1. INTROUDCTION

In a world that is ever-changing and ever-evolving, the need for effective identification of images has become more and more of a demand. From facial recognition to the scanning of QR codes at airports, strong methodologies for such endeavours have been developed. Image classification is the process of categorising and labelling groups of pixels or vectors within image-based data [10]. Here, we use traditional supervised machine learning models on the SVHN dataset to create algorithms to classify image-based information. This dataset contains images of house numbers and their numerical labels. In this report, we highlight the inefficiency of traditional machine learning models for this task, highlighting the need for more advanced techniques.

2. METHODOLODY

This sections details the pre-processing steps taken to transform the data which was then used with traditional machine learning models.

2.1. DATA PRE-PROCESING (REFER TO APPENDIX 6.1)

This section outlines the steps taken to process the data prior to training and predicting the models.

The dataset used for this study was the SVHN dataset, which was loaded from .mat files using the scipy.io module. The dataset was split into training and testing sets, and the feature data was transposed from the shape (32, 32, 3, N) to (N, 32, 32, 3). Both datasets were reduced to 5000 samples each for computational efficiency. The value of 10 within the labels (represents 0 in the SVHN dataset) was remapped to 0.

Initially, visual representations of class distribution using Matplotlib were generated to understand the degree of class imbalance. The classes were balanced using hybrid sampling (a combination of oversampling and under-sampling). Oversampling and under-sampling were achieved using the imbalanced-learn library in Python. Oversampling was implemented using the RandomOverSampler module to match the count of the majority class. Data under-sampling was done using the RandomUnderSampler module to make the class representation equal whilst reducing dataset size.

For contrast enhancement and simplification of feature representation, each image was processed using the OpenCV library in Python. Each image underwent a conversion from RGB to grayscale and then were blurred with a median filter (kernel size = 3) to reduce noise. Adaptive Gaussian thresholding was applied to binarize the image to enhance digit edges under different lighting.

Finally, the images were flattened to and reshaped from 32, 32 into 1D vectors of a size of 1024. This was done to achieve compatibility with traditional machine learning models used in this study.

2.2. BASIC MODEL TRAINING/PREDICTION (REFER TO APPENDIX 6.2)

Using Scikit-learn's pipeline construction function, a machine learning pathway was developed to streamline the model's training and an attempt to mitigate data leakage. Within the pipeline, a feature standardisation was applied to normalise the input features, which is essential for converting the data points to principal components. PCA was done to make the data less noisy whilst retaining the informative structure of the data. The data was then used to train each model (Logistic Regression, Decision Tree, Random Forest, and Gaussian Naïve Bayes). The test features were used to make predictions, and a report of various metrics was run to measure the performance of models

without any hyperparameter tuning. The coding for this can be observed in appendix 6.2. For the full analysis please refer to the attached code.

2.3. HYPERPARAMETER TUNING (REFER TO MAIN CODE FOR FULL ANALYSIS)

Different combinations of hyperparameters were explored using the GridSearchCV module in conjunction with the pipeline established from the previous step. The best hyperparameters were selected based only on the training data. The best hyperparameters were then used to make predictions, and the performance metrics were recorded and compared.

3. DISCUSSION/ANALYSIS

This study focused on four machine learning models: Logistic Regression, Decision Tree Classifier, Gaussian Naïve Bayes, and Random Forest Classifier. Each model was trained and tested on the data without any hyperparameter tuning. This section will provide a comparison of each model, illustrating their basic performance on the dataset. We will then explore the various hyperparameters for each model that were adjusted to achieve optimal results based solely on the training data. Finally, we will compare the performance of each model using their best hyperparameters.

3.1. PERFORMANCE OF BASIC MODELS (NO HYPERPARAMETERS

	Model	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-Score	Support
0	Logistic Regression	0.162025	0.159201	0.162025	0.158934	3160
1	Decision Tree	0.162025	0.159201	0.162025	0.158934	3160
2	Gaussian Naive Bayes	0.211709	0.233065	0.211709	0.208414	3160
3	Random Forest	0.233228	0.243171	0.233228	0.225462	3160

Figure 3.1.1. – The above table depicts the performance of each model. The evaluation metrics used were accuracy, precision, recall, the F1-Score and support.

The results from Figure 3.1.1. depict a general trend of poor performance in classifying image-based data Among these traditional machine-learning models. These models, which use flattened feature vectors as their baseline input, cannot decipher spatial dependencies between pixels, making them unreliable for working with visual data [1]. The feature selection process from image data can be very time-consuming and error-prone. Additionally, a high level of knowledge is required for such endeavours and necessitates a specialised individual to perform such analysis [2]. An alternative to using these models is deep-learning models. They can extract complex features automatically, leading to high accuracy in their classification performance [1].

The Gaussian Naïve Bayes model showed a slight improvement with an accuracy of 21.2%. However, the assumption it carries, that all features are independent of one another, is violated as pixels in an image are highly correlated. It does not represent the image structure accurately due to its inherent assumption. The Gaussian Naïve Bayes method assumes a normal data distribution and may have benefitted from greater separation between class priors. It does not, however, contain a representation of local image structure. In addition to this, Gaussian Bayes failed to classify most images effectively with a low F-1 score of 0.208, though it performed better than Logistic Regression and Decision trees.

Random Forests showed differences in performance in terms of accuracy, precision, recall, and F-1 Score. This can be attributed to the ensemble nature of the model. This allows the model to identify non-linear patterns within the data. Much similar to the other machine learning models used, it does not account for the 2-dimensional spatial structure of the image data [3]. The F-1 score, 0.225, though better than the other models, still indicates much misclassification.

The low macro averaged metrics for all models are a strong indicator they do not often classify the images within the dataset in a correct manner. They also suggest an uneven classification of the data against their digit labels and that they miss certain digits entirely. This further highlights the unsuitability of these models for such image classification tasks. These models cannot effectively identify spatial relationships and visual patterns that are necessary for digit recognition in the images [1].

3.2. HYPERPARAMETER TUNING – LOGISTIC REGRESSION

```
param_grid = {
    'C':[0.1,1,10],
    'penalty':['elasticnet'],
    'solver':['saga'],
    'l1_ratio':[0.5,0.75,1.0],
    'max_iter':[500]
}
```

Figure 3.2.1

Figure 3.2.1. depicts the hyperparameters used in tuning the logistic regression model. The penalty applied was Elastic NET, which combines the penalties of the Ridge and Loss regression methods. It entails both L1 and L2 penalties, influenced by two parameters. Both regression penalties aim to reduce the coefficients to zero; together, they give a comprehensive method of achieving this [4]. The Saga solver is an iterative solver that models the weight for each sample, remembers the past gradients, and utilises that to reduce the noise in each update. It functionally works well with the Elastic NET penalty [5]. In the sci-kit library, the C value is the inverse of the strength of regularisation. The l1 ratio depicts the occurrence of either Ridge or Lasso within the Elastic NET penalty application. If l1 is 1.0, it is purely Lasso, and if it is 0.0, it is Ridge. Each has its benefits and works collectively to shrink the coefficients within the objective function of the machine learning model to zero.

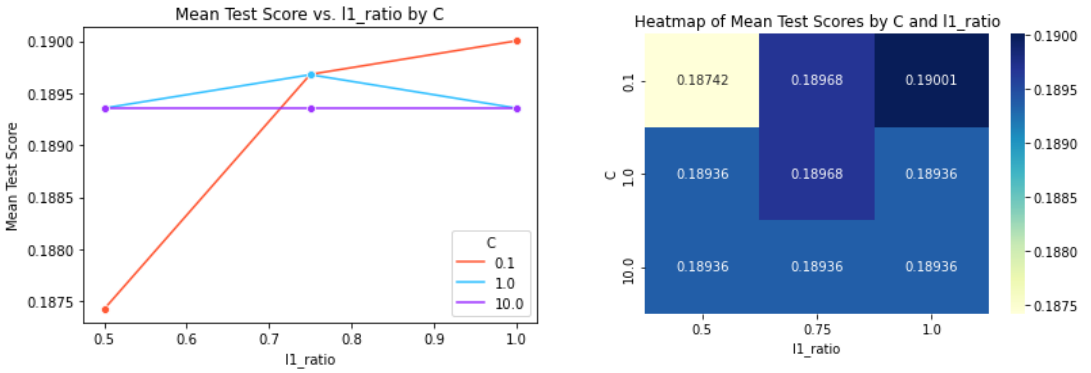

Figure 3.2.2. – Depicts the performance of the model with different C and l1 ratios with the application of the Elastic NET penalty and SAGA solver. The (left) figure portrays the accuracy measurement of the model at different C values. The (right) figure shows a heatmap that depicting the correlation between l1 ratio values and C values with the subject being the accuracy score.

The heatmap is shown in Figure 3.2.2. (right) depicts that the model performs best when the C value is set to 0.1 and the l1 ratio to 1.0. This is reflective of pure L1 regularisation (Lasso) with robust regularisation strength. Both figures show that a low C value combined with a high L1 ratio yields the best generalisation for the Logistic Regression model. Higher performance with pure Lasso regularisation suggests that the model gains more from sparsity, which, in turn, pushes some coefficients to zero [6]. Both figures in 3.2.2. indicate that the performance of the model flatlines at a high C value of 10, thus showing a lack of control over model complexity with weak regularisation. This is further substantiated by the observation of virtually no change to the performance of the model when varying the L1 ratio at a high C value [6]. At C=1, there is a moderate bump to the performance, but it decreases again when the L1 ratio increases, indicating a minimal sensitivity to regularisation at that C value. From these results, it was concluded that the best parameters to use were C = 0.1 and L1 = 1.0.

3.3. HYPERPARAMETER TUNING – DECISION TREE

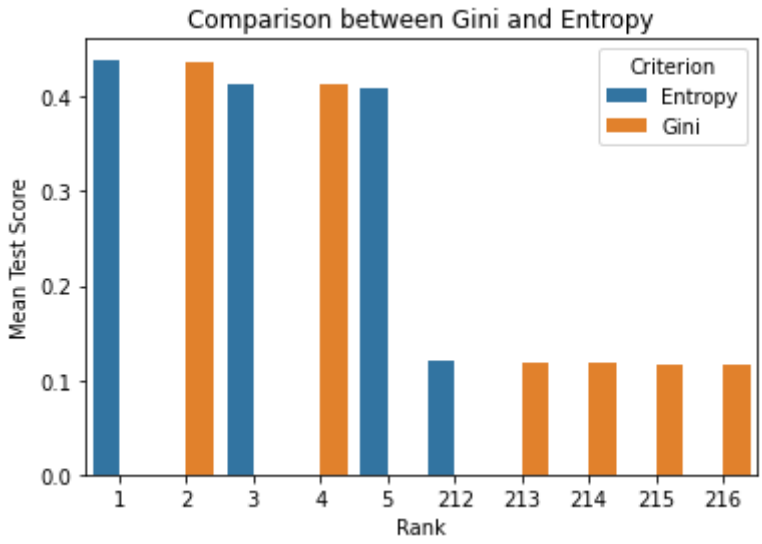

Figure 3.3.1 – This figure depicts the comparison between the two types of criterion for the Decision Tree model.

The two main hyperparameters considered in this configuration were based on the criterion of the model. Namely, the two in consideration were Gini and Entropy. The Gini criterion measures the probability of misclassifying a randomly selected element in the dataset. Entropy looks at the measure of disorder in the dataset. In Figure 3.3.1 We observe that the bottom-performing models had a limit to the depth of the decision tree (limit = 3). The models to the left in the figure depict strong performance irrespective of the model's criterion. This can be attributed to the completion of the tree. The marginal difference in performance due to the model could potentially be due to the dataset's more balanced nature due to preprocessing (under-sampling and oversampling). The best performance can be seen with deeper trees for both criteria, thus showcasing depth as the more prominent factor in influencing model performance. For the purposes of this study, the criterion of Entropy was selected for optimal model execution against predicted data.

3.4. HYPERPARAMETER TUNING – GAUSSIAN NAÏVE BAYES

	Var Smoothing	Mean Test Score	Std Test Score	Rank
0	1e-09	0.236448	0.012541	1
1	1e-08	0.236448	0.012541	1
2	1e-07	0.236448	0.012541	1
3	1e-06	0.236448	0.012541	1
4	1e-05	0.236126	0.012995	5

Figure 3.4.1. – The above tables shows the variance smoothing used as a hyperparameter to tune for Gaussian Naïve Bayes.

In theory, the Var smoothing hyperparameter is a type of regularisation that prevents the denominator of the Naïve Bayes model from approaching zero by introducing a small variance. This prevents numerical instability by making the output asymptotic towards zero [7]. The table in figure 3.4.1. suggests that the data is impervious to the small changes introduced by the varying regularisation technique. This could potentially be attributed to the large variance of the data, thus making it immune to smoothing. However, there is a slight fall in the performance of the model at a higher smoothing rate. This may be due to over-smoothing that starts occurring at higher magnitudes, leading to dampening of important variance signals [8].

3.5. HYPERPARAMETER TUNING – RANDOM FORESTS

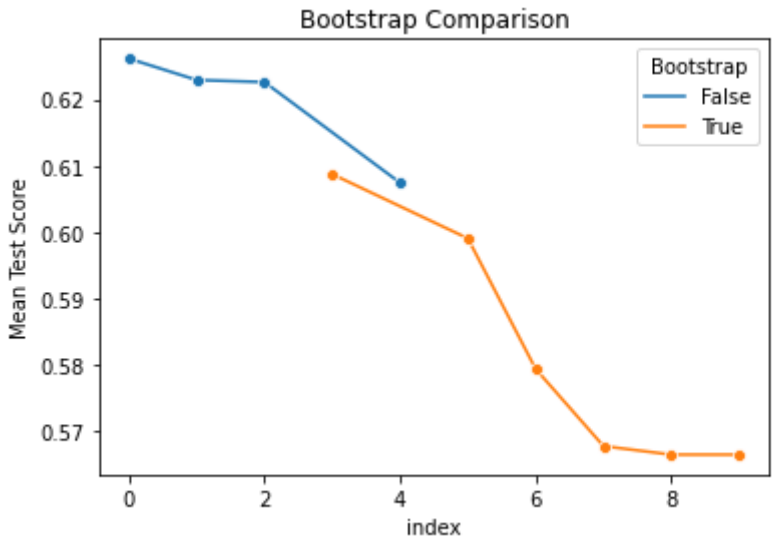

Figure 3.5.1. – The above figure shows a comparison of different instances of bootstrapping and the maximum length of the trees for each bootstrapping instance.

Scenario	Depth	Data Exposure	Performance Impact
bootstrap=False, depth=None	Deep	Full data	**Best** (0.626)
bootstrap=False, depth=10	Shallow	Full data	⚠ Likely underfitting
bootstrap=True, depth=None	Deep	Sampled data	Depends on stability
bootstrap=True, depth=10	Shallow	Sampled data	**Worst** (0.566)

Figure 3.5.2. – depicts the various scenarios with depth restrictions and bootstrapping.

Random Forests generally performs better than the other machine learning models due its ensemble nature. It selects random subsets of data from which it trains various decision trees and then aggregates them into a 'Random Forest'. This is otherwise known as bootstrap aggregation. Figure 3.5.1 depicts various models trained with bootstrapping and without. It was observed that those without bootstrapping generally performed better than those that had bootstrapping applied to them. No bootstrapping brings about better performance likely as a result of the tree having access to the entire training set rather than just a subset. With bootstrapping, it is likely that the variance added is only adding to the already-existing noise from the data thus rendering the performance lower.

Figure 3.5.2 provides a best and worst case scenario combining both bootstrapping and the introduction of a depth restriction. The analysis highlights that both exposure to the data and the complexity of work together to render the model's performance. The model works best without any depth restrictions and without bootstrapping as it has full access to the data. Conversely, with bootstrapping and a depth restrictions, its performance is poorer as a result of smaller datasets training the model and restrictions on capturing the data complexity.

3.6. FINAL MODELS COMPARISON

	Model	Accuracy (Before → After)	Macro Precision	Macro Recall	Macro F1-Score	Support
0	Logistic Regression	0.162 → 0.167	0.159 → 0.165	0.162 → 0.167	0.159 → 0.164	3160
1	Decision Tree	0.162 → 0.205	0.159 → 0.203	0.162 → 0.205	0.159 → 0.203	3160
2	Gaussian NB	0.212 → 0.212	0.233 → 0.233	0.212 → 0.212	0.208 → 0.208	3160
3	Random Forest	0.233 → 0.473	0.243 → 0.490	0.233 → 0.473	0.225 → 0.463	3160

Figure 3.6.1. – The above table shows the various metrics that measure the performance of the models. They show the performance before and after hyperparameter tuning.

The most significant improvement in model performance can be observed with Random Forests. All metrics have approximately doubled for Random Forests after hyperparameter tuning. This can be attributed to the benefits of tuning crucial hyperparameters such as the maximum depth, preventing bootstrapping, etc. The model could capture a fuller dataset complexity by manipulating such parameters, resulting in more effective generalisation [9]. As for Decision Trees, a more basic model of Random Forests, we observe a notable gain in the accuracy and F1 score increase from 0.162 to 0.205 and from 0.159 to 0.203, respectively. Though not as effective as Random Forests, the

improvement in performance metrics can similarly be attributed to manipulating parameters like maximum depth. In addition, by playing around with different criteria, we also observe an effect on the model's performance. The effect of the criterion is marginal, but the slight performance improvement when employing Entropy as the criterion could be due to the uniformity of the data. In essence, depth control and better splitting strategies have helped reduce underfitting [10]. The Gaussian Naïve Bayes model did not show any improvement or decline in performance which was expected. The model relies quite solely on the assumption that all features are independent of each other which does not work well for this given dataset. The Logistic Regression model showed a marginal improvement after hyperparameter tuning. Most likely, the data is too non-linear for the model for it to work effectively even after introducing techniques like regularisation.

4. CONCLUSION

This project looked into the effectiveness of various traditional machine learning models on the classification of the SVHN image dataset. After pre-processing the data, it was used to train the various different models and then make predictions. Hyperparameterisation was conducted to find the best combination of hyperparameters for each model. Those parameters were then applied to the respective models to make predictions. Of all the models, the most significant improvements were observed using the Random Forest model. Despite the higher performance, the overall results showed a weak ability of each model to correctly classify the images in the dataset. This highlights the limitations of traditional machine learning models, which include the models' inability to identify spatial relationships in the data. An alternative approach that could potentially be explored is Convolutional Neural Networks (CNNs) [12].

5. REFERENCES

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
2. Ke He, D.-S. K. (2018). Malware Detection with Malware Images using Deep Learning Techniques. Department of Computer Science and Software Engineering, University of Canterbury.
3. Ye, S., et al. (2018). "Two-Dimensional-Reduction Random Forest." 2018 Eighth International Conference on Information Science and Technology (ICIST).
4. Kakde, A. (2023). "Elastic Net Regression : The Best of L1 and L2 Norm Penalties." from https://adityakakde.medium.com/elastic-net-regression-the-best-of-l1-and-l2-norm-penalties-7a340e2387d6.
5. Defazio, A., et al. (2014). "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives." Advances in neural information processing systems.
6. Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. International Conference on Machine Learning, Stanford University. **21**.
7. . "GaussianNB." from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.
8. Raschka, S. and P. I. Mirjalili (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2.
9. Probst, P., Wright, M. N., & Boulesteix, L. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *9*(3), e1301. https://doi.org/10.1002/widm.1301
10. Biau, G. e. (2012). "Analysis of a Random Forests Model." Journal of Machine Learning Research **13**.

11. Shinozuka, M. and B. Mansouri (2009). Structural Health Monitoring of Civil Infrastructure Systems.
12. Lai, Y. (2019). "A Comparison of Traditional Machine Learning and Deep Learning in Image Recognition." Journal of Physics: Conference Series.

6. APPENDICES

6.1. APPENDIX 1 – PREPROCESSING CODE

```
import numpy as np

import pandas as pd

import scipy.io as sio

!pip install opencv-python

import cv2 as c


train_dat = sio.loadmat("train_32x32.mat")

test_dat = sio.loadmat("test_32x32.mat")


train_feat, train_lab = train_dat['X'], train_dat['y']

test_feat, test_lab = test_dat['X'], test_dat['y']


train_feat = np.transpose(train_feat, (3,0,1,2))

train_feat = train_feat[:5000,:,:,:]

test_feat = np.transpose(test_feat, (3,0,1,2))

test_feat = test_feat[:5000,:,:,:]


train_lab[train_lab == 10] = 0

test_lab[test_lab == 10] = 0

train_lab = train_lab[:5000]

test_lab = test_lab[:5000]


import matplotlib.pyplot as plt



for i in (range(5,10)):
```

```
    plt.imshow(train_feat[i])
    plt.title(f"Label:{train_lab[i]}")
    plt.show()



plt.hist(train_lab, width= 0.8)
plt.xlabel('labels')
plt.ylabel('counts')
plt.title('Frequency of train labels')
plt.xticks(np.unique(train_lab))

plt.show()

plt.hist(test_lab, width = 0.8)
plt.xlabel('labels')
plt.ylabel('counts')
plt.title('Frequency of test labels')
plt.xticks(np.unique(test_lab))

plt.show()



!pip install imbalanced-learn

from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

orig_shape = train_feat.shape
train_feat_flat = train_feat.reshape(orig_shape[0], -1) #Retaining only the number of images and combining the other elements
```

```
#hybrid sampling

train_lab = np.array(train_lab).flatten()

class_counts = Counter(train_lab)


#oversampling

max_class_count = max(class_counts.values())

oversampling_strategy = {label: max_class_count for label in class_counts}


min_class_count = min(class_counts.values())

undersampling_strategy = {label:min_class_count for label in class_counts}


oversample = RandomOverSampler(sampling_strategy=oversampling_strategy, random_state=42)

undersample = RandomUnderSampler(sampling_strategy=undersampling_strategy,
random_state=42)


train_feat_os, train_lab_os = oversample.fit_resample(train_feat_flat, train_lab)

train_feat_resample, train_lab_resample = undersample.fit_resample (train_feat_os, train_lab_os)


#reshape back to original format

train_feat_resample = train_feat_resample.reshape(-1, *orig_shape[1:])


plt.hist(train_lab_resample, width= 0.8)

plt.xlabel('labels')

plt.ylabel('counts')

plt.title('Frequency of train labels')

plt.xticks(np.unique(train_lab))

plt.show()

orig_shape = test_feat.shape

test_feat_flat = test_feat.reshape(orig_shape[0], -1) #flatten images


test_lab = np.array(test_lab).flatten()

class_counts = Counter(test_lab)
```

```
#oversampling

max_class_count = max(class_counts.values())

oversampling_strategy = {label: max_class_count for label in class_counts}


min_class_count = min(class_counts.values())

undersampling_strategy = {label:min_class_count for label in class_counts}


oversample = RandomOverSampler(sampling_strategy=oversampling_strategy, random_state=42)

undersample = RandomUnderSampler(sampling_strategy=undersampling_strategy,
random_state=42)


test_feat_os, test_lab_os = oversample.fit_resample(test_feat_flat, test_lab)

test_feat_resample, test_lab_resample = undersample.fit_resample (test_feat_os, test_lab_os)


#reshape back to original format

test_feat_resample = test_feat_resample.reshape(-1, *orig_shape[1:])


plt.hist(test_lab_resample, width= 0.8)

plt.xlabel('labels')

plt.ylabel('counts')

plt.title('Frequency of train labels')

plt.xticks(np.unique(train_lab))


plt.show()

#application of grey scaling


train_feat_g = []


for image in train_feat_resample:

  im = c.cvtColor(image, c.COLOR_BGR2GRAY)

  im = c.medianBlur(im,3)
```

```
    im = c.adaptiveThreshold(im,255, c.ADAPTIVE_THRESH_GAUSSIAN_C, c.THRESH_BINARY, 11, 2)
    train_feat_g.append(im)


for i in (range(5,10)):
    plt.imshow(train_feat_g[i], cmap = 'gray')
    plt.title(f"Label:{train_lab[i]}")
    plt.show()

test_feat_g = []

for image in test_feat_resample:
    im = c.cvtColor(image, c.COLOR_BGR2GRAY)
    im = c.medianBlur(im,3)
    im = c.adaptiveThreshold(im,255, c.ADAPTIVE_THRESH_GAUSSIAN_C, c.THRESH_BINARY, 11, 2)
    test_feat_g.append(im)

for i in (range(5,10)):
    plt.imshow(test_feat_g[i], cmap = 'gray')
    plt.title(f"Label:{test_lab[i]}")
    plt.show()
#flattening and standardising data
flat_train_feat = train_feat_resample.reshape(train_feat_resample.shape[0],-1)
flat_test_feat = test_feat_resample.reshape(test_feat_resample.shape[0],-1)
```

6.2. APPENDIX 2

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
```

```
pipe = make_pipeline(StandardScaler(), PCA(n_components =
0.95),LogisticRegression(max_iter=500))

pipe.fit(flat_train_feat, train_lab_resample)


predictions = pipe.predict(flat_test_feat)


from sklearn.metrics import accuracy_score, classification_report


accuracy = accuracy_score(predictions, test_lab_resample)

report = classification_report(test_lab_resample,predictions,output_dict = True)

report = pd.DataFrame(report)

report = report.rename(columns = {'macro avg':'Macro Average for Logistic Regression'})


print('accuracy:',accuracy)

print('report:',report)
```