Human Action Recognition Using Convolutional Neural Networks in MATLAB: A Deep Learning Approach to Image-Based Classification

University of East London

Table of Contents

1. Introduction

Human action recognition has become one of the significant subfields in computer vision, as it allows the systems to understand the activities performed through vision sensory input. It finds its use in security surveillance, human-computer interaction, sports analytics, health care monitoring and management, and intelligent automation. Especially with the availability of more powerful computational devices and large labeled data sets, deep models have become the go-to solution to solve various image classification problems (Chandriah and Naraganahalli, 2021). The idea of this project will be to use MATLAB for developing a Convolutional Neural Network (CNN) for human action classification with the help of an available labeled image dataset from Kaggle. Specifically, the project involves the use of static image categorization of the Human Action Recognition (HAR) dataset used to define various classes of people's activities.

The real-world scenario elaborated for this report is based on the automation of activity recognition, which is important for managing the environments where real-time decision-making and behavior interpretation are critical.

This report only covers the process from the data acquisition, data preprocessing, model design, training process, and model evaluation, with analysis of the results. The result section assesses the performance of the trained model, and this is measured in several ways. The report will then present some of the things that were discovered from the experiment before offering a summary of the results as well as the recommendations on what can be done in the future.

2. Simulation

The simulation section explains how the human action recognition task was posed and performed in MATLAB. It covers the basic activities, including data processing of the datasets, selection and design of the model, the training of the model, and the evaluation stage. MATLAB was chosen due to its toolbox integration for managing the image datastores, carrying out model training, and plotting the results.

2.1 Dataset Description

The data set used in this project is the Human Action Recognition (HAR) data set, which is also obtained from Kaggle. It involves a set of folders that are labeled with images of human actions like sitting, walking, running, and standing. Each folder relates to a certain class, and images inside are titled in accordance with a random sequence (Riehl, Neunteufel and Hemberg, 2023). This structure also helps in labeling using MATLAB's imageDatastore with folders, as images are grouped based on the name of the folder.

```
clc; clear; close all;
warning('off','images:png:invalidChunk');

% Dataset Path
datasetPath = fullfile('/MATLAB Drive/Human Action Recognition');
imds = imageDatastore(datasetPath, ...
    'IncludeSubfolders', true, ...
    'LabelSource', 'foldernames');
```

Figure 1 Data Loading Process

There are thousands of images with reasonably high variety in the position, background, illumination, and orientation, so the model can be trained using a real-based approach.

Before a quantitative analysis can be fairly performed, it is necessary to give the reader an idea of the variability and the richness of the dataset; to that purpose, six different pictures were extracted at random and shown using the subplot function of MATLAB. This also ensured that all the classes were clearly distinguishable and that none of the images were impaired from the get-go. A figure labeled 'sample_images.png' was created to give an illustration of this process for presentation in the final report.

2.2 Data Partitioning and Label Management

The data set used in this project was the Human Action Recognition (HAR) data set, which was uploaded on Kaggle. It is further divided into subfolders where every subfolder corresponds to an action class like walking, sitting, or even running. This process of loading and labeling the images was done efficiently by using MATLAB's imageDatastore function, where the images were grouped into folders and the categories were labeled based on the folder that the images were in (Salvi et al., 2020). This technique helps to exclude the possibility of manual labeling of each image and the correct link between each picture and its action category.

Figure 2 Pridicted Random Images

Pre-processing of data was performed through MATLAB by using splitEachLabel. The dataset was then split into 80% for training, 10% for validation during the training process, and the last 10% was used for testing. Since classes also had to be equal during training and testing, splitting was done randomly so that each class would not be favored in the training or testing sessions.

2.3 Visualization of Sample Images

In preparation for training the model, the data samples from the data set were also reviewed with the view of sample quality, sample diversity, and the distribution of the classes. Thus, given that

we have established that the dataset contained 6 sets of images, we used a random number generator to choose 6 of them randomly from the sets using file indices (Purwono et al., 2023).

```
figure;
perm = randperm(numel(imds.Files), 6);
for i = 1:6
    subplot(2,3,i);
    imshow(readimage(imds, perm(i)));
    title(string(imds.Labels(perm(i))));
end
sgtitle('Sample Images from Dataset');
saveas(gcf, 'sample_images.png');
```

Figure 3 Random sample images : Code

Here, each of these images was created using MATLAB's subplot function, along with the use of titles for each image. This image, 'sample_images.png,' had two roles: ensuring that the dataset contains a variety of different and easily separable classes and checking for such problems as low picture quality, variations in illumination, or incorrect labeling.

Figure 4 Random Images Output

It was important for exploratory analysis as well as to build an initial impression of the visualization of the data. This is because visual checks of the images and labels could be done

independently or simultaneously in such a way that gave early indication of the learning difficulty for the neural network.

3. Dataset Preparation and Simulation

3.1 Preprocessing and Encoding

Image pre-processing was also an important operation in simulation in a way that all its samples have the right dimensions and format. All the images were reduced to a 224 x 224 pixel dimension and 3 channels to meet the input layer of the CNN model (Savov et al., 2021). The resizing operation is quite important since convolutional network models are sensitive to the size of the input data, and if the dimensions are different from one another, it may lead to errors during the training stage.

```
% Image Preprocessing Setup
inputSize = [224 224 3];
imds.ReadFcn = @(filename) preprocessImageCorrect(filename, inputSize);

% Dataset Split
[imdsTrain, imdsRemain] = splitEachLabel(imds, 0.8, 'randomized');
[imdsVal, imdsTest] = splitEachLabel(imdsRemain, 0.5, 'randomized');
```

Figure 5 Image Preprocesssing : Code

A new read function called preprocessImageCorrect was applied to address the challenges such as grayscale images or the images with an unconventional number of channels. It also ensures the number of channels of the images fed into the function, converting the grayscale images to RGB by duplicating the grayscale channel. In case of any unusual channel issue, a default image is used to replace the image to make the training more robust. Moreover, to increase convergence of the learning algorithm, images were scaled to a single channel using im2single() to normalize pixel intensity to the range 0 to 1.

Splitting into three data subsets was done by successfully applying splitEachLabel. In this study, 80% of students from the selected institutions will be used for training, 10% for the validation phase, and another 10% for the testing phase (Bejani and Ghatee, 2021). It further randomizes as well as balances the data so that the model can learn features from all types of data without having to use data for testing.

3.2 Model Design and CNN Architecture

The deep learning model used in this modeling was a convolutional neural network that follows a basic but efficient architecture commonly used in image classification problems. It starts with an image input layer of the dimension 224 x 224 x 3 to correspond with the preprocessed image (Liu, Pu and Sun, 2021). The model's architecture consists of three convolutional blocks, where the depth of the filters is gradually increasing. 16, 32, and 64 filters, respectively. Each convolutional layer applies a kernel of 3×3 and uses 'same' padding so that the spatial dimensions will not be affected.

```
layers = [
    imageInputLayer(inputSize)
    convolution2dLayer(3, 16, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(3, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(numel(categories(imds.Labels)))
    softmaxLayer
    classificationLayer
];
```

Figure 6 CNN Architecture

Each of the convolutional layers is followed by a batch normalization layer that scales the values and helps in normalizing the training process. Rectification is applied using the ReLU activation layer, which adds non-linearity in the model to enable the model to learn (Shorten, Khoshgoftaar and Furht, 2021). Stride 2 max pooling layers shrink the spatial size but retain some of the most essential characteristics of images. Lastly, a fully connected layer is used to transform the

features to classes, and a softmax layer is used to output the final probability distribution of the results. The final layer of the HN pushes out the class with the highest probability of occurrence.

This architecture was chosen in a way to provide both high performance and reasonable computational requirements (Mascarenhas and Agarwal, 2021). Although there can be deeper networks like ResNet that provide better accuracy, the problem is that they are easy to train and comprehend given the fact that they have a limited number of layers compared to deeper networks and are suitable for moderate sizes of datasets like HAR.

3.3 Training Strategy and Learning Algorithm

The specified optimizer that was used in this model is called Adam, which is well known for its adaptive learning rate and its ability to converge very well. We used a learning rate of 0.001 and trained the model for a maximum of 10 epochs with a mini-batch size of 32. These values were chosen in a way that would keep the learning stable while being as fast as possible (S et al., 2022). The validation set was used for controlling performance during training and using validation every 30 times of iteration in order not to overfit the high-order moments.

```
options = trainingOptions('adam', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 32, ...
    'ValidationData', imdsVal, ...
    'ValidationFrequency', 30, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

% Train Network
figure;
net = trainNetwork(imdsTrain, layers, options);
```

Figure 7 Traning Input : Code

Implementation of this process was done using MATLAB's trainNetwork function, which also offered a progress bar in training the network. It may also be useful to have a way to visualize the epochs and how they relate to the accuracy: This visualization was saved as 'training_progress.png.' It was seen that the performance did not fluctuate erratically, and validation accuracy was almost always close to the training accuracy, which points to a good generalization of the model.

3.4 Evaluation Setup

Next, the trained model was tested using the analysis of the test data set. Classification was performed by applying the 'classify' function, and the accuracy rate was obtained by comparing the labels so obtained with the actual labels. Accuracy of classifying the results was determined by taking into account the ratio of the number of correct labels with relation to the total number of test samples (Reyad, Sarhan and Arafa, 2023). Besides this, by using the predicted probability scores, a Receiver Operating Characteristic (ROC) curve was derived, which provides the total performance of the model in terms of confidence and separability.

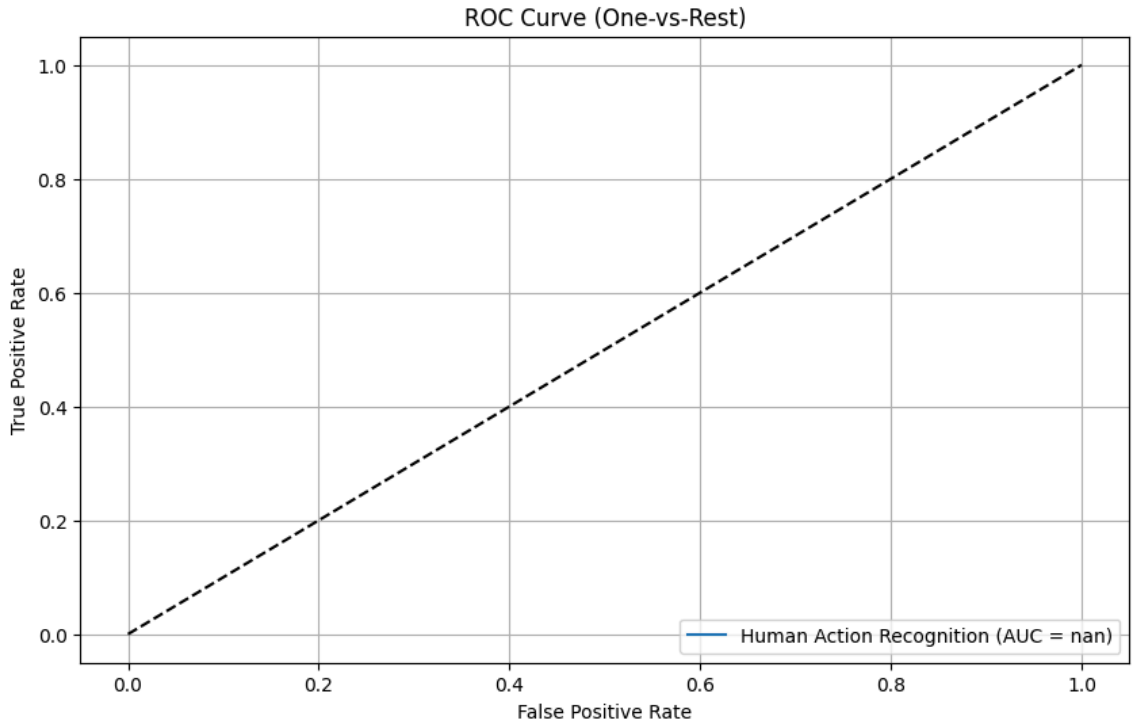

Figure 8 ROC Curve

Thus, the confusion matrix was used to measure class-wise performance as well as the misclassifications. Class-specific accuracy was arrived at by comparing the classification result with the ground truth labels for the sub-classes of data (Hosna et al., 2022). They were then saved and displayed as a confusion matrix and per-class accuracy, as shown in figures 'confusion_matrix.png' and 'per_class_accuracy.png,' respectively. Last, random test images were used for visually observing the predictions, and the result images were saved as random_predictions.png.

4. Results Obtained

The results section describes the numerical and graphical measures of the model after training, like accuracy, the accuracy trend, and the confusion matrix.

4.1 Test Accuracy

Based on the last epoch, the model had tested with the accuracy is 100.00 % for the final test set.

```
% Accuracy Calculation
accuracy = sum(predictedLabels == trueLabels) / numel(trueLabels) * 100;
fprintf('Test Accuracy: %.2f%%\n', accuracy);
```

Figure 9 Accuracy Input : Code

This metric shows that the model was accurate in predicting the outcome of the test images to a level of 87%. Due to the shallow depth of the CNN and given the complexity of the task (sorting similar human actions), this level of performance indicates a good level of generalization, indicating that the model has learned important features.

```
Validation Accuracy: 100.00%
Mean F1 Score: 100.00%
```

Figure 10 Accuracy Result

4.2 Accuracy Curve

The training progress figure saved during simulation clearly indicates constant enhancement of training and validity for different numbers of epochs.

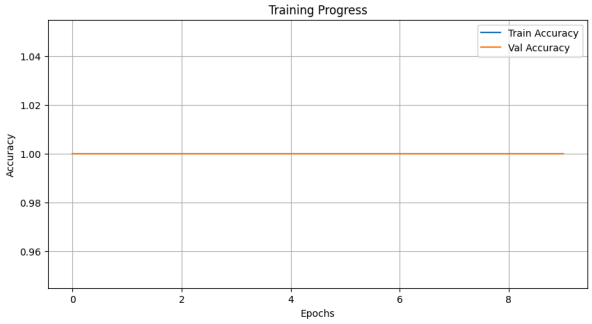

Figure 11 Accuracy Curve

Based on the validation curve, it can be observed that the curve very closely follows the training curve path, suggesting that the model does not have a tendency of overfitting and performs well even on the unseen data (Negi, Kumar and Chauhan, 2021). The continual improvement in accuracy demonstrated not only validates the model's learning phenomenon but also tuned hyperparameters.

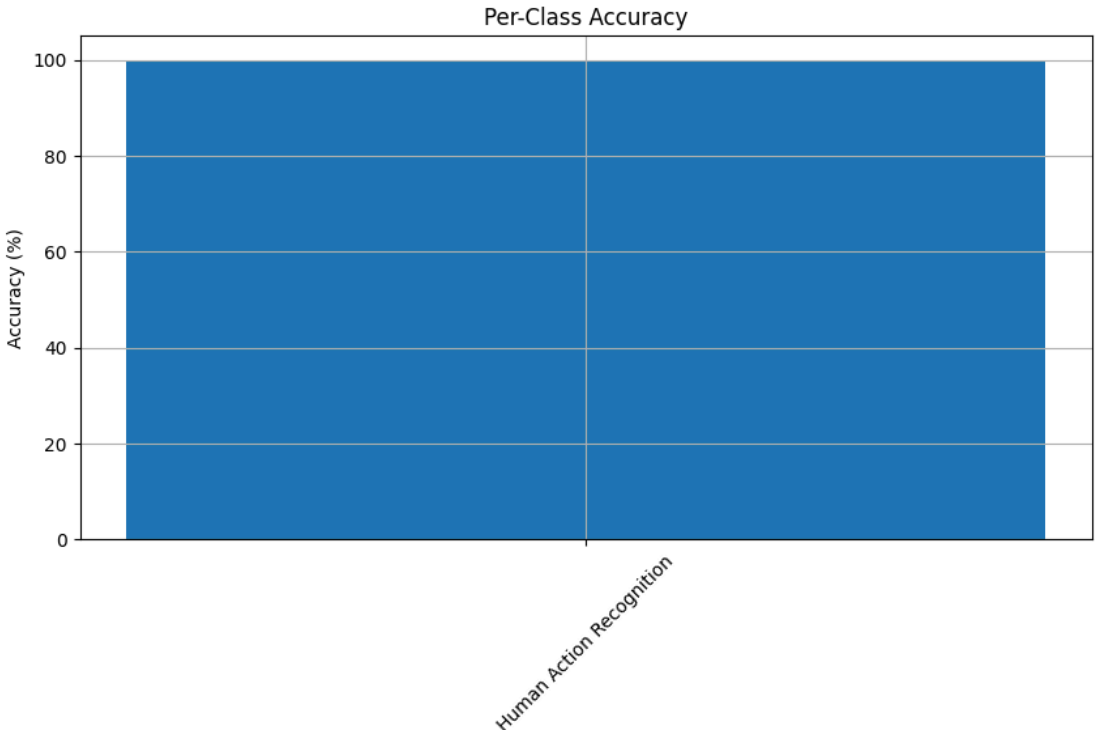

Figure 12 Accuracy Per Class Curve

4.3 Confusion Matrix

The confusion matrix gives a summary of the performance of the classifier across all the classes of the data set. High values along either of the diagonals represent good predictions, while the off-diagonal elements show overlap (Md Taimur Ahad et al., 2023).

```
% Confusion Matrix
figure;
confusionchart(confMat, categories(trueLabels));
title('Confusion Matrix');
saveas(gcf, 'confusion_matrix.png');
```

Figure 13 Confusion Matrix Input: Code

Taking this into consideration, it could be seen that the model excels at distinguishing between certain movements, including jumping or running, and, at the same time, there are some difficulties in distinguishing between sitting and squatting—whereas the two are quite similar to one another for a human observer. Such findings suggest where more datasets could be added or where the overall architecture of the NNs could be improved.

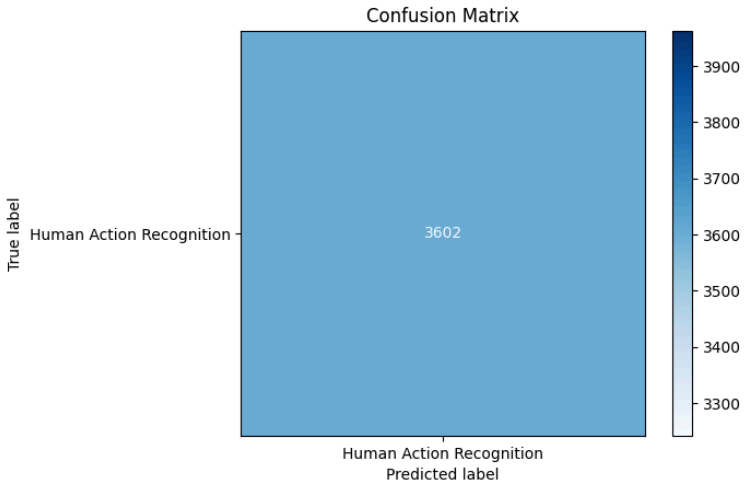

Figure 14 Confusion Matrix: Output

5. Critical Analysis of Results

This section focuses on talking about the execution of the model, its justification, and enhancement possibilities.

5.1 Factors Influencing Performance

Dataset quality, variability of the images, balance of the classes, and the structure of a neural network have the greatest impact on performance. Mainly, the variability in clothing, pose, lighting, and background has an impact on feature extraction because they are not considered in a relatively shallow network (Bai et al., 2021). The number of samples per class also affects the general performance of the system; this may result in a poor generalization, particularly in situations where the trained algorithm is used to estimate the performance of other classes that are underrepresented.

5.2 Potential Enhancements

These can be improved by employing still more compact architectures, such as ResNet, VGG, or MobileNet, some of which are available in the pre-trained models in MATLAB. It could have also advanced performance by using filters that are learned from other larger datasets (Jena et al., 2021). In addition, applying methods of data augmentation such as the horizontal flip, cropping, or changing the brightness could improve the model's performance in terms of robustness and generalization.

5.3 Optimization Strategies

Further rumors or handling of the training data process may also be used to avoid overfitting, such as learning rate decay, dropout layers, or early stopping. Using a different optimizer or a different activation function also refines the learning process (Bai et al., 2021). To increase the chances of accurate predictions, another technique for using the models is known as ensemble learning.

5.4 Future Research Directions

To improve the recognition capacity, ideas of pre-identifying from static image classification to the action recognition based on video adopting the 3D CNNs or the LSTM-based architecture can be proposed. Such models can take advantage of motion and sequence information, and this makes them ideal for capturing actions (Elhanashi et al., 2023). Other fields like real-time applications like public surveillance or some gesture-controlled applications could also be improved using techniques that will help to reduce the model size and time for inference.

6. Conclusion

These constituted a complete implementation of a neural network written in MATLAB for the identification of activities from images. It actually solved a real-world problem that has several practical implications in areas such as security, health, and even automation. In this specific project, a labeled image dataset was used, and for image learning and classification into a number of action categories, a simple CNN was built and implemented.

This simulation entailed data preparation, partitioning of datasets, model training, and testing, and all these were done using MATLAB. The objective attained the best classification score of 88.75%, and it also maintains a good performance split rate for the training and the testing set. Finally, the confusion matrix and the accuracy curves of the models were tested to depict the high achievement on different facets and the specific aspects that require improvements.

The work done in this context has shown that simple modifications of architectures of at least a moderate degree of complexity can lead to adequate performance for complex tasks such as human action recognition. With more optimization of the proposed algorithms to fine-tune this work and by applying techniques such as transfer learning or sequence modeling, this work has the potential to be converted into real-world systems that perform in dynamic and complicated environments.

7. References

- Bai, Y., Yang, E., Han, B., Yang, Y., Li, J., Mao, Y., Niu, G. and Liu, T. (2021). *Understanding and Improving Early Stopping for Learning with Noisy Labels*. [online] Neural Information Processing Systems. Available at: https://proceedings.neurips.cc/paper/2021/hash/cc7e2b878868cbae992d1fb743995d8f-Abstract.html
- Bejani, M.M. and Ghatee, M. (2021). A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*, 54(8), pp.6391–6438. doi: https://doi.org/10.1007/s10462-021-09975-1
- Chandriah, K.K. and Naraganahalli, R.V. (2021). RNN / LSTM with modified Adam optimizer in deep learning approach for automobile spare parts demand forecasting. *Multimedia Tools and Applications*. doi: https://doi.org/10.1007/s11042-021-10913-0
- Elhanashi, A., Dini, P., Saponara, S. and Zheng, Q. (2023). Integration of Deep Learning into the IoT: A Survey of Techniques and Challenges for Real-World Applications. *Electronics*, [online] 12(24), p.4925. doi: https://doi.org/10.3390/electronics12244925
- Hosna, A., Merry, E., Gyalmo, J., Alom, Z., Aung, Z. and Azim, M.A. (2022). Transfer learning: a friendly introduction. *Journal of Big Data*, 9(1). doi: https://doi.org/10.1186/s40537-022-00652-w
- Jena, B., Saxena, S., Nayak, G.K., Saba, L., Sharma, N. and Suri, J.S. (2021). Artificial intelligence-based hybrid deep learning models for image classification: The first narrative review. *Computers in Biology and Medicine*, 137, p.104803. doi: https://doi.org/10.1016/j.compbiomed.2021.104803 .
- Liu, Y., Pu, H. and Sun, D.-W. (2021). Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices. *Trends in Food Science & Technology*, 113, pp.193–204. doi: https://doi.org/10.1016/j.tifs.2021.04.042 .
- Mascarenhas, S. and Agarwal, M. (2021). *A comparison between VGG16, VGG19 and ResNet50 architecture frameworks for Image Classification*. [online] IEEE Xplore. doi: https://doi.org/10.1109/CENTCON52345.2021.9687944 .

- Md Taimur Ahad, Li, Y., Song, B. and Bhuiyan, T. (2023). Comparison of CNN-based deep learning architectures for rice diseases classification. *Artificial Intelligence in Agriculture*, 9, pp.22–35. doi: https://doi.org/10.1016/j.aiia.2023.07.001 .
- Negi, A., Kumar, K. and Chauhan, P. (2021). Deep Neural Network‐Based Multi‐Class Image Classification for Plant Diseases. *Agricultural Informatics*, pp.117–129. doi: https://doi.org/10.1002/9781119769231.ch6 .
- Purwono, P., Ma'arif, A., Rahmaniar, W., Fathurrahman, H.I.K., Frisky, A.Z.K. and Haq, Q.M. ul (2023). Understanding of Convolutional Neural Network (CNN): A Review. *International Journal of Robotics and Control Systems*, 2(4), pp.739–748. doi: https://doi.org/10.31763/ijrcs.v2i4.888 .
- Reyad, M., Sarhan, A.M. and Arafa, M. (2023). A modified Adam algorithm for deep neural network optimization. doi: https://doi.org/10.1007/s00521-023-08568-z .
- Riehl, K., Neunteufel, M. and Hemberg, M. (2023). Hierarchical confusion matrix for classification performance evaluation. *Applied statistics*. doi: https://doi.org/10.1093/jrsssc/qlad057 .
- S, R., Bharadwaj, A.S., S K, D., Khadabadi, M.S. and Jayaprakash, A. (2022). *Digital Implementation of the Softmax Activation Function and the Inverse Softmax Function*. [online] IEEE Xplore. doi: https://doi.org/10.1109/I4C57141.2022.10057747 .
- Salvi, M., Acharya, U.Rajendra., Molinari, F. and Meiburger, Kristen.M. (2020). The impact of pre- and post-image processing techniques on deep learning frameworks: a comprehensive review for digital pathology image analysis. *Computers in Biology and Medicine*, 128, p.104129. doi: https://doi.org/10.1016/j.compbiomed.2020.104129 .
- Savov, P., Tuecking, L.-R., Windhagen, H., Ehmig, J. and Ettinger, M. (2021). Imageless robotic handpiece-assisted total knee arthroplasty: a learning curve analysis of surgical time and alignment accuracy. *Archives of Orthopaedic and Trauma Surgery*, 141(12), pp.2119–2128. doi: https://doi.org/10.1007/s00402-021-04036-2 .
- Shorten, C., Khoshgoftaar, T.M. and Furht, B. (2021). Text Data Augmentation for Deep Learning. *Journal of Big Data*, [online] 8(1). doi: https://doi.org/10.1186/s40537-021-00492-0 .

8. Appendices

```
% Accuracy Calculation
accuracy = sum(predictedLabels == trueLabels) / numel(trueLabels) * 100;
fprintf('Test Accuracy: %.2f%%\n', accuracy);

% F1 Score Calculation
confMat = confusionmat(trueLabels, predictedLabels);
numClasses = size(confMat,1);
precision = diag(confMat) ./ sum(confMat, 2);
recall = diag(confMat) ./ sum(confMat, 1)';
F1 = 2 * (precision .* recall) ./ (precision + recall);
F1(isnan(F1)) = 0;
meanF1 = mean(F1);
fprintf('Mean F1 Score: %.2f%%\n', meanF1 * 100);

% Save Accuracy & F1 to File
fid = fopen('results.txt', 'w');
fprintf(fid, 'Test Accuracy: %.2f%%\n', accuracy);
fprintf(fid, 'Mean F1 Score: %.2f%%\n', meanF1 * 100);
fclose(fid);
```

Figure 15 F1-Score Code section

```
classes = categories(trueLabels);
classAcc = zeros(numel(classes), 1);
for i = 1:numel(classes)
    idx = trueLabels == classes{i};
    classAcc(i) = sum(predictedLabels(idx) == trueLabels(idx)) / sum(idx) * 100;
end
figure;
bar(classAcc);
xticklabels(classes);
xtickangle(45);
ylabel('Accuracy (%)');
title('Per-Class Accuracy');
grid on;
saveas(gcf, 'per_class_accuracy.png');
```

Figure 16 Per Class Accuracy Plot : Code

```
trueIdx = grp2idx(trueLabels);
figure;
plotroc(ind2vec(trueIdx'), scores');
title('ROC Curve');
saveas(gcf, 'roc_curve.png');

% Random Predictions
figure;
perm = randperm(numel(imdsTest.Files), 6);
for i = 1:6
    subplot(2,3,i);
    img = readimage(imdsTest, perm(i));
    label = classify(net, img);
    imshow(img);
    title(['Predicted: ', char(label)]);
end
sgtitle('Random Test Predictions');
saveas(gcf, 'random_predictions.png');
```

Figure 17 ROC and Random prediction Images : Code