IOSR Journal of Polymer and Textile Engineering (IOSR-JPTE)
e-ISSN: 2348-019X, p-ISSN: 2348-0181, Volume 12, Issue 5 (Sep. – Oct. 2025), PP 27-38
www.iosrjournals.org

Comparing the Effectiveness of various Machine Learning Models in the Fault Detection in Steel plates

Argha Chatterjee[1], Koushik Majumder[1], Bidesh Roy[2],Debjit Hazra[2],Swapnanil Das[2],Ronojoy Chakraborty[3],Shiba Dey[4]

Abstract: *With the advent of technology revolution, modern industries are adopting new methods for detecting defects in the steel plates that have affected the inspection quality for a long time. The new methods include use of AI Machine learning, image processing, deep learning, neural network and many other processes. After the steel is freshly cut there could be several defects like stains and scratches on the surface of the steel that could make it ineffective for usage purposes. The manual methods are very time taking and labour intensive and prone to unattended defects hence the attempts to find alternative solutions that are more effective and beneficial for the overall steel industry. Through our technique we try to train machine learning for automatic pattern recognition. The various machine learning algorithms that have been applied for training include Logistic Regression, Decision Tree, Support Vector Machine, K Nearest Neighbour, Gaussian Naive Bayes and Random Forest. Also, various feature extraction techniques like Principal Component Analysis, Linear Discriminant Analysis and Simple classification are applied on the data before the machine learning algorithms are applied. Each of these methods are applied on the training data to see which had more accuracy and then the accuracy is compared, and a final ranking is done for the various machine learning algorithms.*

I. Literature Review

- Jubin Deepakkumar Kothari did a similar project of detecting defects in steel plates using Machine Learning and Computer Vision Algorithms. His studies found that using convolutional Neural Network and U-Net architecture makes the process more efficient resulting in an accuracy of 98.3%
- Yu- Jen Huang, Ko Wei Huang and Shih Hsiung Lee performed a similar study using Deep Learning Technology. They used existing Deep learning techniques for object detection like YOLOv3 and SSD for effectively detecting the defects in stainless steel plates.
- Shuai Wang, Xiaojun Xia, Lanqing Ye and Binbin Yang did a study for detection and classification of steel surface defects using deep convolutional neural networks which resulted in an accuracy of 98.2% and an average faster running time.

I. Introduction

Steel factories today are facing huge problems that arise because of defects in the steel plates. Steel plates are required in several industries like automobile, defence, machinery, chemicals, etc. The various types of defects that these freshly cut steel plates are prone to include stains, scratches, bumps, dirtiness. Manual methods of detection are still prevalent in the industry, but several new methods are coming up.

The objective of this research is to propose a method to understand the defect detection methods in steel plates using various machine learning techniques like Logistic Regression, Decision Tree, Support Vector Machine, K Nearest Neighbour, Gaussian Naive Bayes and Random Forest.

II. Material And Methods

The methodology that we obtained was to run various Machine Learning Algorithms on the data and then verify their scores and Accuracy to see which algorithm gives the best output. The data was standardized and then split into training and testing data. Then Principal component analysis was run on 10 principal components out of the 27 features available. First, we applied Logistic Regression on the data and obtained the scores and accuracy. Then we applied Decision Tree, Support Vector Machine, K Nearest Neighbour, Gaussian Naive Bayes, Random Forest. For dimensionality reduction we applied Linear Discriminant Analysis and performed Logistic Regression, Decision Tree, Support Vector Machine, K Nearest Neighbour, Gaussian Naive Bayes and Random Forest and calculated the accuracy for each case. Next, we did simple classification without dimensionality reduction and then did Logistic Regression, Decision Tree, Support Vector Machine, K Nearest Neighbour, Gaussian Naive Bayes, and Random Forest.

Once the results are obtained from each of the cases the results were compared. First was scores and accuracy between various models using Simple Classification, Principal Component Analysis and Linear Discriminant Analysis. Then scores were compared for each model for each of the techniques Simple

Classification, Principal Component Analysis and Linear Discriminant Analysis. The Accuracy was compared for each model for Simple Classification, Principal Component Analysis and Linear Discriminant Analysis. According to the comparison of the results we were able to reach a conclusion on the ranking of various models as per their accuracy.

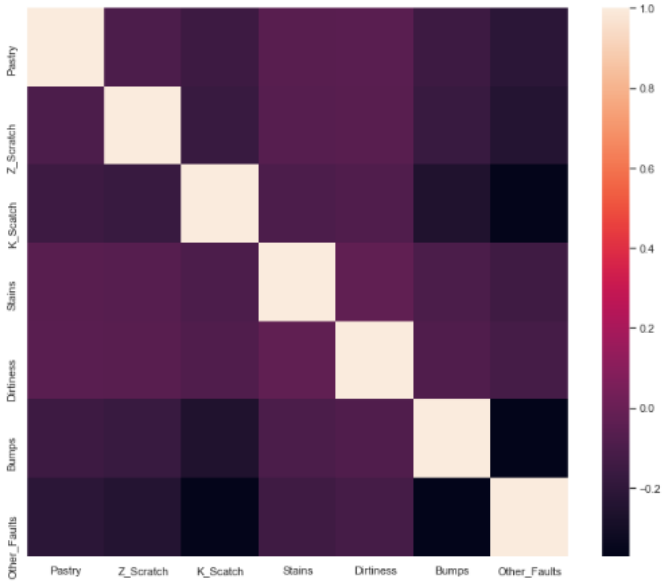
About The Data
The dataset consists of steel plates' faults which are divided in to 7 different types.

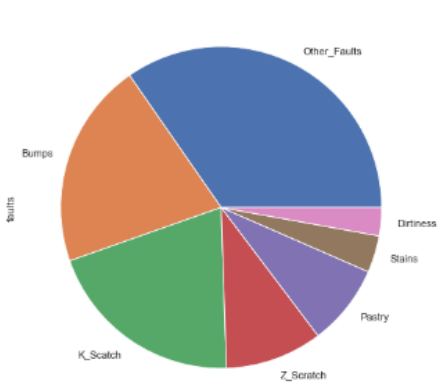
The dataset consists of 27 features describing each fault (location, size, ...) and 7 binary features indicating the type of fault (on of 7: Pastry, Z_Scratch, K_Scatch, Stains, Dirtiness, Bumps, Other_Faults). The latter is commonly used as a binary classification target ('common' or 'other' fault.)

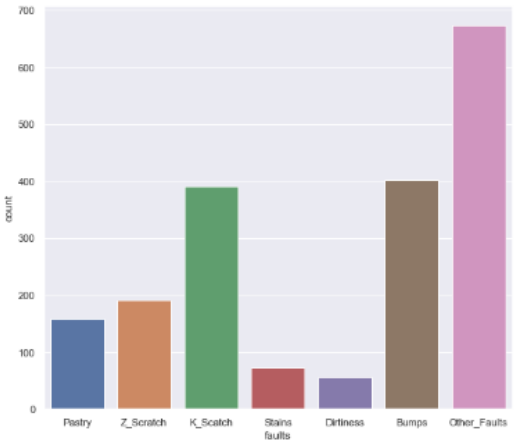
Attribute Information
* V1: X_Minimum
* V2: X_Maximum
* V3: Y_Minimum
* V4: Y_Maximum
* V5: Pixels_Areas
* V6: X_Perimeter
* V7: Y_Perimeter
* V8: Sum_of_Luminosity
* V9: Minimum_of_Luminosity
* V10: Maximum_of_Luminosity
* V11: Length_of_Conveyer
* V12: TypeOfSteel_A300
* V13: TypeOfSteel_A400
* V14: Steel_Plate_Thickness
* V15: Edges_Index
* V16: Empty_Index
* V17: Square_Index
* V18: Outside_X_Index
* V19: Edges_X_Index
* V20: Edges_Y_Index
* V21: Outside_Global_Index
* V22: LogOfAreas
* V23: Log_X_Index
* V24: Log_Y_Index
* V25: Orientation_Index
* V26: Luminosity_Index
* V27: SigmoidOfAreas
* V28: Pastry
* V29: Z_Scratch
* V30: K_Scatch
* V31: Stains
* V32: Dirtiness
* V33: Bumps
* Class: Other_Faults

III. Result

Visualizing the Fault Distribution and Correlation







Creating Training and Testing Data Set

The data set is first divided into training and testing data set and then the various machine learning techniques are applied on the training data.

Spliting the Data for Training and Testing

```
In [53]: from sklearn.model_selection import train_test_split

In [54]: x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.50, random_state = 42)

In [55]: x_train.shape
Out[55]: (970, 27)

In [57]: y_train.shape
Out[57]: (970,)

In [58]: x_test.shape
Out[58]: (971, 27)

In [59]: y_test.shape
Out[59]: (971,)
```

Principal Component Analysis

Next, we apply the Principal Component Analysis method for reducing the data dimensionality of the data. We will take 10 Principal Components out of the given 27 features available in the dataset.

Applying Principal Component Analysis

We will take 10 Principal Components, out of 27 Features available with us.

```
In [60]: from sklearn.decomposition import PCA

In [62]: pca=PCA(10)

In [63]: pca.fit(x_test)

Out[63]: PCA(copy=True, iterated_power='auto', n_components=10, random_state=None,
             svd_solver='auto', tol=0.0, whiten=False)

In [64]: pca_train=pca.transform(x_train)

In [65]: pca_test=pca.transform(x_test)

In [66]: pca_train=x_train

In [67]: pca_test=x_test

In [68]: pca_score= np.zeros(6)

In [69]: pca_accuracy= np.zeros(6)
```

Logistic Regression

After Principal Component Analysis is done, we apply the Logistic Regression method on the data.

Applying Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable

```
In [70]: from sklearn.linear_model import LogisticRegression as LR

In [71]: Logistic_Regression = LR().fit(pca_train,y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

In [72]: pca_score[0]=Logistic_Regression.score(pca_train,y_train)

In [73]: predictions_LR = Logistic_Regression.predict(pca_test)
```

Accuracy of Prediction using Logistic Regression

```
In [74]: from sklearn.metrics import accuracy_score

In [75]: pca_accuracy[0]=accuracy_score(y_test, predictions_LR)

In [77]: pca_accuracy[0]*100

Out[77]: 73.01750772399588
```

Decision Tree

Decision Tree method is applied on the same data and accuracy is predicted.

Applying Decision Tree

```
In [78]: from sklearn.tree import DecisionTreeClassifier as DTC

In [79]: Decision_Tree_Classifier = DTC().fit(pca_train,y_train)

In [181]: pca_score[1]=Decision_Tree_Classifier.score(pca_train,y_train)

In [81]: predictions_DTC = Decision_Tree_Classifier.predict(pca_test)
```

Accuracy of Prediction using Logistic Regression

```
In [83]: pca_accuracy[1]=accuracy_score(y_test, predictions_DTC)

In [84]: pca_accuracy[1]*100
Out[84]: 63.027806385169924
```

We get 63.02 % Accuracy using "Decision Tree"

Support Vector Machine

A similar analysis is done applying the Support Vector Method and K Nearest Neighbours method and Gaussian Naive Bayes and Random Forest method.

Applying Support Vector Machine

```
In [85]: from sklearn.svm import SVC

In [86]: SVC = SVC().fit(pca_train,y_train)

In [87]: pca_score[2]=SVC.score(pca_train,y_train)

In [88]: predictions_SVC = SVC.predict(pca_test)
```

Accuracy of Prediction using Support Vector Machine

```
In [91]: pca_accuracy[2]=accuracy_score(y_test, predictions_SVC)

In [92]: pca_accuracy[2]*100
Out[92]: 77.34294541709578
```

We get 77.34 % Accuracy using "Support Vector Machine"

Applying K-Nearest Neighbours

```
In [93]: from sklearn.neighbors import KNeighborsClassifier as KNC

In [94]: K_Neighbors_Classifier = KNC(8).fit(pca_train,y_train)

In [95]: pca_score[3]=K_Neighbors_Classifier.score(pca_train,y_train)

In [96]: predictions_KNC = K_Neighbors_Classifier.predict(pca_test)
```

Accuracy of Prediction using K-Nearest Neighbours

```
In [97]: pca_accuracy[3]=accuracy_score(y_test, predictions_KNC)

In [98]: pca_accuracy[3]*100
Out[98]: 71.98764160659114
```

We get 71.98 % Accuracy using "K-Nearest Neighbours"

Applying Gaussian Naive Bayes

```
In [99]: from sklearn.naive_bayes import GaussianNB as GNB
In [100]: GNB = GNB().fit(pca_train,y_train)
In [101]: pca_score[4]=GNB.score(pca_train,y_train)
In [102]: predictions_GNB= GNB.predict(pca_test)
```

Accuracy of Prediction using Gaussian Naive Bayes

```
In [103]: pca_accuracy[4]=accuracy_score(y_test, predictions_GNB)
In [104]: pca_accuracy[4]*100
Out[104]: 59.320288362512876
```

We get 59.32 % Accuracy using "Gaussian Naive Bayes"

Applying Random Forest

```
In [105]: from sklearn.ensemble import RandomForestClassifier as RF
In [106]: RF = RF().fit(pca_train,y_train)
In [107]: pca_score[5]=RF.score(pca_train,y_train)
In [108]: predictions_RF= RF.predict(pca_test)
```

Accuracy of Prediction using Random Forest

```
In [109]: pca_accuracy[5]=accuracy_score(y_test, predictions_RF)
In [110]: pca_accuracy[5]*100
Out[110]: 78.57878475798145
```

We get 78.57 % Accuracy using "Random Forest"

Linear Discriminant Analysis

We then perform Linear Discriminant Analysis on the given data to reduce the number of features into more manageable numbers.

Applying Linear Discriminant Analysis

Linear Discriminant Analysis is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning purposes.

```
In [111]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
In [112]: lda=LinearDiscriminantAnalysis()
In [113]: lda_train = lda.fit(x_test, y_test)
In [114]: lda_train=lda.transform(x_train)
In [115]: lda_test=lda.transform(x_test)
In [116]: lda_score= np.zeros(6)
In [117]: lda_accuracy= np.zeros(6)
```

Next, we apply the machine Learning methods like Logistic Regression, Decision Tree, Support Vector,K Nearest Neighbour, Gaussian Naive Bayes and Random Forest on the data obtained from the Linear Discriminant Analysis and calculate the Accuracy of prediction.

Applying Logistic Regression ¶

In [118]:
```
Logistic_Regression = LR().fit(lda_train,y_train)
lda_score[0]=Logistic_Regression.score(lda_train,y_train)
predictions_LR = Logistic_Regression.predict(lda_test)
lda_accuracy[0]=accuracy_score(y_test, predictions_LR)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Prediction Accuracy

In [119]:
```
lda_accuracy[0]*100
```

Out[119]:
```
72.39958805355305
```

We get 72.39 % Accuracy using LDA- "Logistic Regression"

Applying Decision Tree

In [120]:
```
Decision_Tree_Classifier = DTC().fit(lda_train,y_train)
lda_score[1]=Decision_Tree_Classifier.score(lda_train,y_train)
predictions_DTC = Decision_Tree_Classifier.predict(lda_test)
lda_accuracy[1]=accuracy_score(y_test, predictions_DTC)
```

Prediction Accuracy

In [121]:
```
lda_accuracy[1]*100
```

Out[121]:
```
63.64572605561277
```

We get 63.64 % Accuracy using LDA- "Decision Tree"

Applying Support Vector Machine

In [123]:
```
from sklearn.svm import SVC
SVC = SVC().fit(lda_train,y_train)
lda_score[2]=SVC.score(lda_train,y_train)
predictions_SVC = SVC.predict(lda_test)
lda_accuracy[2]=accuracy_score(y_test, predictions_SVC)
```

Prediction Accuracy

In [128]:
```
lda_accuracy[2]*100
```

Out[128]:
```
69.51596292481977
```

We get 69.51 % Accuracy using LDA- "Support Vector Machine"

Applying K-Nearest Neighbours

```
In [130]: K_Neighbors_Classifier = KNC(10).fit(lda_train,y_train)
          lda_score[3]=K_Neighbors_Classifier.score(lda_train,y_train)
          predictions_KNC = K_Neighbors_Classifier.predict(lda_test)
          lda_accuracy[3]=accuracy_score(y_test, predictions_KNC)
```

Prediction Accuracy

```
In [131]: lda_accuracy[3]*100
Out[131]: 74.35633367662204
```

We get 74.35 % Accuracy using LDA- "K-Nearest Neighbours"

Applying Gaussian Naive Bayes

```
In [133]: from sklearn.naive_bayes import GaussianNB as GNB

          GNB = GNB().fit(lda_train,y_train)
          lda_score[4]=GNB.score(lda_train,y_train)
          predictions_GNB= GNB.predict(lda_test)
          lda_accuracy[4]=accuracy_score(y_test, predictions_GNB)
```

Prediction Accuracy

```
In [134]: lda_accuracy[4]*100
Out[134]: 54.994850669412976
```

We get 54.99 % Accuracy using LDA- "Gaussian Naive Bayes"

Applying Random Forest

```
In [135]: from sklearn.ensemble import RandomForestClassifier as RF
          RF = RF().fit(lda_train,y_train)
          lda_score[5]=RF.score(lda_train,y_train)
          predictions_RF= RF.predict(lda_test)
          lda_accuracy[5]=accuracy_score(y_test, predictions_RF)
```

Prediction Accuracy

```
In [136]: lda_accuracy[5]*100
Out[136]: 76.41606591143152
```

We get 76.41 % Accuracy using LDA- "Random Forest"

Simple Classification Method:

Simple Classification method without dimension Reduction is applied on the data and then the Machine Learning algorithms are applied one by one like Logistic Regression, Decision Tree, Support Vector, K Nearest Neighbour, Random Forest and Gaussian Naive Bayes.

Applying Simple Classification without Dimension Reduction

```
In [138]: simple_score= np.zeros(6)
          simple_accuracy= np.zeros(6)
```

Applying Logistic Regression

In [139]:
```
Logistic_Regression = LR().fit(x_train,y_train)
simple_score[0]=Logistic_Regression.score(x_train,y_train)
predictions_LR = Logistic_Regression.predict(x_test)
simple_accuracy[0]=accuracy_score(y_test, predictions_LR)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Predicting Accuracy

In [140]:
```
simple_accuracy[0]*100
```

Out[140]:
```
73.01750772399588
```

We get 73.01 % Accuracy using Simple Classification- "Logistic Regression"

Applying Decision Tree

In [141]:
```
Decision_Tree_Classifier = DTC().fit(x_train,y_train)
simple_score[1]=Decision_Tree_Classifier.score(x_train,y_train)
predictions_DTC = Decision_Tree_Classifier.predict(x_test)
simple_accuracy[1]=accuracy_score(y_test, predictions_DTC)
```

Predicting Accuracy

In [143]:
```
simple_accuracy[1]*100
```

Out[143]:
```
64.57260556127703
```

We get 64.57 % Accuracy using Simple Classification- "Decision Tree"

Applying Support Vector Machine

In [144]:
```
from sklearn.svm import SVC
SVC = SVC().fit(x_train,y_train)
simple_score[2]=SVC.score(x_train,y_train)
predictions_SVC = SVC.predict(x_test)
simple_accuracy[2]=accuracy_score(y_test, predictions_SVC)
```

Predicting Accuracy

In [145]:
```
simple_accuracy[2]*100
```

Out[145]:
```
77.34294541709578
```

We get 77.34 % Accuracy using Simple Classification- "Support Vector Machine"

Applying K Nearest Neighbour

In [146]:
```
K_Neighbors_Classifier = KNC(10).fit(x_train,y_train)
simple_score[3]=K_Neighbors_Classifier.score(x_train,y_train)
predictions_KNC = K_Neighbors_Classifier.predict(x_test)
simple_accuracy[3]=accuracy_score(y_test, predictions_KNC)
```

Predicting Accuracy

In [147]:
```
simple_accuracy[3]*100
```

Out[147]:
```
71.26673532440783
```

We get 71.26 % Accuracy using Simple Classification- "K Nearest Neighbour"

Applying Gaussian Naive Bayes

```
In [149]: from sklearn.naive_bayes import GaussianNB as GNB

          GNB = GNB().fit(x_train,y_train)
          simple_score[4]=GNB.score(x_train,y_train)
          predictions_GNB= GNB.predict(x_test)
          simple_accuracy[4]=accuracy_score(y_test, predictions_GNB)
```

Predicting Accuracy

```
In [150]: simple_accuracy[4]*100
Out[150]: 59.320288362512876
```

We get 59.32 % Accuracy using Simple Classification- "Gaussian Naive Bayes"

Applying Random Forest

```
In [151]: from sklearn.ensemble import RandomForestClassifier as RF
          RF = RF(10).fit(x_train,y_train)
          simple_score[5]=RF.score(x_train,y_train)
          predictions_RF= RF.predict(x_test)
          simple_accuracy[5]=accuracy_score(y_test, predictions_RF)
```

Predicting Accuracy

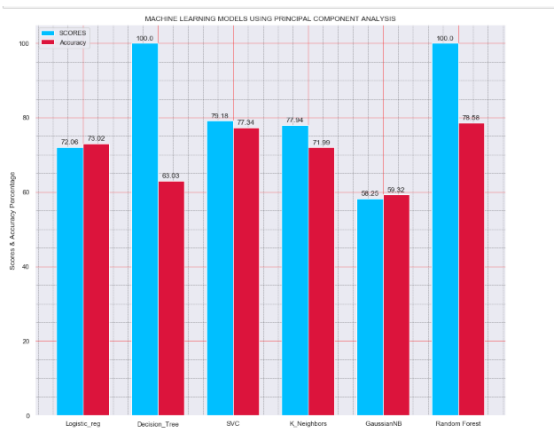
```
In [152]: simple_accuracy[5]*100
Out[152]: 71.67868177136972
```

We get 71.67 % Accuracy using Simple Classification- "Random Forest"

Comparison of the Results

After we applied the various Machine Learning models, we tried to compare the score and accuracy of each of the models. First, we compared the scores and accuracy for each of the models with simple classification method applied for classification.

Next, we compare the results for the Machine Learning Algorithms when we have applied Principal Component Analysis as given in the below diagram.



And then we compare the results for the Machine Learning Algorithms when we have applied Linear Discriminant Analysis as given in the below diagram.

IV. Conclusion

From the visualization on the accuracy given above we can rank the models as given below:
Rank 1: Random Forest
Rank 2: Support Vector
Rank 3: K Nearest Neighbor
Rank 4: Logistic Regression
Rank 5: Decision Trees
Rank 6: Gaussian Naive Bayes
We can conclude that Random Forest method is the most effective method for detection of faults in the steel plates for the given data with accuracy of 78.58 % for PCA, 76.42 for LDA and 71.68 for Simple classification.

References

[1]. Luo, Q.; Fang, X.; Liu, L.; Yang, C.; Sun, Y. Automated Visual Defect Detection for Flat Steel Surface: A Survey. IEEE Trans.
[2]. Instrum. Meas. 2020, 69, 626–644. [CrossRef]
[3]. Thomas, G.B.; Jenkins, M.S.; Mahapatra, R.B. Investigation of strand surface defects using mould instrumentation and modelling.
[4]. Ironmak. Steelmak. 2004, 31, 485–494. [CrossRef]
[5]. Shi, T.; Kong, J.; Wang, X.; Liu, Z.; Zheng, G. Improved Sobel algorithm for defect detection of rail surfaces with enhanced efficiency and accuracy. J. Cent. South Univ. 2016, 23, 2867–2875. [CrossRef]
[6]. Liu, Y.; Xu, K.; Wang, D. Online Surface Defect Identification of Cold Rolled Strips Based on Local Binary Pattern and Extreme
[7]. Learning Machine. Metals 2018, 8, 197. [CrossRef]
[8]. Wang, Z.; Zhu, D. An accurate detection method for surface defects of complex components based on support vector machine and spreading algorithm. Measurement 2019, 147, 106886. [CrossRef]
[9]. Kang, G.; Liu, H. Surface defects inspection of cold rolled strips based on neural network. In Proceedings of the 2005 International
[10]. Conference on Machine Learning and Cybernetics(ICMLC 2005), Guangzhou, China, 18–21 August 2005; pp. 5034–5037.
[11]. Di, H.; Ke, X.; Peng, Z.; Zhou, D. Surface defect classification of steels with a new semi-supervised learning method. Opt. Laser.
[12]. Eng. 2019, 117, 40–48. [CrossRef]
[13]. Schlegl, T.; Seeböck, P.; Waldstein, S.M.; Schmidt-Erfurth, U.; Langs, G. Unsupervised Anomaly Detection with Generative
[14]. Adversarial Networks to Guide Marker Discovery. In Proceedings of the 25th International Conference on Information Processing in Medical Imaging (IPMI 2017), Boone, NC, USA, 25–30 June 2017; pp. 146–157.
[15]. Lee, S.Y.; Tama, B.A.; Moon, S.J.; Lee, S. Steel Surface Defect Diagnostics Using Deep Convolutional Neural Network and Class
[16]. Activation Map. Appl. Sci. 2019, 9, 5449. [CrossRef]
[17]. Tabernik, D.; Šela, S.; Skvarˇc, J.; Skoˇcaj, D. Segmentation-based deep-learning approach for surface-defect detection. J. Intell.
[18]. Manuf. 2020, 31, 759–776. [CrossRef]
[19]. Prappacher, N.; Bullmann, M.; Bohn, G.; Deinzer, F.; Linke, A. Defect Detection on Rolling Element Surface Scans Using Neural
[20]. Image Segmentation. Appl. Sci. 2020, 10, 3290. [CrossRef]
[21]. Li, J.; Su, Z.; Geng, J.; Yin, Y. Real-time Detection of Steel Strip Surface Defects Based on Improved YOLO Detection Network.
[22]. IFAC PapersOnLine 2018, 51, 76–81. [CrossRef]
[23]. Wei, R.; Song, Y.; Zhang, Y. Enhanced Faster Region Convolutional Neural Networks for Steel Surface Defect Detection. ISIJ Int.
[24]. 2020, 60, 539–545. [CrossRef]
[25]. Oh, S.-J.; Jung, M.-J.; Lim, C.; Shin, S.-C. Automatic Detection of Welding Defects Using Faster R-CNN. Appl. Sci. 2020, 10, 8629. [CrossRef]
[26]. Borselli, A.; Colla, V.; Vannucci, M.; Veroli, M. A fuzzy inference system applied to defect detection in flat steel production. In Proceedings of the 2010 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2010), Barcelona, Spain, 18–23 July 2010; pp. 1–6.
[27]. Zapata J., Vilar R., and Ruiz R., "Performance evaluation of an automatic inspection system of weld defects in radiographic images based on neuro-classifiers ", Expert Systems with Applications, Vol. 38, pp. 8812 – 8824, 2011.
[28]. Liao, T.W.," Improving the accuracy of computer-aided radiographic weld inspection by feature selection ", NDT&E International, Vol. 42, pp. 229–239, 2009.

[29]. Ankit Narendrakumar Soni 2018. Data Center Monitoring using an Improved Faster Regional Convolutional Neural Network. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 7, Issue 4, April 2018.
[30]. Vilar, R., Zapata, J., and Ruiz, R.," An automatic system of classification of weld defects in radiographic images ", NDT & E International, Vol. 42, pp. 467–476, 2009.
[31]. Shafeek H.I., Gadelmawla E.S., AbdelShafy A.A. and Elewa I.M.," Assessment of welding defects for gas pipeline radiographs using computer vision ", NDT&E International, Vol. 37, pp. 291–299, 2004.