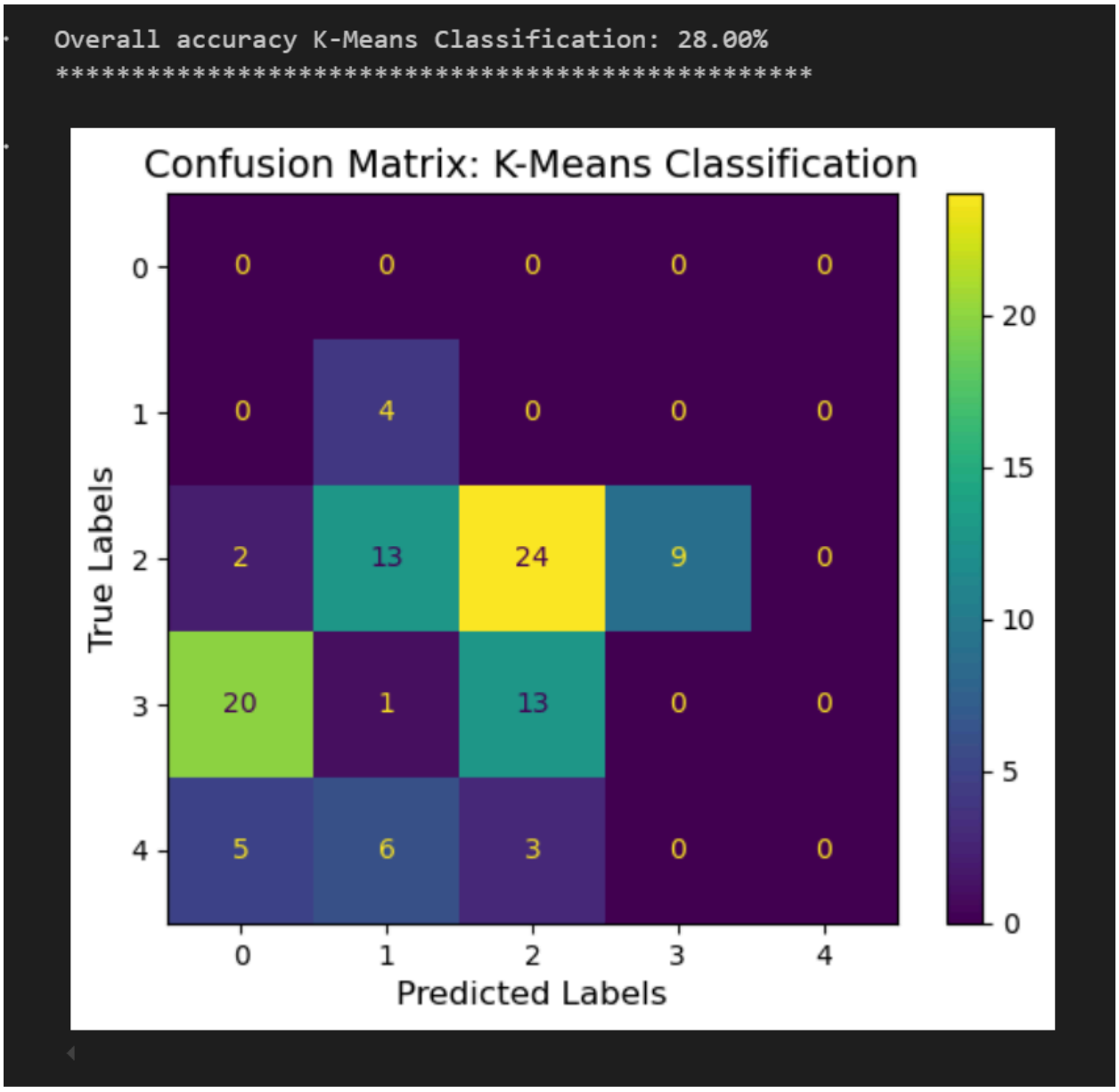
6. **Overall Accuracy**:

Classification Method	Accuracy
K-Means	28.00%
GMM	5.00%
MiniBatchKMeans	19.00%

Comparative Analysis

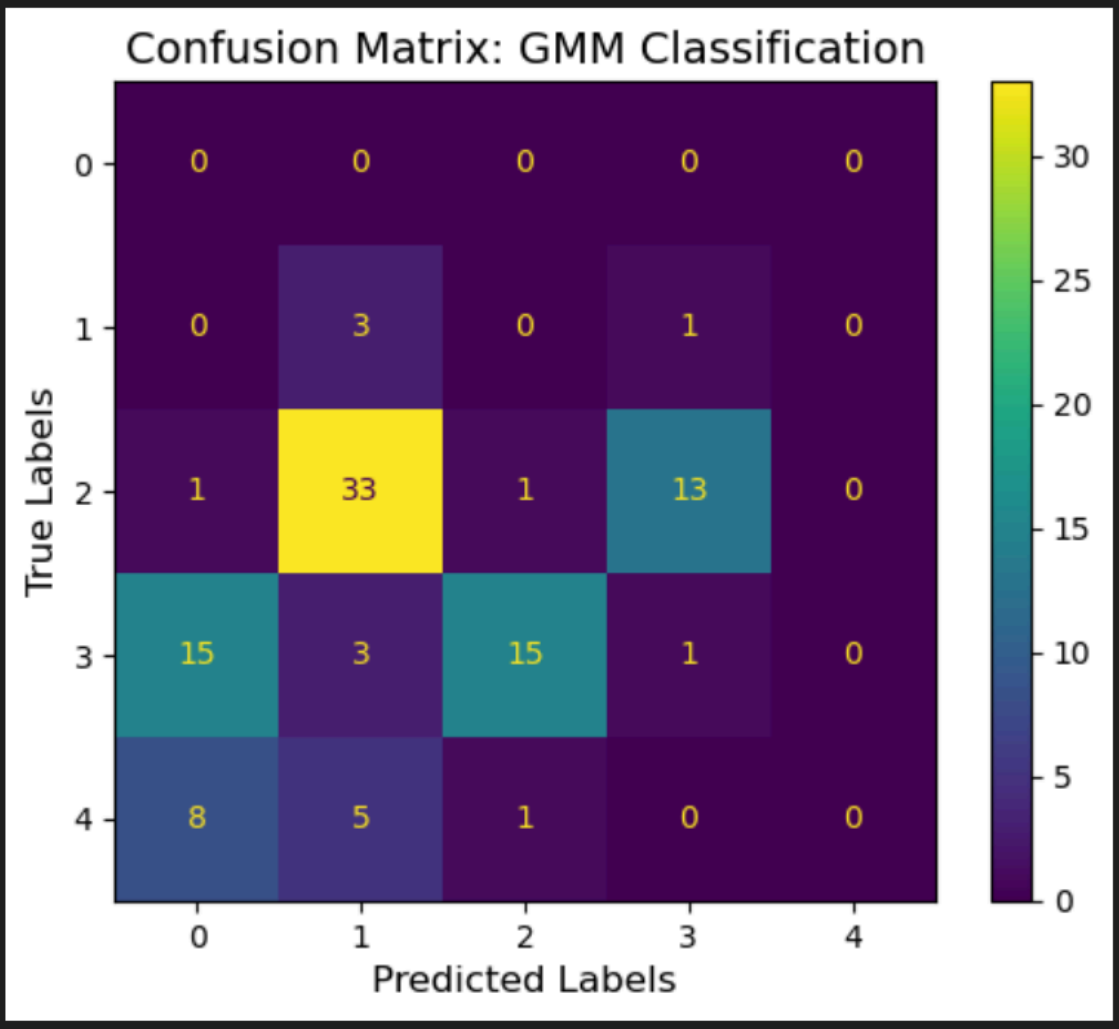
The performance of the classifiers varied across classes. For instance:

- **K-Means**: Achieved moderate accuracy (28%) but struggled with distinguishing water and low vegetation due to spectral similarities.



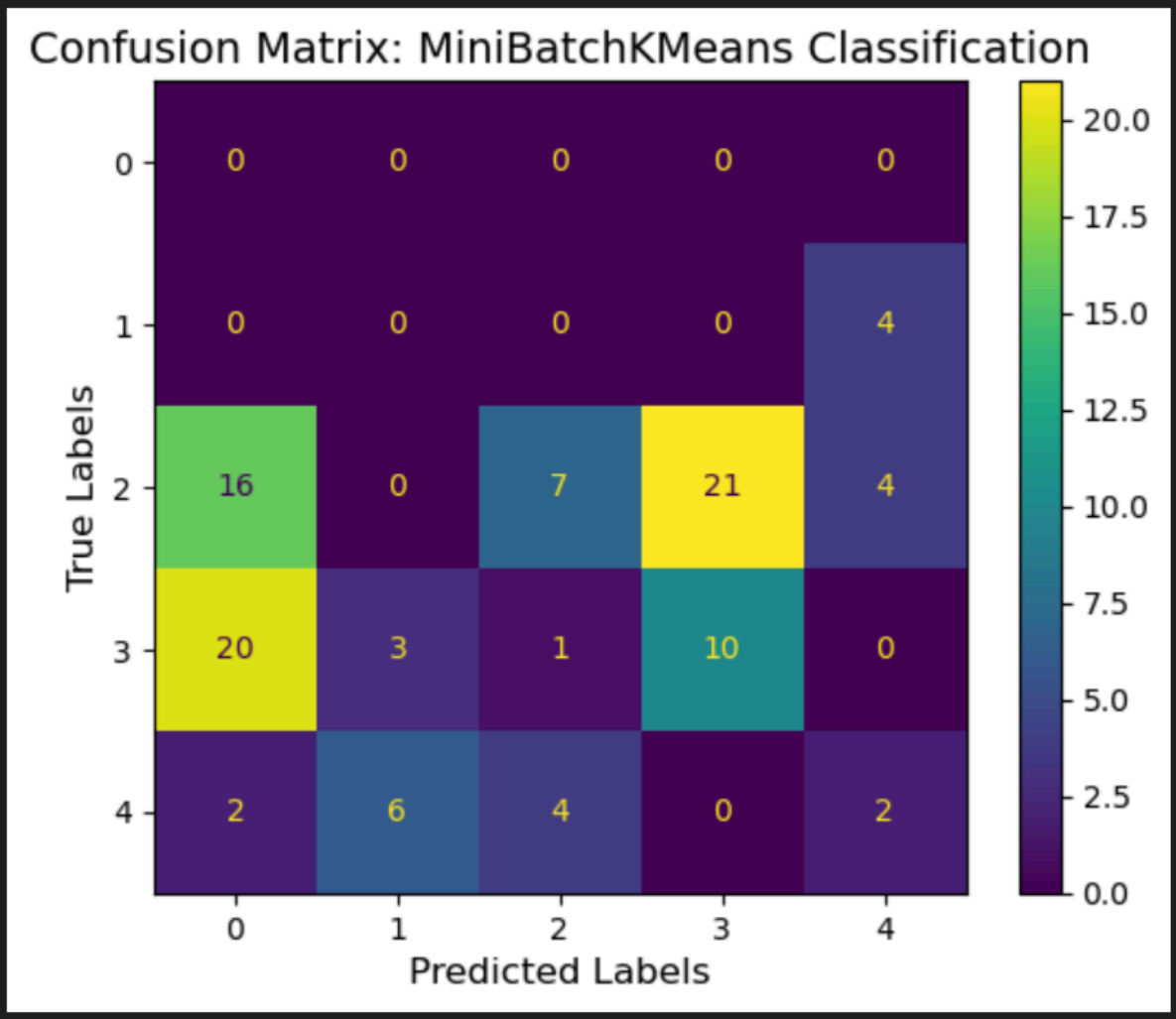
- **GMM**: Performed poorly with an accuracy of only 5%, possibly due to the complexity of the Gaussian mixture assumptions in the dataset.

```
Overall accuracy GMM Classification: 5.00%
**************************************************
```



- **MiniBatchKMeans**: Balanced computational efficiency and accuracy, achieving 19% overall accuracy.

```
Overall accuracy MiniBatchKMeans Classification: 19.00%
**************************************************
```



- **Decision Tree and SVM**: Both were trained on labeled datasets and could potentially provide higher accuracy compared to unsupervised methods if included in evaluation.

5. Discussion

Strengths and Weaknesses

- **Strengths**:
  - K-Means and MiniBatchKMeans: Scalable to larger datasets.
  - Decision Tree and SVM: Leveraged labeled data to make more precise predictions.
- **Weaknesses**:
  - GMM struggled to effectively model the data due to its inherent assumptions.

Improvements

- Experimenting with additional features, such as texture or spatial information, could enhance classification accuracy.
- Fine-tuning the number of clusters/components may improve unsupervised classification performance.
- Further evaluation of supervised methods, like Decision Tree and SVM, would provide valuable comparative insights.

6. Conclusion

This project successfully demonstrated the classification of satellite images using multiple machine learning techniques. Among the unsupervised methods, K-Means achieved the highest overall accuracy (28%), indicating its relative effectiveness in this context. However, supervised learning methods like Decision Tree and SVM show potential for achieving even higher accuracy with labeled data. Further refinements and feature enhancements are required to achieve higher classification accuracy.