THE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

WELLINGTON FACULTY OF ENGINEERING

COMP309- ML TOOLS AND TECHNIQUES

PROJECT REPORT

Student Name: LAM QUANG THINH
Student ID: 300538520

I. Introduction

The realm of computational vision presents a myriad of oppotunities and also challenges, with the accurate classification of visual entities standing at the forefront. While humans navigate these distinctions with ease, machines require intricate algorithms to emulate such capabilities (Reid, 2022). Deep convolutional neural networks (CNNs) have emerged as a potent solution, transforming our approach to image classification. Anchoring our project in this transformative technique, we take a focused dive into a specific classification conundrum: differentiating between images of cherries, tomatoes, and strawberries. Through building, training, and evaluating our models, including MLP, CNN, and pre-trained models, we explore the nuances of model tuning, aiming to achieve unparalleled precision in classifying these three unique fruit categories on unseen datasets.

II. Problem investigation

1. EDA

1.1 Random images

Figure 1.1 presents random visuals from each class of the train data, by checking its label, it is claimed that they are most likely to be stored correctly.

Figure 1.1 Random images

1.2 Image sizes

After loading the images for analysis, there is an obvious variability in their dimensions. The dataset contains images of diverse sizes, with over 30 unique dimensions identified. This variance in image sizes highlights the potential need for preprocessing steps, such as resizing or padding, to ensure consistency when feeding the images into machine learning models. Figure 1.2 illustrates the difference in image sizes of the data.

Figure 1.2 Examples of difference in size of images

1.3 Class distribution

Based on Figure 1.3, it is observed that the distribution of classes is well-balanced. Each class contains precisely 1,500 images, ensuring that no particular category dominates the dataset. Therefore, there is no need in applying balancing techniques to the data.

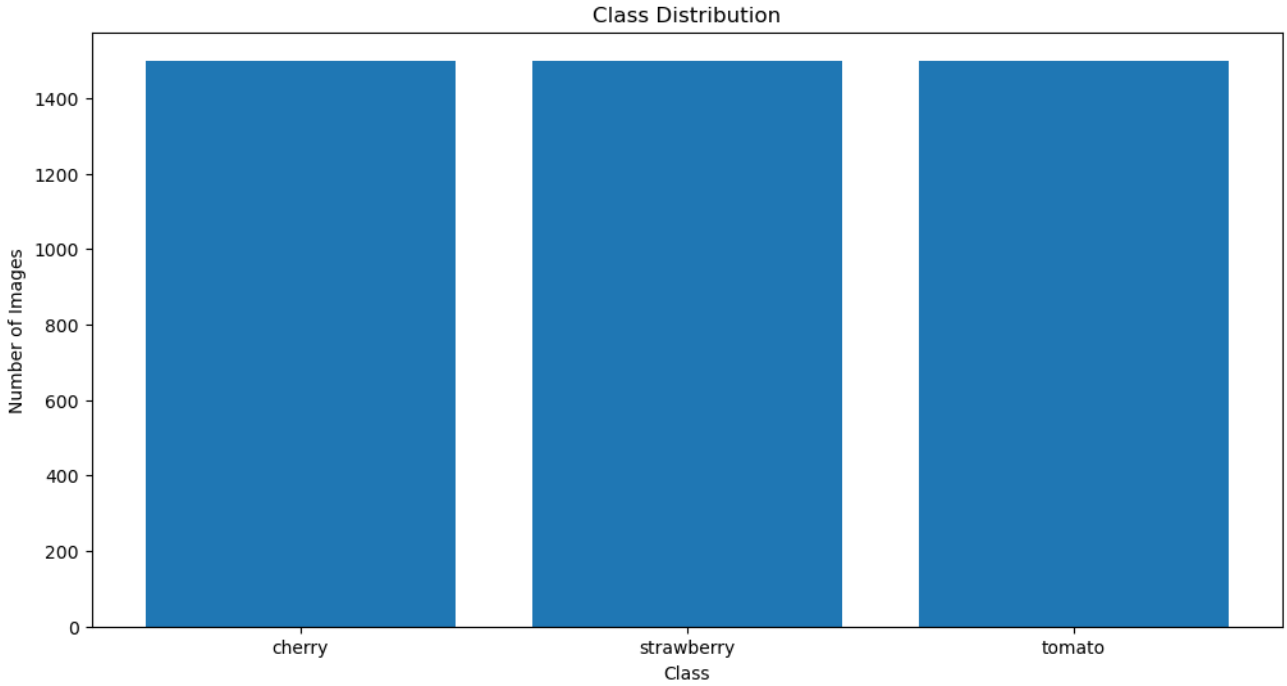

Figure 1.3 Class distribution

1.4 Basic statistics

Based on the computed statistics, the dataset exhibits a diverse range of mean values across its channels, ranging from as low as approximately 0.13 to as high as around 0.55. Additionally, the standard deviations also show variability, with values spanning from about 0.13 to roughly 0.31. This variation in the dataset's intensity distribution indicates that the data points might exist in different scales and ranges. Therefore, it is essential to normalise the data. Normalising ensures that all features have a consistent scale, which is crucial for many machine learning algorithms to converge faster and perform optimally, especially in the context of deep learning models.

1.5 Noisy Image Detection

As part of the Exploratory Data Analysis (EDA) process, it is essential to ensure the quality of the images in the dataset. In specific, presence of noisy images might be a potential issue that can affect the performance of machine learning models (Dodge & Karam, 2022). To address this, a simple statistical method was implemented to detect potentially noisy images. By calculating the standard deviation of pixel intensities for each image and comparing it to a predefined threshold (set at 15 in this case), images that might be overly uniform or lacking in detail would be investigated. Upon running this detection method, four potentially noisy images were identified in the dataset, which are visualised in figure 1.5 below:

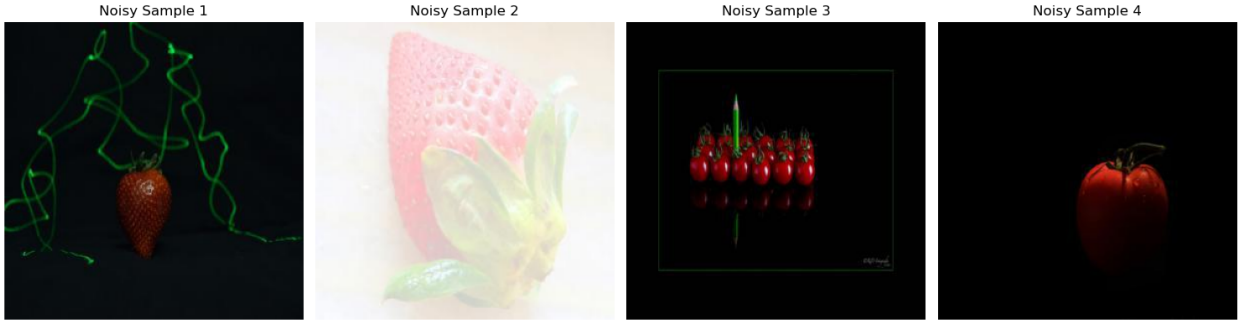

Figure 1.5 Noisy images

2. Data Preprocessing

2.1 Resize image

After examining the dataset, it was observed that the images varied in dimensions. To ensure uniformity during model training and improve consistency, all images in the dataset were resized to a standardised resolution of 300x300 pixels.

2.2 Normalisation

In a similar vein, to ensure the data values fall within a certain range, the ‘transforms.Normalize’ function was utilised, which normalised the dataset values with a mean and standard deviation of (0.5, 0.5, 0.5) for each of the three colour channels.

2.3 Noisy Image Removal

After detection from the EDA, the noisy images were removed, ensuring a cleaner and more cohesive dataset.

III. Methodology

1. Data Usage

To manage the given image dataset for model training and validation, the dataset was initially partitioned into a training set and a validation set. This partitioning was done using a stratified split to ensure that the distribution of classes in both subsets mirrors that of the complete dataset. Specifically, 80% of the images were allocated to the training set, while the remaining 20% were designated for the validation set. This split was performed using the random_split method from PyTorch's utility functions, ensuring randomness in the selection of images for each set.

For the training process, images were loaded in batches using a DataLoader. It's notable that the training data loader was set to shuffle images in each epoch. This shuffling is crucial as it helps prevent the model from generalising too much on a specific class order, aiding in better model training. On the other hand, the test set's data loader did not employ shuffling, as the order of images during validation does not impact the evaluation metrics.

2. The loss function

The Cross-Entropy Loss, implemented as ‘nn.CrossEntropyLoss()’ in PyTorch, is a standard choice for classification tasks. It evaluates how well the predicted probabilities from the model match the ground truth labels. Specifically, for each sample, it considers the predicted probability for the true class and aims to maximise it. As the name suggests, it quantifies the "entropy" or difference between two probability distributions: the predicted probabilities and the actual labels. A lower Cross-Entropy Loss indicates that the model's predictions are closer to the true labels, making it a powerful tool to train models to make accurate class predictions.

3. The optimisation methods

In the process of training machine learning models, optimization algorithms play a pivotal role in updating the model's weights to reduce the error. There are three optimisation methods were attempted in the project, which indicates differences performance on the models:

Stochastic Gradient Descent (SGD): A foundational optimization method, SGD updates the model's weights using a portion of data at each step, instead of the entire dataset. This can lead to faster convergence but might be sensitive to the learning rate and other hyperparameters. While SGD is versatile and can be employed in various architectures, in the project, it performs well for simpler models like Multi-Layer Perceptrons (MLP) due to its straightforward nature, with the accuracy of around 56% in the validation test, compared to 33% with the RMSprop optimizer. However, SGD might struggle with more complex landscapes, which is the CNN model.

RMSprop: Root Mean Square Propagation (RMSprop) is an adaptive learning rate optimization algorithm which divides the learning rate by an exponentially decaying average of squared gradients. While it often overcomes the issues of SGD, such as oscillations, it

did not exhibit satisfactory performance in this context. This might be due to various factors like hyperparameter settings or the specific nature of the dataset and model architecture.

Adamax: An extension of the Adam optimizer, Adamax is a variant that uses the infinity norm. It addresses certain limitations of Adam in terms of convergence. Similar to its predecessor, Adamax computes adaptive learning rates for different parameters, making it apt for intricate architectures such as Convolutional Neural Networks (CNN). In this particular scenario, Adamax is a considerable choice for the image classification task because it showcased satisfactory efficacy for the CNN models, registering approximately 67% prediction accuracy on the validation dataset.

4. The regularisation strategy

Regularisation techniques are crucial in machine learning to prevent overfitting, making the model generalise better to unseen data. Among the regularisation methods, L1 (Lasso) and L2 (Ridge) regularisation are the most popular, and combining them leads to the Elastic Net regularisation (Collimator, 2023).

L1 Regularization (Lasso): This technique adds a penalty equal to the absolute value of the magnitude of the coefficients. This can lead to some coefficients becoming exactly zero, effectively leading to feature selection. L1 regularisation tends to produce sparse weight matrices, promoting simpler and more interpretable models.

$$\textit{L1 Regularization Term} = \lambda * (|w_1| + |w_2| + \ldots + |w_n|)$$

L2 Regularization (Ridge): L2 regularisation adds a penalty proportional to the square of the magnitude of the coefficients. This approach tends to produce small weights but does not force them to become exactly zero. It ensures that the model does not fit the training data too closely by penalising large weight values.

$$\textit{L2 Regularization Term} = \lambda * (w_1^2 + w_2^2 + \ldots + w_n^2)$$

In the context of the project, the use of the L1+L2 strategy aimed to produce a model that is robust against overfitting while potentially benefiting from feature selection (due to the L1 component) and ensuring weight coefficients do not grow disproportionately large (due to the L2 component). This balanced approach helps in making the model more generalizable and resistant to the pitfalls of complex datasets.

5. The activation function

ReLU (Rectified Linear Unit) is a popular activation function for deep neural networks, especially convolutional neural networks (CNNs). It is a simple non-linear function, which for an input x returns max(0,x). The function replaces all negative values in the output with zeros and remains positive values unchanged. Due to its simplicity, it's

computationally efficient and allows the model to converge faster during training. In the project, ReLU is applied in the following manner:

-Convolutional Layers: After both self.conv1 and self.conv2, the ReLU activation function has been used. This is a common practice in CNNs to introduce non-linearity after convolutional layers, enabling the network to learn more complex patterns.

-Fully Connected Layers: ReLU were applied after self.fc1 and self.fc2, which are fully connected (dense) layers. This again introduces non-linearities, allowing the neural network to learn and approximate complex functions. On the other hand, self.fc3 was not applied to the ReLU activation function because the output of this layer is passed to the cross-entropy loss function, which implicitly applies the softmax function to convert raw logits into class probabilities.

6. Data Enrichment

The purpose of data augmentation is to artificially expand the dataset by introducing slight modifications to the original images, thereby enhancing the ability of the model to generalise and reducing overfitting (Khoshgoftaar, 2019). Augmentor library was used for augmentation. For each class, the process implemented a set of augmentation operations:

+Flip left to right: With a 50% probability, images are horizontally flipped.

+Black and white: A 10% chance to convert the images into grayscale.

+Rotate: A 30% probability to rotate images within a range of ±10 degrees.

+Skew: With a 40% likelihood, images undergo a skewing operation with a magnitude up to 0.5.

+Zoom: There's a 20% chance that the images are zoomed in or out, with a zoom factor ranging from 1.1 to 1.5.

As a result, the process generates 100 augmented samples each class based on these operations, which results in a total increase of 300 images in the data. Figure 3.6 presents a sample of the original image and the output from the augmentation process:

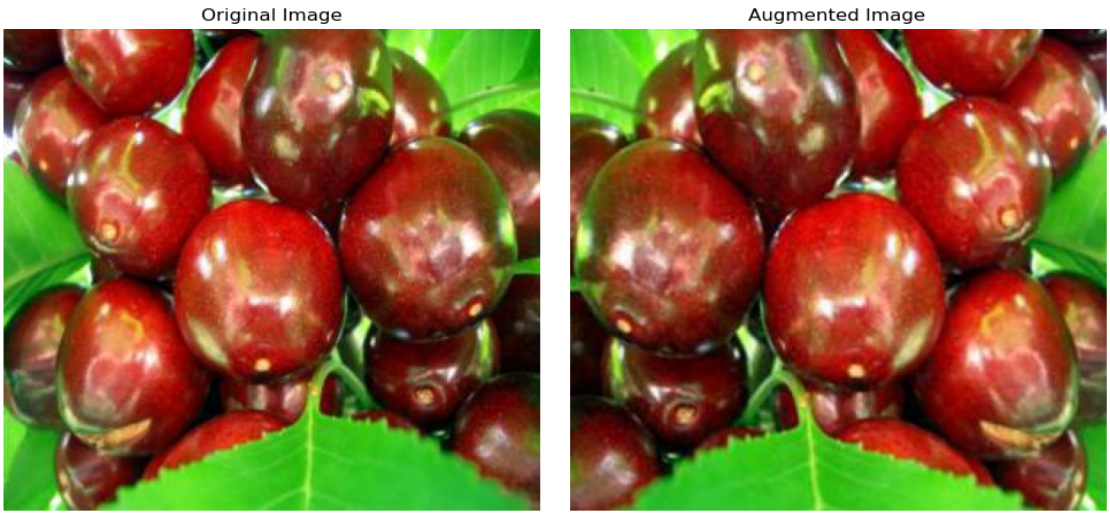

Figure 3.6- Original and Augmented (Flip left to right) image

7. Transfer learning

Transfer learning leverages the knowledge of a model pre-trained on a vast dataset, in this case, ImageNet, to benefit a new but related task. In the project, the ResNet-50 model's initial layers, already adept at capturing intricate image features from its prior training, are frozen to retain their learned behaviour. The model's final layer, originally designed for ImageNet's 1000 classes, is replaced to cater to a specific three-class classification problem in the project, which are cherry, tomato, and strawberry. This tailored model is then trained on a new dataset, allowing it to map the robust features extracted by the ResNet's frozen layers to the new dataset's classes. By harnessing the power of a pre-trained model and fine-tuning its final layers, this approach can achieve higher accuracy with fewer training samples and iterations compared to training a deep model from scratch.In terms of performance, the ResNet-50 model generate output with accuracy of approximately 82%, which is dominant compared to the MLP and CNN models. However, due to the size (more than 100MB) and training time (more than 800 minutes on my computer), it can be considered a drawback of this model when deploying solutions in environments with limited computational capabilities or under strict time constraints.

IV. Summary and Discussion

1. Baseline MLP model

The neural network architecture, MLP, comprises two fully connected layers. The input to the network is a flattened version of the 300x300 image, resulting in a vector of length 3×300×300. This input is passed through the first layer (layer1), which has neurons equal to the batch size (45 in this case). This layer uses the ReLU activation function. The output of this layer is then passed to the second layer (layer2), which has 3 neurons, corresponding to the 3 potential classes of the images. For training, Cross-Entropy Loss function was used and the optimization method chosen was Stochastic Gradient Descent (SGD) with a learning rate of 0.01. The model is trained using batches of images, with each batch containing 45 images, resulting in 100 batches for the entire training dataset of 4500 images. The result in Figure 4.1 shows that the training set has higher Loss and lower Accuracy that those of the validation set. Specifically, the result peaks at around 51%.

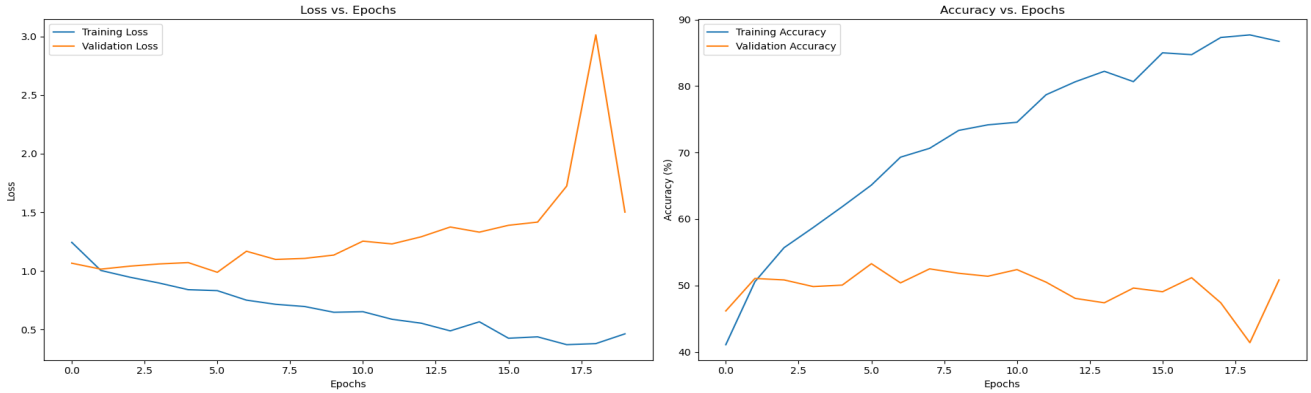

Figure 4.1- Loss and Accuracy curve of the baseline MLP model

2. Best CNN model

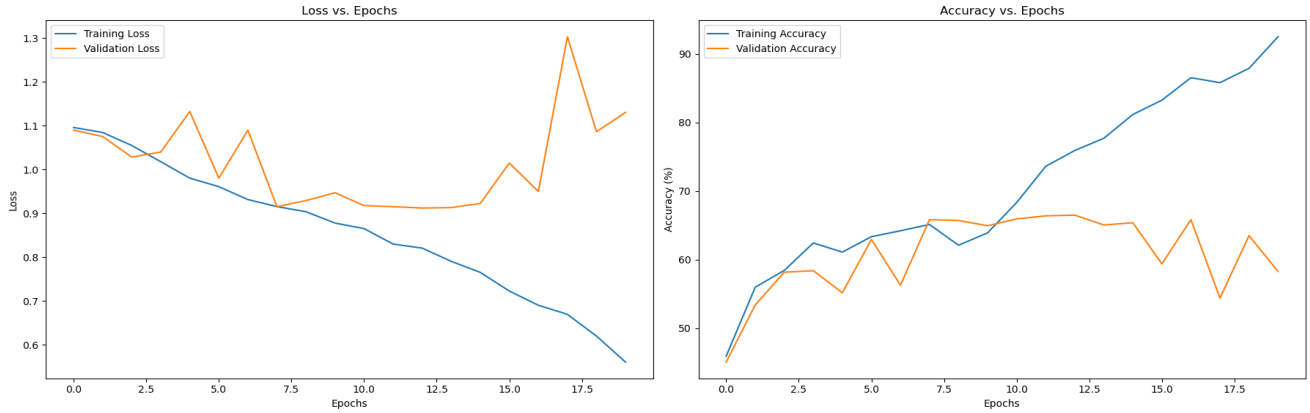

Figure 4.2 - Loss and Accuracy curve of the CNN model

The neural network architecture, Net, is a convolutional neural network (CNN) designed for image classification. The initial input to the model is a 3-channel image of size 300×300. The first layer, conv1, is a convolutional layer that transforms this input into 6 channels using a kernel of size 5. This is followed by a max-pooling layer, pool, which reduces the dimensions of the feature maps by half. Subsequently, the conv2 layer further processes the feature maps and outputs 16 channels. After the convolutional layers, the feature maps are flattened and passed through three sequential fully connected layers: fc1, fc2, and fc3. The fc1 layer reduces the features to a dimensionality of 120, which is further reduced to 84 by fc2. The final layer, fc3, outputs a vector of size 3, corresponding to the three potential classes of the images. Throughout the network, the ReLU activation function is employed for non-linearity, except for the final layer. For training, the model employs the Cross-Entropy Loss function and optimization is achieved using the Adamax algorithm with a learning rate of 0.008. Training batches are constructed with 32 images each, ensuring a good mix of data in each forward and backward pass. In terms of performance, the CNN model peaks at 67%, which is considerably higher than the MLP model (51%). However, the performance is likely to be more stable across the epochs, compared to the MLP model. In the aspect of training time, the MLP turns out to outperform the CNN model, with 27 minutes while CNN requires more than 77 minutes for 20 epochs.

The possible explanation is that the Convolutional Neural Networks (CNNs) are particularly tailored for image classification tasks, offering advantages over Multi-Layer Perceptrons (MLPs). CNNs excel in capturing spatial hierarchies of features, benefiting from weight sharing across spatial locations, which introduces translation invariance and reduces the model's complexity. Their architecture is also adept at dimensionality reduction, concentrating on essential image features while reducing susceptibility to overfitting. In contrast, MLPs lack these inherent spatial and hierarchical feature detections, making CNNs more efficient and effective for image data.

V. Conclusions and future work

In conclusion, while the MLP architecture demonstrated adequate capabilities, it ultimately fell short of the demands of complex image classification tasks. On the other hand, the pre-trained model, ResNet50, showcased superior performance, living up to its reputation. However, its training was time-consuming, making it potentially less feasible for real-time or real-life applications. The custom CNN architecture emerged as a balanced choice. Its performance was commendable, and it trained in a reasonable time frame, a combination that offers practical utility. In addition, the flexibility inherent to the CNN design makes it an ideal candidate for future enhancements and adjustments. As the field of deep learning continues to evolve, it's crucial to strike the right balance between performance, efficiency, and adaptability (Andreetto, 2017), and the CNN model seems poised to balance those.

In future endeavours, with more sufficient time and enhanced computational resources, It is aspiring to fortify the model's performance through three potential strategies. First, by amassing a more expansive and diverse dataset, it is promising to capture intricate patterns, enhancing the model's proficiency. Second, systematic hyperparameter tuning, such as adjusting learning rates, layers or batch sizes, offers a pathway to refine the model's configuration for optimum results. Lastly, It is interesting to harness the collective prowess of multiple models through stack ensemble techniques, for example, stacking MLP, CNN, RNN and possibly pretrained models, in order to leverage their combined strengths to bolster accuracy and reliability in predictions.

VI. Reference list

Andreetto, M. (2017, April 17). *(PDF) MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. ResearchGate. Retrieved October 31, 2023, from https://www.researchgate.net/publication/316184205_MobileNets_Efficient_Convolutional_Neural_Networks_for_Mobile_Vision_Applications

Collimator. (2023, August 22). *What is L1 vs L2 regularisation?* Collimator. Retrieved October 31, 2023, from https://www.collimator.ai/reference-guides/what-is-l1-vs-l2-regularization

Dodge, S., & Karam, L. (2022, October 2). *.Understanding How Image Quality Affects Deep Neural Networks*. Retrieved October 30, 2023, from https://arxiv.org/pdf/1604.04004.pdf

Khoshgoftaar, T. M. (2019, July 6). *A survey on Image Data Augmentation for Deep Learning - Journal of Big Data*. Journal of Big Data. Retrieved October 31, 2023, from https://doi.org/10.1186/s40537-019-0197-0

Reid, P. (2022, May 27). *The Difference Between Computer Vision and Human Vision - visionAI*. Vision AI Suite. Retrieved October 30, 2023, from https://visionaisuite.net/blog/the-difference-between-computer-vision-and-human-vision