Project Implementation Report

CSG2341 – Intelligent Systems

Jordan Farrow (10653054)

Ming Ren Leong (10652365)

26 October, 2025

Table of Contents

1. Summary

The goal of this project was to create a learning model capable of accurate abdominal organ classifications, minimally impacted by unintended artifacts. A random forest (RF) and convolutional neural network (CNN) were implemented with figure 1's steps.

All images and label files were loaded and pre-processed, removing black pixel noise and resizing to 32x32 (RF) or 128x128 (CNN). The 32x32 pre-processed images had features extracted using Histogram of gradients (HOG) for RF training. Both models were hyperparameter tuned, with a grid search (RF) and hand-tuning (CNN). Model's were then trained using the pre-processed images (CNN) or extracted HOG features (RF) of the training dataset; this was followed by predicting the labels of the validation dataset. Finally, both models were evaluated individually using classification reports, confusion matrices and learning curve graphs, followed by a comparison.

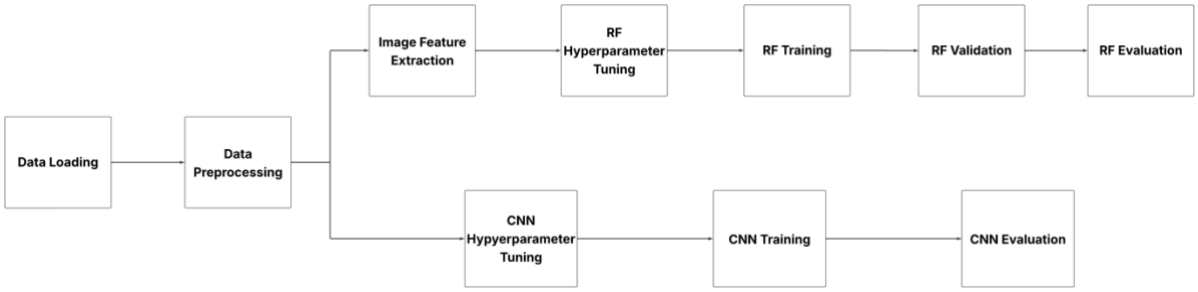
The CNN classified with a 99% accuracy, 6% higher than the RF (93%). In addition, it achieved greater precision and recall values, indicating greater organ classification ability. Despite this, the CNN achieved an 89% accuracy with the Kaggle test dataset, showing the model was unable to generalise to unseen data.

Implementation differed from the project proposal; tensorflow replaced pytorch for CNN creation due to integrated metric tracking and prior experience with it from similar Python exercises. In addition, different HOG extraction and image size configurations were untested, instead prioritising time into achieving higher CNN metrics.

2. Solution Flowchart

Figure 1

Flowchart of Implemented Steps



3. Implementation Steps

3.1. Data Loading (Performed by Ming & Jordan)

Input: folder paths for train, validation and test datasets; csv files for train and validation image labels. Output: image file paths loaded with corresponding labels. Figure 2 shows the label file is loaded and image paths of a dataset extracted. Image paths are iterated through, assigning the correct label. Source is part of a large function in appendix A.

Figure 2

Data Loading Pseudocode
Create array storing subset labels
With panda, read csv file
Extract all image paths of parent folder, stored as a list
For each image path, loop:
Load raw image and extract file name
    Locate file name within loaded csv

Append label to end of label array

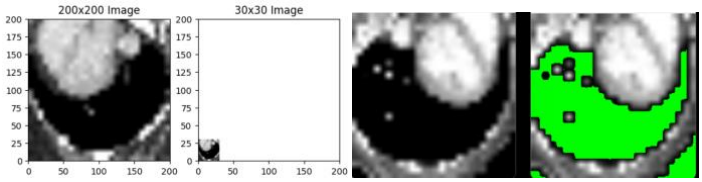
Data loading implementation used python 3.12.11 with Pandas 2.3.2, OpenCV 4.12.0.88, pathlib and imutils 0.5.4 packages; computed with a Ryzen 9800x3d and 64gb RAM, using Windows 11 in anaconda environment.

3.2. Data Preprocessing (Performed by Jordan)

Input: raw 200x200 grayscale images from training, validation and test dataset. Output: grayscale images resized to 32x32 for RF and 128x128 for CNN with black pixels masked. Tuned parameters: image resize dimensions = 32x32 (RF), 128x128 (CNN); pixel masking = RGB(0,0,0). To reduce image size, neighbouring pixels values were averaged, producing 32x32 or 128x128 pixels. Afterwards, black pixels were removed by changing their binary values to 0 and removed from images (GeeksforGeeks, 2025). The results are seen in figure 3, with source code in appendix A.

Figure 3

Resizing & Black Pixel asking Image Processing Results



This was implemented using python 3.12.11 with modules Pandas 2.3.2, OpenCV 4.12.0.88, pathlib, imutils 0.5.4 and numpy 2.2.6 for image loading and preprocessing. Source code was ran on a Ryzen 9800x3d and 64gb RAM, performed on Windows 11 and an anaconda environment.

3.3. Image Feature Extraction (Performed by Ming & Jordan)

Input: 32x32 masked grayscale training, validation and test images. Output: Array of vectors with image HOG features. Tuned parameters: orientations = 9, pixels_per_cell_block = (8, 8), cells_per_block = (2, 2), block_norm = 'L2-HYS', visualize = False, feature_vector =True. To extract HOGs from an image, it was first divided into 8x8 pixel cells (Kadota et al., 2009; Sai et al., 2023). Within the cells, the horizontal and vertical gradients of each pixel was calculated using the formulas (Kadota et al., 2009; Sai et al., 2023):

$$f_x(x,y)=f(x+1,y)-f(x-1,y)$$
$$f_y(x,y)=f(x,y+1)-f(x,y-1)$$

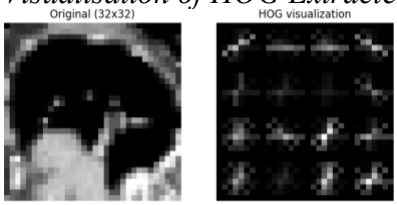
For each gradient, direction was calculated (Kadota et al., 2009; Sai et al., 2023):

$$m(x,y)= \text{sqrt}\ (f_x(x,y)^2+f_y(x,y)^2)$$
$$\theta(x,y)= \arctan\ (f_y(x,y) / f_x(x,y))$$

placed within a histogram, becoming that cell's HOG (Kadota et al., 2009; Sai et al., 2023). Each histogram shows the distribution of gradient orientations, providing greater weightings to more frequent orientations (Kadota et al., 2009; Sai et al., 2023). Finally, neighbouring cell histograms within the 2x2 block were normalised, entering the vector (Kadota et al., 2009; Sai et al., 2023). The results are visually seen in figure 4.

Figure 4

Visualisation of HOG Extracted Features of Image



The source code was part of larger function in appendix A, using python 3.12.11 with Pandas 2.3.2, OpenCV 4.12.0.88, pathlib, imutils 0.5.4 and numpy 2.2.6 packages. Execution used a Ryzen 9800x3d CPU and 64gb RAM, using Windows 11 and anaconda environments.

3.4. RF Hyperparameter Tuning (Performed by Jordan)

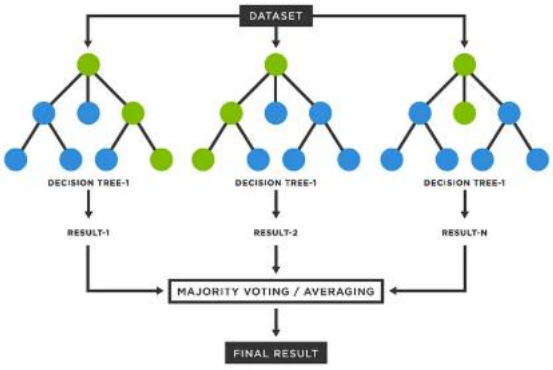
Input: array of HOG vectors from training dataset; grid parameters: Max_depth = [50], N_estimators = [400, 800, 1000, 1200], Max_features = [sqrt], Bootstrap = True, False. Output: Hyperparameter tuned RF using training HOG vectors. Tuned parameters: Max_depth = 50, N_estimators = 1000, Max_features = sqrt, Bootstrap = false. This was performed using a grid search; all parameter combinations of the grid were used to train the RF and compared (Hastie et al., 2009). For each combination, training data was split into 3 cross-validation folds. For 3 iterations, each fold validated the two used for training; the 3 scores were averaged, producing an accuracy score for the parameter combination. Appendix B source code was implemented using python 3.12.11 with packages scikit-learn 1.7.1 for model training and joblib 1.5.2 to use all processor cores; this was ran on a Ryzen 9800x3d and 64gb RAM, performed on Windows 11 and an anaconda environment. The output of the source code is seen in appendix C and D.

3.5. RF Training (Performed by Ming & Jordan)

Input: Tuned RF model and array containing training dataset HOG features. Output: Tuned RF model trained for organ classification. Tuned parameters: Max_depth = 50, N_estimators = 1000, Max_features = sqrt, Bootstrap = false. To form a tree, the square root of 324 image HOG features are randomly selected, choosing the feature with best gain for classification (Russel et al., 2022). The chosen feature is assigned to the node, with uniform distributions of n being randomly selected and compared as threshold values (Russel et al., 2022). The threshold splits the features into subsets for node children, repeating the above steps until assigned max depth is reached; this results in trees similar to figure 5. Through random selection, diverse prediction paths are created (Genuer & Poggi, 2020; Russel et al., 2022). The source code for model training is seen in appendix E.

Figure 5

Random Forest Model Exemplar (Data Science Dojo, 2024)



This was implemented using python 3.12.11 with packages scikit-learn 1.7.1 and joblib 1.5.2; execution occurred on a Ryzen 9800x3d CPU and 64gb RAM, using Windows 11 and anaconda environment.

3.6. RF Validation (Performed by Jordan)

Input: Tuned RF model trained on HOG training dataset; array of HOG validation dataset images. Output: Label predictions for validation dataset. Tuned parameters: Max_depth = 50, N_estimators = 1000, Max_features = sqrt, Bootstrap = false. Using the developed trees, images were classified using similar pseudocode below:

Extract HOG features from image
Set currentNode to root node
Set currentImageFeature to HOG feature matching currentNode value
Whilst currentNode is not a leaf node, continue the following:
    IF currentImageFeature value is > current node threshold, THEN traverse to left child (set currentNode to currentNode's left child)
    ELSE traverse to right child (set currentNode to currentNode's right child)
Give prediction to majority vote
Majority vote across all trees

The source code is shown in appendix F, implemented using python 3.12.11 with packages scikit-learn 1.7.1 joblib 1.5.2. Source code was ran using a Ryzen 9800x3d CPU and 64gb RAM, performed on Windows 11 and anaconda environment.

3.7. RF Evaluation (Performed by Jordan)

Input: Trained RF model; array of HOG features extracted from validation dataset. Output: Classification report containing precision, recall and accuracy scores; confusion matrix predicted labels. Tuned parameters: Max_depth = 50, N_estimators = 1000, Max_features = sqrt, Bootstrap = false. To analyse the RF's performance, a classification report and confusion matrix was produced. The classification report included precision and recall scores for each label, in addition to total accuracy of the model. The formula for calculating each metric is shown below (Russell et al., 2022):

Accuracy	Precision	Recall
$\frac{Total\ correct\ predictions}{Total\ no.predictions}$	$\frac{True\ positives}{True\ positives + False\ positives}$	$\frac{True\ positive +}{True\ positives + false\ negatives}$

The confusion report showed for each label, the frequency of predictions that were classified as each of the labels. This was used to determine which images were misclassified more frequently as another. Implementation used appendix G source code with python 3.12.11 and scikit-learn 1.7.1 package; it was ran with a Ryzen 9800x3d CPU and 64gb RAM, performed on Windows 11 within an anaconda environment.

3.8. CNN Hyperparameter Tuning (Performed by Jordan)

Input: pre-processed test and validation image subsets. Output: hyperparameter tuned CNN model achieving 0.056 validation loss and 99% validation accuracy when trained. Tuned parameters: Image resize = 128x128; Convolutional layers = 3x3 kernel with 16 → 32 → 64 → 128 filters and ReLU activation; Pooling layers = after each convolutional layer; Dense layer 1 = 128 neurons, relu activation and 1e-8 regularizer; Dropout = 0.4; Dense layer 2 = 11 classes, softmax activation. Hyperparameter tuning was performed with hand-tuning, repeatedly implementing different CNN layer and parameter combinations to achieve low validation loss and high accuracy. The source code for the tuned model is shown in appendix

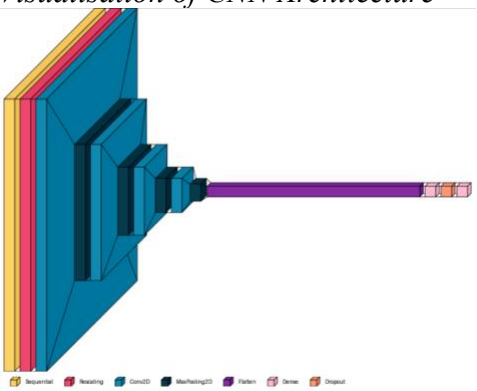
I, implemented using python 3.12.11 with module tensorflow 2.18.1. Source code was ran on a Ryzen 9800x3d and 64gb RAM, performed on Windows 11 within an anaconda environment.

3.9. CNN Training (Performed by Jordan)

Input: pre-processed test and validation image subsets. Output: hyperparameter tuned and trained CNN model achieving 0.056 validation loss and 99% validation accuracy. Tuned parameters: Image resize = 128x128; Convolutional layers = 3x3 kernel with 16 → 32 → 64 → 128 filters and ReLU activation; Pooling layers = after each convolutional layer; Dense layer 1 = 128 neurons, relu activation and 1e-8 regularizer; Dropout = 0.4; Dense layer 2 = 11 classes, softmax activation.

Figure 6

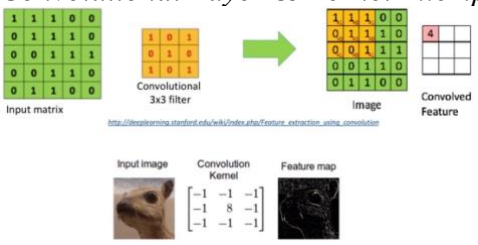
Visualisation of CNN Architecture



The CNN architecture can be seen in figure 6. Within each convolutional layer, a 3x3 kernel (9-pixel area) slides across the image, multiplying the values of each pixel to a section of the kernel; the values are then summed, producing a convolved feature as shown in figure 7 (Abu-Khalaf, n.d.b; O'shea & Nash, 2015). After sliding across the image, the result is a feature map seen similarly to figure 7. Whilst using the kernel, edge pixels are viewed less therefore, padding is used to reduce the biased weighting of inner pixels being seen more frequently (Abu-Khalaf, n.d.b; Dumoulin & Visin, 2018).

Figure 7

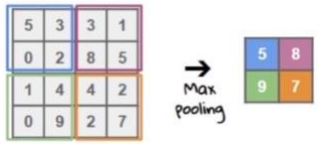
Convolutional Layer & Kernel Exemplar (Abu-Khalaf, n.d.b)



Pooling is then used after each convolutional layer, choosing the maximum value within rectangular neighbours seen in figure 8; this reduces the size of feature maps whilst retaining its critical information (Abu-Khalaf, n.d.b; Zhao & Zhang, 2024).

Figure 8

Max Pooling Exemplar (Abu-Khalaf, n.d.b)



After all convolutional layers, the final feature map is flattened and processed through 128 neurons based on their weightings (Vianello et al., 2019). 40% of the neuron paths are turned off, with regularizers used to prevent overfitting by penalising large weights (Abu-Khalaf, n.d.b).

The model then uses the validation image dataset, passing through the convolutional, pooling and dense layers; the feature map is then passed through the trained neurons (Ding et al., 2024). After processing, the softmax determines the probability of the image to each label (Reng & Wang, 2023). After all validation images are processed, a validation loss and accuracy is produced. The weightings created within the previous epoch are then passed to the next epoch for further tuning in an attempt reduce loss (Azam et al., 2024).

The source code for the CNN training is shown in appendix J, implemented using python 3.12.11 with the tensorflow 2.18.1 package, running on a Ryzen 9800x3d and 64gb RAM, performed on Windows 11 within an anaconda environment.

3.10. CNN Evaluation (Performed by Jordan)

Input: Trained CNN model and pre-processed validation dataset images. Output: Classification report containing precision, recall and accuracy scores; confusion matrix of all predicted labels; Learning curves showing evolving validation loss and accuracy across epochs. Tuned parameters: Image resize = 128x128; Convolutional layers = 3x3 kernel with 16 → 32 → 64 → 128 filters and ReLU activation; Pooling layers = after each convolutional layer; Dense layer 1 = 128 neurons, relu activation and 1e-8 regularizer; Dropout = 0.4; Dense layer 2 = 11 classes, softmax activation. For evaluation, a classification report, confusion matrix and learning curve graph was produced. The learning curve graph showed the performance of the CNN model over multiple epochs, with lower validation loss implying the model was more confident and correct in its predictions of the validation dataset. The classification report included precision and recall scores for each label, with the total accuracy of the model. The formula for calculating each metrics is shown below (Russell et al., 2022):

Accuracy	Precision	Recall
$\frac{Total\ correct\ predictions}{Total\ no.predictions}$	$\frac{True\ positives}{True\ positives + False\ positives}$	$\frac{True\ positive +}{True\ positives + false\ negatives}$

The confusion report showed for each label, the frequency of predictions that were classified as each of the labels. This was used to determine which images were misclassified more frequently as another. The evaluation source code is shown in appendix K, implemented using python 3.12.11 with tensorflow 2.18.1, numpy 2.2.6, matplotlib 3.10.6 and scikit-learn 1.7.1 packages. It was executed using a Ryzen 9800x3d and 64gb RAM, using Windows 11 within an anaconda environment.

4. Performance Evaluation

The 'IS_2025_OrganAmnist' dataset was accessed through Canvas, unzipped and loaded into a parent folder with the source code. Images and label files were checked for corruption and naming misaligned to the image file names. The predefined dataset contained 58,830 CT scan images separated into training validation and test subsets containing 34,561 (58.75%), 6,491 (11.03%) and 17,778 (30.22%) images. Table 1 shows multiple organs had lower training frequencies, causing the limited learning of adreal gland, bladder, brain and breast organs (Johnson & Khoshgoftaar, 2019). In addition, multiple organs had larger validation frequencies, causing less sensitive metrics scores to successes and failures.

Table 1

Organ Image Frequencies in Test & Validation Subset

Organ	Testing Subset Freq.	Validation Subset Freq.
Andreal Gland	1,956	321
Bladder	1,390	233
Brain	1,357	225
Breast	1,474	392
Colon	3,963	568
Esophagus	3,817	637
Eye	6,164	1,033
Kidney	3,919	1,033
Liver	3,929	1,009
Lung	3,031	529
Ovary	3,561	511

The random seed for both models was fixed to '10653054', ensuring reproducibility and models unimpacted by randomness.

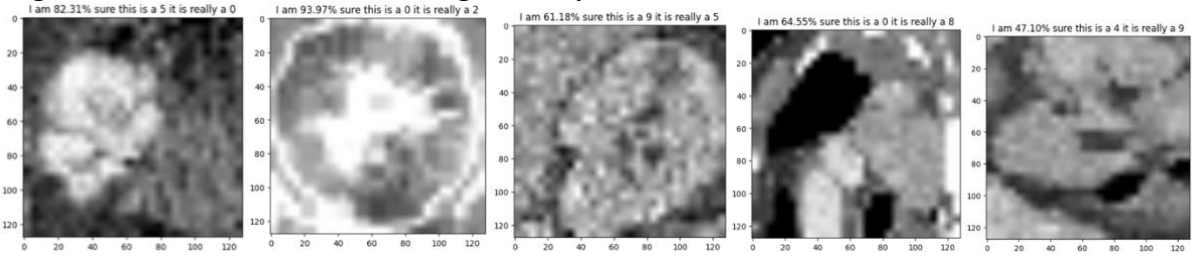
To evaluate model performance, the following was used; accuracy measured the percentage of correct predictions against all predictions performed, determining the model with overall better classification capability (Russel et al., 2022). Precision measured the frequency of correct predictions against the number of incorrect predictions of a label (Russel et al., 2022), used to find which model avoided false positives. Recall measured the frequency of correct predictions against the number of actually true classifications meant for a label (Russel et al., 2022), used to find which model achieved true positives. Through precision and recall, the models were evaluated in its ability to overcome the imbalanced dataset, with each label individually evaluated. Although accuracy provided less evaluative capability, it was used to determine which model overall produced greater image classification.

4.1. RF Training and Validation Evaluation

This model achieved in table 2 a 93% accuracy in classifying the validation dataset, misclassifying 1,245 (7%) images. 252 images are misclassified as organ 6 in figure 11, resulting in a 0.8 precision score; this is followed by organs 10 and 4, which had 74 and 82 misclassifications respectively. These classifications would have occurred from organs 0, 1, 2, 5, 8 and 9 due to their recall values in table 2. Due to class imbalance, organs 8 and 9 had similar misclassification amounts but different recall values, impacted by frequency sensitivity. The misclassified organs in figure 9 are similar in shape and noise; this would have impacted feature extraction. Limited HOG feature extraction in figure 4 would also have impacted model learning.

Figure 9

Organ 0, 2, 5, 8 & 9 Validation Images classified with RF model



Max precision score for organs 3 and 8 were achieved, showing the model effectively learnt the unique organ characteristics in figure 11 and table 2; the recall scores are lower, likely caused by image noise, orientations HOG features unseen during training. Organs 0, 1, 2, 7 and 9 have similar precision scores, showing the model is somewhat capable in classifying organs. All organ 6 images were correctly classified as seen in figure 10 possibly caused by its unique structure.

Figure 10

Organ 6 High Confidence Predictions

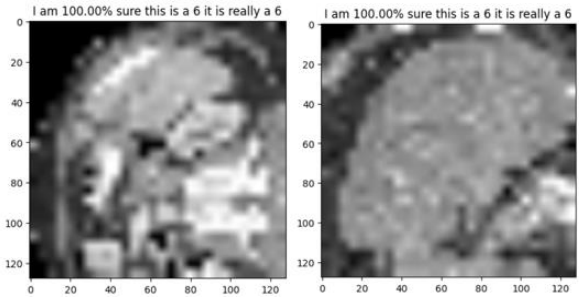


Table 2

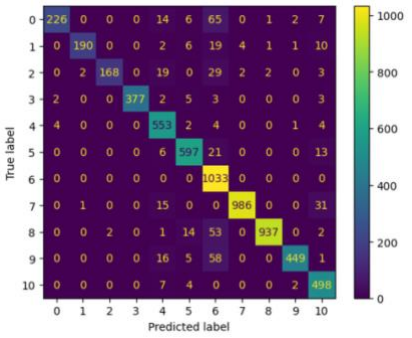
RF Model Classification Report

Clasification report:

	precision	recall	f1-score	support
0	0.97	0.70	0.82	321
1	0.98	0.82	0.89	233
2	0.99	0.75	0.85	225
3	1.00	0.96	0.98	392
4	0.87	0.97	0.92	568
5	0.93	0.94	0.94	637
6	0.80	1.00	0.89	1033
7	0.99	0.95	0.97	1033
8	1.00	0.93	0.96	1009
9	0.99	0.85	0.91	529
10	0.87	0.97	0.92	511
accuracy			0.93	6491
macro avg	0.95	0.90	0.91	6491
weighted avg	0.94	0.93	0.93	6491

Figure 11

RF Model Confusion Matrix

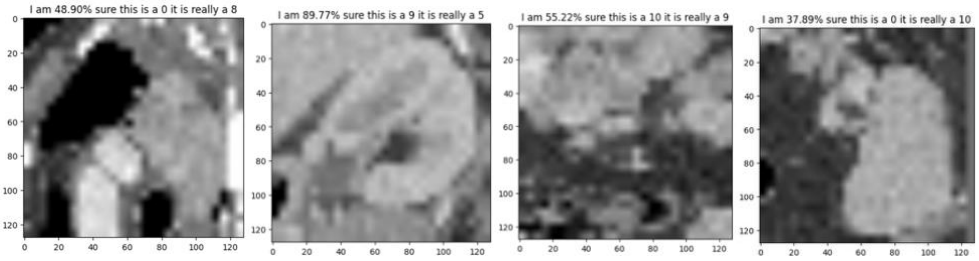


4.2. CNN Training and Validation Evaluation

Figure 13 shows the model achieving high validation accuracy and low loss within initial training epochs; the model is overfitting, training effectively for high validation accuracy and neglecting generalisation for unseen datasets. A 99% validation accuracy was achieved, misclassifying 477 (2.68%) images in table 3 and figure 14. Organs 0, 5, 8, 9 and 10 were primarily misclassified as organs 4, 5, 6 and 10, causing lower precision and recall scores. The model varies in confidence across the organs in figure 12.

Figure 12

Organ's 8, 5, 9 and 10 Validation Dataset Predictions from CNN



Full 2, 7 and 8 organ precision scores show the model is capable of distinguishing unique organ characteristics; all 6, 7 and 1 organ images were also correctly classified. Other organs share similar values, showing strong model use in classifying validation dataset organs.

Figure 13

CNN Accuracy and Loss Learning Curve Throughout Epochs

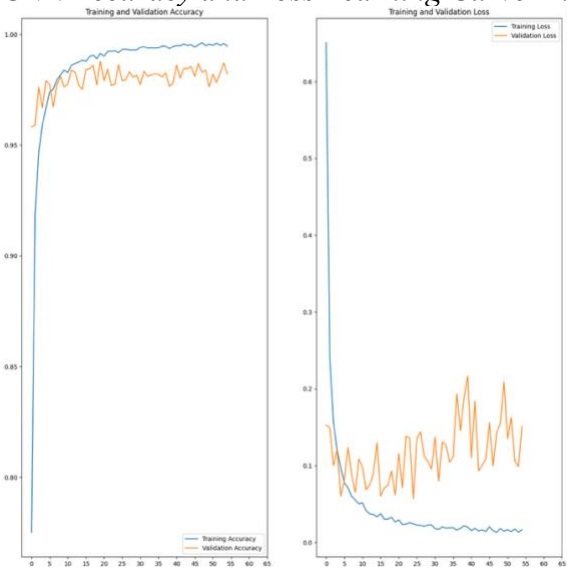


Table 3

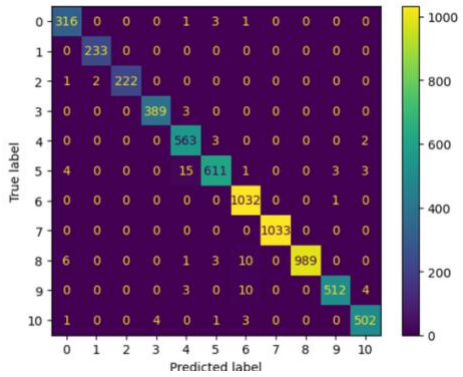
CNN Classification Report

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	321
1	0.99	1.00	1.00	233
2	1.00	0.99	0.99	225
3	0.99	0.99	0.99	392
4	0.96	0.99	0.98	568
5	0.98	0.96	0.97	637
6	0.98	1.00	0.99	1033
7	1.00	1.00	1.00	1033
8	1.00	0.98	0.99	1009
9	0.99	0.97	0.98	529
10	0.98	0.98	0.98	511
accuracy			0.99	6491
macro avg	0.99	0.99	0.99	6491
weighted avg	0.99	0.99	0.99	6491

Figure 14

CNN Confusion Matrix



4.3. Model Comparison

Table 4

CNN & RF Precision, Recall and Accuracy Scores

	Precision		Recall		Accuracy	
	CNN	RF	CNN	RF	RF	CNN
0	0.96	0.97	0.98	0.70		
1	0.99	0.98	1.00	0.82		
2	1.00	0.99	0.99	0.75		
3	0.99	1.00	0.99	0.96		
4	0.96	0.87	0.99	0.97		
5	0.98	0.93	0.96	0.94	0.93	0.99
6	0.98	0.80	1.00	1.00		
7	1.00	0.99	1.00	0.95		
8	1.00	1.00	0.98	0.93		
9	0.99	0.99	0.97	0.85		
10	0.98	0.87	0.98	0.97		

The CNN was the greatest performing model, correctly classifying 6% more images than the RF. In addition, the CNN achieves greater precision and recall scores across all organs, showing it could better distinguish between organ characteristics and correctly classify more validation dataset images. Two instances exist where the RF achieved better results; precision values for organs 0 and 3 were 0.01 greater than the CNN. This was minor thus, disregarded. The CNN may have outperformed the RF due to the non-reliance of HOG feature vectors which captured little detail of organs as seen in figure 4. In addition, the CNN had multiple feature extraction layers, considering large and small features where the RF may have not.

4.4. Kaggle Comparison

Figure 15

CNN Performance with Kaggle Test Dataset

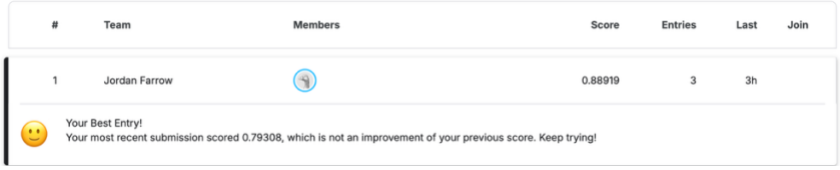


Figure 15 shows the CNN model achieved an 89% accuracy score with an unseen test dataset, 10% lower than the validation accuracy. This could occur due to unseen image noise or organ orientations which the model was not trained on; the model was also seen overfitting in figure 13, reducing generalisation and impacting predictions with test datasets. With further parameter tuning, it is believed the CNN can achieve greater generalisation.

5. Appendix

Appendix A

Data Loading, Pre-Processing and Feature Extraction Function Source Code

```
# extract Histogram of Gradients
def extract_histogram_of_Grad(image):
    # convert the image to grayscale (if it isn't already)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Compute HOG features
    hog_features = hog(
        gray_image,
        orientations=9,
        pixels_per_cell=(8, 8),
        cells_per_block=(2, 2),
        block_norm='L2-Hys',
        visualize=False,
        feature_vector=True
    )
    return hog_features

# extract features of all test & validation images + labels in a path (also reduces size - 32x32
-> hog)
def imageFeatureExtraction(imagePaths, labelPaths, labelFlag, extractionFlag):

    # arrays to store image + label extraction
    hog_features = []
    images = []
    labels = []

    # reads label files for train & val subset
    if labelPaths != False:
        #allows excel table to be read
        df = pd.read_csv(labelPaths)
        df.set_index('file', inplace=True)

    # gets all image paths of subset
    imagePaths = list(paths.list_images(imagePaths))

    #loop over images
    for (i, imagePath) in enumerate(imagePaths):
        image = cv2.imread(imagePath) #load raw image
        if extractionFlag == True:
            image = cv2.resize(image, (32, 32)) #resize image from 200x200 to 32x32 (RF)
        else:
            image = cv2.resize(image, (128, 128)) #resize image from 200x200 to 128x128 (CNN)


        # gets labels for train & val subset images
```

```
        if labelFlag == True:
            fileLabel = pathlib.PurePath(imagePath).name # extract name of file
            if "-checkpoint" in fileLabel: # jupyter adds 'checkpoint' files. this stops that from
messing with function.
                continue
            label = df.loc[fileLabel, 'label'] # with file name, find organ label
            labels.append(label)

        # removes background noise in image (black pixels; masking out)
        tempImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        __, mask = cv2.threshold(tempImage, 1, 255, cv2.THRESH_BINARY) # creates
background mask
        image = cv2.bitwise_and(image, image, mask=mask) # masks background

        if extractionFlag == True:
            # extracts feature of image (Histogram of Gradients) for RF
            hog_f=extract_histogram_of_Grad(image)
            hog_features.append(hog_f)
        else:
            # appends image for CNN
            images.append(image)

        # show progress every 1000 images
        if i > 0 and i % 1000 == 0:
            print("processed {}/{}".format(i, len(imagePaths)))


    if extractionFlag == True:
        return np.array(hog_features), np.array(labels)
    else:
        return np.array(images)

#training subset image loading,pre-processing & feature extraction | Label extraction
print('\nTraining subset loading, pre-processing and feature extraction \n')
hog_train_features, labels_train =
imageFeatureExtraction('IS_2025_OrganAMNIST/train/images_train/',
"IS_2025_OrganAMNIST/train/labels_train.csv", True, True)

#CNN train images
print('\nTraining CNN image preprocessing \n')
train_CNN_Images =
imageFeatureExtraction('IS_2025_OrganAMNIST/train/images_train/',
"IS_2025_OrganAMNIST/train/labels_train.csv", True, False)

#validation subset image loading,pre-processing & feature extraction | Label extraction
print('\nValidation subset loading, pre-processing and feature extraction \n')
hog_val_features, labels_val =
imageFeatureExtraction('IS_2025_OrganAMNIST/val/images_val/',
'IS_2025_OrganAMNIST/val/labels_val.csv', True, True)
```

```
#CNN validation images
print('\nValidation CNN image preprocessing \n')
val_CNN_Images = imageFeatureExtraction('IS_2025_OrganAMNIST/val/images_val/',
'IS_2025_OrganAMNIST/val/labels_val.csv', True, False)

#test subset image loading,pre-processing & feature extraction
print('\nTest subset loading, pre-processing and feature extraction \n')
hog_test_features = imageFeatureExtraction('IS_2025_OrganAMNIST/test/images',
False,False, True)

#CNN test images
print('\nTest CNN image preprocessing \n')
test_CNN_Images = imageFeatureExtraction('IS_2025_OrganAMNIST/test/images',
False,False, False)
```

Note: Used to extract images and their features. From Module 6 Workshop Exercise: Machine Learning by Abu-Khalaf, J (https://courses.ecu.edu.au/), Used to remove black pixel noise from images. From Python Exercise: Image Masking with OpenCV by Rosebrock, A (https://pyimagesearch.com/2021/01/19/image-masking-with-opencv/), Used to remove black pixel noise from images. From Python Exercise: Removing Black Background and Make Transparent using Python OpenCV by GeeksforGeeks (https://www.geeksforgeeks.org/python/removing-black-background-and-make-transparent-using-python-opencv/)

Appendix B

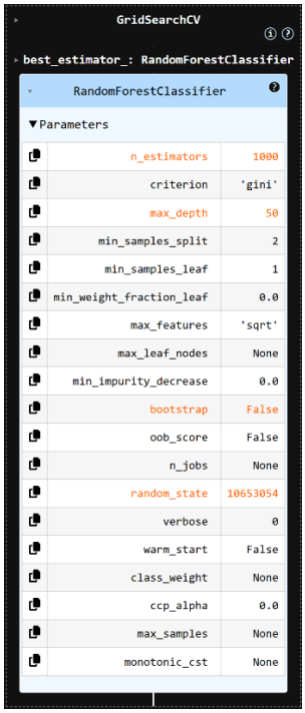
RF Grid Search Source Code

```
#Parameters to be used in grid search
RFModel = RandomForestClassifier(random_state = 10653054)
param_grid = {
    'max_depth': [50],
    'n_estimators': [400, 800, 1000, 1200],
    'max_features': ['sqrt'],
    'bootstrap': [True, False]
}

# Performing grid search
grid_search = GridSearchCV(estimator=RFModel, param_grid=param_grid, cv=3,
scoring='accuracy', n_jobs=-1)
grid_search.fit(hog_train_features, labels_train)

print('best parameters of grid search: ', grid_search.best_params_)
```

Note: Used to perform a grid search. From Python Exercise: Tuning Random Forest with Grid Search by Soni, P (https://www.blog.trainindata.com/random-forest-with-grid-search/).

Appendix C

RF Grid Search Output



Appendix D

RF Grid Search Output

```
best param:  {'bootstrap': False, 'max_depth': 50, 'max_features': 'sqrt', 'n_estimators': 1000}
best score:  0.8944186411207232
```

Appendix E

RF Training Source Code

```
#Random forest model using HOG extracted features
model = RandomForestClassifier(max_depth=50, n_estimators=1000, max_features='sqrt',
n_jobs=-1, bootstrap=False, random_state = 10653054)
rfHog = model
rfHog.fit(hog_train_features, labels_train) # Training Classifier
acc = rfHog.score(hog_val_features, labels_val)

print("hog  accuracy with Random Forset: {:.2f}%".format(acc * 100))
```

Note: Used for training the RF. From Module 6 Workshop Exercise: Machine Learning by Abu-Khalaf, J (https://courses.ecu.edu.au/).

Appendix F

RF Prediction Source Code

```
#RF classification report
rfHog_prediction = rfHog.predict(hog_val_features)
print( '\nClasification report:\n', classification_report(labels_val, rfHog_prediction))

#RF confusion matrix
confusion_matrix = metrics.confusion_matrix(labels_val, rfHog_prediction)
display_matrix = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
display_matrix.plot()
plt.show()
```

Note: Used for generating the classification report. From Module 6 Workshop Exercise: Machine Learning by Abu-Khalaf, J (https://courses.ecu.edu.au/), Used for producing a confusion matrix. From Python Exercise: Machine Learning – Confusion Matrix by W3Schools (https://www.w3schools.com/python/python_ml_confusion_matrix.asp)

Appendix G

RF Evaluation Source Code

```
#RF classification report
rfHog_prediction = rfHog.predict(hog_val_features)
print( '\nClasification report:\n', classification_report(labels_val, rfHog_prediction))


#RF confusion matrix
confusion_matrix = metrics.confusion_matrix(labels_val, rfHog_prediction)
display_matrix = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
display_matrix.plot()
plt.show()
```

Appendix H

RF Prediction Confidence Source Code

```
tmp = rfHog.predict(hog_val_features)

#prints confidence + prediction for 10 images
for i in range(10):
    image = val_CNN_Images[i]
    plt.imshow(image) #show image

    predicted_index = np.argmax(validationPredicting[i]) #gets predicted label
    confidence = validationPredicting[i][predicted_index] #confidence of predicted label
        plt.title('I am {:.2%} sure this is a '.format(confidence) + str(predicted_index) + ' it is
really a ' + str(labels_val[i]))
        plt.show()
```

Note: Used to view confidence for each image. From Module 9 Workshop Exercise: Convolutional Neural Networks (https://courses.ecu.edu.au/)

Appendix I

CNN Model Layers & Parameters Source Code

```
imgSize=128
rows = 128
columns = 128
num_classes = 11
epochs = 40
data_augmentation = keras.Sequential(
   [
   tf.keras.layers.RandomFlip("horizontal", input_shape=(rows, columns, 3)),
   ])

model = Sequential([
  data_augmentation,
  tf.keras.layers.Rescaling(1./255, input_shape=(rows, columns, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'), #convolutional 1
  layers.MaxPooling2D(), #pool 1
  layers.Conv2D(32, 3, padding='same', activation='relu'), #convolutional 2
  layers.MaxPooling2D(), #pool 2
  layers.Conv2D(64, 3, padding='same', activation='relu'), #convolutional 3
  layers.MaxPooling2D(), #pool 3
  layers.Conv2D(128, 3, padding='same', activation='relu'), #convolutional 4
  layers.MaxPooling2D(), #pool 4
  layers.Flatten(),
  layers.Dense(128, activation='relu',kernel_regularizer=regularizers.l2(1e-8)), #Neurons +
regulairzers
  layers.Dropout(0.4), #drop 40% of neurons
  layers.Dense(num_classes, activation='softmax') #predicts label
])

model.summary()

visualkeras.layered_view(model, legend=True) #visualises CNN shape

plot_model(model, show_shapes=True, show_layer_names=True)
```

Note: Used for creating the CNN model. From Module 9 Workshop Exercise: Convolutional Neural Networks by Abu-Khalaf, J (https://courses.ecu.edu.au/)

Appendix J
CNN Model Training Source Code

```
#runs CNN training
tf.random.set_seed(10653054)

model.compile(optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
        metrics=['accuracy'])

callback =tf.keras.callbacks.EarlyStopping(
   monitor='val_loss', min_delta=1e-6, patience=10, restore_best_weights=True)

history = model.fit(
  train_CNN_Images, labels_train,
  validation_data=(val_CNN_Images, labels_val),
  epochs=epochs,
  callbacks=[callback], shuffle = True
)
```

Appendix K
CNN Evaluation Source Code

```
#Validation accuracy & loss graphs
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)


plt.figure(figsize=(16, 16))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot( val_acc, label='Validation Accuracy')
plt.xticks(range(0,30))
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xticks(range(0,epochs)[0::2])
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

```
# Lowest validation loss achieved
val_loss = np.array(history['val_loss'])
print('Lowest validation loss: ', val_loss.min())
```

Note: Used for graphing epoch accuracy and loss. From Module 9 Workshop Exercise: Convolutional Neural Networks by Abu-Khalaf, J (https://courses.ecu.edu.au/)

Appendix L
Saving Newly Trained CNN Source Code

```
model.save('my_model.keras')

with open("history.pkl", "wb") as f:
    pickle.dump(history.history, f)

print('done')
```

Note: Used for saving the epoch history of trained CNN model. From Python Exercise: How to use Pickle to save and load Variables in Python? by GeeksforGeeks, Used for saving trained CCN model. From Module 9 Workshop Exercise: Convolutional Neural Networks by Abu-Khalaf, J (https://courses.ecu.edu.au/).

Appendix M
Saved CNN Loading Source Code

```
model = load_model('my_model.keras') #loads saved trained CNN

visualkeras.layered_view(model, legend=True) #visualises CNN shape

model.summary()
```

Note: *Used* for loading trained CNN model. From Python Exercise: How to use Pickle to save and load Variables in Python? by GeeksforGeeks (https://www.geeksforgeeks.org/python/how-to-use-pickle-to-save-and-load-variables-in-python/)

Appendix N
Saved CNN Evaluation Source Code

```
#loads CNN training history
with open("history.pkl", "rb") as f:
    history = pickle.load(f)

epochs = 70
acc = history['accuracy']
val_acc = history['val_accuracy']
loss = history['loss']
val_loss = history['val_loss']
epochs_range = range(epochs)

# Validation accuracy & loss graphs
plt.figure(figsize=(16, 16))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot( val_acc, label='Validation Accuracy')
plt.xticks(range(0, epochs, 5))
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xticks(range(0, epochs, 5))
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

# Lowest validation loss achieved
val_loss = np.array(history['val_loss'])
print('Lowest validation loss: ', val_loss.min())

# CNN classification report
validationPredicting = model.predict(val_CNN_Images)
labelPredictions = np.argmax(validationPredicting, axis=1)
print("\nClassification Report:\n")
print(classification_report(labels_val,labelPredictions))

# CNN confusion matrix
#https://www.w3schools.com/python/python_ml_confusion_matrix.asp
confusion_matrix = metrics.confusion_matrix(labels_val, labelPredictions)
display_matrix = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

display_matrix.plot()
plt.show()
```

Note: Used for graphing epoch accuracy and loss. From Module 9 Workshop Exercise: Convolutional Neural Networks by Abu-Khalaf, J (https://courses.ecu.edu.au/), Used for loading trained CNN training history. From Python Exercise: How to use Pickle to save and

load Variables in Python? by GeeksforGeeks (https://www.geeksforgeeks.org/python/how-to-use-pickle-to-save-and-load-variables-in-python/), Used for generating the classification report. From Module 6 Workshop Exercise: Machine Learning by Abu-Khalaf, J (https://courses.ecu.edu.au/), Used for producing a confusion matrix. From Python Exercise: Machine Learning – Confusion Matrix by W3Schools (https://www.w3schools.com/python/python_ml_confusion_matrix.asp)

Appendix O
CNN Prediction Confidence Source Code

```
validationPredicting = model.predict(val_CNN_Images)

#prints confidence + prediction for 10 images
for i in range(10):
    image = val_CNN_Images[i]
    plt.imshow(image)

    predicted_index = np.argmax(validationPredicting[i]) #gets predicted label
    confidence = validationPredicting[i][predicted_index] #confidence of predicted label
        plt.title('I am {:.2%} sure this is a '.format(confidence) + str(predicted_index) + ' it is
really a ' + str(labels_val[i]))
        plt.show()
```

Note: Used to view confidence for each image. From Module 9 Workshop Exercise: Convolutional Neural Networks (https://courses.ecu.edu.au/)

Appendix P
Kaggle Prediction Saving Source Code

```
# testing model with kaggle dataset
model = load_model('my_model.keras') # load trained CNN model
testPredicting = model.predict(test_CNN_Images) # predict CNN
labelPredictions = np.argmax(testPredicting, axis=1) #gets predicted label

#Places predictions
submission = pd.DataFrame({
    "index": range(len(labelPredictions)),
    "id": labelPredictions
})

submission.to_csv("submission.csv", index=False)
```

Note: Used for storing predictions in correct Kaggle competition format. From Python Exercise: Saving a Pandas Dataframe as a CSV by GeeksforGeeks (https://www.geeksforgeeks.org/pandas/saving-a-pandas-dataframe-as-a-csv/)

6. References

Abu-Khalaf, J. (n.d.a). *wk6.ipynb*. Canvas.

https://courses.ecu.edu.au/

Abu-Khalaf, J. (n.d.b). *wk9.ipynb*. Canvas.

https://courses.ecu.edu.au/

Azam, M., Sadman Sakib, Nur Mohammad Fahad, Abdullah Al Mamun, Md. Anisur Rahman, Swakkhar Shatabda, & Hossain, S. (2024). A systematic review of hyperparameter optimization techniques in Convolutional Neural Networks. *Decision Analytics Journal*, 11.

https://doi.org/10.1016/j.dajour.2024.100470

Data Science Dojo. (2024, August 22). Understanding the Random Forest Algorithm – A Comprehensive Guide. Retrieved September 18, 2025, from https://datasciencedojo.com/blog/random-forest-algorithm/

Ding, W., Abdel-Basset, M., Ali, A. M., & Moustafa, N. (2024). A survey of intelligent multimedia forensics for internet of things communications: Approaches, strategies, perspectives, and challenges for a sustainable future. *Engineering Applications of Artificial Intelligence*, 138. https://doi.org/10.1016/j.engappai.2024.109451

Dumoulin, V., & Visin, F. (2018). *A Guide to Convolution Arithmetic for Deep Learning*. https://arxiv.org/abs/1603.07285

GeeksforGeeks. (2025a, April 12). *Bitwise Operations on Binary Images in OpenCV2*. Retrieved October 20, 2025, from https://www.geeksforgeeks.org/python/arithmetic-operations-on-images-using-opencv-set-2-bitwise-operations-on-binary-images/

GeeksforGeeks. (2025b, July 23). *Removing Black Background and Make Transparent using Python OpenCV*. Retrieved October 1, 2025, from https://www.geeksforgeeks.org/python/removing-black-background-and-make-transparent-using-python-opencv/

GeeksforGeeks. (2025c, July 23). *How to use Pickle to save and load Variables in Python?*. Retrieved October 2, 2025, from https://www.geeksforgeeks.org/python/how-to-use-pickle-to-save-and-load-variables-in-python/

GeeksforGeeks. (2025d, July 11). *Saving a Pandas Dataframe as a CSV*. Retrived October 19, 2025, from https://www.geeksforgeeks.org/pandas/saving-a-pandas-dataframe-as-a-csv/

Genuer, R., & Poggi, J. M. (2020). *Random forests with R*. Springer. https://doi.org/10.1007/978-3-030-56485-8

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1). https://doi.org/10.1186/s40537-019-0192-5

Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R., & Nakamura, Y. (2009). hardware architecture for HOG feature extraction. *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. https://doi.org/10.1109/iih-msp.2009.216

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. https://arxiv.org/abs/1511.08458

Ren, J., & Wang, H. (2023). Chapter 3 – calculus and optimization. *Mathematical Methods in Data Science*, 51-89. https://doi.org/10.1016/B978-0-44-318679-0.00009-0

Rosebrock, A. (2021, January 19). *Image Masking with OpenCV.* PyImageSearch. Retrieved October 1, 2025, from https://pyimagesearch.com/2021/01/19/image-masking-with-opencv/

Russell, S. J., Norvig, P., Chang, M.-W., Devlin, J., Dragan, A., Forsyth, D., Goodfellow, I., Malik, J., Mansinghka, V., Pearl, J., & Wooldridge, M. J. (2022). *Artificial intelligence : a modern approach* (4th ed.). Pearson

Sai, A., Khan, M. K., Reddy, K. K., & Chand, G. S. (2023). Artistry in detail: mastering HOG feature extraction. *Journal of Advances in Computational Intelligence Theory*, 6(1), 25-30. https://doi.org/10.5281/zenodo.10421202

Soni, P. (2025, January 1). Tuning Random Forest with Grid Search. *Train in Data*. Retrieved October 2, 2025, from https://www.blog.trainindata.com/random-forest-with-grid-search/

Vianello, E., Perniola, L., & De Salvo, B. (2019). 15 – emerging memory technologies for neuromorphic hardware. *Advances in Non-Volatile Memory and Storage Technology* (2nd ed.). Woodhead Publishing, 585-602. https://doi.org/10.1016/B978-0-08-102584-0.00016-4

W3Schools. (n.d.a). *Machine Learning – Confusion Matrix*. Retrieved October 19, 2025, from https://www.w3schools.com/python/python_ml_confusion_matrix.asp

Zhao, L., & Zhang, Z. (2024). *A Improved Pooling Method for Convolutional Neural Networks*.

https://doi.org/10.1038/s41598-024-51258-6