ISSN: 2995-3286

Research Article

International Journal of Media and Networks

A Systematic Review of Enhancing CNN Performance in Automated Fabric Defect Detection Through Sampling Techniques for Imbalanced Datasets with the Developed CNN Model

Saima Saleem*, David Williams and Satya Prakash

Department of artificial intelligence, Ireland

***Corresponding Author**
Saima Saleem, Department of Artificial Intelligence, Ireland.

Submitted: 2024, Oct 06; **Accepted:** 2024, Oct 28; **Published:** 2024, Nov 08

Citation: Saleem, S., Prakash, S., Williams, D. (2024). A Systematic Review of Enhancing CNN Performance in Automated Fabric Defect Detection Through Sampling Techniques for Imbalanced Datasets with the Developed CNN Model. *Int J Med Net, 2*(10), 01-43.

Abstract

In the textile industry, manual fabric inspection poses significant challenges. Incomplete and faulty inspections can compromise both product cost and quality. With the advancements in deep learning, various machine learning algorithms have emerged as successful tools for image classification and analysis tasks. Nevertheless, there are several persistent issues, including the complexity and time-consuming nature of training methods, the requirement for large datasets, and difficulties in achieving generalization. What's needed is an accurate and swift automatic machine learning algorithm suitable for real-time detection in industrial setups. To tackle these challenges, this research successfully developed a straightforward Convolutional Neural Network (CNN) machine learning algorithm.

The algorithm's performance was evaluated on two different image sizes: 150 x 700, and 245 x 345. It became evident that image size significantly influences the model's performance. Additionally, the dataset's inherent imbalance had an adverse impact on the model's performance due to inadequate training and overfitting. To address the issue of imbalanced dataset and enhance the model's performance, various sampling techniques were experimented with. Among these, the CNN model exhibited its most outstanding performance when paired with a smaller image size of 245x345 and when utilizing the SMOTEENN sampling technique. The results demonstrated remarkable accuracy, precision, recall, and F1 score, with values of 98.00%, 98.00%, 98.00%, and 98.00%, respectively. Moreover, the time required for modelling and prediction was impressively low, at 1.57 seconds and 0.09 seconds, respectively. The research also proposed a method to deploy the algorithm and automate the entire quality inspection process within the textile industry.

Keywords: Fabric-Defect Detection, Fabric Inspection, Deep Learning, Convolutional Neural Networks, Imbalanced Classification

Abbreviations

CNN: Convolutional Neural Network
TP: True Positive
FP: False Positive
TN: True Negative
FN: False Negative
P: Precision
r: Recall
SSD: Single Shot Detector
VLSTM: Visual Long-Short-Term Memory
VP: Visual Perception
SCAE: Stacked Convolutional Autoencoders
R-CNN: Region Based Convolutional Neural Network
RPN: Region Proposal Network
ROI: Region of Interest
PRAN-Net: Priori Anchor Convolutional Neural Network
MSCNN with K-clustering: Multi-scale CNN with K-clustering
CBAM: Convolutional Block Attention Module
ROC, Receiver Operator Characteristic
ReLU: Rectified Linear Unit
ADASYN: Adaptive Synthetic Sampling SMOTE
PCA: Principal Component Analysis

1. Introduction

A textile fabric manufacturing process is depicted in Fig. 1 below:

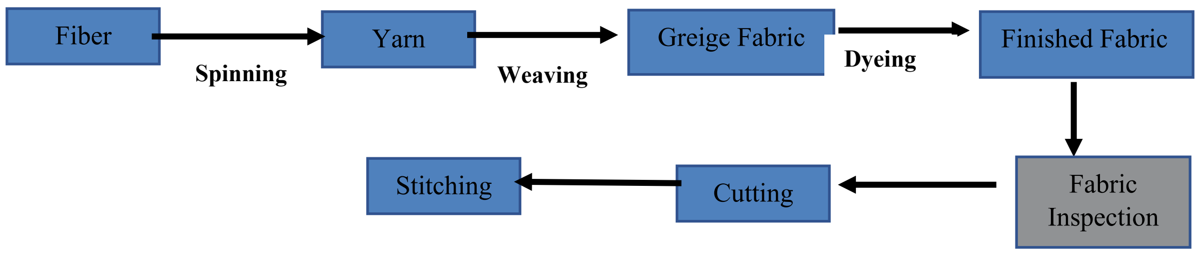


Figure 1: Fabric Manufacturing Process

Fabric inspection is conducted at each stage of the textile manufacturing process, including weaving, dyeing, printing, and finishing. This ensures product quality and minimizes potential losses [1]. The structure of fabric is inherently repetitive; therefore, any defects disrupt this repetitive pattern [2]. Early defect detection is crucial to prevent wastage in the final product [1]. Jing et al. (2022, pp. 2-3) categorized the task of defect detection into four levels [3]:

- The first level involves the task of classifying defects into categories;
- The second level involves detecting and locating these defects;
- The third level identifies defective pixels within images, known as segmentation;
- The fourth level entails defect semantic segmentation, combining segmentation and classification.

Fabric defect detection algorithms can be categorized into two types:

- Traditional algorithms and
- Learning-based algorithms.

Traditional algorithms include statistical, spectral, structural, and model-based methods. These techniques process individual images and require manual feature selection. In contrast, learning-based methods automate feature selection. Learning-based algorithms can be further divided into:

- Classical machine learning algorithms;
- Deep machine learning algorithms

Deep learning methods have the advantage of automatic feature extraction; However, they often require a substantial amount of data compared to classical machine learning algorithms, which, in turn, demand extensive parameter tuning [1,2].

Research has shown that CNN predictions for face recognition, image classification and real time object detection can be as accurate as human [4,5].

2. Related Works

Liu et al. (2018) proposed a method that utilized an enhanced Single Shot Detector (SSD) with a shallower convolutional feature layer [6]. They combined the deep neural networks for single detection with YOLO's regression approach and Faster R-CNN's anchor principle. Regression simplified the algorithm's complexity and improved its efficiency, while anchors effectively captured characteristics related to scales and aspect ratios. SSD also employed a multiscale object feature extraction technique. However, this method encountered challenges in accurately detecting small defects.

Zhang et al. (2018) conducted a comparison among the YOLO 9000, YOLO-VOC, and Tiny-YOLO models, and they proposed an improved YOLO-VOC model [7]. This enhanced model employed super-parameter optimization for defect classification and localization by utilizing a single-stage CNN for image prediction increased detection speed, although the proposed method's accuracy was comparatively lower.

In a study cited by Minhas and Zelek (2020, p. 507), Tan et al. (2018) identified network-based transfer learning as the most practically useful class among instance-based, mapping-based, network-based, and transfer learning methods [8]. This approach involves constructing a target network from a source network, consisting of feature extractor and classification sub-networks. Such an approach enables defect classification even when working with limited labeled data.

Liu et al. (2019, p. 2) emphasized the accuracy of Convolutional Neural Network (CNN) models in fabric defect detection tasks [9]. According to Prakash et al. (2021), deep learning, especially Convolutional Neural Networks (CNN), was a highly accurate machine learning method for image detection and multi-classification tasks, with real-time implementation capability [10].

Zhao et al. (2019) designed a visual Long-Short-Term Memory (LSTM) integrated CNN model utilizing visual perception (VP) for classification [11]. Stacked Convolutional Autoencoders (SCAE) were employed to extract features. Experimental results on datasets containing 500, 1,000, and 10,500 images demonstrated the highest performance on the 500-image dataset, achieving an accuracy of 99.47%. However, this complex method is sensitive to intricate fabric backgrounds.

Liu et al. (2019) proposed optimising the VGG16 convolutional network with a deconvolutional network to accurately detect defects [9]. Their model, LZFNet, incorporated a deconvolutional network attached to each VGG16 layer, establishing a continuous path back to the image pixel space. Despite its low parameter count, obtaining a substantial amount of labeled training data for CNN models in real-world scenarios remains challenging.

Wei et al. (2019) introduced a Faster R-CNN model that consisted of convolution layers, a Region Proposal Network (RPN), ROI pooling, and classification. Feature extraction was performed using VGG16 [12]. The output score of the classification layer represented the background for each anchor, while the output of the regression layer indicated the coordinates of fabric defects. However, this approach suffered from extended training times. Chen et al. (2020) proposed an improved Faster R-CNN model that incorporated Gabor kernels within Faster R-CNN and was trained using a two-stage backpropagation method [13].

Minhas and Zelek (2020) introduced a specialised VGG16 model optimised through deconvolutional networks using transfer learning [8]. Their research highlighted two techniques: fixed feature extraction and full network fine-tuning. Full network fine-tuning, which involves updating the entire network's parameters during training, outperformed fixed feature extraction. Notably, the model's detection speed was not considered.

Peng et al. (2020) introduced the Priori Anchor Convolutional Neural Network (PRAN-Net), which achieved high accuracy in detecting tiny defects. They utilized multi-scale feature maps from a Feature Pyramid Network (FPN) to generate sparse priori anchors based on fabric defect ground truth boxes. Feature extraction was performed using ResNet-101-FPN [14].

In contrast, Zhao et al. (2020) employed a K-means clustering method known as Multi-scale CNN with K-clustering (MSCNN with K-clustering) to define defect bounding boxes of known sizes. However, this approach is data-driven and requires a substantial number of labeled fabric images for model training, and it covers only five defect types [11].

Almeida, Moutinho, and Matos-Carvalho (2021) proposed an operator-assisted CNN model based on the premise that undetected defects (False Negatives - FNs) typically incur higher costs than non-defective items classified as defective (False Positives - FPs) [15]. This model achieved an average accuracy of 75%. However, with operator assistance, it reached an accuracy of 95%.

He et al. (2021) introduced another Faster R-CNN algorithm that incorporated the Convolutional Block Attention Module (CBAM) in conjunction with ResNet50. ResNet50 was utilized for feature extraction, classification, and regression [16]. The extracted feature map was then input into RoI pooling and RPN. To eliminate redundant boxes, Soft-NMS and RPN were employed. This model can be integrated with CNN and trained end-to-end alongside the basic CNN. Although this method is time-consuming, the Channel Attention Module compresses the feature map in the spatial dimension to obtain a one-dimensional vector before performing the operation.

3. Convolutional Neural Network (CNN) Model

The CNN algorithm showed huge success in identifying objects within images and was therefore considered in the present study [17]. According to Yamashita et al. (2018, pp. 612-617), a typical CNN architecture comprised convolution layers, pooling layers, and fully connected layers [18]. It employs a kernel, a small grid of parameters, to process input tensors. The kernel is applied to the input tensor, and at each location, it calculates the product of each element in the kernel and the corresponding element in the input tensor, followed by summation. This process yields a feature map. Two critical hyperparameters to consider are the size and number of kernels. Padding is another consideration, which involves reducing the dimensions of the output feature map by overlapping the center of each kernel with the outermost element of the input tensor.

Convolutional neural network CNN consists of convolution layers, pooling layers, and fully connected layers. A convolution layer is a basic layer of the CNN which is used for feature extraction [18].

A Two-dimensional (2D) grid is an array used for storing the pixels of the images [18].

Kernel is a small grid of parameters that is used for the feature extraction [18].

As the output of one layer is fed to the next layer, extracted features can hierarchically and progressively become more complex. Training process minimises the difference between the output and ground truth by means of backpropagation and gradient descent. Training process identifies best kernels. Kernels learn automatically during the training process. Hyperparameters that need to be adjusted are the size of the kernels, number of kernels, padding, and stride [18].

Convolution is a linear operation used for feature extraction. In this process the kernel is applied on the input number array called tensors. A product between each element of the kernel and the input tensor is calculated at each location of the tensor and then summed. The output is called a feature map. Two key hyperparameters are the size and number of kernels. These are usually 3×3, 5×5 or 7×7 [18].

Padding process reduces height and width of the output feature map by overlapping the centre of each kernel with the outermost element of the input tensor [18].

Stride is the distance between two successive kernels. A stride that is larger than 1 is used to downsample the feature maps [18].

3.1 Nonlinear Activation Function

The outputs of a linear operation such as convolution are passed through a nonlinear activation function. These are sigmoid or hyperbolic tangent (tanh) functions, rectified linear unit (RELU) [18].

3.2 Pooling Layer

A pooling layer reduces the dimensionality of the feature maps and decreases the number of subsequent learnable parameters [18].

Weight Sharing is used for the following purpose:

● To allow the local feature patterns extracted by kernels to travel across all the images for the detection of patterns.

● To learn spatial hierarchies of feature patterns by downsampling in conjunction with a pooling operation, resulting in capturing an increasingly larger field of view.

● To increase model efficiency by reducing the number of parameters to learn in comparison with fully connected neural networks [18].

Max pooling extracts patches from the input feature maps, discards the other and outputs the maximum value in each patch. A max pooling with a filter of size 2×2 with a stride of 2 is commonly used. This down samples the dimension of feature maps by a factor of 2 [18].

3.3 Fully Connected Layer

The output feature maps of the final convolution or pooling layer is flattened by means of transformation into a one-dimensional (1D) array of number and then connected to one or more fully connected layers. This fully connected layer is also known as dense layers, in which every input is connected to every output by a learnable weight. The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as RELU [18].

3.4 Last Layer Activation Function

Selection of an appropriate activation function, applied in the last of fully connected layer, is crucial and must be tailored to the specific task at hand. For multiclass classification tasks, the SoftMax function is commonly employed. This function normalizes the output values from the last fully connected layer into target class probabilities, ensuring each value ranges between 0 and 1, with the sum of all values equating to 1 [18].

3.5 Data and Ground Truth Labels

Data and ground truth labels are collected to train and test a model. Available data is divided into three sets: a training, a validation, and a test set [18].

3.6 Overfitting

It occurs when a model memorises irrelevant data instead of learning the signal, and, therefore, performs less well on a subsequent new dataset. An overfitted model does not generalize to new data. If the model performs well on the training set compared to the validation set, then the model has likely been overfit to the training data. The best solution for reducing overfitting is to obtain more training data. The other solutions including dropout or batch normalisation and data augmentation as well as reducing architectural complexity. During dropout, random activations are set to zero during training, reducing the model's reliance on specific weights. Weight decay helps prevent overfitting by penalizing large weights. Batch normalization adjusts input values in each layer, reducing overfitting and improving gradient flow, allowing faster learning rates and less reliance on initial values. Data augmentation also combats overfitting by modifying training data with random changes like flipping, translation, and rotation, ensuring the model encounters varied inputs during training [18].

3.7 Loss Function or Cost Function

It measures the compatibility between output predictions of the network through forward propagation and given ground truth labels. Commonly used loss function for multiclass classification is cross entropy [18].

Yamashita et al. (2018, pp. 612-617) mentioned that ReLU function operates by multiplying input values by weights and then summing them to produce specific outputs [18]. Activation functions like sigmoid or tangent were unsuitable for multi-layered CNNs due to the vanishing gradient problem.

According to Prakash et al. (2021, p.4), the Adaptive Moment (Adam) function is used to minimize error loss [10]. Yamashita et al. (2018, pp. 619-620) mentioned that Gradient Descent is employed to update the learnable parameters, including kernels and weights, of the network to minimize loss [18].

3.8 Gradient Descent

It is used to update the learning parameters to minimize the loss [18].

3.9 Epochs in Deep Learning

An epoch represents a full cycle through the entire training dataset in neural network training. During each epoch, the network adjusts its parameters (weights and biases) based on the training data and chosen optimization method to minimize the loss function. Data is passed through the network, predictions are made, and compared with actual values using a loss function. Gradients of the loss function are computed through techniques like backpropagation, guiding parameter updates to minimize loss. Parameters are updated using an optimization algorithm like stochastic gradient descent (SGD). This process is repeated for each batch of data until all batches are processed, completing one epoch. Multiple epochs are typically run to allow the model to learn from the entire dataset. The number of epochs is a hyperparameter adjusted during training and validation to optimize model performance while considering computational resources [19].

3.10 Test and Validation Set

In machine learning, data is often divided into three main subsets: the training set, the validation set, and the test set. Training set is used to train the model. The model learns the relationships between input features and the target variable through iterative optimization on this dataset, The validation set is used to tune the model's hyperparameters and to evaluate its performance during the training process. By assessing the model on a separate validation set, one can detect issues like overfitting and adjust such as changing the model architecture, adjusting hyperparameters, or implementing regularization techniques. The key purpose of the validation set is to provide an unbiased evaluation of the model fit during the training process and to assist in model selection. After the model has been trained and tuned using the training and validation sets, it is evaluated on the test set. The test set provides a final, unbiased evaluation of the model's performance. Since the test set is not used during training or validation, it serves as a measure of how the model is expected to perform on unseen data. The performance metrics obtained from the test set give a realistic estimate of the model's generalization ability [20].

4. Imbalance Dataset and Resampling Techniques

The dataset in which classification categories are not equal is an imbalance dataset. Accuracy is not a useful performance measurement in case of imbalanced dataset. Resampling the dataset by different sampling techniques is used to tackle this issue of imbalance dataset [21]. The dataset should be balanced in which all sample sizes of positive and negative examples are roughly equivalent. This balanced dataset is required to avoid systematic error and bias [22].

4.1 Data Sampling

Imbalanced dataset can either be absolute or relative [23].
● Absolute: minority class samples are less and not well represented.
● Relative: minority samples are well represented but these are larger in number as compared to majority class samples.
Imbalance can be:
● Between-class: number of samples representing a class differs from the number of samples representing the other class.
● Within Class: when a class is composed of several different subclusters which, in turn, do not contain the same number of samples [23].

4.2 Problems of Data Imbalance

● The classifier is biased towards the majority classes.
● Class imbalance hinders the recognition of minority classes since the minority class samples may be insufficient to represent the boundaries between the two classes.
● Imbalanced datasets are more deeply impacted by noisy data [23].
Resampling is commonly used to adjust the class distribution when dealing with unbalanced datasets [23].

4.3 Imbalanced Dataset Approaches

Resampling (oversampling and undersampling) is commonly employed to adjust class distribution when dealing with unbalanced datasets [23]. Jing et al. (2020, p. 3) recommended two categories of methods to address class imbalance: hard and soft sampling techniques [2]. Hard sampling methods involve down-sampling positive or negative samples, whereas soft sampling methods use a weighted loss function that updates parameters using the entire dataset [2].

In a study on fabric defect detection mechanisms, Rasheed et al. (2020, p. 2) discussed the challenges of acquiring datasets with fabric defects and emphasized how the dataset's imbalanced nature can cause traditional supervised machine learning algorithms to fail. As a result of this imbalance, the mean average precision and ROC curve were considered superior evaluation methods [24].

Lin et al. (2023, pp. 2-10) recommended a generalized focal loss function to tackle issues caused by imbalanced datasets. This function aims to enhance the learning of positive samples while reducing the impact of less informative negative samples. In contrast, Jing et al. (2020) proposed using the median frequency loss function [2].

Cheng et al. (2022, pp. 3101-3122) introduced an intriguing deep learning algorithm called Separation Convolution UNet (SCUNet), representing an enhanced iteration of UNet initially designed for medical image segmentation [25]. UNet featured a distinctive structure for stitching low-level and high-level semantic features, employing four max-pooling operations. In each pooling operation, image pixels were grouped into 2x2 pixels, retaining the maximum pixel values within each group while discarding the others. This process resulted in information loss, particularly affecting the representation of small defects. In contrast, SCUNet opted for convolutional downsampling to reduce the feature map size. To address overfitting and minimize parameter count, all convolutional layers were substituted with depth-separable convolution. The choice of the Intersection over Union (IoU) Loss function, as opposed to cross-entropy loss, was made to accommodate irregular boundaries in defected images. The resulting model demonstrated an accuracy and recall of 98.01% and 98.07%, respectively. SCUNet's improvements over UNet were noteworthy. Instead of cropping the feature map, SCUNet stitched the entire feature map. All down-sampling layers were replaced with convolutional layers, and rather than discarding information, SCUNet compressed and fused feature image pixels. Down-sampling was achieved through a max-pooling layer with a stride of 2. The use of depth-separable convolution divided the process into channel-by-channel and point-by-point convolutions, extracting features from different locations within the same channel feature map and combining them to extract features of different channels at the same spatial location.

4.4 Resampling Techniques

The most common methods are:
● Oversampling means that we increase the number of samples in the minor classes so that the number of samples in different classes become equal or close to it thus get more balanced [26].
● Undersampling methods resample the data to reduce some samples from the majority class but this results in removal of useful data samples [23]. To cope with this issue, different techniques are used as:

4.5 Imbalanced-learn Tool

Imbalanced-learn is a tool that offers various methods to address the challenges posed by imbalanced datasets. It provides techniques such as oversampling, undersampling, a combination of oversampling and undersampling, and ensemble sampling for data preprocessing. Within these methods, there is a category labeled 'Miscellaneous,' which includes the Function Sampler technique.

4.6 Instance Selection Algorithms

In this unimportant samples are eliminated. This algorithm selects a subset of the samples that preserves the underlying distribution, so that the remaining data is still representative of the characteristics of the overall data [23].

4.7 Combination Method
Ensemble method combines several weak classifiers to get a better and more comprehensive ensemble classifier [27].
Oversampling Methods

5. Synthetic Minority Oversampling Technique SMOTE
It is a technique to over sample the minority class by creating more examples that are slightly different from the original data points [21]. SMOTE chooses a minority class instance at random and finds its k-nearest neighbors. One of these k-nearest neighbors is then selected at random [28].

5.1 Random Over-Sampling
This method repeats some samples and balances the number of samples. Samples are selected at random from minority classes with replacement. In case of multiple classes, each class is sampled independently (imbalanced-learn, n.d., i).

5.2 Borderline SMOTE
This finds borderline samples to generate new synthetic samples. Model learns the borderline of each class in the training process. Borderline of the minority class is determined and then synthetic examples are generated to add to the original training set [29].

5.3 Synthetic Minority Over-Sampling Technique for Nominal and Continuous (SMOTENC)
In this method, median of standard deviations of the minority class (continuous) is calculated. Euclidean distance is calculated between minority of one class for which k-nearest neighbors are being identified and the other minority class samples using the continuous feature space [21].

5.4 Synthetic Minority Over-sampling Technique for Nominal (SMOTEN)
It is used to resample the categorical data features. The nearest neighbors are computed using the modified version of Value Difference Metric. The Value Difference Metric (VDM) looks at the overlap of feature values over all feature vectors. A matrix defining the distance between corresponding feature values for all feature vectors is created [21].

5.5 SVM SMOTE
It is a kind of SMOTE that uses a support vector machine algorithm to select samples. In SVM-SMOTE, borderline is determined by the support vectors after training SVMs classifier on the original training set (Imbalanced learn, n.d., a).

5.6 Drawback of SMOTE:
The oversampling of SMOTE ignores within-class imbalance. Algorithm does not enforce the decision boundary. Sample instances far from the border are oversampled with the same probability as those close to the boundary [23].

SMOTE produces new samples with certain blindness and may make class overlapping more serious [27].

5.7 Adaptive Synthetic (ADASYN) Algorithm
It is like SMOTE but it generates a different number of samples depending on an estimate of the local distribution of the class to be oversampled. ADASYN adds a random value and the samples are somewhat scattered [23].

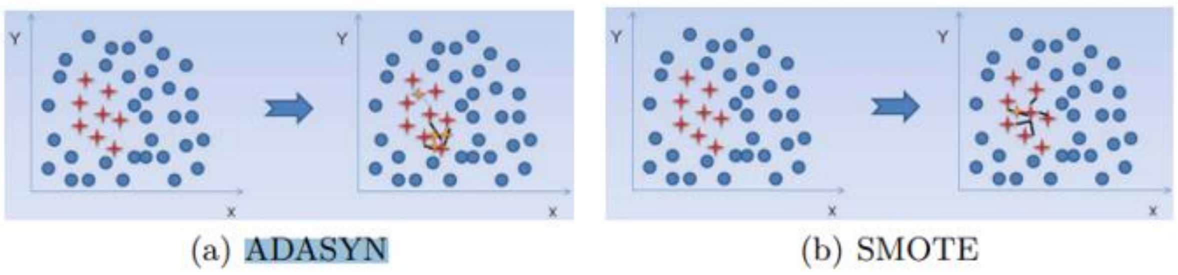


Figure 2: Difference between ADASYN and SMOTE Algorithms (Letteri et al., 2020, p. 7) [23]

It generates minority data samples according to their distributions. Thus, more minority class samples are generated that learn hardly as compared to those minority samples that are easier to learn. The ADASYN method can not only reduce the learning bias introduced by the original imbalance data distribution but can also adaptively shift the decision boundary to focus on those difficult to learn sample [30].

5.8 K-Means SMOTE Oversampling
K-Means SMOTE works in three steps:
1. Cluster the entire input space using k-means.
2. Distribute samples to:
● Select clusters which have a high number of minority class samples.
● Assign more synthetic samples to clusters where minority class samples are sparsely distributed.

The method implements SMOTE and random oversampling as limit cases (kmeans-smote.readthedocs.io, n.d.).

6. Undersampling
6.1 Cluster Centroids
This technique decreases the influence of the majority class by replacing several of its samples with centroids generated by a KMeans algorithm. It selects a set number (N) of majority samples to form N clusters using KMeans. Then, it substitutes these samples with the coordinates of the cluster centroids (imbalanced learn, n.d., b).

6.2 Condensed Nearest Neighbour

Condensed Nearest Neighbor reduces the dataset for k-NN classification by using a subset of examples [31]. Nearest neighbor rule decides how to remove a sample or not. The algorithm is runs as given below:
1. Get all minority samples in a set.
2. Add a sample from the targeted class.
3. Classify each sample using nearest neighbor rule.
4. Add a sample if it is misclassified.
5. Repeat the procedure until there are no samples to be added.
(imbalanced learn, n.d. f)

6.3 Edited Nearest Neighbor

This method cleans dataset by removing samples close to the decision boundary (imbalanced learn, n.d., c).

Samples are classified using nearest neighbor rule and then classified using single nearest neighbor rule of the pre-classified samples [32].

6.4 Repeated Edited Nearest Neighbor

This repeats the Edited Nearest Neighbor many times (imbalanced learn, n.d., d).

7. AIIKNN

This method applies Edited Nearest Neighbor several times and will vary the number of nearest neighbours (imbalanced learn, n.d. e). AIIKNN differs from the previous Repeated Nearest Neighbor as the number of neighbors of the internal nearest neighbors algorithm is increased at each iteration (imbalanced learn, n.d. e).

7.1 Instance Hardness Threshold

Samples of the class with low probabilities are removed from the dataset. Sampling strategy is based on the result. If the result is float, then it is the ratio of majority to minority class. There is a problem of class overlap [33].

7.2 NearMiss Undersampling Technique

This method randomly eliminates samples from the larger class. When two points belonging to different classes are very close to each other in the distribution, this algorithm eliminates the samples of the larger class to balance the distribution. The steps taken by this algorithm are:
1. Calculates the distance between all the points in the larger class and compares them with the points in the smaller class.
2. Samples of the larger class that have the shortest distance with the smaller class are selected. These n classes need to be stored for elimination.
3. If there are m instances of the smaller class, then the algorithm will return m*n instances of the larger class [34].

7.3 Tomek Links

Tomek Links is a modification from Condensed Nearest Neighbors. CNN method only randomly selects the samples with its k nearest neighbors from the majority class that are removed. Tomek Links method uses the rule to selects the pair of observation (a and b) that are fulfilled these properties:
1. The observation a's nearest neighbor is b.
2. The observation b's nearest neighbor is a.
3. Observations a and b belong to a different class. That is, a and b belong to the minority and majority class vice versa [35].

7.4 Neighbourhood Cleaning Rule Undersampler

This class uses Edited Nearest Neighbor and a k-NN to remove irrelevant samples from the datasets (imbalanced learn, n.d. g). Neighborhood Cleaning Rule (NCL) is an undersampling method to overcome imbalance class distribution by reducing the data based on cleaning. Cleaning process is improvised by removing the three closest neighbors from the data which are incorrectly classified. The data cleaning process is for both majority and minority class. Basically, the principle of NCL is based on the concept of One-Sided Selection (OSS), which is one technique for reducing data based on the instances to reduce classes carefully [36].

7.5 Hybrid Sampling: Oversampling Combined with Undersampling

This method can generate noisy samples which is solved by cleaning the space resulting from over-sampling. After SMOTE over-sampling, Tomek's link and edited nearest-neighbours are the two cleaning methods are applied to the pipeline.The two ready-to use classes imbalanced-learn implements for combining over- and undersampling methods are: (i) SMOTETomek and (ii) SMOTEENN (imbalanced learn, n.d. h)

7.6 Stratified k-Fold Cross Validation

Stratified k-fold cross-validation is a technique used in machine learning for model evaluation and hyperparameter tuning. It is particularly useful when dealing with imbalanced datasets, where the distribution of classes is uneven. The method involves dividing the dataset into k-folds while ensuring that each fold maintains the same class distribution as the original dataset. This helps to mitigate the risk of bias in model evaluation by ensuring that each class is adequately represented in both the training and testing subset
- The dataset is first divided into k equal-sized folds.
- For each fold, the class distribution is preserved, meaning that each fold contains approximately the same proportion of each class as the original dataset.
- The model is trained on k-1 folds and evaluated on the remaining fold
- This process is repeated k times, with each fold serving as the validation set exactly once
- The final performance metric is computed by averaging the performance across all k folds

Stratified k-fold cross-validation helps to produce more reliable estimates of model performance, especially in scenarios where the class distribution is skewed [37].

8. Evaluation Metrics

8.1 Confusion Matrix

In a confusion matrix, columns correspond to the predicted class, while rows correspond to the actual class. The components of the matrix are as follows:
● True Negatives (TN): The count of negative examples that are

correctly classified as negative.

● False Positives (FP): The count of negative examples that are incorrectly classified as positive.

● False Negatives (FN): The count of positive examples that are incorrectly classified as negative.

● True Positives (TP): The count of positive examples that are correctly classified as positive.

Evaluation metrics used in image identification are typically accuracy, precision, recall, F1- score (Al-Sarayreh et al., 2018; Larsen et al., 2014; Ropodi et al., 2015; Setyono et al., 2018; Wang et al., 2019).

Accuracy

Accuracy and F1 scores are described in Eq. 1 and 2 respectively [10].

$$Overall\ Accuracy = \frac{\sum_{i=1}^{n=5} \quad TP_i}{N} \tag{1}$$

In Eq. 1, TP_i or the true positive is the number of instances predicted correctly for instance i and N is the total number of predictions.

F1 Score

$$F1_i = 2 * \frac{Precision_i * Recall_i}{Precision_i + Recall_i} \tag{2}$$

where,

Precision

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \tag{3}$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{4}$$

[10].

• Precision

Precision is defined as the ratio of true positive defects correctly detected to the total number of detected defects [14]. Precision indicates how effectively a model identifies true positives, ensuring that the positive detections are accurate [21].

• Recall

Recall, often referred to as sensitivity or true positive rate, is the ratio of correctly detected true defects (true positives) to the total number of true defects [14]. Recall showcases the model's capacity to identify all positive instances, thus assessing its ability to capture all actual positives [21].

• F-Score

The F-value, also known as the F-score or F-measure, represents the harmonic average of precision and recall [8,11].

The F-value combines both recall and precision, and it reaches its highest value when both recall and precision are high [29].

For imbalanced datasets, accuracy can be misleading. Even if a classifier correctly classifies all majority examples but misclassifies all minority examples, the high accuracy results from the significant number of majority examples. This renders accuracy unreliable for predicting the minority class [29].

For balanced datasets, the error rate is utilized as a performance metric:

Error Rate = 1 - Accuracy [21]

In imbalanced datasets, the error rate isn't an appropriate performance measure. Instead, precision and recall are more meaningful metrics [21].

9. Data Visualization

9.1 Principal Component Analysis

It is a statistical method to reduce the dimensionality of a dataset. It determines the direction where the variation in the dataset is maximum. This direction is called the “principal component”. It determines the different principal components (directions) within the dataset and then uses these principal components to represent the samples. In this way samples can be plotted to check the similarities or differences between the samples. This enables to group the samples. Principal components are the linear combinations (new variables) of the original variables [38]. PCA is a technique for feature extraction. It combines all the variables and then drops less important variables. In this way new independent variables are created. It identifies the relationships between variables and then determines direction of dispersion in the dataset [39].

9.2 Visual Studio Code

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. It is widely used by developers for writing and debugging code, and it supports a wide range of programming languages and frameworks. Here are some key features and aspects of Visual Studio Code. While it is technically a code editor, VS Code offers many features commonly found in integrated development environments (IDEs) (Microsoft, n.d.).

9.3 Tensorflow and Keras

Tensorflow is an open source machine learning framework designed for building and training of machine learning models and neural networks. It consists of libraries and tools for machine learning tasks [66].

Keras is an open-source neural network library written in Python. It is designed to be user-friendly, modular, and extensible, allowing users to quickly prototype and build deep learning models with minimal code. Keras was originally developed by François Chollet and was integrated into TensorFlow as its official high-level API starting from TensorFlow version 2.0. Keras provides a simple and intuitive API that allows users to define neural networks using high-level building blocks like layers, activations, optimizers, and loss functions. Keras supports both convolutional and recurrent neural networks, as well as combinations of the two. It also provides support for various types of layers, such as dense, convolutional, recurrent, and more [40].

9.4 Research Question

The objective of this research is to address the following research questions:

● Is it feasible to create a deep learning methodology for fabric defect detection that combines?

a) efficient speed,

b) heightened accuracy and

c) comprehensive training encompassing various fabric defect types.

● Can this method incorporate a balanced approach to minimize false negatives in detection rates?

The focus was placed on deep learning algorithms. While a considerable amount of research has been done, several limitations persist, and a system that meets the criteria for deployment within industry has not been achieved.

10. Materials and Methods

10.1 Dataset

The data collected for this project were from :

● Fabric Defect Dataset from Kaggle [41].

● Fabric Stain Dataset from Kaggle [42].

● Aitex fabric image database [43]

● Dataset from the author of the literature review paper: only the non-defect images from the dataset [14].

10.2 Aitex Fabric Image Database

This textile fabric dataset consisted of 245 images of 7 different fabrics with image sizes of 4096×256 pixels. There were 140 defect-free images, 20 for each type of fabric. In each of the defected category, there were 105 images [43].

10.3 Fabric Stain Dataset

This was taken from Kaggle [42]. The dataset was built as a part of the fabric defect detection project of the Intelligence Lab of University of Moratuwa, Sri Lanka. The dataset consisted of images with resolution of 1920x1080 or 1080x1920. It consisted of 398 defected images with different types of stains and 68 defect free images.

10.4 Fabric Defect Dataset

The dataset was also taken from Kaggle and it was supplied by the Intelligence Lab, Department of computer science and Engineering, University of Moratuwa [41]. This dataset consisted of 3 classes of defective images namely horizontal, vertical and holes along with 3 mask images for each defective image sample. The folder named 'captured' consisted of raw images. Images have a size of 640x360.

10.5 Dataset from the x of the Literature Review Paper

Authors of the research papers in the literature review were contacted through emails for the dataset. With the consent of the author of the paper "Automatic fabric defect detection method using pran-net" named Troy Peng, the annotated data was used [14].

After collecting the images from all these sources, the distribution of the images is as given below:

● Defect free 1,666

● Defected 1,073

Defected hole 281

Defected horizontal 136

Defected lines images 157

Defected stain images 398

Defected verticle images 101

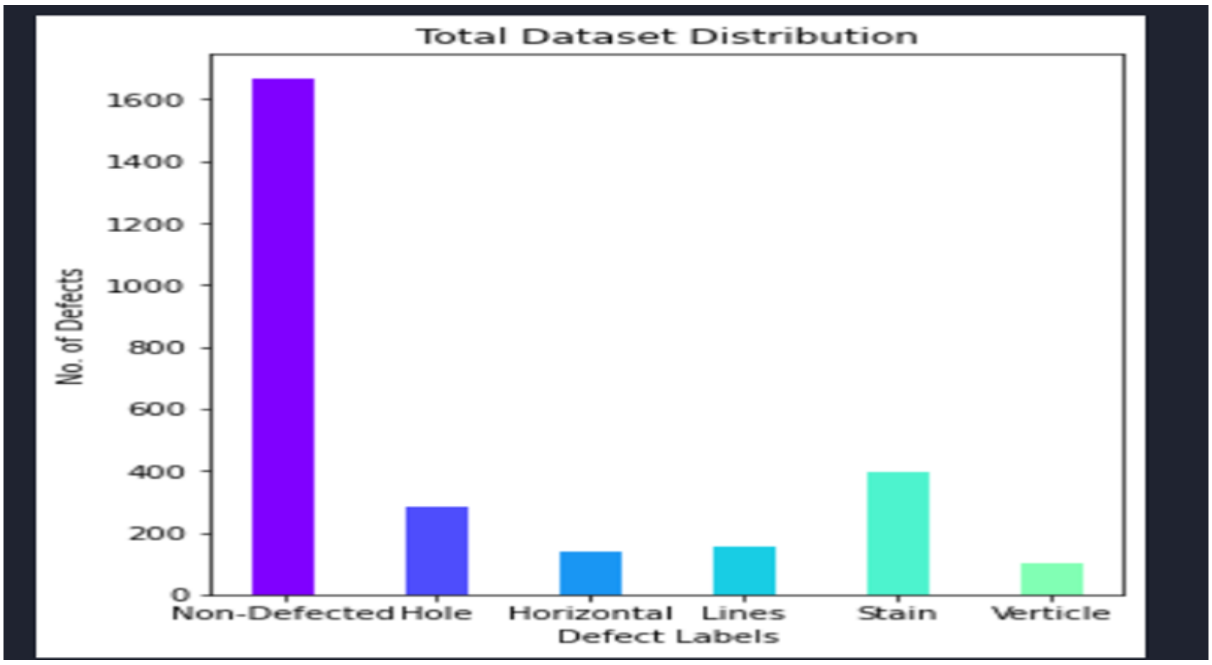


Figure 3: Number of Images in Each Class of the Datase

Total Images = 2,739

10.6 Experimental Setup and Evaluation

The images were experimented with two distinct sizes:

● 150x700
● 245x345

The entire experiment was carried out using the Python programming language . Keras , in conjunction with TensorFlow , was employed for the development and training of CNN. Various libraries, including numpy, pandas, and scikit-learn, were used for supporting tasks. The experimentation was conducted within the Visual Studio Code (VS Code) environment.

10.7 Data Pre-processing

The acquired dataset exhibited an imbalance in its distribution across different classes. Consequently, various sampling techniques from the imbalanced-learn module were applied and tested on the three chosen image sizes i.e. 150x700, and 245x345. The objective was to determine the most effective sampling technique that yields high performance.

10.8 Data Visualization

Data visualisation was facilitated through Principal Component Analysis (PCA).

Principal Component Analysis (PCA) is considered good for data visualization for several reasons:

- PCA reduces the number of dimensions in the data while preserving as much variance as possible. This simplifies complex datasets and makes them easier to visualize, typically in 2D or 3D.
- By projecting data onto principal components that capture the most variance, PCA highlights the underlying structure and patterns, making relationships in the data more apparent.
- PCA can help in removing noise and redundant information, which can clarify the visual representation of the data.

10.9 Modelling- Convolutional Neural Network

A CNN model's architecture was designed to facilitate multi-classification and consisted of five two-dimensional convolutional layers for feature extraction.

● The initial convolutional layer was configured with 16 filters, each having a 7x7 kernel size. The second convolutional layer used a set of 32 filters with a 5x5 kernel size.
● The third layer used 64 filters with a 3x3 kernel size.
● The fourth layer comprised 128 filters with a kernel size of 3x3.
● Lastly, the fifth layer was equipped with 256 filters and used a 3x3 kernel size.

Padding with the hyperparameter 'same' was applied to allow the filter kernels to traverse the images by including additional pixels. Using 'same' as the padding hyperparameter ensured that the size of the output feature map remained the same as that of the input feature map.

Each successive convolutional layer was accompanied by batch normalization, an activation function, max pooling, and dropout layers. Batch normalization was used to standardize and normalize the dataset within batches, contributing to improved training and learning speed.

The outputs of linear operations, such as convolutions, passed through nonlinear activation function Rectified Linear Unit (ReLU). Rectified Linear Unit (ReLU) activation function was employed due to the multi-layered nature of the CNN.

For this novel model, Max pooling was adopted to reduce image dimensions and achieve downsampling, creating lower-resolution images with essential features. A 2x2 filter shape was employed for max pooling since it should be smaller than the feature map's dimensions. This pooling layer generated new feature maps. Dropout layers were introduced to address overfitting.

In the final phase of the model, a fully connected layer, also known as a dense layer, was used to align the outputs from the pooling layers with the dataset labels. Since multiple pooling layers were employed, a flattening layer was introduced to organize these outputs sequentially into a vector. After passing through all the convolutional layers, the output took the form of a multidimensional array, which was then fed into a dense layer. To interpret the results as a probability distribution, a SoftMax function was applied in the fully connected layer.

To evaluate compatibility between network output predictions and given labels, a loss function i.e. Adaptive Moment (Adam) function was employed.

10.10 Parameter Settings
For this novel idea, for each experiment, the dataset was divided into a training set and a test set using a 95:05 stratified sampling ratio. The validation data was used to examine if the hyperparameters required further tuning. The test data was used as an unseen dataset to examine the results of the model. The training process encompassed 100 epochs , with a batch size of 6 samples per batch.

Using a stratified k-fold cross-validation approach with 10 folds, the dataset is split into training and validation sets for each fold. For every fold iteration, a Convolutional Neural Network (CNN) model is created using the specified image shape and trained on the training data for 100 epochs with a batch size of 6. The model's performance is evaluated on the validation set, and both the validation loss and accuracy are recorded. This iterative process allows for a comprehensive assessment of the model's performance across multiple folds, contributing to a more reliable evaluation of its generalization capabilities."

10.11 Evaluation Metrics
Evaluation metrics used were accuracy, precision, recall, F1-score, confusion matrix.

10.12 Validation Time
This is the time taken to evaluate the performance of the model on the validation dataset after each epoch (or at specified intervals during training). The validation dataset is a separate set of data not used for training but for assessing how well the model is generalizing. This helps in tuning the model's hyperparameters and prevents overfitting. The validation time includes the time required to compute metrics like accuracy, precision, recall, and F1 score on this dataset

10.13 Modelling Time
Also known as training time, this is the duration required for the model to learn from the training dataset. It includes the time taken for forward and backward propagation through the network for all epochs until the model converges. This phase involves the optimization of the model parameters (weights and biases) through iterative updates. Efficient modelling time is critical for developing models quickly, especially when dealing with large datasets or complex architectures.

10.14 Prediction Time
This is the time it takes for the trained CNN model to make predictions on new, unseen data. It measures how quickly the model can infer the class labels or other outputs for input data during deployment. Low prediction time is essential for real-time applications, where the model needs to provide results almost instantaneously, such as in autonomous driving or live video analysis.

11. Results
CNN Modelling was performed for the two selected sizes of images dataset without any sampling technique. In order to optimize the performance of the CNN model, sampling techniques available in the imbalanced-learn module were experimented to select the best sampling technique with high performance. The results were as given in table 1, 2 and 3.

	Parameters		
	Total	**Trainable**	**Non-trainable**
245x345	2,698,310	2,697,062	1,248
150x700	3,157,062	3,155,814	1,248

Table 1: Total number of Trainable and Non-Trainable Parameters for the Two Image Sizes

	245x345				150x700			
	Accuracy	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
	%	**%**	**%**		**%**	**%**	**%**	
Without Sampling	94	96	94	94	91	91	91	91
Undersampling								
Cluster Centroids	89	91	89	90	73	87	73	76
AllKNN	93	**97**	93	94	94	95	94	94
Condensed Nearest Neigbor	65	83	65	69	69	73	69	69
Edited Nearest Neighbours	90	93	90	89	87	91	87	87
RepeatedEditedNearestNeighbours	88	90	88	88	91	93	91	91
Instance Hardness Threshold	87	88	87	87	85	87	85	86
Near Miss	73	87	73	75	77	85	77	78
Neighbourhood Cleaning Rule	95	95	95	95	91	92	91	91
TomekLinks	92	93	92	92	91	91	91	90
Random Undersampler	93	94	93	93	90	93	90	90
Oversampling								
ADASYN	95	95	95	95	90	90	90	90
Random Oversampling	96	**97**	96	96	95	96	95	95
Borderline SMOTE	96	96	96	95	93	93	93	93
kMeans SMOTE	94	95	94	94	95	96	95	95

SMOTEN	91	92	91	91	93	94	93	93
SMOTENC								
SVM SMOTE	96	96	96	96	95	95	95	95
Combined Oversampling and Undersampling								
SMOTEENN	**98**	**98**	**98**	**98**	**96**	**96**	**96**	**96**
SMOTETomek	96	96	96	96	91	91	91	91
Ensemble								
Pipeline	93	93	93	93	87	89	87	87
Miscellaneous								
Function Sampler	94	95	94	94	96	97	96	96

Table 2: Model Performance Without and With Data Pre-Processing

	245x345			**150x700**		
	Validation Time	**Model Time**	**Pred Time**	**Validation Time**	**Model Time**	**Pred Time**
	Sec	**Sec**	**Sec**	**Sec**	**Sec**	**Sec**
Without Sampling	2.64	0.22	0.06	3.26	0.34	0.06-0.1
Undersampling						
ClusterCentroids	2.8	0.29	0.06-0.1	3.11	0.205	0.07-0.08
AllKNN	2.69	0.21	0.07	3.52	0.24	0.07-0.08
Condensed Nearest Neigbor	2.52	0.2	0.07	3.18	0.20	0.06-0.15
Edited Nearest Neighbour	2.5	0.2	0.09	3.3	0.22	0.09
Repeated Edited Nearest Neighbout	2.6	0.2	0.06	3,12	0.21	0.05-0.7

Instance Hardness Threshold	2.50	0.21	0.07	3.32	0.2	0.08
Near Miss	2.69	0.24	0.07	3.2	0.2	0.07
Neighbourhood Cleaning Rule	2.82	0.24	0.07	3.4	0.25	0.06-0.14
TomekLinks	2.45	0.27	0.06-0.1	2.93	0.3	0.08-0.1
Random Undersampler	2.55	0.23	0.06-0.08	3.86	2.51	0.09
Oversampling						
ADASYN	2.43	0.56	0.08-0.1	3.53	0.89	0.08
Random Oversampling	2.90	0.7	0.06-0.1	2.98	0.23	0.07
Borderline SMOTE	3.25	1.96	0.06-0.096	3.31	1.74	0.087-0.065
kMeans SMOTE	3.2	1.1	0.09	3.47	5.66	0.077
SMOTEN	3	1.9	0.07	3.36	2.02	0.07-0.09
SMOTENC						
SVM SMOTE	3.15	1.90	0.085-0.1	3.40	1.86	0.07-0.086
Combined Oversampling and Undersampling						
SMOTEENN	3,4	1.57	0.09	3.5	1.15	0.07-0.1
SMOTETomek	2.77	2.21	0.06	3.67	2.44	0.07-0.12
Ensemble						
Pipeline (AIKNN+Neighbouhood CleaningRule)	2.8	0.31	0.07	3.26	0.35	0.06-0.09
Miscellaneous						
Function Sampling	2.59	0.21	0.07-0.1	3.26	0.2	0.07-0.15

Table 3: Model Time Performance With Data Pre-Processing

12. Discussion

The aims of this study were twofold: to amass an image dataset containing fabric defects and to create a model capable of accurately categorizing these defects, ultimately leading to an automated system for identifying and detecting fabric defects.

To do this, state-of-the-art data sampling techniques were applied to the dataset consisting of a total of 2,739 images, including a total of 1666 non-defected images and 1,073 defected images. These sampling techniques were experimented on a novel CNN image detection methodology. These results demonstrated some interesting findings relating to AI and implementation strategies for future commercial deployment strategies.

As depicted in table 2, when analysing the dataset without pre-processing, the CNN model's performance was impacted by the imbalanced nature of the data. For images size 150 x 700, the model achieved 91% accuracy, precision, recall, and F1 score, respectively. However, with images sized 245x345, the model's performance notably improved, reaching 94% accuracy, 96% precision, 94% recall and F1 score. Remarkably, the 245x345 image size demonstrated superior performance compared to the larger 150x700 image size. But even then, the performance was not outstanding and needed further improvement. For this reason, different sampling techniques were experimented to improve the performance of the model.

12.1 Oversampling

In table 2, it is evident that ADASYN, Random Oversampling, Borderline SMOTE, and SVM SMOTE yielded superior performance when applied to images sized 245x345 compared to those sized 150x700, across metrics such as accuracy, precision, recall, and F1 score. Unfortunately, SMOTENC encountered memory size limitations and could not be executed. Notably, kMeans SMOTE and SMOTEN did not exhibit enhanced performance.

Among the oversampling techniques, Random Oversampling, Borderline SMOTE, and SVM SMOTE showcased exceptional performance. Notably, Random Oversampling achieved a precision of 97%, outperforming the other techniques in this regard.

As indicated in Table 2, both Random Oversampling and ADASYN exhibited notably quicker validation and modelling times for images sized 245x345 compared to other oversampling methods, as well as compared to the 150x700 size across all methods. The longest validation times were observed for k Means SMOTE, Borderline SMOTE, SMOTEN, and SVM SMOTE.

12.2 Under Sampling

None of the under sampling techniques demonstrated satisfactory performance for either of the two image sizes. However, Neighbourhood Cleaning Rule and AIIKNN showcased notable improvements. Neighbourhood Cleaning Rule exhibited superior performance, achieving 95% across accuracy, precision, recall, and F1 Score for images sized 245x345. AIIKNN displayed enhanced precision, reaching 97% and 95% for images sized 245x345 and 150x700, respectively.

The validation, modelling, and prediction times for both AIIKNN and Neighbourhood Cleaning Rule were nearly equivalent to the model's performance without sampling. This held true for both image sizes, 245x345 and 150x700. Among the under sampling techniques, Tomek links exhibited the shortest validation, modelling, and prediction times.

12.3 Combined Undersampling and Oversampling

Remarkably, both SMOTEENN and SMOTE Tomek surpassed the performance of combined oversampling and undersampling techniques. Particularly, SMOTEENN demonstrated superior performance compared to SMOTE Tomek.

12.3.1 Ensemble

As depicted in Table 2, the Pipeline sampling utilized undersampling techniques AIKNN and Neighbourhood Cleaning Rule. Due to limited memory space, oversampling techniques could not be evaluated. Consequently, the Pipeline could not demonstrate any performance across both image sizes.

12.3.2 Miscellaneous

The Function sampler did not exhibit any performance improvement with the image size 245x345. However, for the larger image size of 150x700, performance was notably better, with 96% accuracy, recall, and F1 score, and 97% precision.

From Table 2, it's apparent that among the best-performing sampling techniques across various experimented methods, SMOTEENN, combining oversampling and undersampling, delivered the most promising performance compared to other selected top-performing methods. The CNN model's performance with the SMOTEENN sampling technique was notably strong for the image size of 245x345.

The following table (Table 4) provides an overview of the best-performing techniques among the different sampling methods tested with two image sizes: 150x700 and 245x345. As shown in the table, the combined oversampling and undersampling techniques yielded the best results. Among the two combined techniques tested, SMOTEENN and SMOTETomek, SMOTEENN achieved the highest performance with 98% accuracy, precision, recall, and F1 score for the 245x345 image size.

For SMOTEENN, the validation, modeling, and prediction times for the 245x345 image size were nearly the same as for the 150x700 image size. However, for SMOTETomek, the validation time for the 245x345 size was lower (2.7 seconds) compared to SMOTEENN (3.4 seconds). Conversely, the modeling time for SMOTETomek was higher (2.21 seconds) compared to SMOTEENN (1.57 seconds).

The performance of SVM SMOTE, Random Oversampling, and Borderline SMOTE—techniques categorized under oversampling for the 245x345 image size—and the miscellaneous sampling technique, Function Sampler, for the 150x700 image size, remained nearly identical in terms of accuracy, precision, recall, and F1 score. The validation, prediction, and modeling times for SVM SMOTE, Random Oversampling, and Borderline SMOTE were relatively shorter compared to the combined oversampling and undersampling technique, SMOTEENN.

For images sized 245x345, the validation, modeling, and prediction time for SMOTEENN was longer compared to SMOTE Tomek, being the maximum among the undersampling and oversampling techniques. SMOTE Tomek exhibited shorter validation times, outperforming other undersampling and oversampling techniques. However, as SMOTEENN has surpassed SMOTETomek in terms of accuracy and precision, therefore, SMOTEENN remains on the top of SMOTETomek and among other sampling techniques in terms of overall performance.

Sampling	**Image Size**		**Accuracy**	**Precision**	**Recall**	**F1**	**Time**		
							Model S	**Predict s**	**Valid s**
Oversampling and Undersampling	245x345	SMOTEENN	0.98	0.98	0.98	0.98	1.57	0.09	3.4
	150x700		0.96	0.96	0.96	0.96	1.15	0.07-0.1	3.5
	245x345	SMOTETomek	0.96	0.96	0.96	0.96	2.21	0.06	2.77
Oversampling	245x345	SVM SMOTE	0.96	0.96	0.96	0.96	1.90	0.09-0.1	3.15
	150x700		0.95	0.96	0.95	0.95	1.86	0.07-0.09	3.4
	245x345	Random Oversampling	0.96	0.97	0.96	0.96	0.7	0.06-0.1	2.90
	150x700		0.95	0.96	0.95	0.95	0.23	0.07	2.98
	245x345	Borderline SMOTE	0.96	0.96	0.96	0.95	1.96	0.06-0.09	3.25
Miscellaneous	150x700	Function Sampler	0.96	0.97	0.96	0.96	0.2	0.07-0.15	3.26

Table 4: Performance Comparison of the Best Performing Sampling Techniques

When analysing results, PCA was used for the data visualization. Figure 4 shows the data set before data pre-processing.

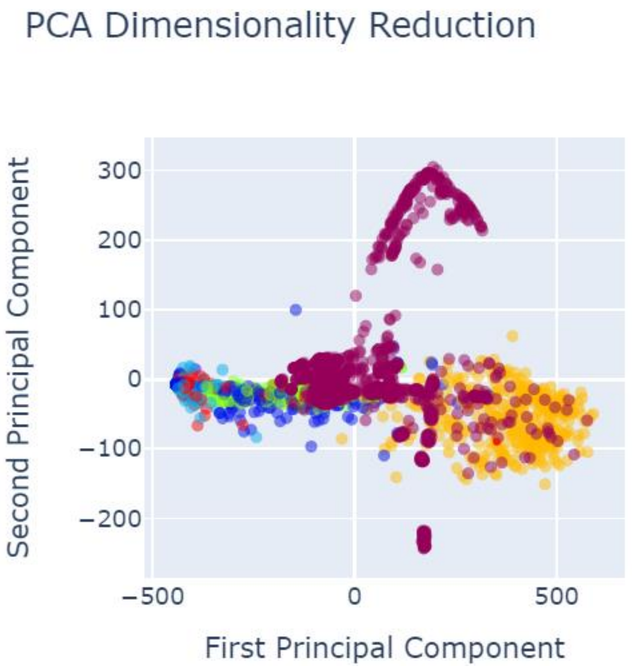


Figure 4: PCA before Data Pre-Processing

- Red coloured narrates the defect free images.

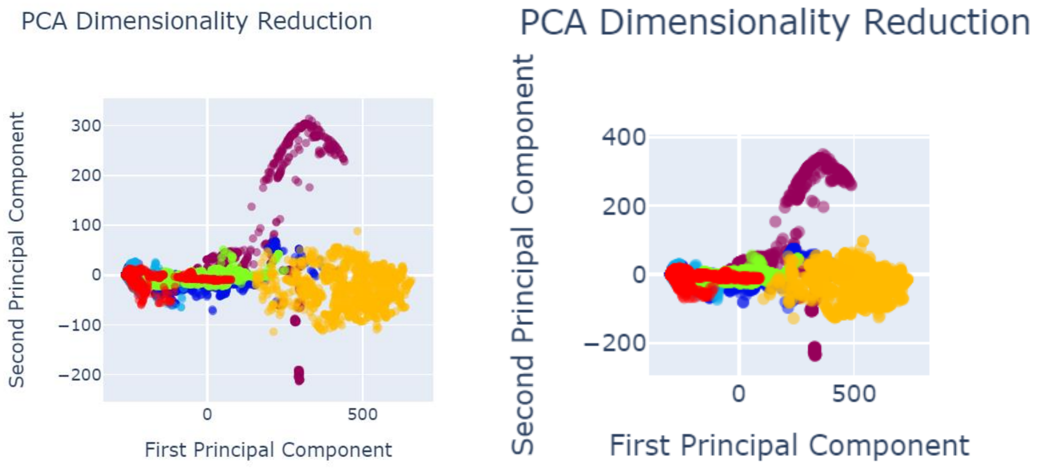
Defected images are shown as:

- Royal Blue shows the holes,
- Orange shows vertical
- Green shows lines
- Light blue shows horizontal
- Yellow shows stains

As shown in the figure 4, defects are not properly segregated and therefore, model could not predict the defect classes accurately and precisely. For example, the yellow dots representing stains could be misclassified as defect-free since they are mixed with the red dots. Similarly, the royal blue dots are mixed with the green dots, leading to potential misclassification of defects. A similar issue occurs with the royal blue dots, which are mixed with the green dots, leading to potential misclassification of these defects.

The following figures 5 and 6 give the data visualization using Principal Component Analysis for the best performing techniques.

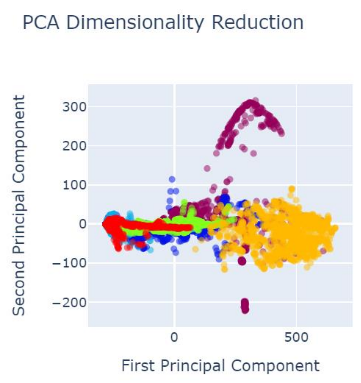
Fig. 5 shows the data visualization when data was pre-processed using best performed sampling technique SMOTEENN.



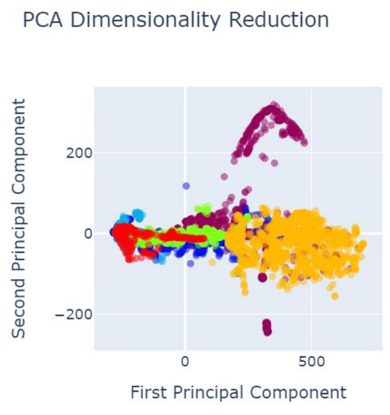
a) Image size 245x345 b) Image size 150x700

Figure 5: PCA After Data Pre-Processing with SMOTEENN Sampling Technique

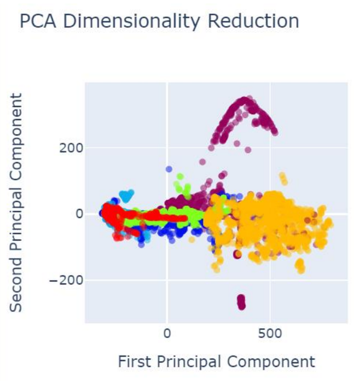
As shown in the fig. 5a. There is a clear separation between the different colors showing that all the classes are completely seggragated that led to high accuracy. Similary, fig 5b also shows the similar but somewhere like there is a small merge of two classed. For example, the red color is somewhat mixed with other colors.



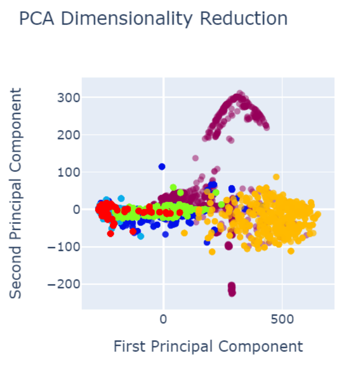
a



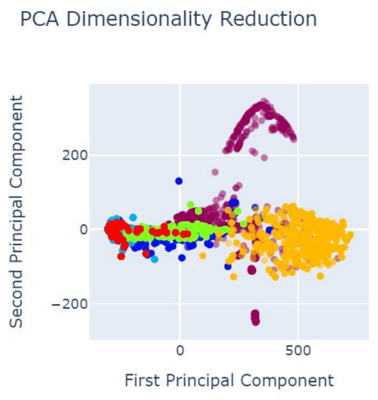
b



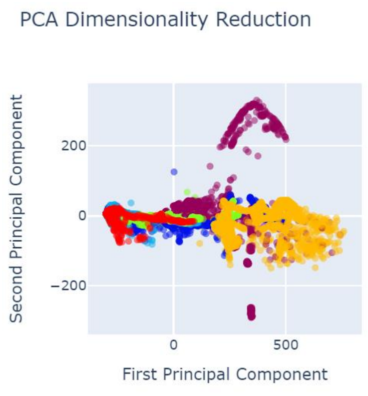
c



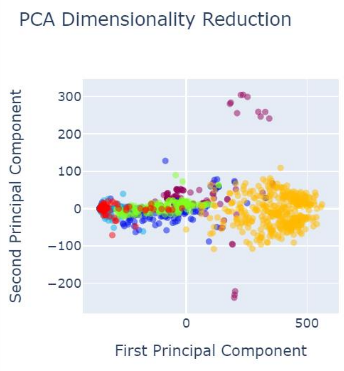
d



e



f



g

Figure 6: PCA After Data Pre-Processing with:

a) SMOTETomek image size 245x345
b) SVM SMOTE Oversampling image size 245x345
c) SVM SMOTE Oversampling image size 150x700
d) Random Oversampling image size 245x345
e) Random Oversampling image size 150x700
f) Borderline SMOTE image size 245x345
g) Function Sampler image size 150x700

It is obvious from the Fig 6 that the model with the different resampling techniques like SMOTETomek (image size 245x345), SVM SMOTE (245x345 and 150x700), Random sampling (image size 245x345, 150x700), Borderline SMOTE (image size 245x345) and Function Sampler (image size 150x700), the red colored dots representing the defect free class are mixed with the other classes that is the reason for the decrease in the accuracy of the model performance.

PCA hence is a very effective method to understand the model performance. This method gives a clear picture of the multiclassification behaviour of the model.

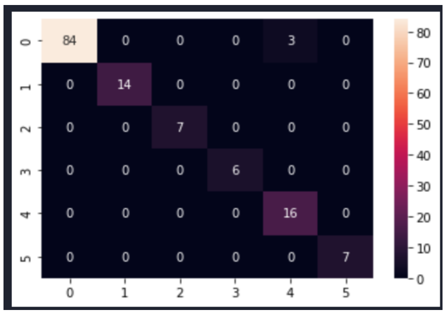
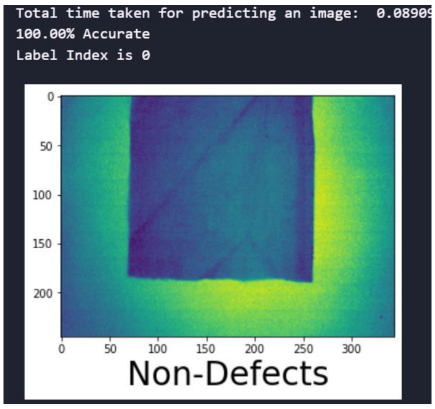


Figure 7: Shows the Confusion Matrices After Modelling with CNN for the Selected Size 245x345

Fig. 7 Confusion matrix of the CNN model sampled with SMOTEENN method for image size 245x345



```
Total time taken for predicting an image:  0.08909
100.00% Accurate
Label Index is 0
```

Figure 8: Prediction Time and Accuracy of the CNN Model with Smoothen Sampling Technique Image Size 245x345

Fig.8 shows the time taken and accuracy for predicting the defects and the accuracy of predictions for the developed CNN with SMOTEENN sampling techniques for the image size 245x345. It is evident that the trained model performance remained 100% accurate.

Verification from Literature Review

The findings from the experimentation align with the conclusions drawn by Luke, Joseph, and Balaji (2019, p. 70), which are elaborated below [44]:

1. CNN significantly influences image resolution accuracy.
2. Image sizes exert a substantial impact on model accuracy. Changes in size can either enhance or degrade model performance.
3. Accuracy tends to increase as image sizes are enlarged, but this trend reverses beyond a certain threshold due to reduced receptive field.
4. Larger image sizes impact the ability to learn medium and high-level features.
5. To achieve higher accuracy, the optimal image size falls within the range between the default size and 512.
6. Resizing images affects low-level features due to information loss, thereby influencing accuracy. The accuracy is particularly influenced by inter-class similarities, which are closely tied to low-level features [44].

Image resolution notably influences the model's performance. Thambawita et al. (2021) discovered that CNN achieved its best performance with a larger image size of 512x512 compared to 32x32 [45].

Horwath et al. (2020, p. 4) noted that image segmentation becomes challenging with high-resolution images due to issues recognizing interface pixels without considering the background. The complexity of features increases accordingly [22].

Sabottke and Spieler (2020, p.2) emphasized that a lower number of parameters leads to improved model performance by reducing overfitting risk [46]. However, excessive resolution reduction may result in a loss of valuable information. They found that CNN performance was optimal within the range of resolutions between 256x256 and 448x448 image sizes. Their conclusion was that there's a trade-off between increasing image resolution and the maximum feasible batch size. A high number of parameters affects model performance not solely due to overfitting but also because of the intricacies involved in optimization [46].

Rukundo (2021, p. 17) highlighted that determining the optimal image size for training datasets is a significant challenge. He observed that a size of 256x256 outperformed 128x128, attributing this improvement to the lower cross-entropy loss in a U-net trained on a 256x256 training image set. He emphasized that lower loss corresponds to greater model accuracy [47].

12.4 Result Comparison with Models from the Related Works

Until today, fabric defect detection tasks have primarily been carried out manually. Numerous research studies have proposed defect detection algorithms; however, these methods have certain limitations. They often suffer from slow processing speeds and lack the capability to effectively detect a wide range of defect types. Consequently, these methods have not found practical applicability within the textile industry.

The following table gives a brief view of the evaluation of the models discussed in the section of related works.

	Accuracy %	Precision %	Recall %	F1 Score %	Detection Time
Peng et. al (2020) PRAN-NET Denim	91.99				9.7 Frame per sec
Peng et. al (2020) PRAN-NET Plain	98.66				25.4 Frame per sec
Almeida, Moutinho and Matos-Carvalho (2021) CNN-FN Reduction	75.45	72.76	88.47		5.69 ms per image
Zhao et. al (2019) CNN-VLSTM	95.73-99.47	95.74-100	95.73-100	95.73-100	1.8-3.5 e2
Wei et. al (2019) Faster RCNN	95.88				0.3 sec
Liu et al. (2019) VGG16	97.7				
Liu et al. (2019) LZFNet	98				13.8 ms
Cheng et al. (2022) SCUNet	98.01		96.86		
Our Model	98	98	98	98	0.06-0.07 sec

Table 5: Performance Comparison of Different Models from the Literature Review with our Model

All the models discussed in the related works section exhibited a common drawback: they are slow and time-consuming, rendering them unsuitable for various types of fabric defects. Although CNNs based on visual VLSTM presented by Zhao et al. (2019) demonstrated superior performance in terms of accuracy and speed but they also exhibit significant sensitivity to intricate fabric structures [11].

The performance of the model developed by Zhao et al. (2019) is given in the table 8.

Dataset	Accuracy	Precision	Recall	F1
	%	%	%	%
500	99.5	100	100	100
1000	97.7	99.26	99.20	99.20
10500	95.7	95.74	95.73	95.73

Table 6: Performance Metrics of Zhao et al. (2019) Model

The model developed by Zhao et al. (2019) was based on visual long-short term memory (VLSTM) [11]. In contrast to the model proposed by Zhao et al. (2019), the current model is:

● notably simpler,
● lacking the integration of the CNN architecture with other mechanisms.

● Zhao et al. (2019) conducted experiments with varying image quantities, such as 500, 1000, and 10500, and identified the dataset consisting of only 500 images as yielding the best performance. However, the performance of the larger image datasets, particularly the 1000-image dataset, was suboptimal [11].

In contrast, the CNN model devised in this research exhibited superior performance when trained with a substantial dataset comprising 2,739 images, particularly excelling with the image size of 245x345. Consequently, the model discussed in this research paper emerges as the most suitable candidate for deployment within the textile industry.

The SCUNet model, developed by Cheng et al. (2022), exhibits several limitations. Notably, the experimentation involved a singular image size of 256x256, a small dataset comprising only 106 grayscale images, and data processing utilizing image cropping and augmentation. Importantly, the model's performance on colored data images remains unknown, as the experiments focused solely on grayscale images [25]. Cheng et al. (2022, p. 3115) acknowledge a limitation wherein the model requires fabric to be "flat"; however, the term is inaccurately used, as the authors seem to refer to "plain" fabric—fabric devoid of patterns or designs [25]. The model's inadequacy in handling designed fabrics is evident. Cheng et al. (2022, p. 3117) claim time efficiency compared to manual visual inspection but fail to present metrics for detection time, leaving the efficiency and speed of defect detection unmeasured [25]. In contrast, our research paper not only addresses detection time but also evaluates the model's efficiency. Our model, tested on both colored and grayscale images, encompasses all defect sizes. The dataset, curated from various sources, includes plain and patterned fabrics. In direct comparison, our model outperforms SCUNet with an overall accuracy of 98% and recall of 98%. This achievement is particularly noteworthy given our comprehensive experimentation with diverse fabric types, defect sizes, and image variations, demonstrating superior adaptability and performance.

13. Conclusion

This research introduces an innovative approach for the automated detection of fabric defects within the textile industry. Within this sector, the journey begins with yarn from the spinning department, which is then utilized in the weaving unit to create greige fabric. The subsequent stages involve processes such as scouring, bleaching, dyeing, and finishing the fabric. Once the fabric reaches the finished state, it undergoes a quality assessment before proceeding to the stitching department for garment production. During the quality inspection phase, the fabric is placed across quality tables, where inspectors meticulously examine it and identify any instances of fabric defects, marking the fabric as necessary.

The objective of this study was to identify a suitable deep learning algorithm capable of automating fabric quality inspection in the textile industry, thereby replacing the labor-intensive manual process. After conducting a comprehensive literature review, a convolutional neural network (CNN) algorithm was chosen as the basis for the research. To determine the optimal image size for the algorithm, two sizes—150x700, and 245x345—were tested to achieve the best model performance. Addressing the challenge of dataset imbalance, various sampling techniques were employed and evaluated.

Remarkably, the research revealed that the CNN model yielded the most impressive results when combined with the SMOTEENN sampling technique. This outcome signifies a significant advancement in automating fabric quality inspection within the textile industry.

The research highlighted the substantial impact of image size on the model's performance. The CNN model that was developed demonstrated its most exceptional performance when paired with the SMOTEENN sampling technique. Among the two tested image sizes—150x700, and 245x345—the 245x345 size emerged as the most optimal performer.

In terms of practical application, this model stood out for its swift processing times. Specifically, for the image sizes 245x345, the model achieved a validation time of just 3.4 seconds and a prediction time of 0.09 seconds. While the 150x700 size demanded 3.5 seconds and 1.15 seconds for the same tasks. This efficiency positions the model as a viable candidate for deployment within the industry.

Deployment Strategy

A novel approach is introduced to revolutionise the fabric quality inspection process. This method integrates the utilisation of an infrared camera, which scans the entire fabric width while it is conveyed along the table's length. Capturing high-resolution infrared images, the camera provides data for the developed machine learning algorithm, which is responsible for identifying fabric defects. Upon detecting a defect, the algorithm triggers an automatic labelling machine to mark the defective section. The entire workflow is illustrated in fig. 9 below:

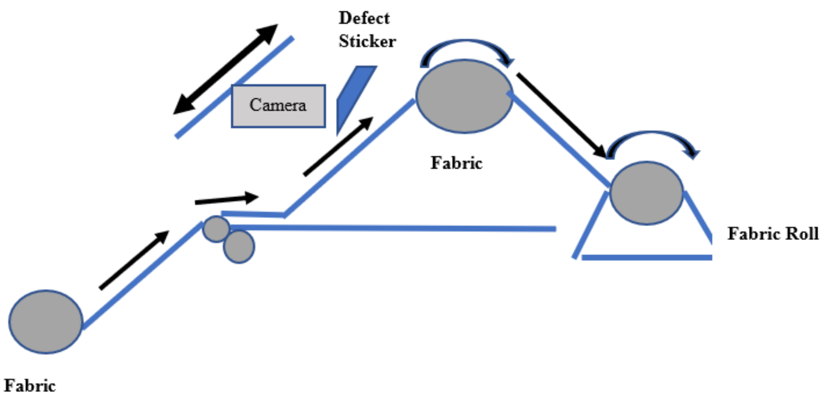


Figure 9: Automated Fabric Defect Detection

Future Research

Further research is warranted to enhance the robustness of the model by expanding the image dataset and incorporating a broader array of defect types for multi-classification purposes.

Additionally, more extensive research is needed to facilitate the deployment of this model within the textile industry. This would involve the implementation of infrared cameras and defect labelling machines to fully automate the fabric quality inspection process. Such endeavours would contribute to the comprehensive integration of the developed algorithm into practical industrial settings.

Acknowledgments

I extend a special thanks to Peiran Peng, the author of "Automatic Fabric Defect Detection Method Using PRAN-Net," as well as to Kaggle for their invaluable support in facilitating the data collection process.

Conflict of Interest Statement

The author has no conflict of interest to declare.

References

1. Li, C., Li, J., Li, Y., He, L., Fu, X., & Chen, J. (2021). Fabric defect detection in textile manufacturing: a survey of the state of the art. *Security and Communication Networks, 2021*(1), 9948808.
2. Wang, Z., & Jing, J. (2020). Pixel-wise fabric defect detection by CNNs without labeled training data. *IEEE Access, 8,* 161317-161325.
3. Jing, J., Wang, Z., Rätsch, M., & Zhang, H. (2022). Mobile-Unet: An efficient convolutional neural network for fabric defect detection. *Textile Research Journal, 92*(1-2), 30-42.
4. Du, J. (2018, April). Understanding of object detection based on CNN family and YOLO. In *Journal of Physics: Conference Series* (Vol. 1004, p. 012029). IOP Publishing.
5. Fan, H., & Zhou, E. (2016). Approaching human level facial landmark localization by deep learning. *Image and Vision Computing, 47,* 27-35.
6. Liu, Z., Liu, S., Li, C., Ding, S., & Dong, Y. (2018, October). Fabric defects detection based on SSD. In *Proceedings of the 2nd international conference on graphics and signal processing* (pp. 74-78).
7. Zhang, H. W., Zhang, L. J., Li, P. F., & Gu, D. (2018, May). Yarn-dyed fabric defect detection with YOLOV2 based on deep convolution neural networks. In *2018 IEEE 7th data driven control and learning systems conference (DDCLS)* (pp. 170-174). IEEE.
8. Minhas, M. S., & Zelek, J. S. (2020, February). Defect Detection using Deep Learning from Minimal Annotations. In *VISIGRAPP* (4: VISAPP) (pp. 506-513).
9. Liu, Z., Zhang, C., Li, C., Ding, S., Dong, Y., & Huang, Y. (2019). Fabric defect recognition using optimized neural networks. *Journal of Engineered Fibers and Fabrics, 14*, 1558925019897396.
10. Prakash, S., Berry, D. P., Roantree, M., Onibonoje, O., Gualano, L., Scriney, M., & McCarren, A. (2021). Using artificial intelligence to automate meat cut identification from the semimembranosus muscle on beef boning lines. *Journal of Animal Science, 99*(12), skab319.
11. Zhao, Y., Hao, K., He, H., Tang, X., & Wei, B. (2020). A visual long-short-term memory based integrated CNN model for fabric defect image classification. *Neurocomputing, 380,* 259-270.
12. Wei, B., Hao, K., Tang, X., S. and Ren, L. (2019) 'Fabric defect detection based on faster RCNN' in *Advances in Intelligent Systems and Computing, 849*, pp. 45–51.
13. Chen, M., Yu, L., Zhi, C., Sun, R., Zhu, S., Gao, Z., ... & Zhang, Y. (2022). Improved faster R-CNN for fabric defect detection based on Gabor filter with Genetic Algorithm optimization. *Computers in Industry, 134*, 103551.
14. Peng, P., Wang, Y., Hao, C., Zhu, Z., Liu, T. and Zhou, W. (2020) 'Automatic fabric defect detection method using pran-net' *Applied Sciences (Switzerland), 10* (23), pp. 1–13.
15. Almeida, T., Moutinho, F., & Matos-Carvalho, J. P. (2021). Fabric defect detection with deep learning and false negative reduction. *IEEE Access, 9,* 81936-81945.
16. He, Y. (2021). Fabric Defect Detection based on Improved Faster RCNN. *International Journal of Artificial Intelligence & Applications (IJAIA), 12*(4).
17. Wallelign, S., Polceanu, M., & Buche, C. (2018, May). Soybean plant disease identification using convolutional neural network. In *The thirty-first international flairs conference*.

18. Yamashita, R., Nishio, M., Do, R., K., G. and Togasi, K. (2018), 'Convolutional neural networks: an overview and application in radiology', *SpringerOpen.*
19. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.
20. Goodfellow, I. (2016). Deep learning.
21. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research, 16*, 321-357.
22. Horwath, J. P., Zakharov, D. N., Mégret, R., & Stach, E. A. (2020). Understanding important features of deep learning models for segmentation of high-resolution transmission electron microscopy images. *npj Computational Materials, 6*(1), 108.
23. Letteri, I., Di Cecco, A., Dyoub, A., & Della Penna, G. (2020). A novel resampling technique for imbalanced dataset optimization. *arXiv preprint arXiv:2012.15231*.
24. Rasheed, A., Zafar, B., Rasheed, A., Ali, N., Sajid, M., Dar, S. H., ... & Mahmood, M. T. (2020). Fabric defect detection using computer vision techniques: a comprehensive review. *Mathematical Problems in Engineering, 2020*(1), 8189403.
25. Cheng, L., Yi, J., Chen, A., & Zhang, Y. (2023). Fabric defect detection based on separate convolutional UNet. *Multimedia Tools and Applications, 82*(2), 3101-3122.
26. Vladimir, P. (2020),'Handling imbalanced dataset in image classification', Analytics Vidhya. Available at: Handling imbalanced dataset in image classification | by Privalov Vladimir | Analytics Vidhya | Medium
27. Duan, H., Wei, Y., Liu, P. and Yin, H. (2020), 'A Novel Ensemble Framewok Based on K-Means and Resampling for Imbalanced Data', *Applied Sciences, 10*, pp. 1-16.
28. Pahren, L., Thomas, P., Jia, X., & Lee, J. (2022). A novel method in intelligent synthetic data creation for machine learning-based manufacturing quality control. *IFAC-PapersOnLine, 55*(19), 73-78.
29. Han, H., Wang, W. Y., & Mao, B. H. (2005, August). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (pp. 878-887). Berlin, Heidelberg: Springer Berlin Heidelberg.
30. He, H., Bai, Y., Garcia, E. A., & Li, S. (2008, June). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (pp. 1322-1328). Ieee.
31. Gowda, K., & Krishna, G. (1979). The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (corresp.). *IEEE Transactions on Information Theory, 25*(4), 488-490.
32. Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics, (3)*, 408-421.
33. Sisters, Li. (2020),'Instance Hardness Threshold: An Undersampling Method to Tackle Imbalanced Classification Problems', *Towards Data Science*.
34. Madhukar, B. (2020),'Using Near-Miss Algorithm For Imbalanced Datasets', *Developers Corner.*
35. Viadinugroho, R. A.A. (2021),'Imbalanced Classification in Python: SMOTE-Tomek Links Method', *TowardsData Science.*
36. Agustianto, K., & Destarianto, P. (2019, October). Imbalance data handling using neighborhood cleaning rule (NCL) sampling method for precision student modeling. In *2019 International conference on computer science, Information Technology, and Electrical Engineering (ICOMITEE)* (pp. 86-89). IEEE.
37. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Morgan Kaufman Publishing*.
38. Ringnér, M. (2008). What is principal component analysis?. *Nature biotechnology, 26*(3), 303-304.
39. Brems, M. (2017). A one-stop shop for principal component analysis. *Medium towards Data Science, 17.*
40. Chollet, F., et al. (2015). Keras. GitHub Repository.
41. Ranathunga, S. (2020), 'Fabric defect dataset', Kaggle.
42. Pathirana, P. (2020), 'Fabric stain dataset', Kaggle.
43. Silvestre-Blanes, J., Albero-Albero, T., Miralles, I., Pérez-Llorens, R. and Moreno, J. (2019), ' Aitex Fabric Image Database', Aitex.
44. Luke, J., Joseph, R., & Balaji, M. (2019). Impact of image size on accuracy and generalization of convolutional neural networks. *Int. J. Res. Anal. Rev.(IJRAR), 6*(1), 70-80.
45. Thambawita, V., Strümke, I., Hicks, S. A., Halvorsen, P., Parasa, S., & Riegler, M. A. (2021). Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images. *Diagnostics, 11*(12), 2183.
46. Sabottke, C. F., & Spieler, B. M. (2020). The effect of image resolution on deep learning in radiography. *Radiology: Artificial Intelligence, 2*(1), e190015.
47. Rukundo, O. (2021), 'Effects of image size on Deep Learning', *Arxiv*, pp.19.
48. Andalib, A. S., Salekin, A., Islam, M. R., & Abdulla-Al-Shami, M. (2012, December). Novel approaches for detecting fabric fault using artificial neural network with k-fold validation. In *2012 15th International Conference on Computer and Information Technology (ICCIT)* (pp. 55-60). IEEE.
49. Atwya, M., & Panoutsos, G. (2019). Transient thermography for flaw detection in friction stir welding: a machine learning approach. *IEEE Transactions on Industrial Informatics, 16*(7), 4423-4435.
50. Bovik, A. C., Clark, M., & Geisler, W. S. (1990). Multichannel texture analysis using localized spatial filters. *IEEE transactions on pattern analysis and machine intelligence, 12*(1), 55-73.
51. Campbell, J. G., Fraley, C., Stanford, D., Murtagh, F., & Raftery, A. E. (1999). Model-based methods for textile fault detection. *International Journal of Imaging Systems and Technology, 10*(4), 339-346.
52. Chakraborty, S. (2021). *Automatic Printed Fabric Defect Detection Using a Convolutional Neural Network.* North Carolina State University.
53. Chetverikov, D., & Hanbury, A. (2002). Finding defects

in texture using regularity and local orientation. *Pattern Recognition, 35*(10), 2165-2180.
54. Conci, A., & Proença, C. B. (1998). A fractal image analysis system for fabric inspection based on a box-counting method. *Computer Networks and ISDN Systems, 30*(20-21), 1887-1895.
55. Imbalanced Learn (n.d.a), 'SVMSMOTE'.
56. Imbalanced Learn (n.d.b),'ClusterCentroids.
57. Imbalanced Learn (n.d.c),'EditedNearestNeighbor'.
58. Imbalanced Learn (n.d. d), 'RepeatedEditedNearestNeighbors'.
59. Imbalanced Learn (n.d.e),'AIIKNN'.
60. Imbalanced Learn (n.d.,f), 'Under-sampling'.
61. Imbalanced Learn (n.d.h)', Combination of Over- and Under-Sampling'.
62. Imbalanced Learn (n.d.i)', 'RandomOverSampler'.
63. kmeans-smote.readthedocs.io (no date),'kmeans_smote module'.
64. Lin, G., Liu, K., Xia, X. and Tan, R. (2023). 'An Efficient and Intelligent Detection Method for Fabric Defects based on Improved YOLOv5', *Sensors, 23* (1), pp. 1-16.
65. Microsoft. (n.d.). Visual Studio Code - Code Editing. Redefined.
66. TensorFlow. (n.d.).
67. Zhao, S., Yin, L., Zhang, J., Wang, J., & Zhong, R. (2020). Real-time fabric defect detection based on multi-scale convolutional neural network. *IET Collaborative Intelligent Manufacturing, 2*(4), 189-196.

Appendices

Appendix A- Link to Data and Experiments

https://drive.google.com/drive/folders/1paR6OtnxIoPmi-hnHLoAaq1HitH7Xypu?usp=sharing

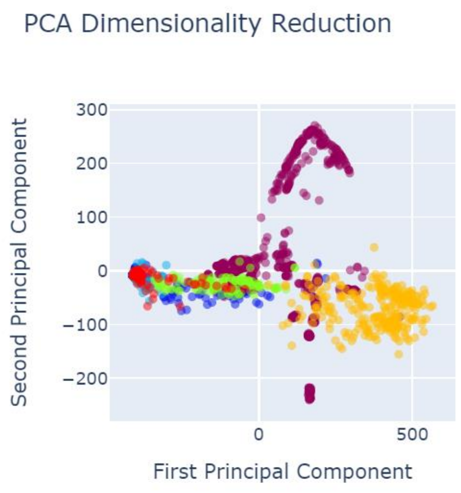
Appendix B-Figures

Appendix C- Tables

Appendix D- Sampling Techniques (PCA, Confusion Matrix)

Undersampling Techniques

AIKNN



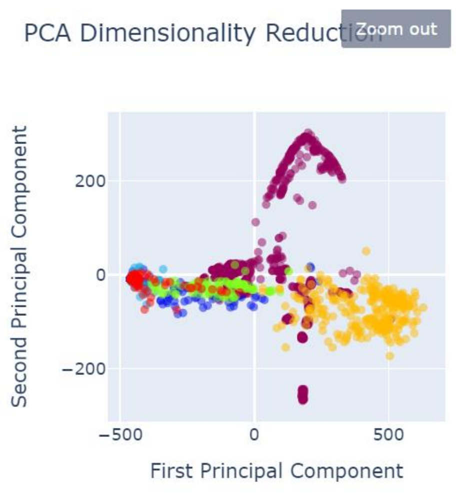
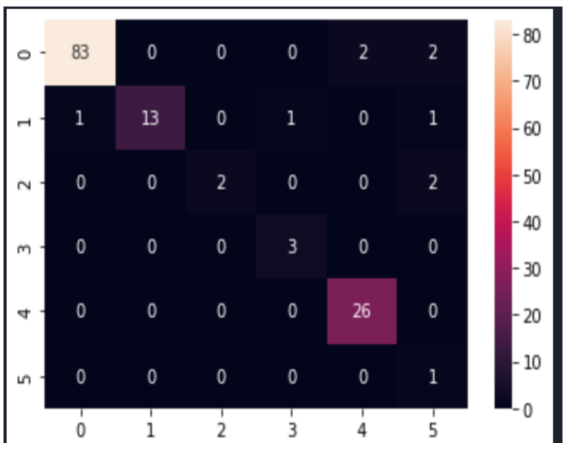


Image size 245x345

Image size 150x700



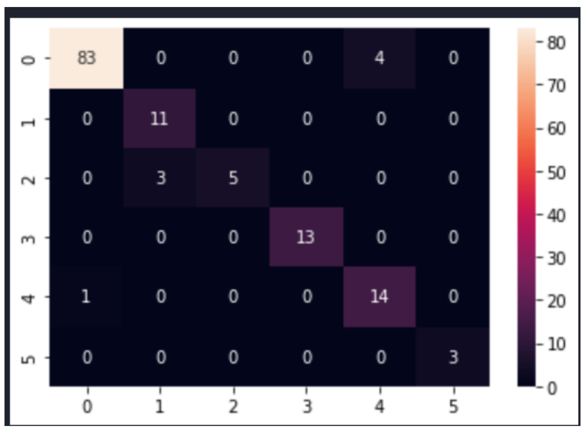
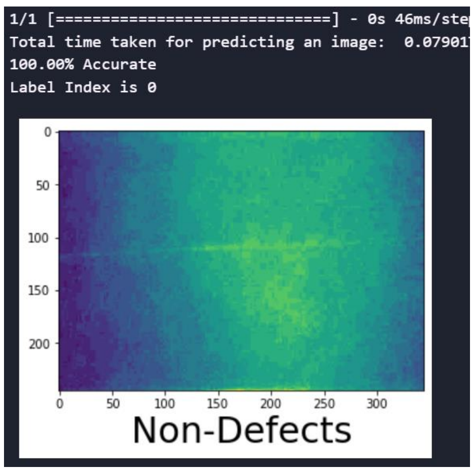
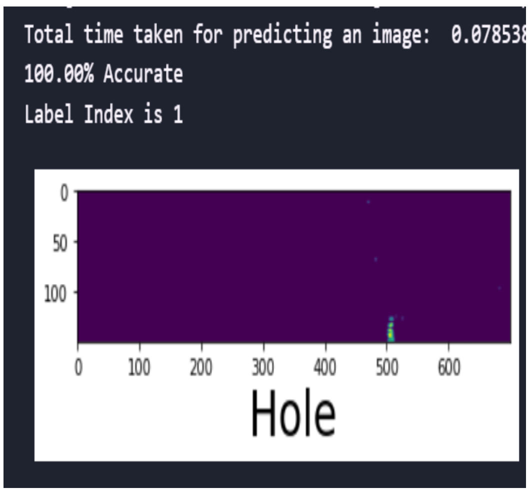


Image size 245x345

Image size 150x700



```
1/1 [==============================] - 0s 46ms/ste
Total time taken for predicting an image:  0.07901
100.00% Accurate
Label Index is 0
```

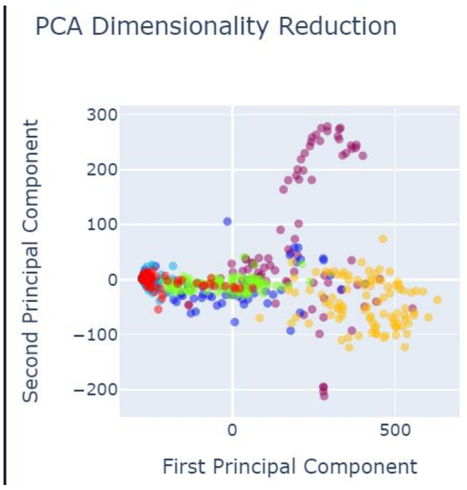


```
Total time taken for predicting an image:  0.078538
100.00% Accurate
Label Index is 1
```

Image size 245x345

Image size 150x700

Cluster Centroids



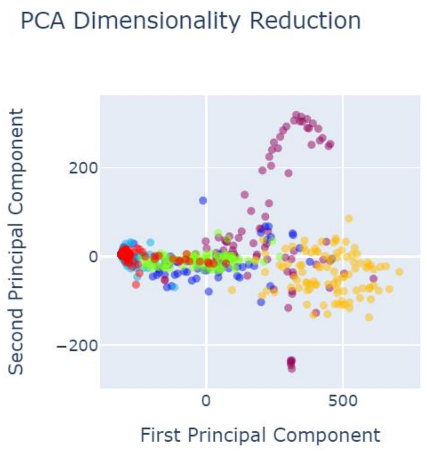
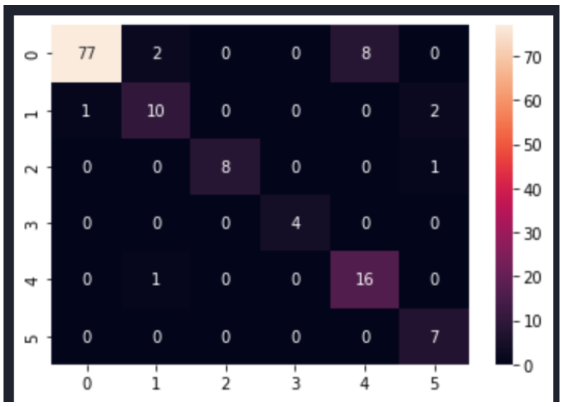


Image size 245x345

Image size 150x700



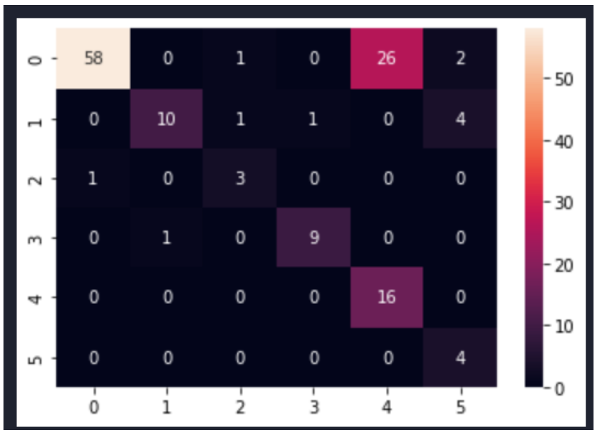
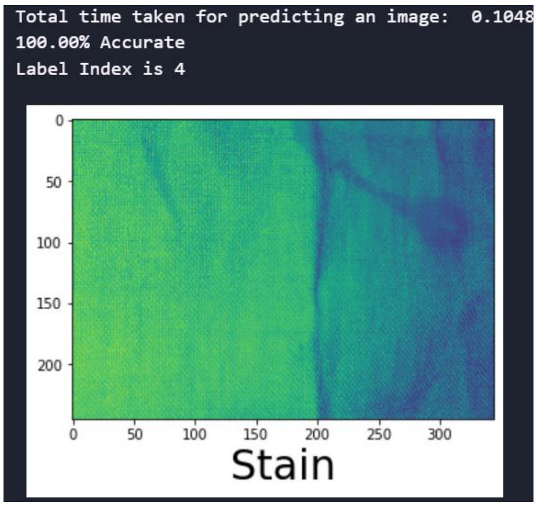
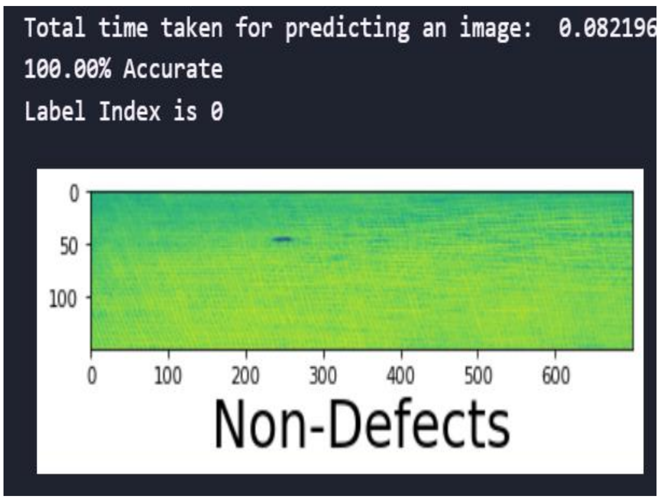


Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.1048
100.00% Accurate
Label Index is 4
```

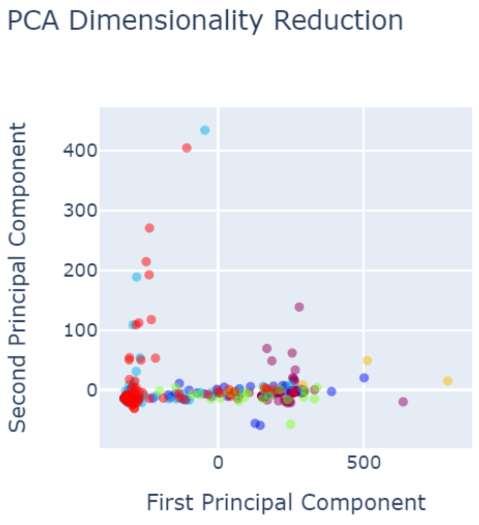


```
Total time taken for predicting an image:  0.082196
100.00% Accurate
Label Index is 0
```

Image size 245x345

Image size 150x700

Condensed Nearest Neighbour



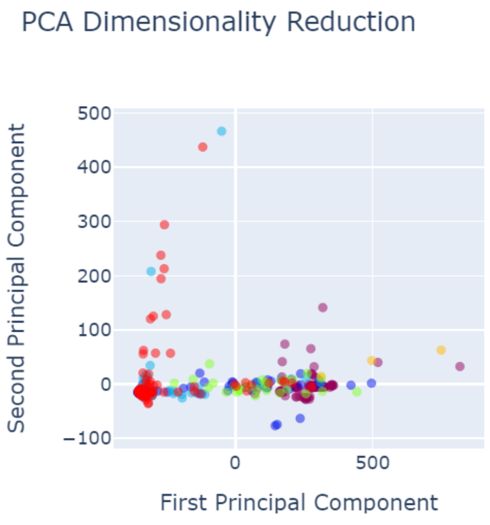
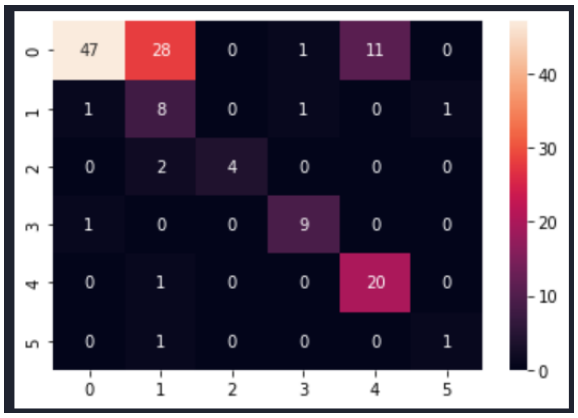
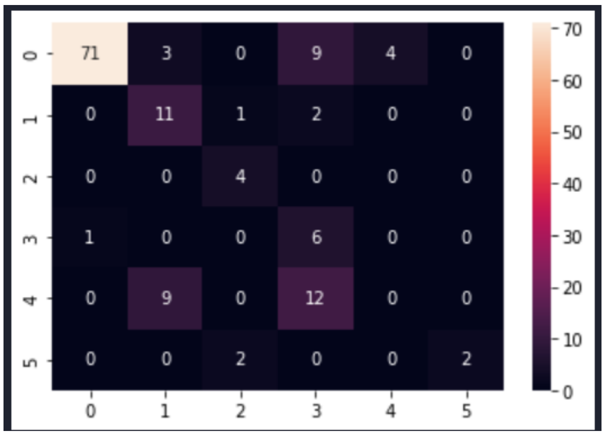


Image size 245x345

Image size 150x700



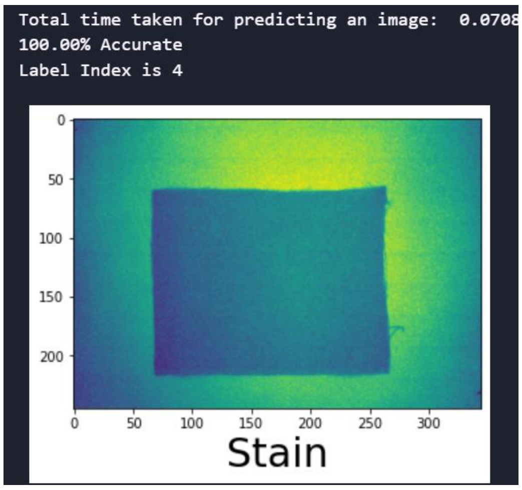
	0	1	2	3	4	5
0	47	28	0	1	11	0
1	1	8	0	1	0	1
2	0	2	4	0	0	0
3	1	0	0	9	0	0
4	0	1	0	0	20	0
5	0	1	0	0	0	1



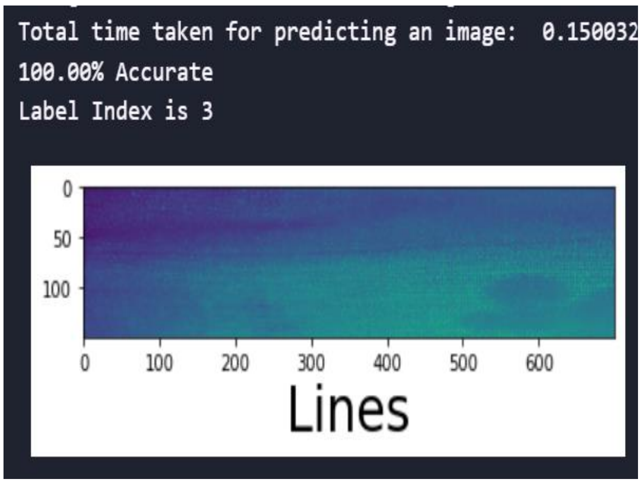
	0	1	2	3	4	5
0	71	3	0	9	4	0
1	0	11	1	2	0	0
2	0	0	4	0	0	0
3	1	0	0	6	0	0
4	0	9	0	12	0	0
5	0	0	2	0	0	2

Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.0708
100.00% Accurate
Label Index is 4
```

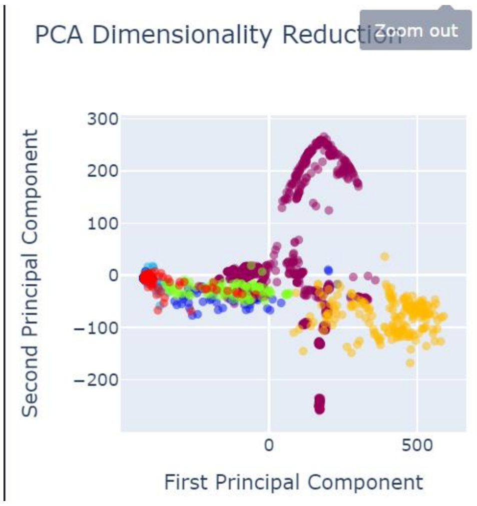


```
Total time taken for predicting an image:  0.150032
100.00% Accurate
Label Index is 3
```

Image size 245x345

Image size 150x700

Edited Nearest Neighbour



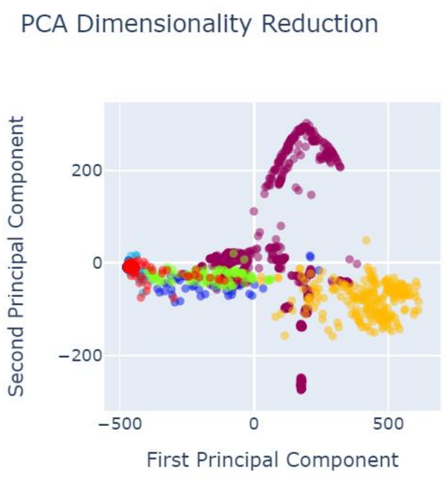
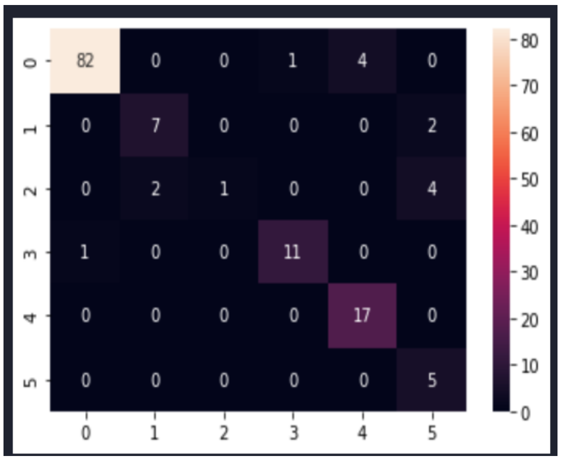


Image size 245x345 Image size 150x700



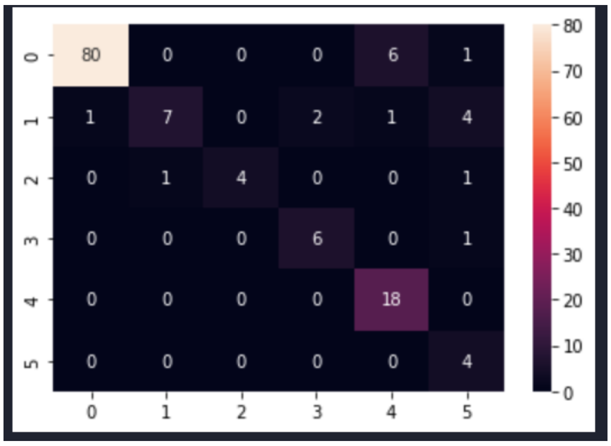
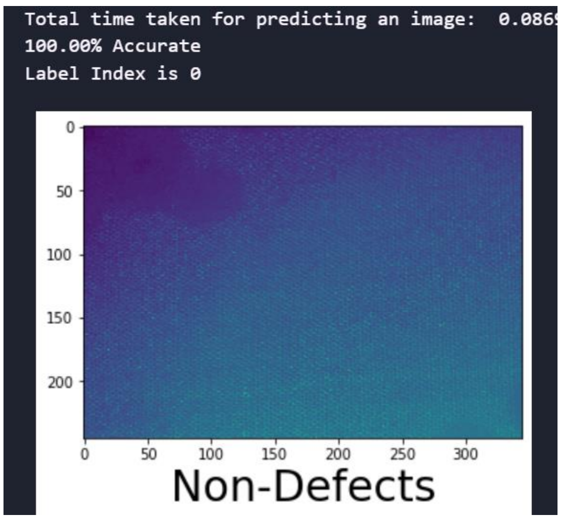
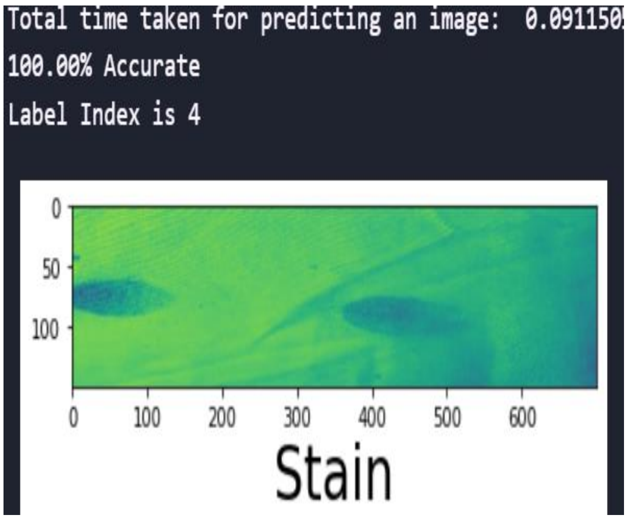


Image size 245x345 Image size 150x700



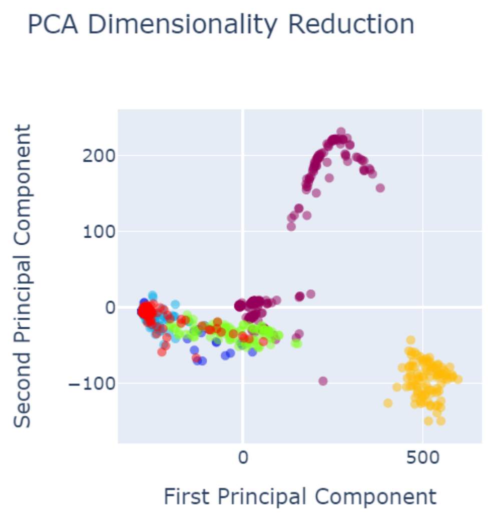
```
Total time taken for predicting an image:  0.086
100.00% Accurate
Label Index is 0
```



```
Total time taken for predicting an image:  0.091150
100.00% Accurate
Label Index is 4
```

Image size 245x345 Image size 150x700

Instance Hardness Threshold



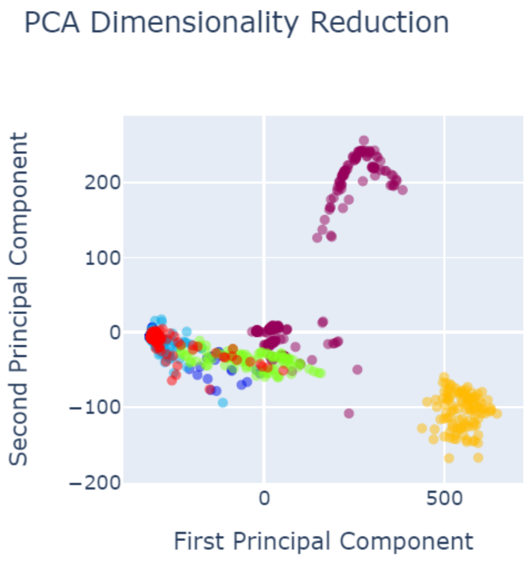
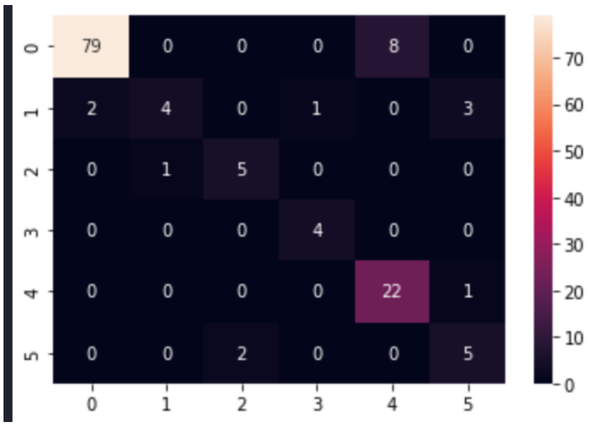
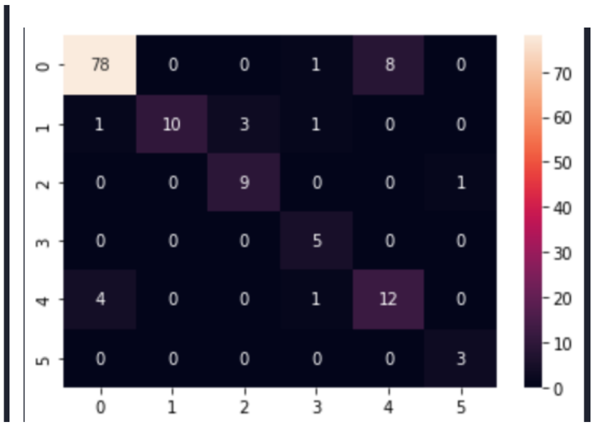


Image size 245x345

Image size 150x700



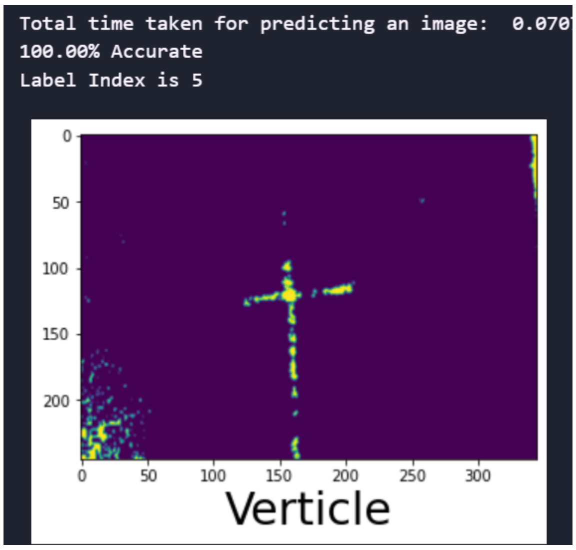
	0	1	2	3	4	5
0	79	0	0	0	8	0
1	2	4	0	1	0	3
2	0	1	5	0	0	0
3	0	0	0	4	0	0
4	0	0	0	0	22	1
5	0	0	2	0	0	5



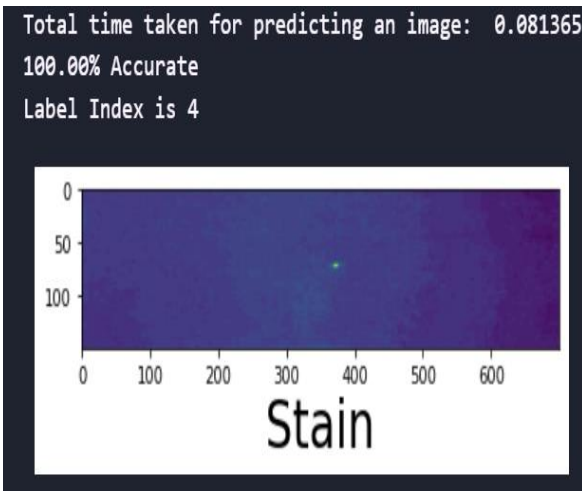
	0	1	2	3	4	5
0	78	0	0	1	8	0
1	1	10	3	1	0	0
2	0	0	9	0	0	1
3	0	0	0	5	0	0
4	4	0	0	1	12	0
5	0	0	0	0	0	3

Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.070
100.00% Accurate
Label Index is 5
```

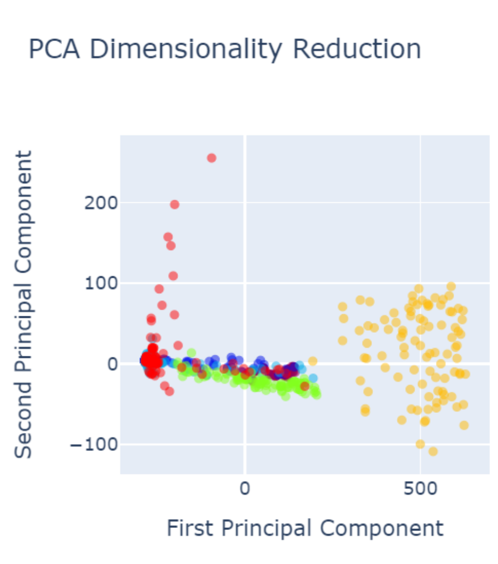


```
Total time taken for predicting an image:  0.081365
100.00% Accurate
Label Index is 4
```

Image size 245x345

Image size 150x700

NearMiss



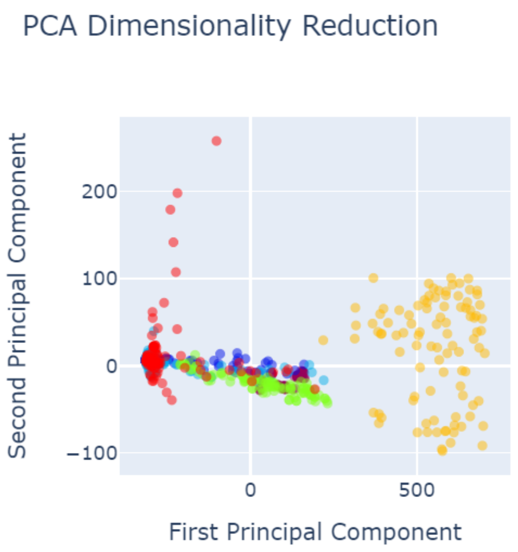
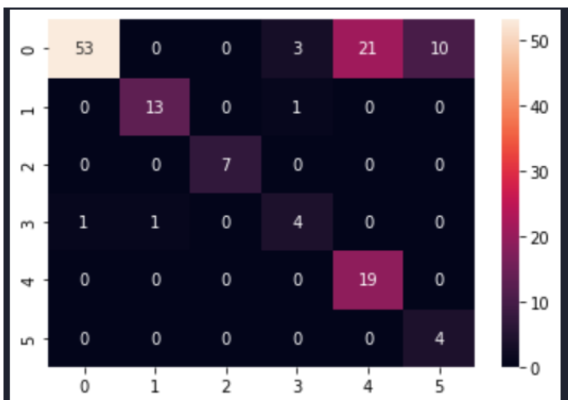


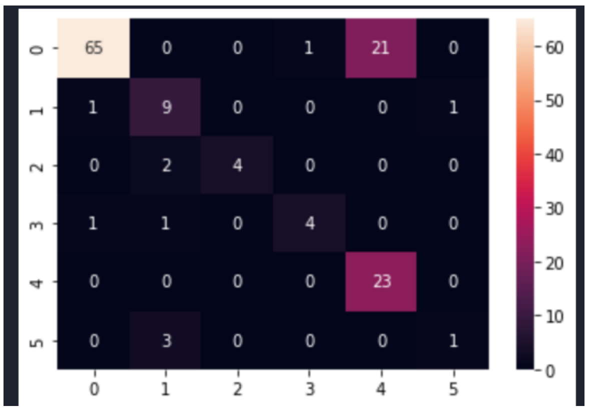
Image size 245x345

Image size 150x700



	0	1	2	3	4	5
0	53	0	0	3	21	10
1	0	13	0	1	0	0
2	0	0	7	0	0	0
3	1	1	0	4	0	0
4	0	0	0	0	19	0
5	0	0	0	0	0	4

50
40
30
20
10
0

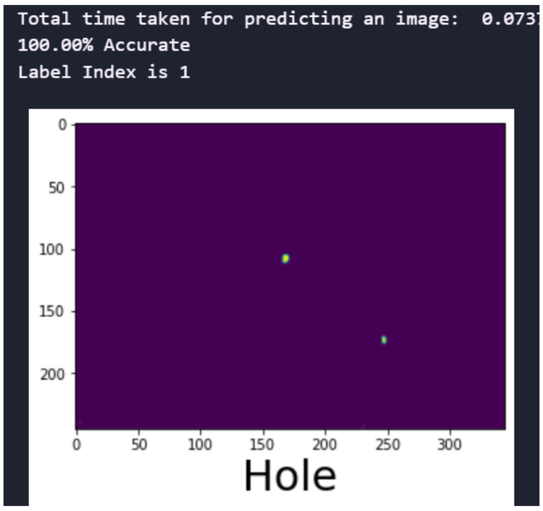


	0	1	2	3	4	5
0	65	0	0	1	21	0
1	1	9	0	0	0	1
2	0	2	4	0	0	0
3	1	1	0	4	0	0
4	0	0	0	0	23	0
5	0	3	0	0	0	1

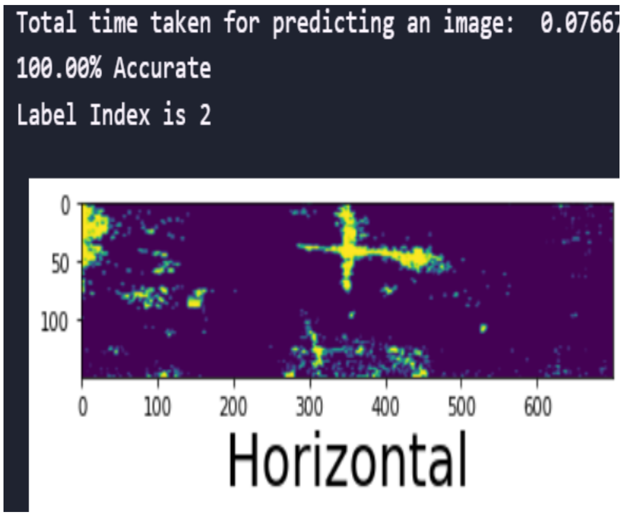
60
50
40
30
20
10
0

Image size 245x345

Image size 150x700



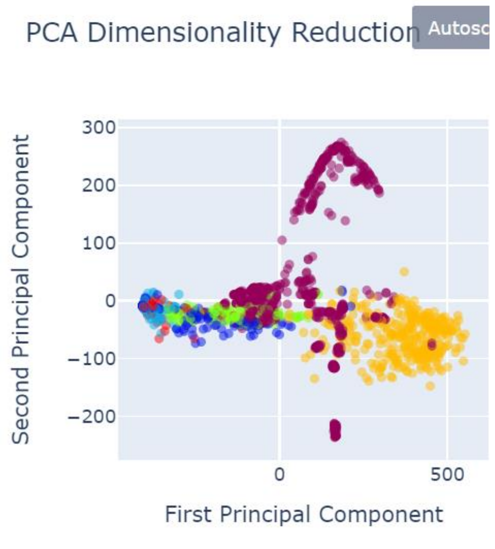
```
Total time taken for predicting an image:  0.073
100.00% Accurate
Label Index is 1
```

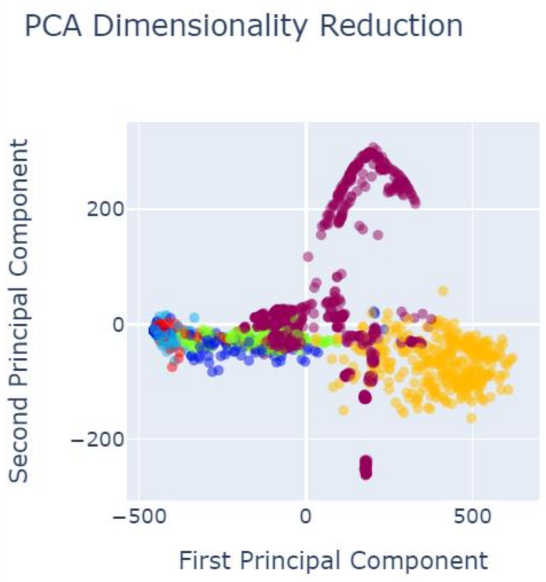


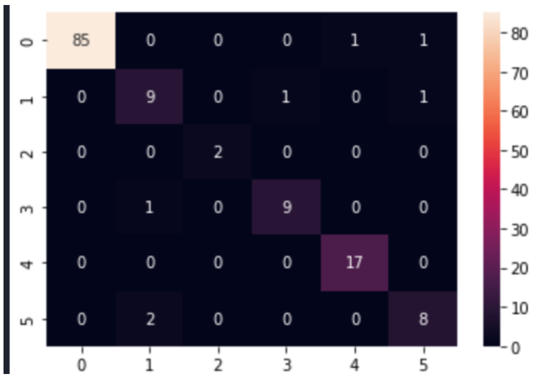
```
Total time taken for predicting an image:  0.0766
100.00% Accurate
Label Index is 2
```

Image size 245x345 Image size 150x700

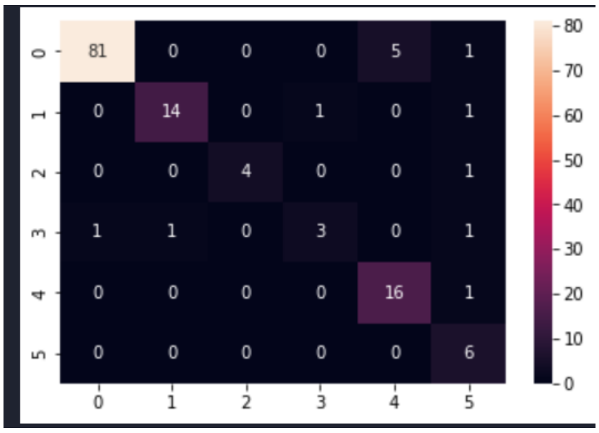
Neighbourhood Cleaning Rule





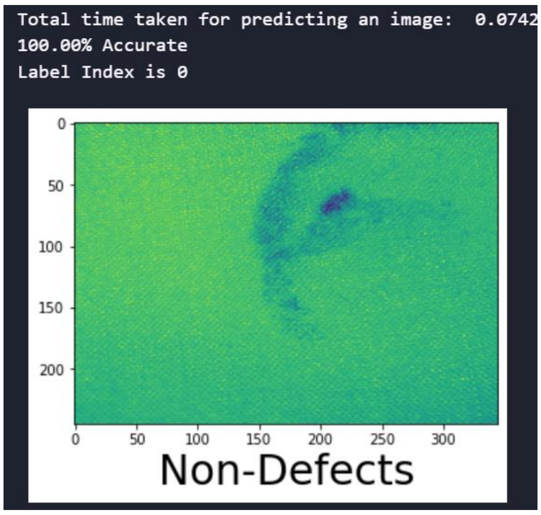


	0	1	2	3	4	5
0	85	0	0	0	1	1
1	0	9	0	1	0	1
2	0	0	2	0	0	0
3	0	1	0	9	0	0
4	0	0	0	0	17	0
5	0	2	0	0	0	8

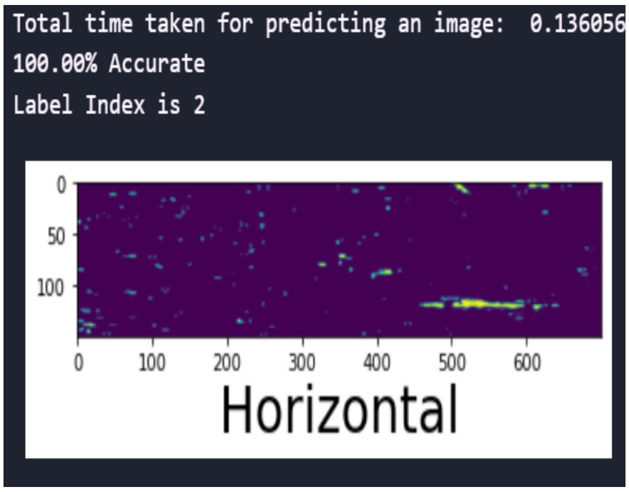


	0	1	2	3	4	5
0	81	0	0	0	5	1
1	0	14	0	1	0	1
2	0	0	4	0	0	1
3	1	1	0	3	0	1
4	0	0	0	0	16	1
5	0	0	0	0	0	6

Image size 245x345 Image size 150x700

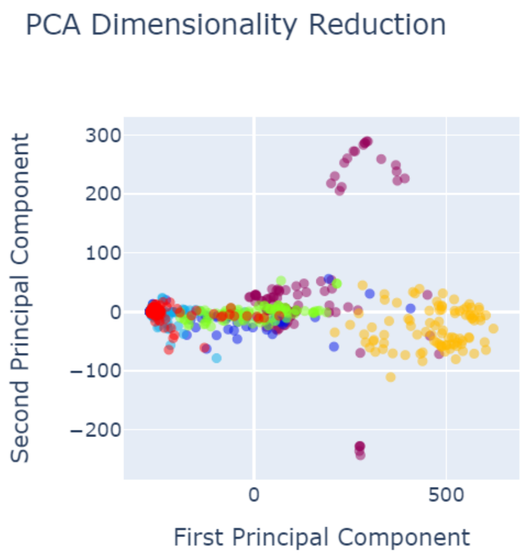


```
Total time taken for predicting an image:  0.0742
100.00% Accurate
Label Index is 0
```



```
Total time taken for predicting an image:  0.136056
100.00% Accurate
Label Index is 2
```

Random Undersampler



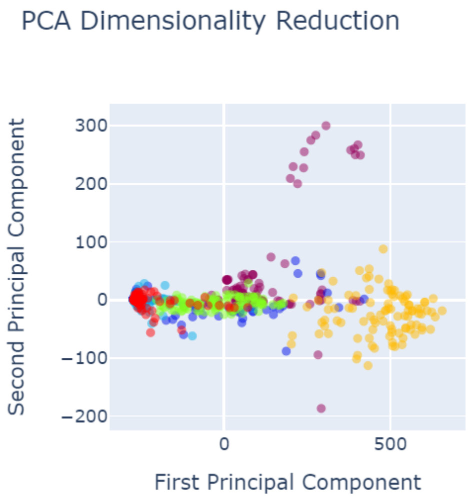
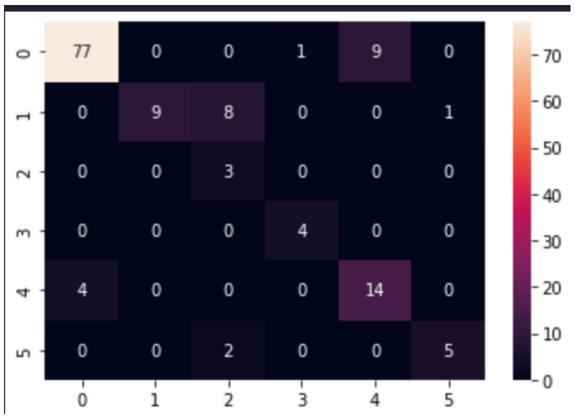
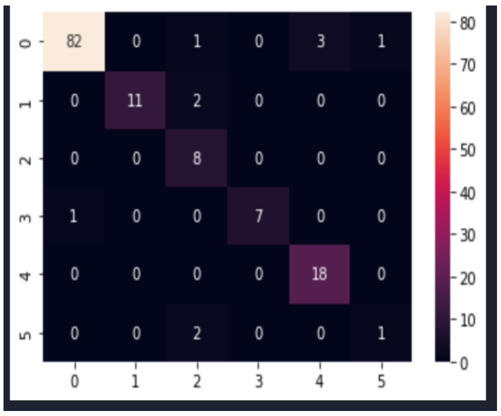


Image size 245x345

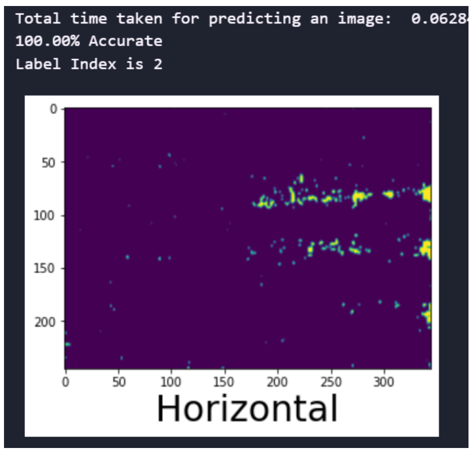
Image size 150x700



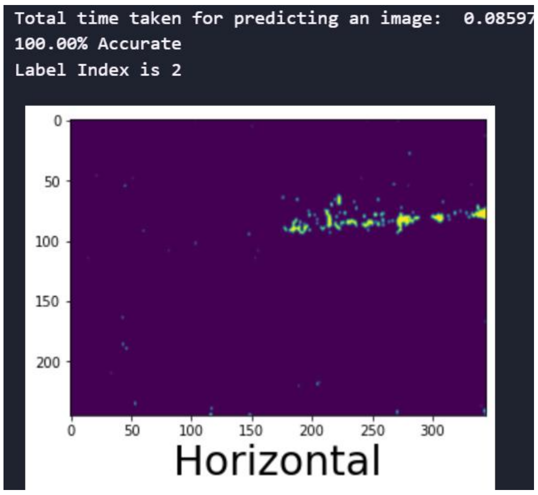
	0	1	2	3	4	5
0	77	0	0	1	9	0
1	0	9	8	0	0	1
2	0	0	3	0	0	0
3	0	0	0	4	0	0
4	4	0	0	0	14	0
5	0	0	2	0	0	5



	0	1	2	3	4	5
0	82	0	1	0	3	1
1	0	11	2	0	0	0
2	0	0	8	0	0	0
3	1	0	0	7	0	0
4	0	0	0	0	18	0
5	0	0	2	0	0	1



```
Total time taken for predicting an image:  0.0628
100.00% Accurate
Label Index is 2
```

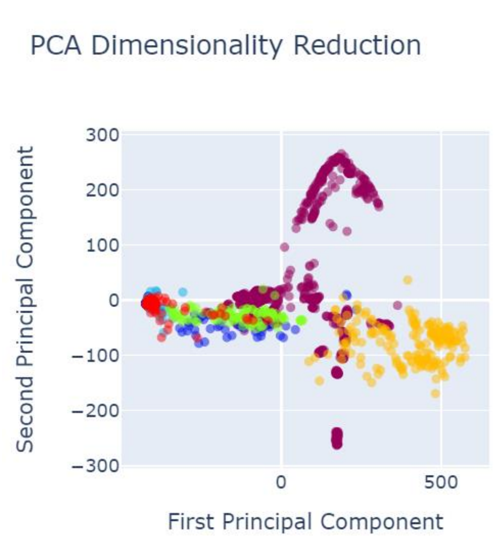


```
Total time taken for predicting an image:  0.08597
100.00% Accurate
Label Index is 2
```

Image size 245x345

Image size 150x700

Repeated Edited Nearest Neighbour



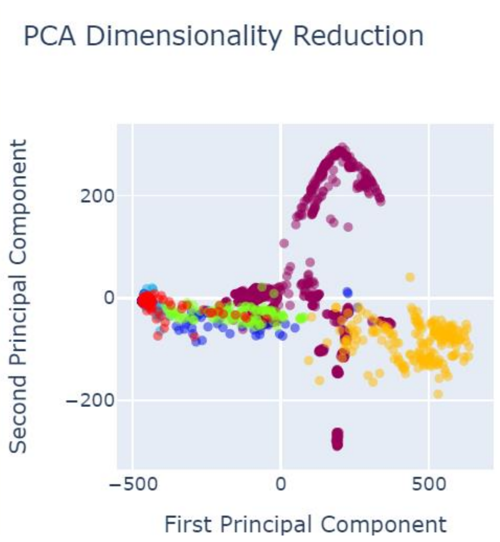
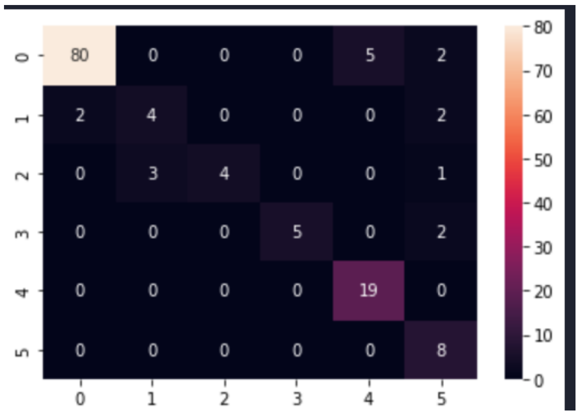


Image size 245x345

Image size 150x700



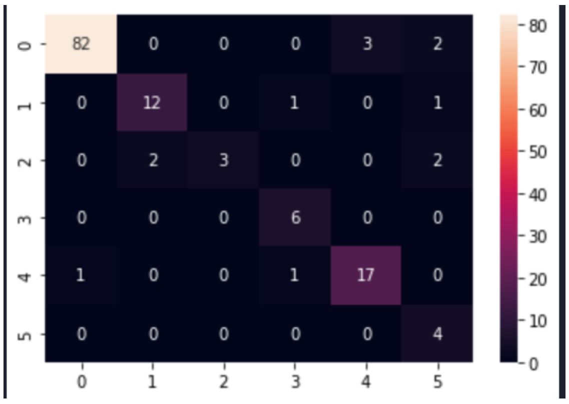
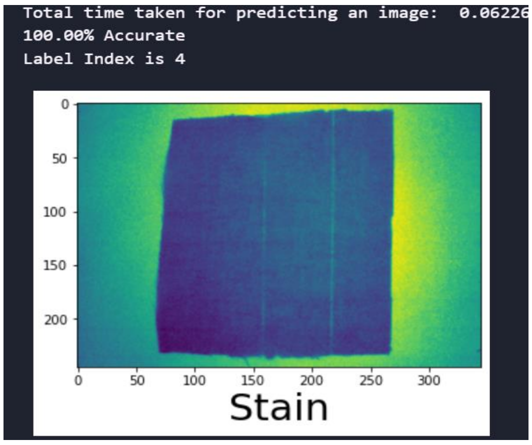
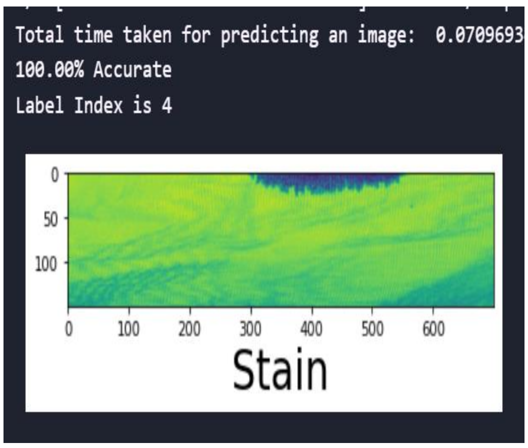


Image size 245x345

Image size 150x700



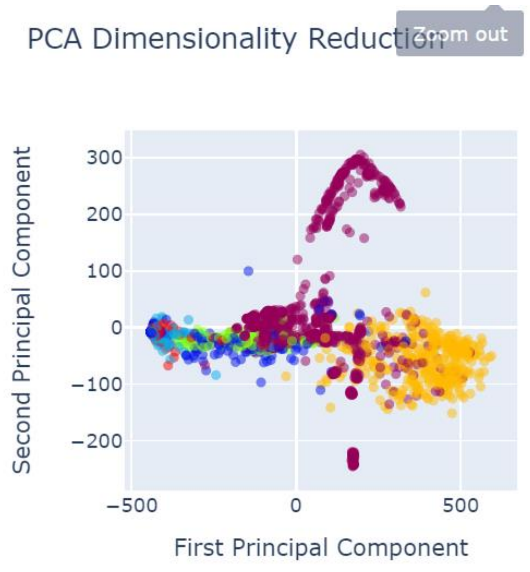
```
Total time taken for predicting an image:  0.06226
100.00% Accurate
Label Index is 4
```



```
Total time taken for predicting an image:  0.0709693
100.00% Accurate
Label Index is 4
```

Image size 245x345 Image size 150x700

Tomeklinks



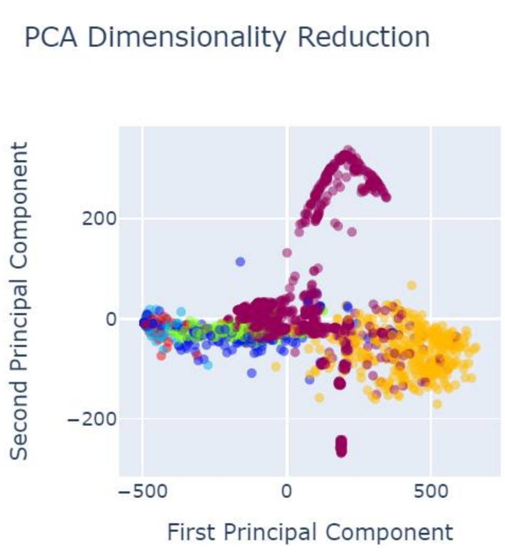
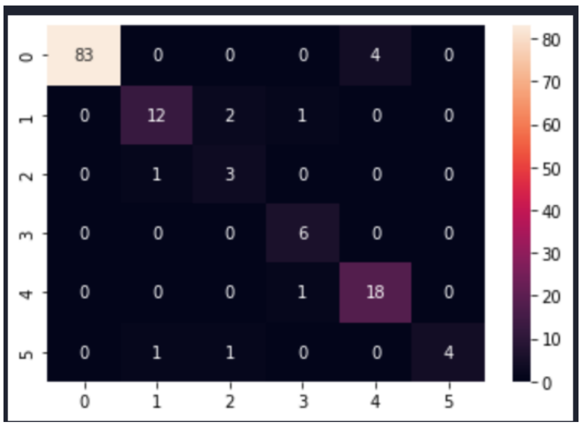
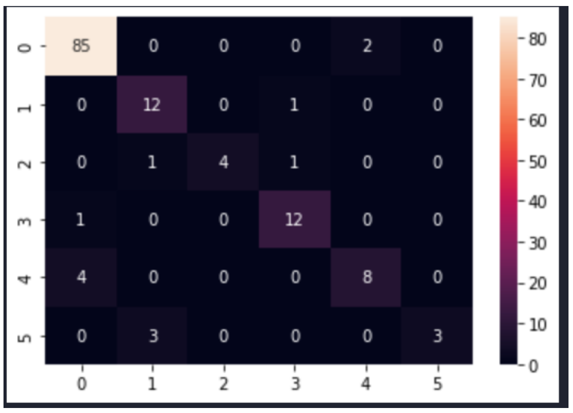


Image size 245x345 Image size 150x700

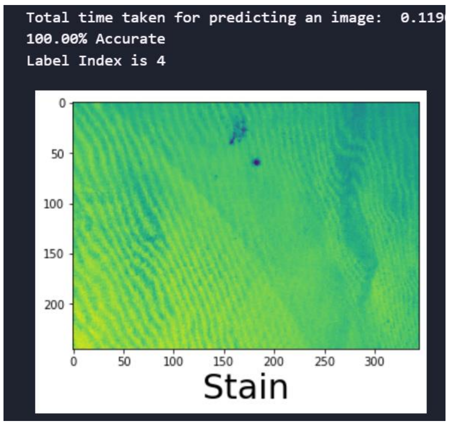


	0	1	2	3	4	5
0	83	0	0	0	4	0
1	0	12	2	1	0	0
2	0	1	3	0	0	0
3	0	0	0	6	0	0
4	0	0	0	1	18	0
5	0	1	1	0	0	4

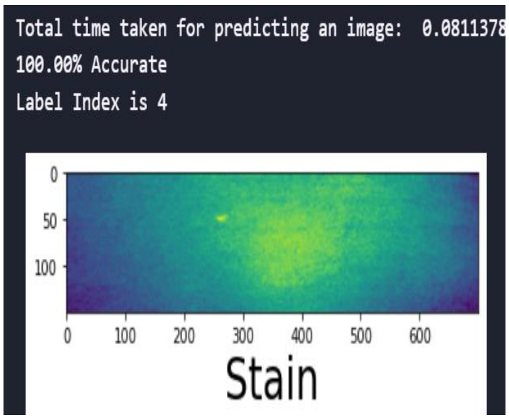


	0	1	2	3	4	5
0	85	0	0	0	2	0
1	0	12	0	1	0	0
2	0	1	4	1	0	0
3	1	0	0	12	0	0
4	4	0	0	0	8	0
5	0	3	0	0	0	3

Image size 245x345 Image size 150x700



```
Total time taken for predicting an image:  0.119
100.00% Accurate
Label Index is 4
```

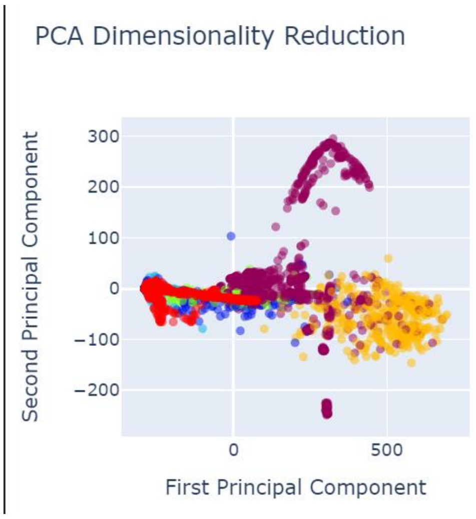


```
Total time taken for predicting an image:  0.0811378
100.00% Accurate
Label Index is 4
```

Image size 245x345 Image size 150x700

Oversampling

ADASYN



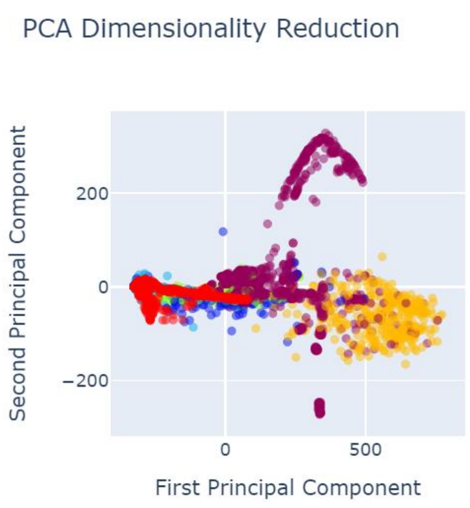
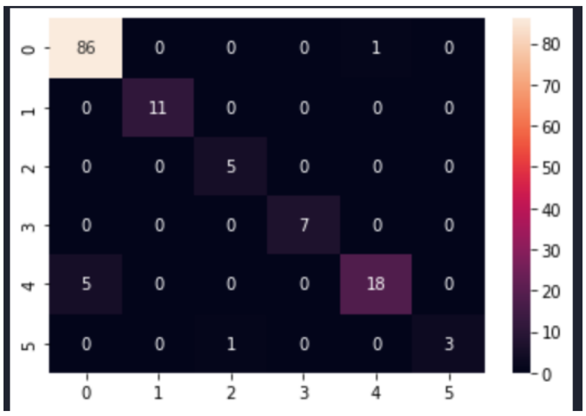
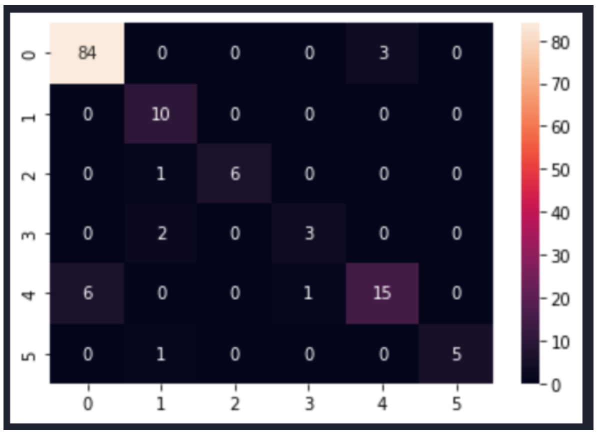


Image size 245x345 Image size 150x700

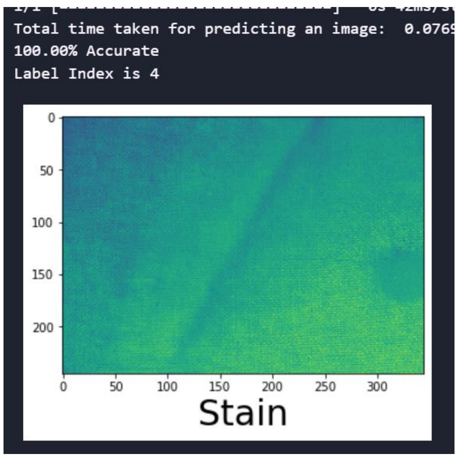


	0	1	2	3	4	5
0	86	0	0	0	1	0
1	0	11	0	0	0	0
2	0	0	5	0	0	0
3	0	0	0	7	0	0
4	5	0	0	0	18	0
5	0	0	1	0	0	3

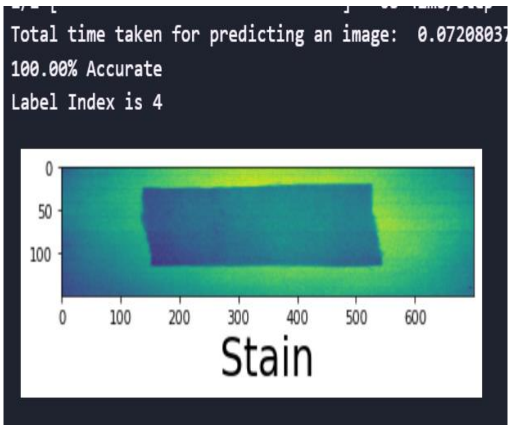


	0	1	2	3	4	5
0	84	0	0	0	3	0
1	0	10	0	0	0	0
2	0	1	6	0	0	0
3	0	2	0	3	0	0
4	6	0	0	1	15	0
5	0	1	0	0	0	5

Image size 245x345 Image size 150x700



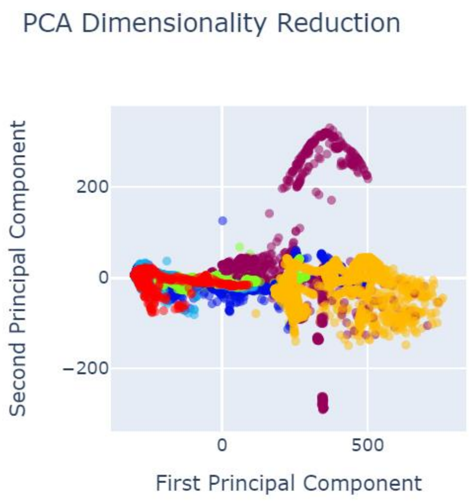
```
Total time taken for predicting an image:  0.0769
100.00% Accurate
Label Index is 4
```



```
Total time taken for predicting an image:  0.07208037
100.00% Accurate
Label Index is 4
```

Image size 245x345 Image size 150x700

Borderline SMOTE



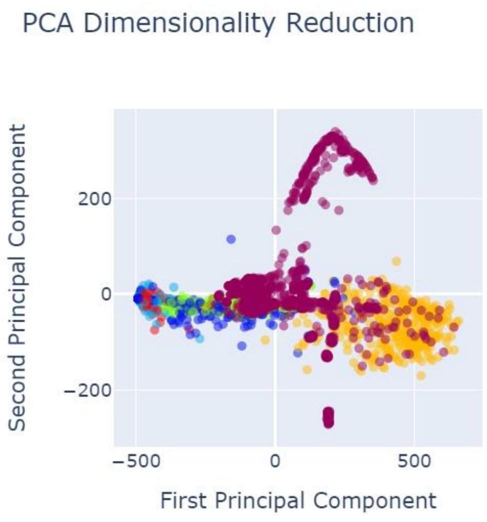
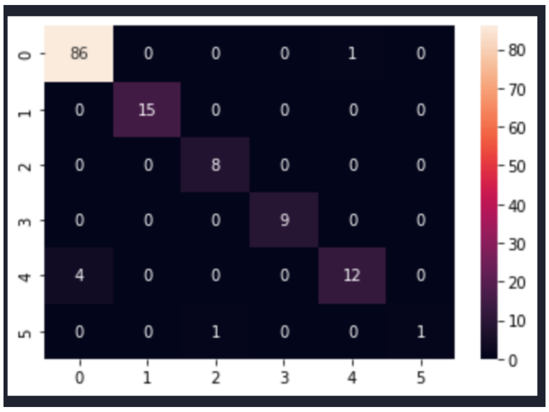
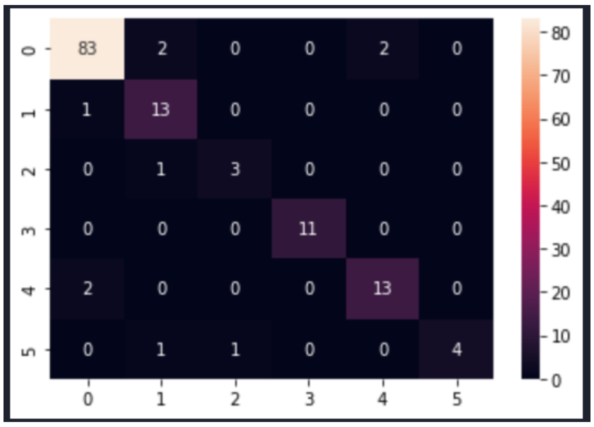


Image size 245x345 Image size 150x700

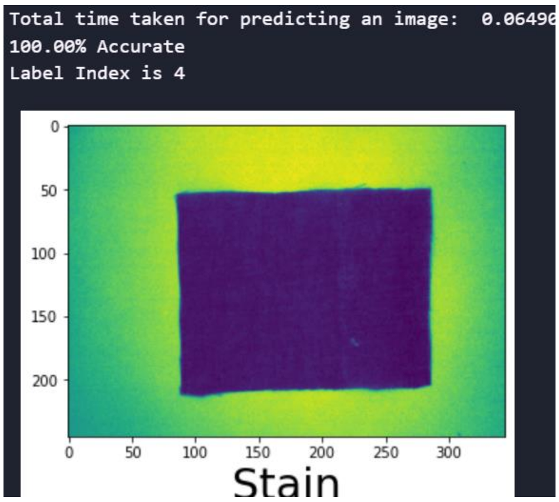


	0	1	2	3	4	5
0	86	0	0	0	1	0
1	0	15	0	0	0	0
2	0	0	8	0	0	0
3	0	0	0	9	0	0
4	4	0	0	0	12	0
5	0	0	1	0	0	1

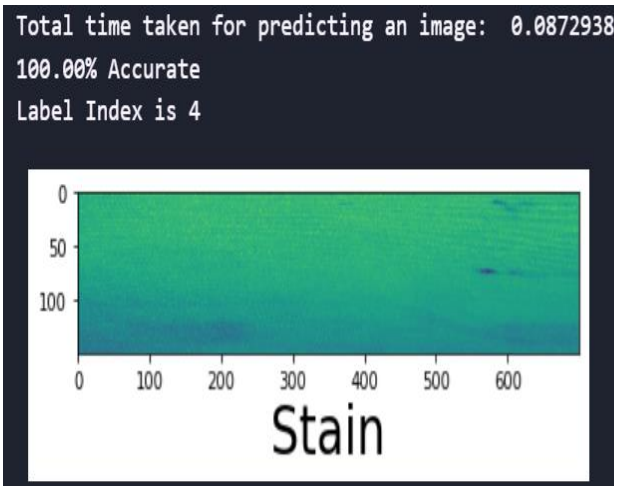


	0	1	2	3	4	5
0	83	2	0	0	2	0
1	1	13	0	0	0	0
2	0	1	3	0	0	0
3	0	0	0	11	0	0
4	2	0	0	0	13	0
5	0	1	1	0	0	4

Image size 245x345 Image size 150x700



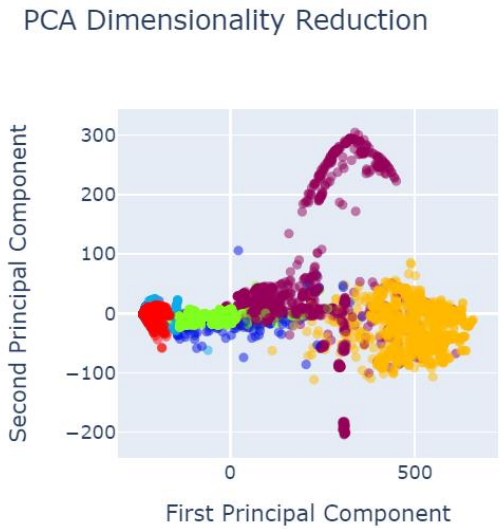
```
Total time taken for predicting an image:  0.06496
100.00% Accurate
Label Index is 4
```



```
Total time taken for predicting an image:  0.0872938
100.00% Accurate
Label Index is 4
```

Image size 245x345   Image size 150x700

kMeans SMOTE



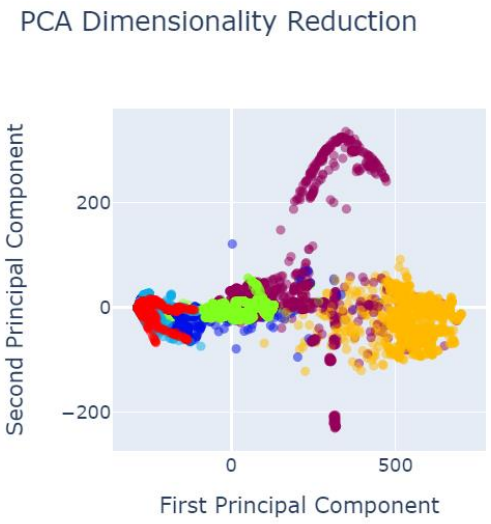
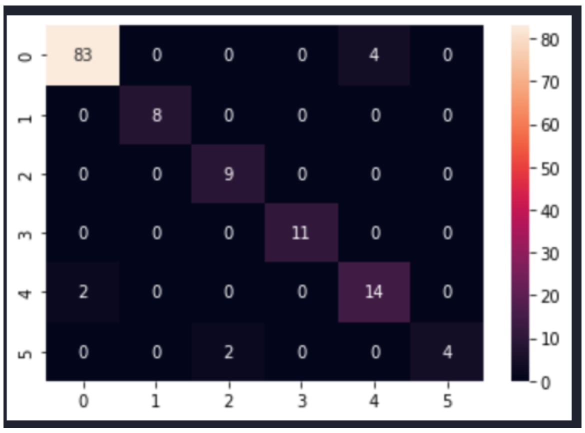
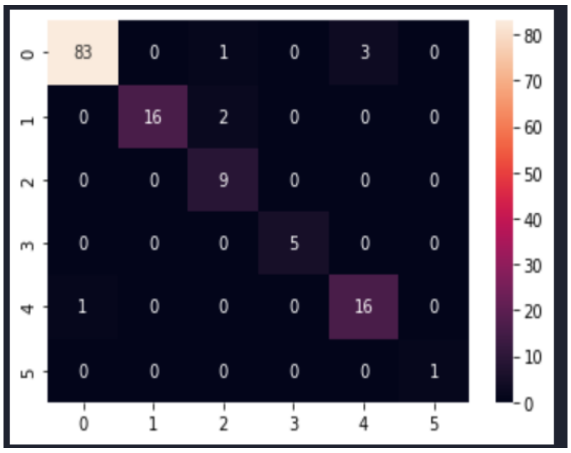


Image size 245x345   Image size 150x700

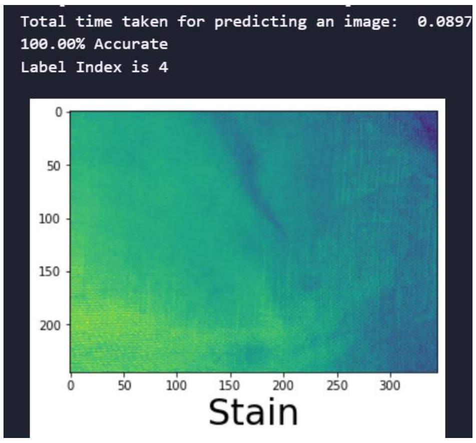


	0	1	2	3	4	5
0	83	0	0	0	4	0
1	0	8	0	0	0	0
2	0	0	9	0	0	0
3	0	0	0	11	0	0
4	2	0	0	0	14	0
5	0	0	2	0	0	4

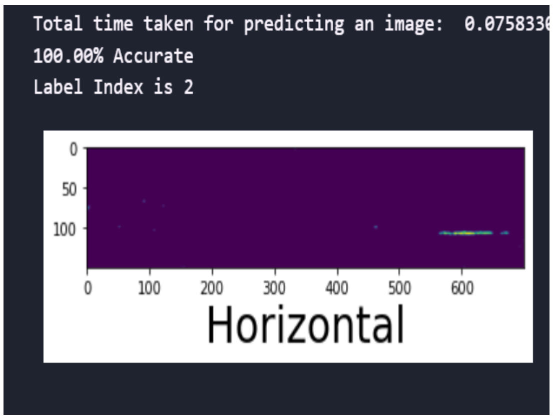


	0	1	2	3	4	5
0	83	0	1	0	3	0
1	0	16	2	0	0	0
2	0	0	9	0	0	0
3	0	0	0	5	0	0
4	1	0	0	0	16	0
5	0	0	0	0	0	1

Image size 245x345   Image size 150x700



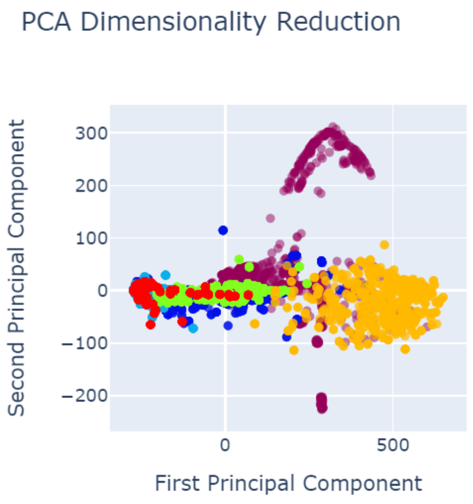
```
Total time taken for predicting an image:  0.0897
100.00% Accurate
Label Index is 4
```



```
Total time taken for predicting an image:  0.075833
100.00% Accurate
Label Index is 2
```

Image size 245x345 Image size 150x700

Random Oversampling



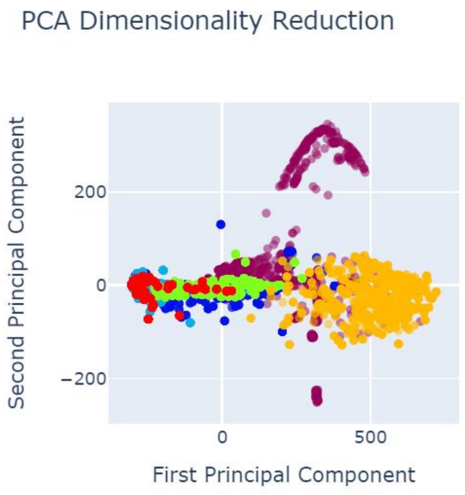
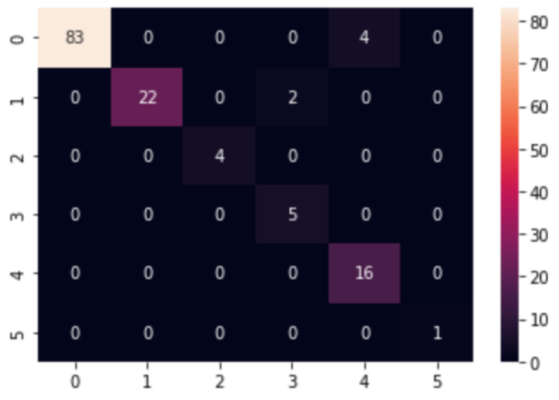


Image size 245x345 Image size 150x700



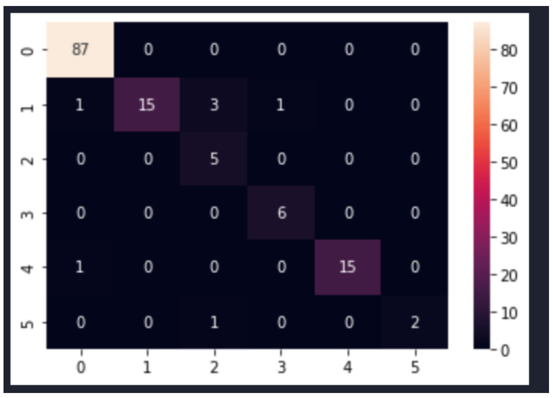
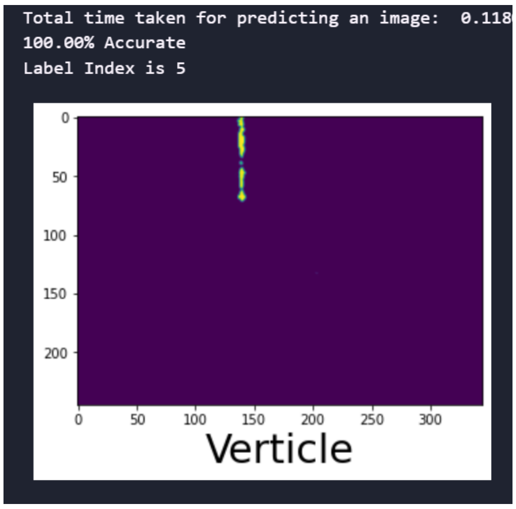
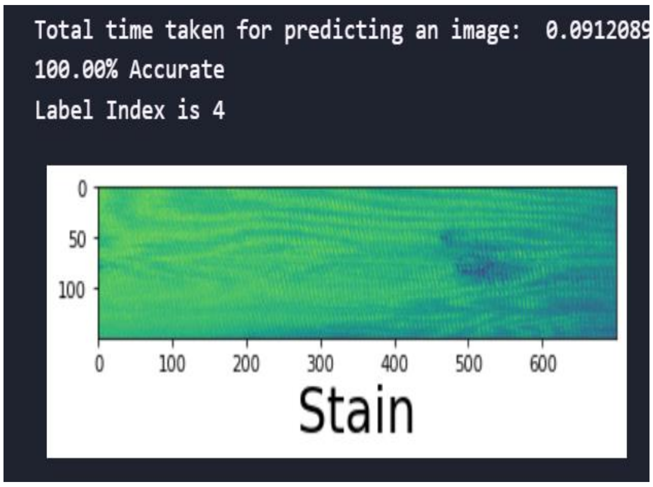


Image size 245x345 Image size 150x700



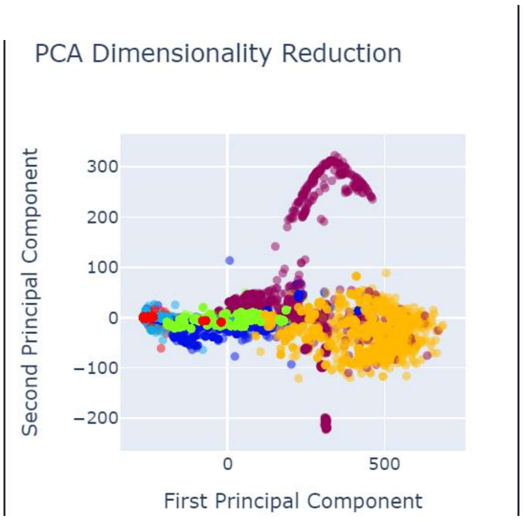
```
Total time taken for predicting an image:  0.118
100.00% Accurate
Label Index is 5
```



```
Total time taken for predicting an image:  0.0912089
100.00% Accurate
Label Index is 4
```

Image size 245x345 Image size 150x700

SMOTEN



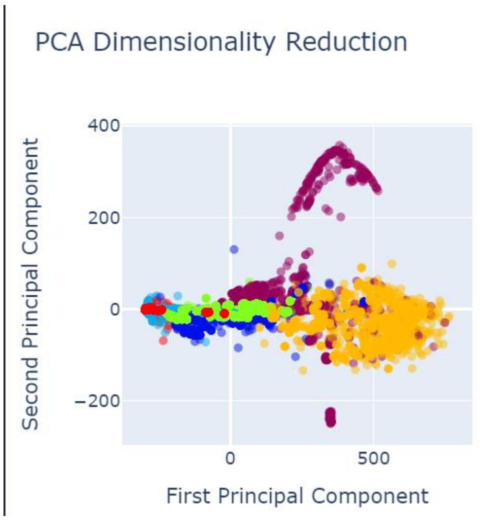
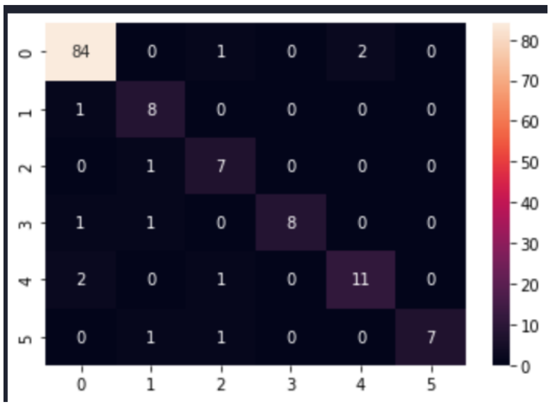
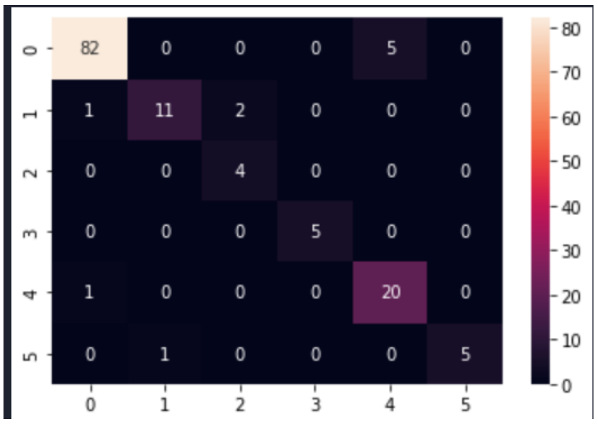


Image size 245x345 Image size 150x700

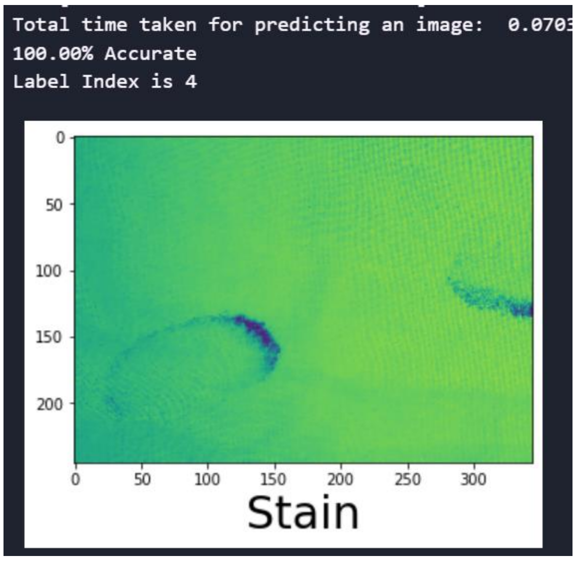


	0	1	2	3	4	5
0	84	0	1	0	2	0
1	1	8	0	0	0	0
2	0	1	7	0	0	0
3	1	1	0	8	0	0
4	2	0	1	0	11	0
5	0	1	1	0	0	7

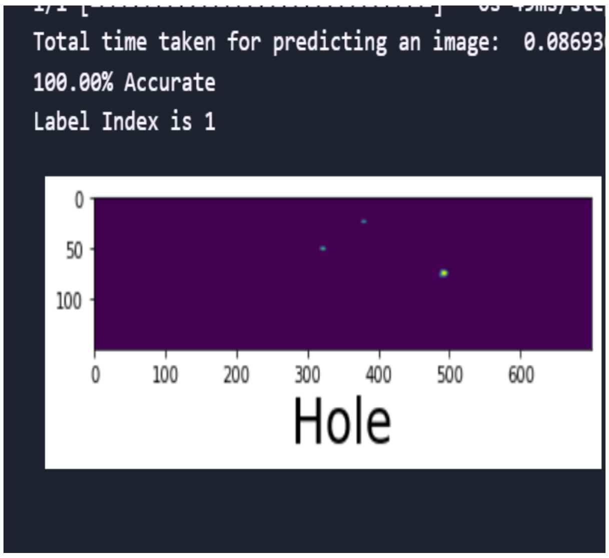


	0	1	2	3	4	5
0	82	0	0	0	5	0
1	1	11	2	0	0	0
2	0	0	4	0	0	0
3	0	0	0	5	0	0
4	1	0	0	0	20	0
5	0	1	0	0	0	5

Image size 245x345 Image size 150x700



```
Total time taken for predicting an image:  0.0703
100.00% Accurate
Label Index is 4
```

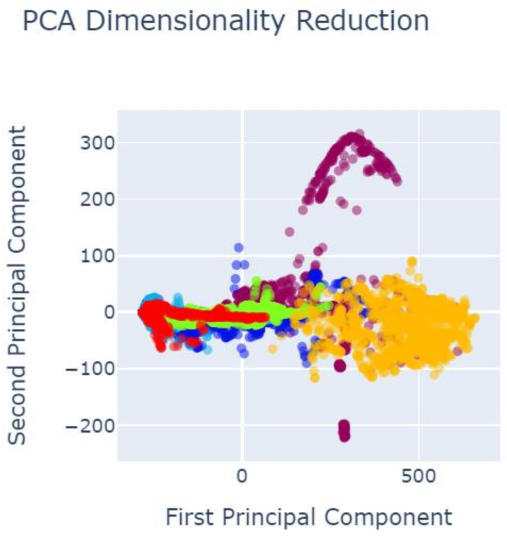


```
Total time taken for predicting an image:  0.08693
100.00% Accurate
Label Index is 1
```

Image size 245x345 Image size 150x700

Combined Oversampling and Undersampling

SMOTETomek



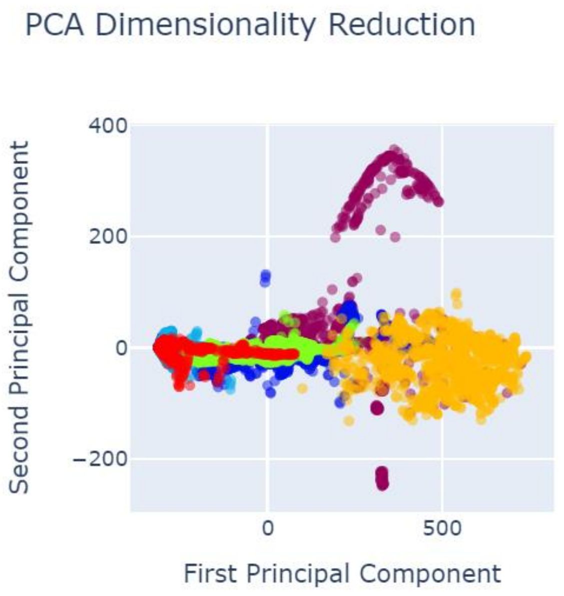
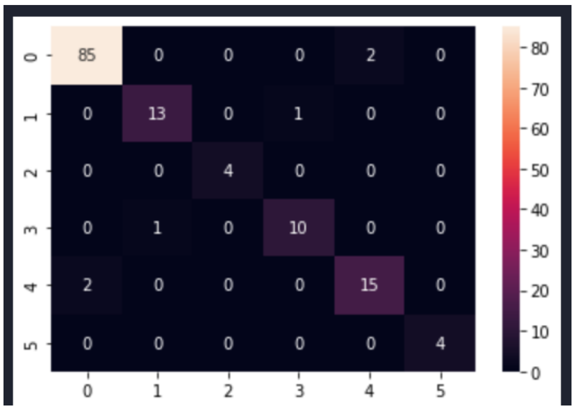
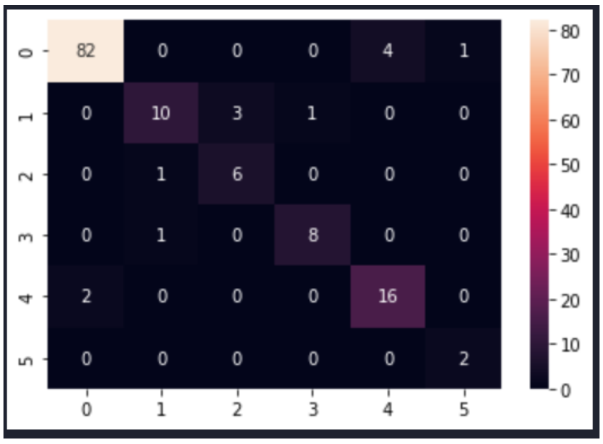


Image size 245x345 Image size 150x700



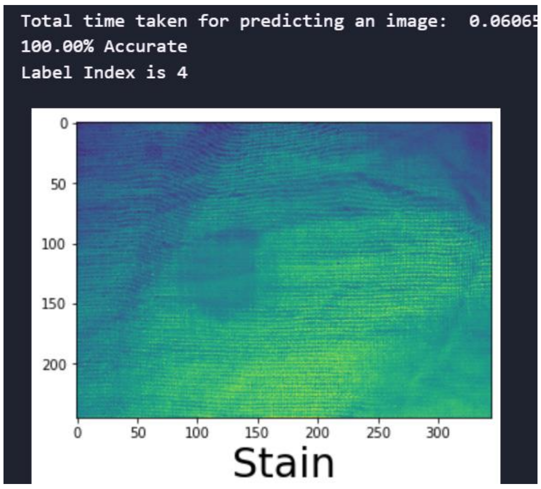
	0	1	2	3	4	5
0	85	0	0	0	2	0
1	0	13	0	1	0	0
2	0	0	4	0	0	0
3	0	1	0	10	0	0
4	2	0	0	0	15	0
5	0	0	0	0	0	4



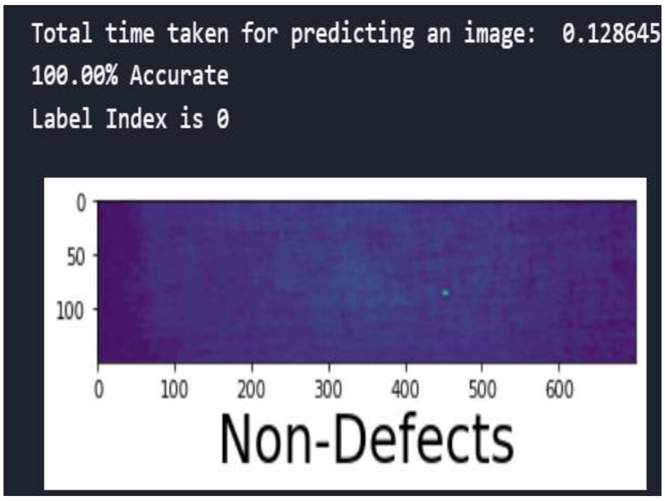
	0	1	2	3	4	5
0	82	0	0	0	4	1
1	0	10	3	1	0	0
2	0	1	6	0	0	0
3	0	1	0	8	0	0
4	2	0	0	0	16	0
5	0	0	0	0	0	2

Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.06065
100.00% Accurate
Label Index is 4
```



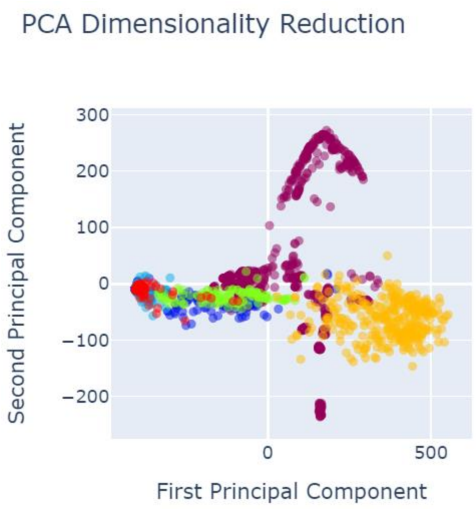
```
Total time taken for predicting an image:  0.128645
100.00% Accurate
Label Index is 0
```

Image size 245x345

Image size 150x700

Ensemble

Pipeline



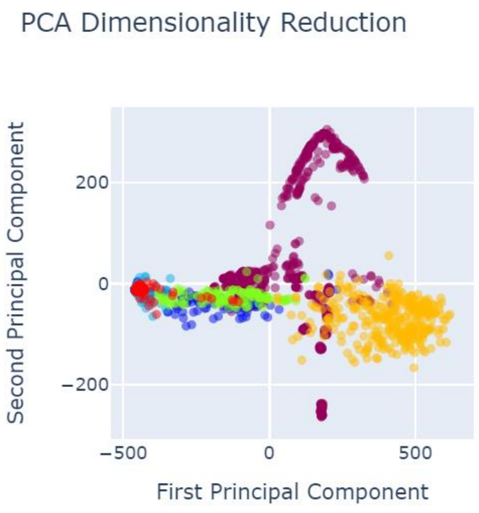
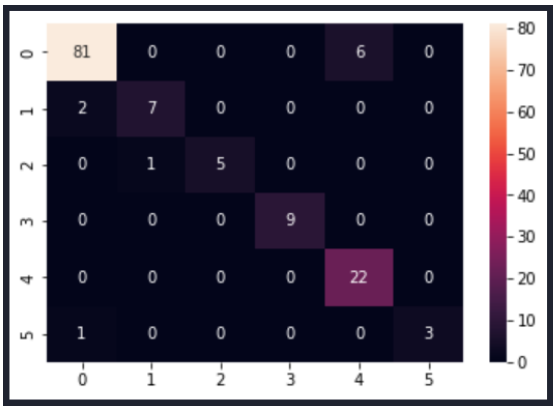
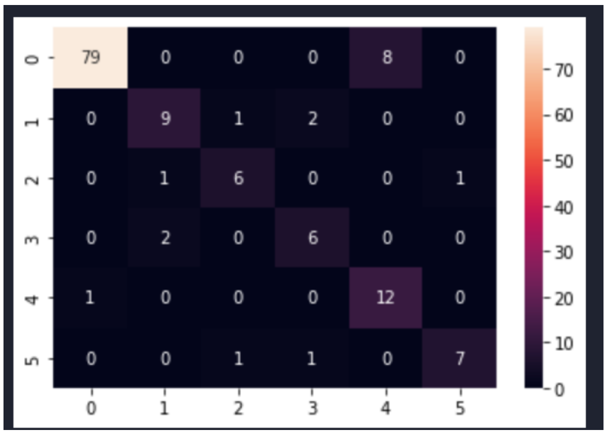


Image size 245x345

Image size 150x700



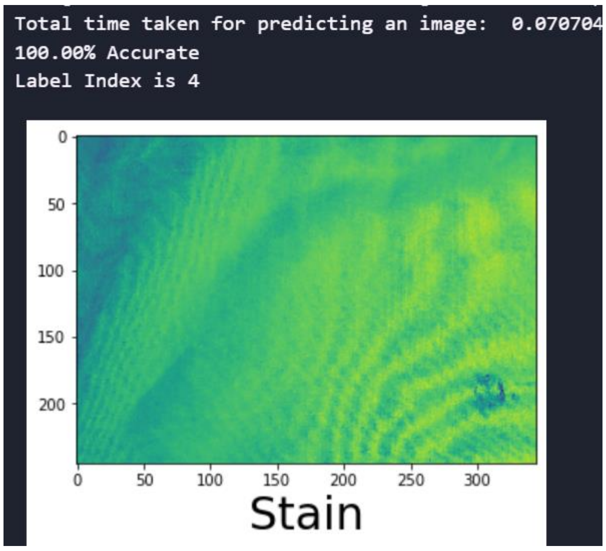
	0	1	2	3	4	5
0	81	0	0	0	6	0
1	2	7	0	0	0	0
2	0	1	5	0	0	0
3	0	0	0	9	0	0
4	0	0	0	0	22	0
5	1	0	0	0	0	3



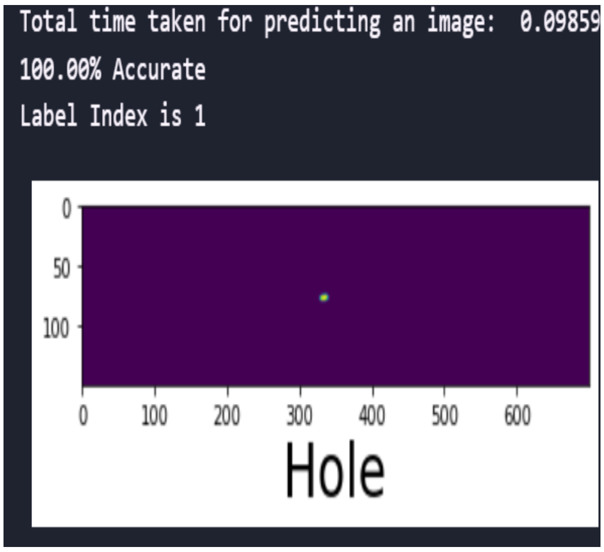
	0	1	2	3	4	5
0	79	0	0	0	8	0
1	0	9	1	2	0	0
2	0	1	6	0	0	1
3	0	2	0	6	0	0
4	1	0	0	0	12	0
5	0	0	1	1	0	7

Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.070704
100.00% Accurate
Label Index is 4
```



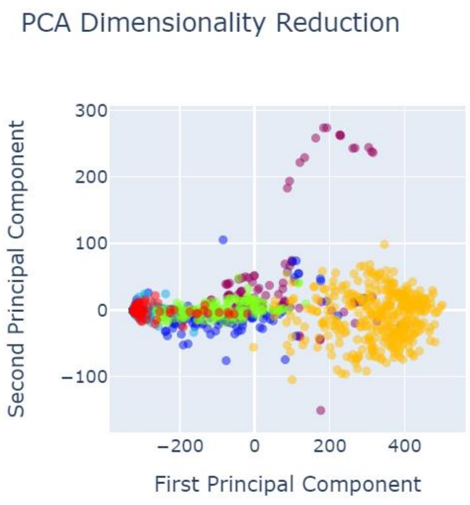
```
Total time taken for predicting an image:  0.09859
100.00% Accurate
Label Index is 1
```

Image size 245x345

Image size 150x700

Miscellaneous

Function Sampler



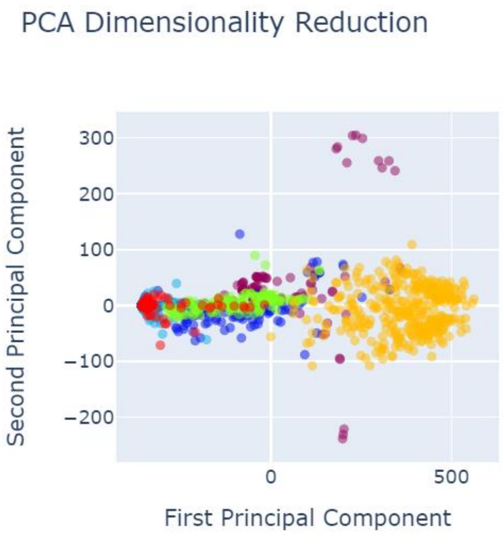
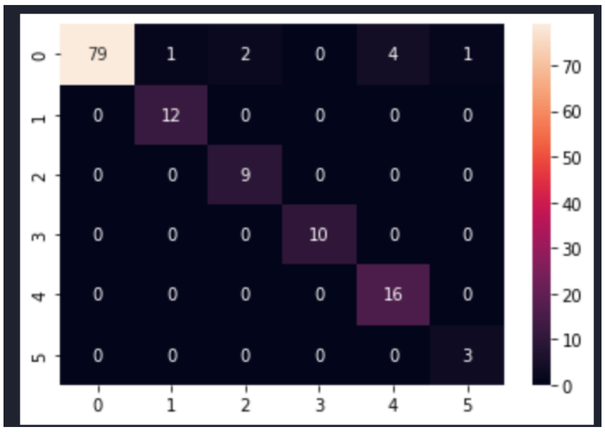


Image size 245x345

Image size 150x700



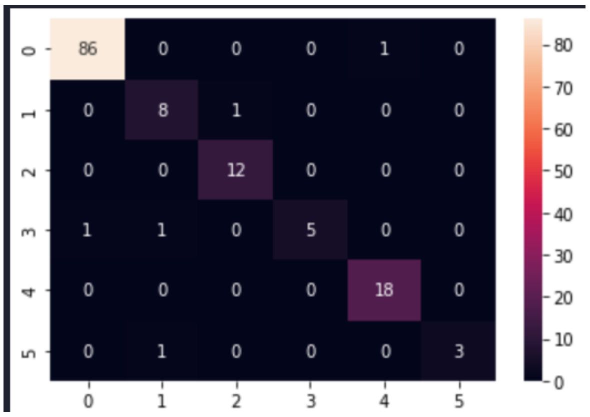
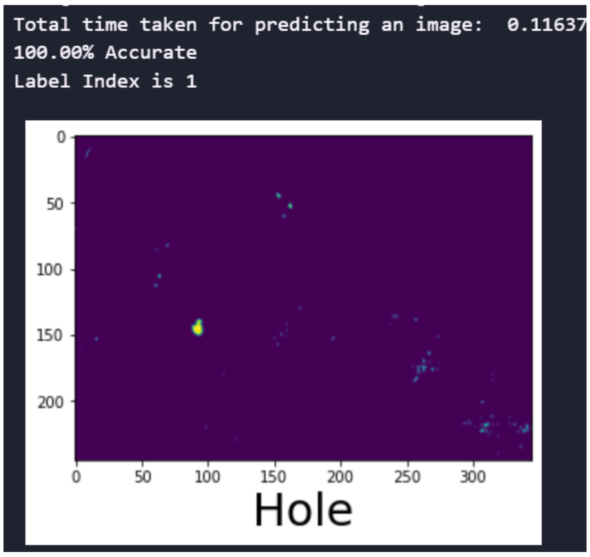
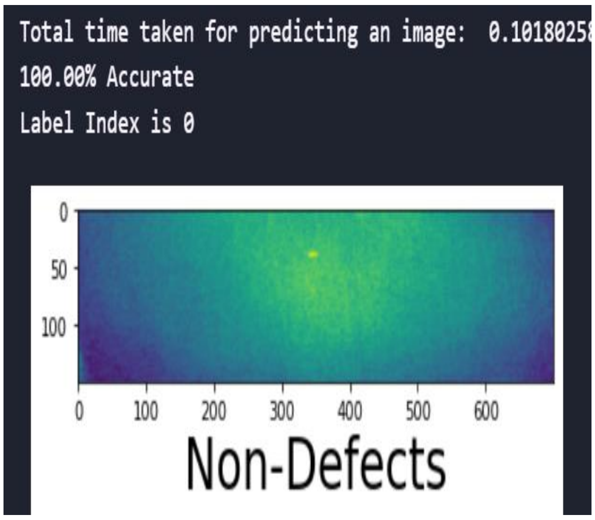


Image size 245x345

Image size 150x700



```
Total time taken for predicting an image:  0.11637
100.00% Accurate
Label Index is 1
```



```
Total time taken for predicting an image:  0.1018025
100.00% Accurate
Label Index is 0
```

Image size 245x345

Image size 150x700