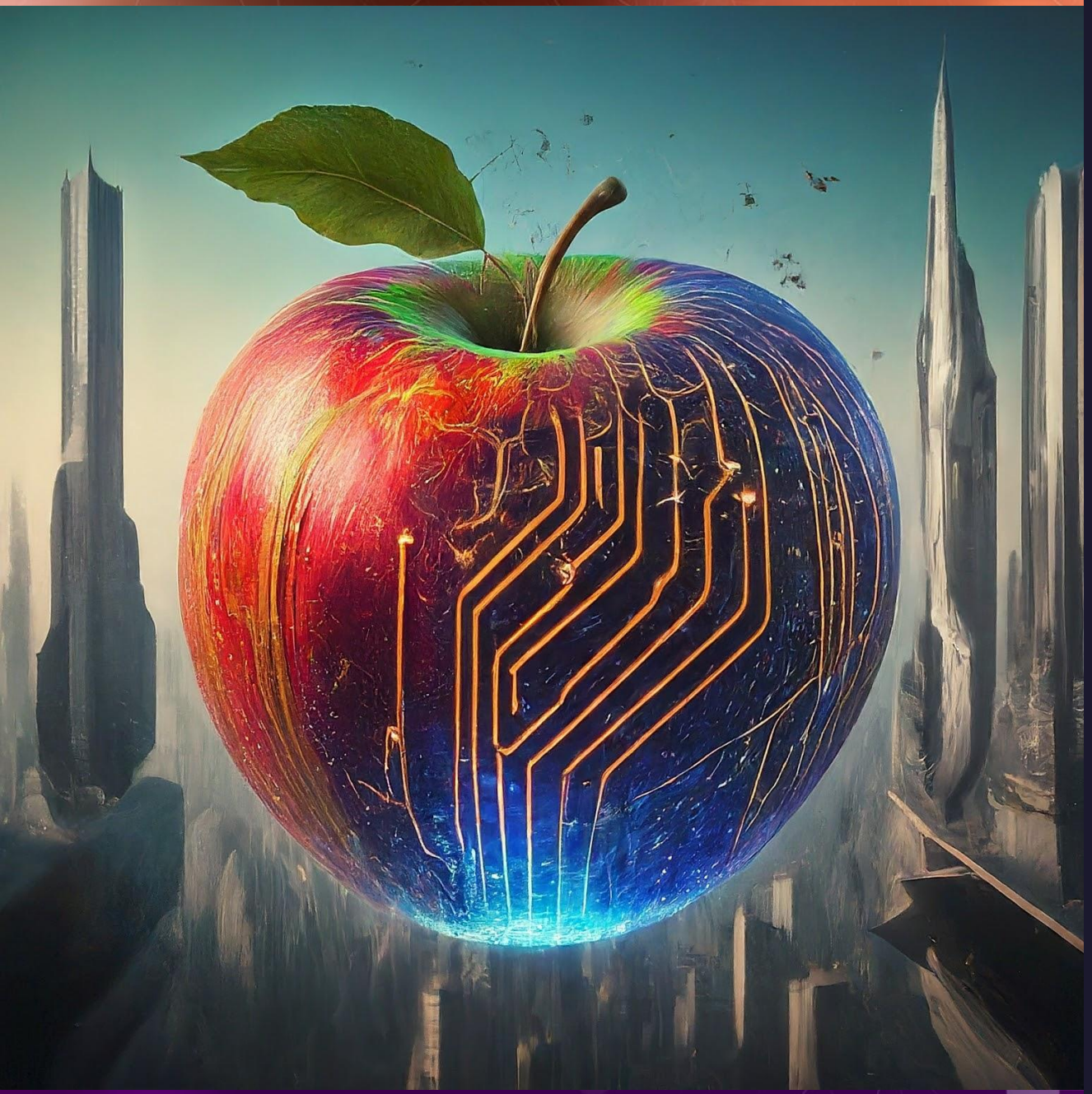
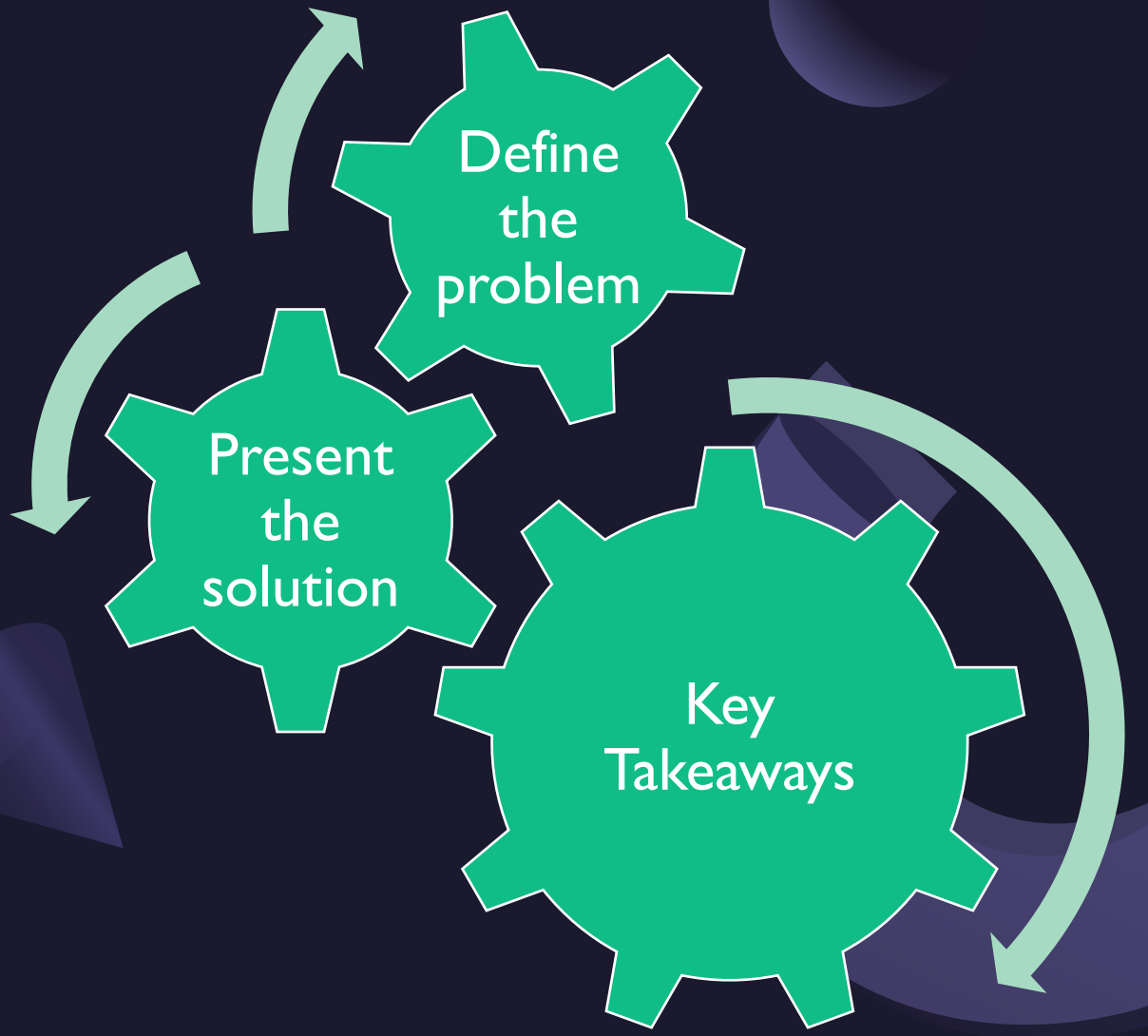


AI ML Capstone

Computer Vision for produce quality

Agenda



1. Introduction and Project Overview

Define the Problem

Problem statement

In today's fast-paced world, businesses and organizations face the challenge of extracting meaningful insights from the overwhelming amount of data at their disposal. Traditional methods often fall short, necessitating a more sophisticated and efficient solution. Your project aims to address this gap by creating an advanced AI-driven data analytics system capable of automatically analyzing large datasets, uncovering patterns, and generating actionable insights. The primary goal is to empower decision-makers with a tool that enhances their ability to make informed, data-driven decisions in various industries.

Quality and Safety

Consumers in the region are increasingly demanding high-quality, safe fruits, including apples. Factors like appearance, freshness, and the absence of harmful chemicals are key considerations. By prioritizing these factors, apple suppliers can build trust and meet consumer expectations.

Red Delicious, a popular variety, have seasonal availability in the region due to factors like growing seasons, transportation, and import regulations. Cold chain logistics are crucial for maintaining fruit quality during transportation, involving temperature-controlled storage and transport.

AI CV ML can be used to automate batch apple quality visual inspection via Deep Learning CNN unsupervised Image Classification. By analyzing images of apples, algorithms can accurately identify defects like bruises, blemishes, and wormholes, ensuring that only high-quality fruits reach consumers.

Apple Supply Chain

A general overview of the apple supply chain. The specific stages and processes may vary depending on the region, the scale of production, and the distribution channels involved

Farming and Harvesting

Planting: Apple trees are planted in orchards, often on a commercial scale.
Growing: The trees are cared for throughout the growing season, which includes pruning, pest control, and irrigation.
Harvesting: Ripe apples are harvested, typically by hand or with specialized machines

Sorting and Grading

Sorted by size, color, and quality.

Graded based on their appearance, firmness, and other quality factors.

Packaging

Packaged in various containers and Labels are affixed to the packaging, providing information such as the variety, weight, and country of origin.

Transportation

Local Transportation: Apples may be transported from the farm to a packing facility or distribution center.

Long-Distance Transportation: to markets or distribution centers, often using trucks, trains, or ships.

Storage and distribution

Storage: Apples may be stored in cold storage facilities to maintain their freshness and quality. **Distribution:** Apples are distributed to wholesalers, retailers, or directly to consumers.

Retailer

Wholesalers: Wholesalers purchase apples from suppliers and distribute them to retailers.

Retailers: Retailers, such as supermarkets, grocery stores, and farmers' markets, sell apples to consumers.

Consumer

Purchase: Consumers purchase apples from retailers.

Consumption: Consumers enjoy eating the apples.

Sort Defect Apple Process - Farm

Manual by hand → Manual roller → Automated roller → Manual roller

Weight (total) → Defect Sortation → Dip, wax, and Dry → 100% Fruit QC → Grade (by Weight) → Packing (by Grade)

In the farm, apples are typically checked for freshness immediately after harvesting. AI Computer Vision Machine Learning can be used to automate apple quality visual inspection. By analyzing images of apples, algorithms can accurately identify defects like bruises, blemishes, and wormholes, ensuring that only high-quality fruits reach consumers.

2. Topic Selection Rationale

Define the Problem

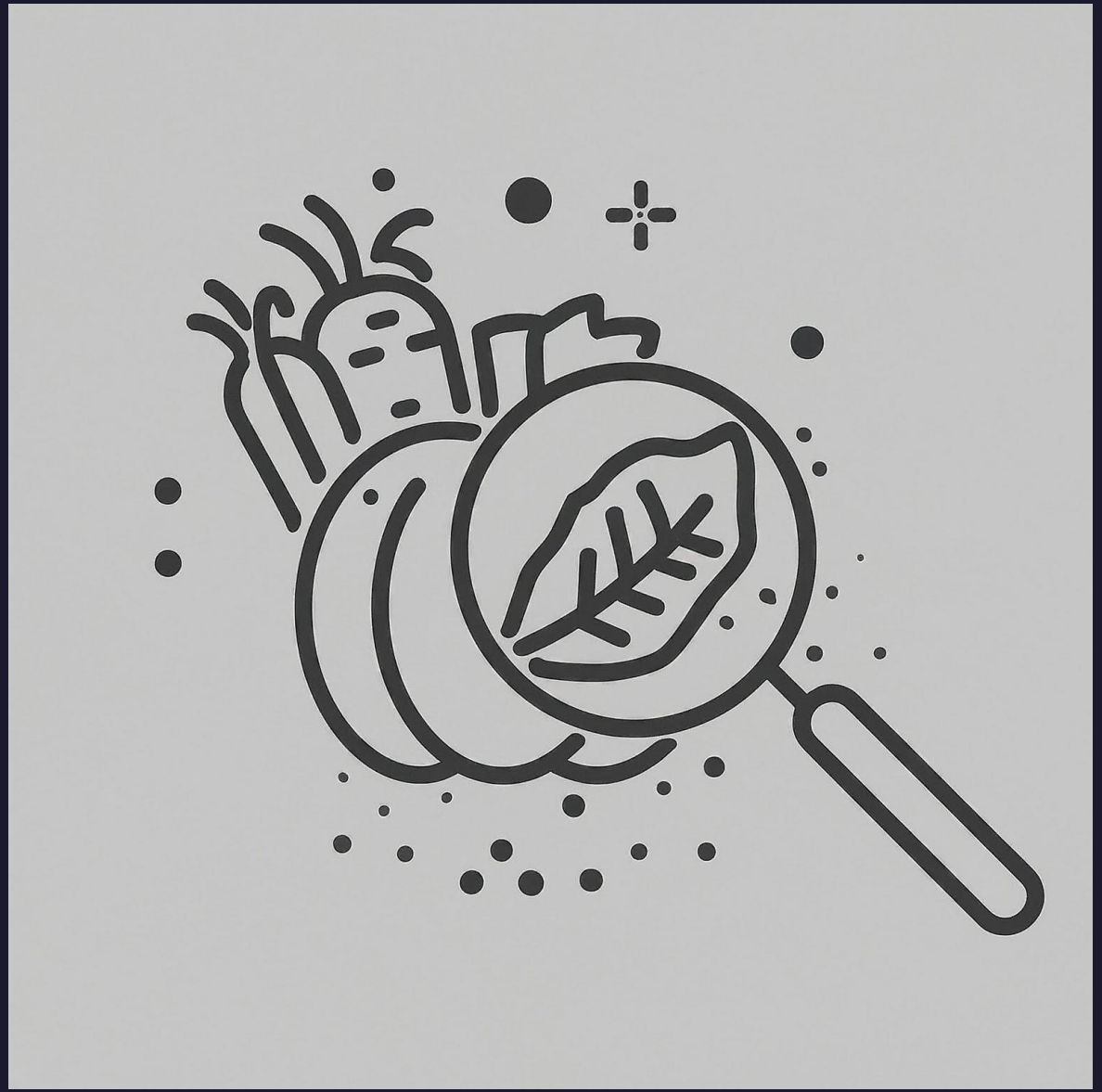
AI ML Business Innovation Process

Defining business goals and target audience.	• introduce automation to SCM process • improve visual inspection processing • farmers, cold chain, retail
Identifying data sources and ensuring data quality.	• leverage on popular fruits 360 dataset for training and validation • research work done • for implementation to allow realtime online capture via opencv
Choosing the appropriate machine learning algorithm.	• TF KERAS API • CNN, image classification • Accuracy loss diagram • Prediction sample
Training and testing the model for accuracy and fairness.	• CNN model • Adam optimizer • Batch image prediction
Deploying and monitoring the model for continuous improvement.	• Increase training image dataset, add diseased apple images as defect, • when real-time online check error to inspect for out of sample images e.g. sensor spoil

The Data

Fruits 360 dataset (Version: 2020.05.18.0), is a comprehensive collection of images featuring various fruits and vegetables. The dataset is publicly available. This dataset is organized into three main subsets: training, test, and validation, with a diverse array of classes representing different types of fruits and vegetables. For instance, in the training set, you may find separate folders for specific fruit classes, such as "apple red," "apple hit," and "apple rotten." This granularity allows for the exploration of multiple characteristics, such as freshness or damage, within a single fruit type. Possible Fruit Classification Use Cases include the following 1: Sorting Ripe Fruits, 2: Detecting Spoiled Fruits, 3: Inventory Management, 4: Fruit Disease Detection.

Detecting Spoiled Fruits is selected. This project proposes a method involving the deep learning technique which is CNN for feature extraction and classification of defect fruits. It is one of the applications of image classification problems. This approach uses an RGB channel image of the fruit under examination. ML - Unsupervised classification, batch, model based.



Visual Inspection

Timing of Freshness Checks: By conducting regular freshness checks at various stages of the supply chain, businesses and consumers can help to ensure that the apples they purchase and consume are of the highest quality.

Harvesting: Apples are typically checked for freshness immediately after harvesting.

Transportation and Storage: Freshness checks are conducted throughout the transportation and storage process to ensure that the apples remain in good condition.

Retail: Retailers typically check the freshness of apples when they receive shipments and periodically throughout the day.

Consumer: Consumers can check the freshness of apples at any time before purchasing or consuming them.

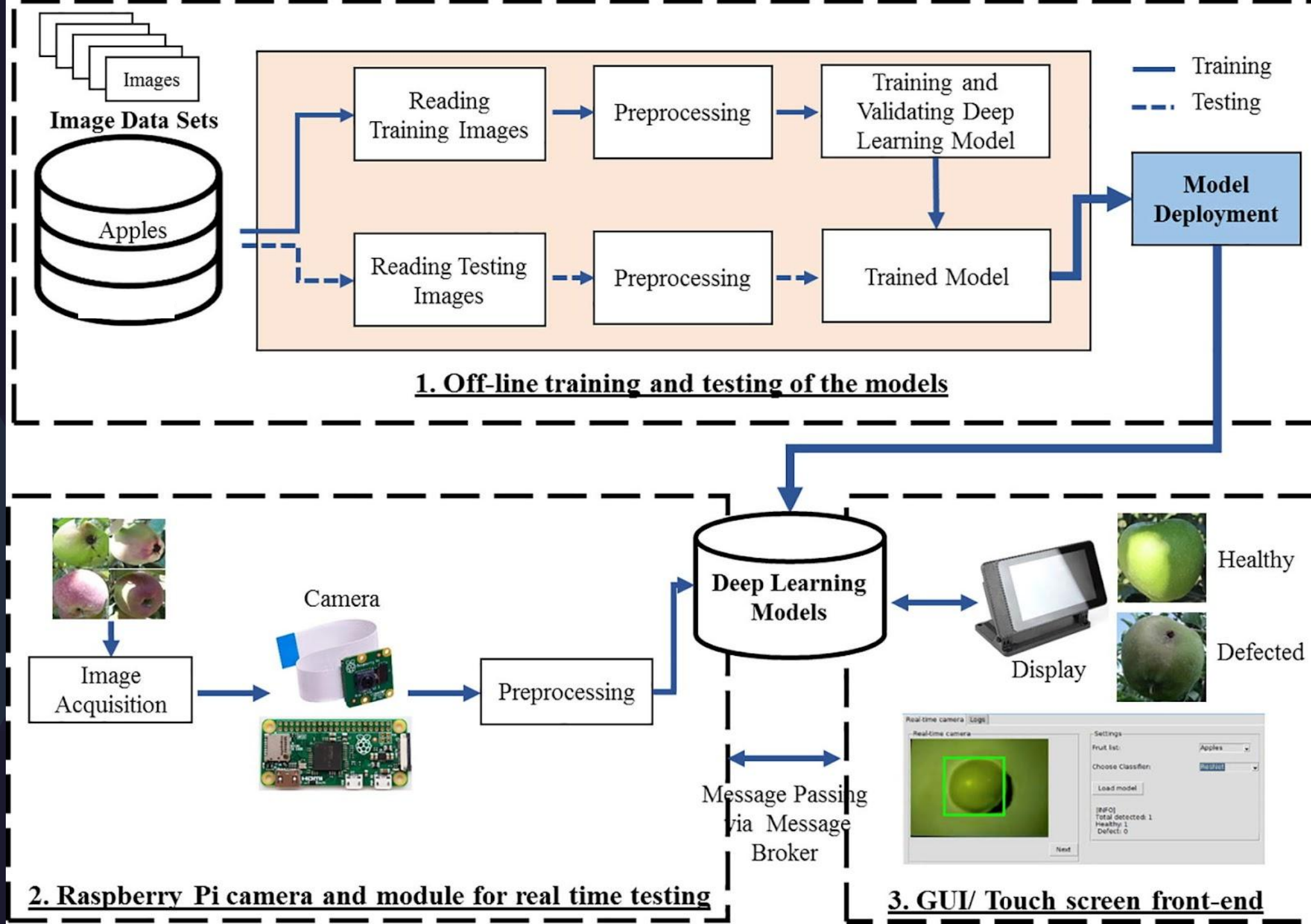
3. System Architecture and Design

Present the Solution

ML - Unsupervised classification, batch, model based

System Design

Source
Science Direct, online, 03/09/24



AI ML – CV Image classification CNN

Source Science Direct, online, 03/09/24

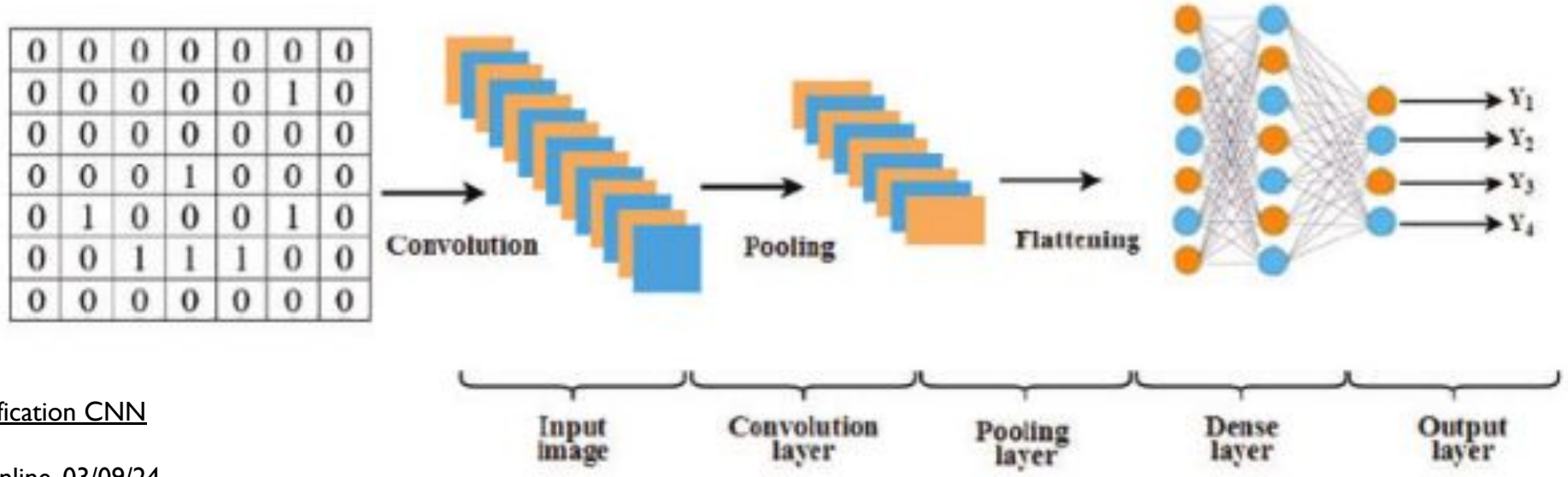


Figure 1: CNN architecture

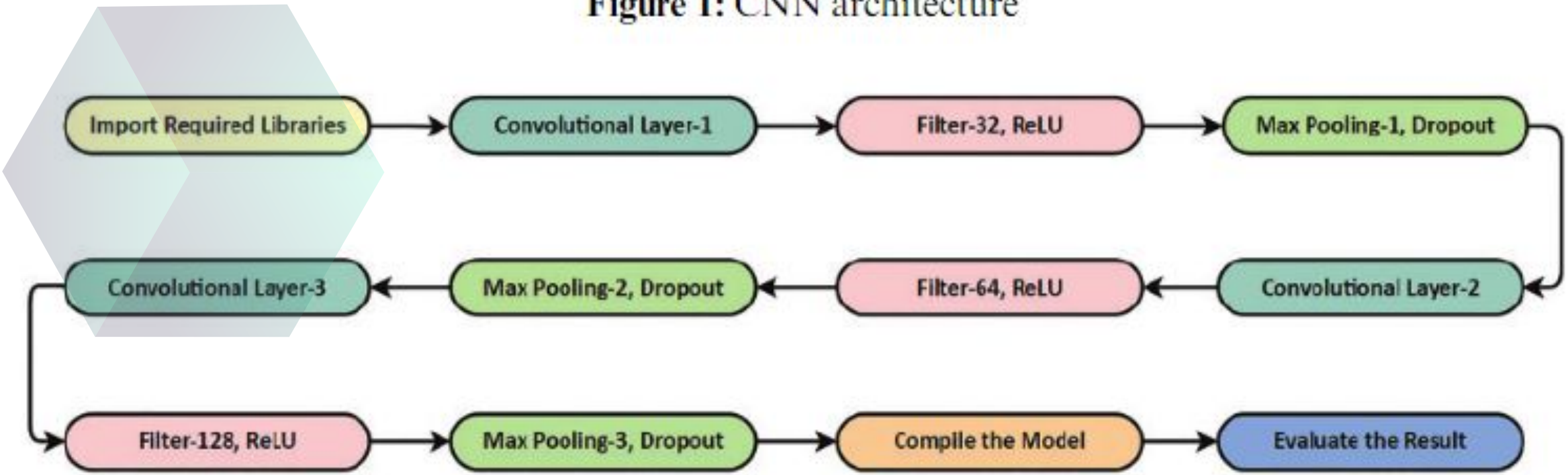


Figure 2: Proposed CNN model

4. Coding and Implementation Highlights
5. Testing and Validation Results
6. Visualization Demonstration

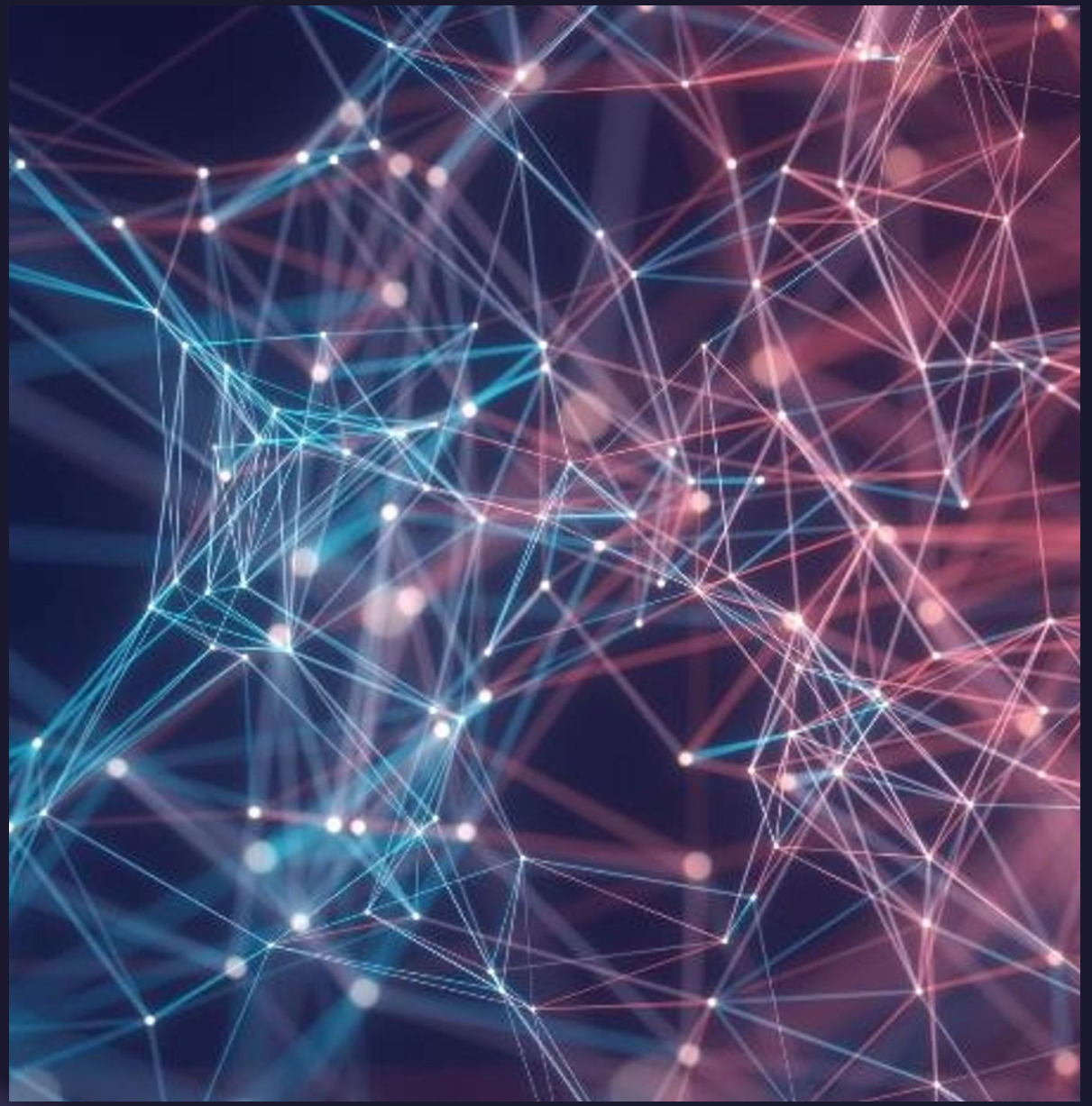
Present the Solution

The Solution

GitHub

SCAN ME

https://github.com/sallez9/SCTPAIMLcapstone



Highlights CNN Model

Classifier using the Sequential API: Use TF Keras API to load the dataset, provides utility functions to fetch and load common datasets

```
from tensorflow import keras
from tensorflow.keras import layers

data_augmentation = keras.Sequential(
    [
        layers.Input(shape=(img_height, img_width, 3)),  # Define the input shape here
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
    ]
)
```

```
#CNN model
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='tanh'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

Display all the model layers including layer names, output shape and number of parameters dense layer often have a lot of parameters and give the model flexibility to fit training data but means runs risk of overfittting especially if training data is small.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3,965,056
dense_1 (Dense)	(None, 128)	16,512
outputs (Dense)	(None, 2)	258

Total params: 12,016,232 (45.84 MB)

Trainable params: 4,005,410 (15.28 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 8,010,822 (30.56 MB)

+ Code + Markdown

```
# specifying the optimizer and model metrics
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

AI ML – CV Image classification CNN

Source Science Direct, online, 03/09/24

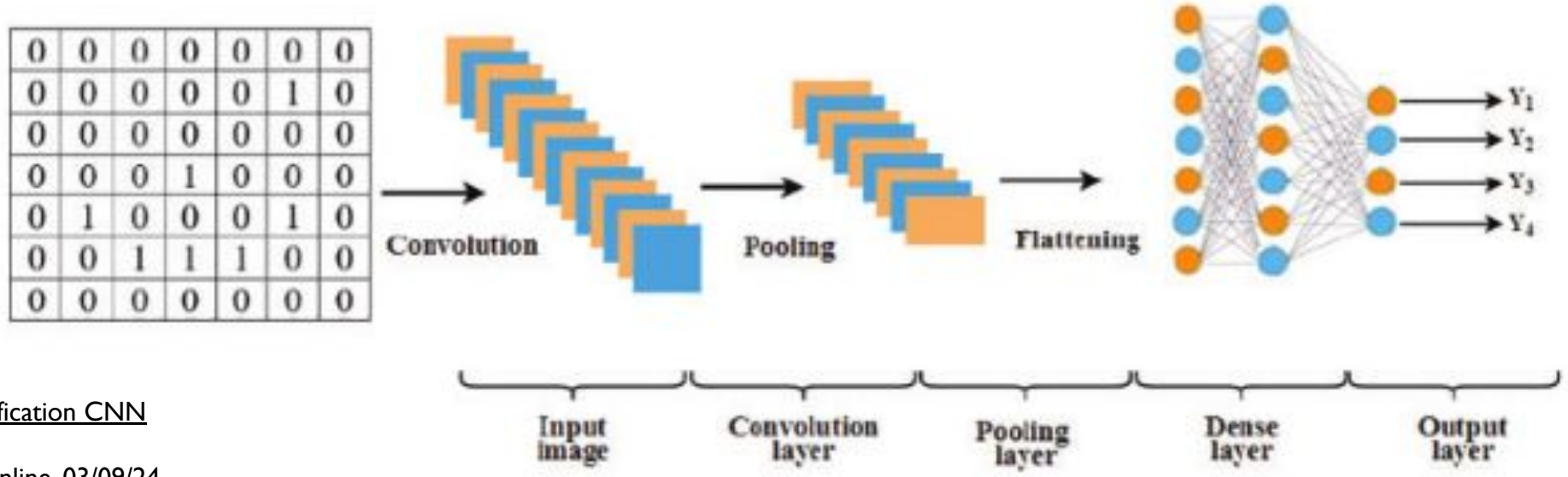


Figure 1: CNN architecture

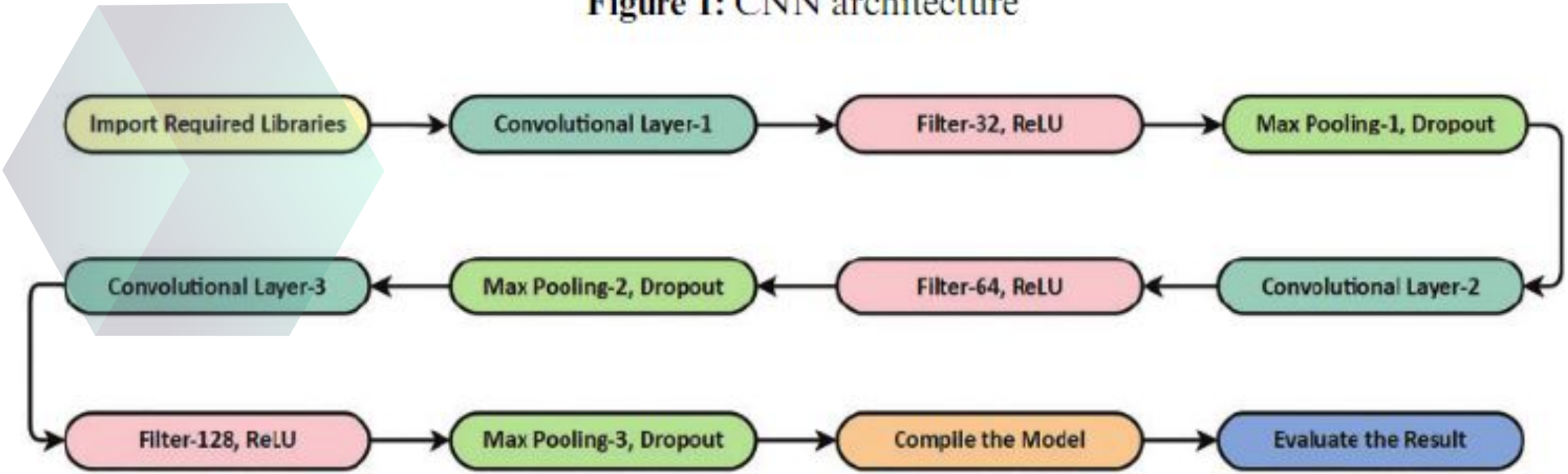


Figure 2: Proposed CNN model

Sorting and filtering dataset

Consolidate certain classes allowed us to simplify the task of classifying defect versus fresh apples. To achieve this, we merge class folders and rename files. This step was crucial for balancing the dataset and ensuring that the classifier could distinguish between healthy and defective fruits effectively. Thus demonstrate effective data cleaning, normalization and transformation to incur quality and readiness of data.

The preprocessing involved:

File Renaming: To avoid filename conflicts, we automatically renamed the files during the merging process, ensuring unique identifiers for each image.

Merging Classes: We wrote Python code to combine the images from the "apple hit" and "apple rotten" classes into "apple defect".

Making file directory and replacing for fresh and defect apples

```
[5] !mkdir SCTPAIMLcapstone/dataset/apple_fresh
    !mkdir SCTPAIMLcapstone/dataset/apple_defect

    # Paths to the merged class folders
    fresh_folder_path = 'SCTPAIMLcapstone/dataset/apple_fresh'
    defect_folder_path = 'SCTPAIMLcapstone/dataset/apple_defect'

    # Paths of individual folders that need to be merged into the two classes
    folders_to_replace = {
        fresh_folder_path: [
            'SCTPAIMLcapstone/dataset/apple_red_1',
            'SCTPAIMLcapstone/dataset/apple_red_2',
            'SCTPAIMLcapstone/dataset/apple_red_3'
        ],
        defect_folder_path: [
            'SCTPAIMLcapstone/dataset/apple_hit_1',
            'SCTPAIMLcapstone/dataset/apple_rotten_1'
        ]
    }
```

```
[6] #file rename files with replace folder instead of only merge files into folder method
    # Function to rename files in a folder by adding a prefix, suffix, and ensuring unique names
    def rename_files_in_folder(folder_path, prefix='', suffix=''):
        with os.scandir(folder_path) as entries:
            for entry in entries:
                if entry.is_file():
                    name, ext = os.path.splitext(entry.name)
                    unique_id = uuid.uuid4().hex[:8]  # Generate a short unique identifier
                    new_name = f"{prefix}{name}_{unique_id}{suffix}{ext}"
                    os.rename(entry.path, os.path.join(folder_path, new_name))
```

```
[7] # Function to replace files from source folder into destination folder
    def replace_files(src_folder, dest_folder):
        with os.scandir(src_folder) as entries:
            for entry in entries:
                if entry.is_file():
                    dest_file_path = os.path.join(dest_folder, entry.name)
                    # If the file exists in the destination folder, delete it before replacing
                    if os.path.exists(dest_file_path):
                        os.remove(dest_file_path)
                    shutil.move(entry.path, dest_file_path)
            # Erase the source directory after replacing files
            shutil.rmtree(src_folder)
```

```
[8] # Rename files within the main class folders with unique names
    rename_files_in_folder(fresh_folder_path, prefix='fresh_')
    rename_files_in_folder(defect_folder_path, prefix='defect_')

    # Replace the files in the main class folders with files from the additional folders
    for dest_folder, src_folders in folders_to_replace.items():
        for src_folder in src_folders:
            # Rename files before replacing with unique names
            rename_files_in_folder(src_folder, prefix=os.path.basename(dest_folder) + '_')
            replace_files(src_folder, dest_folder)

    print("Files have been uniquely renamed and replaced in two classes: 'fresh' and 'defect'.")

→ Files have been uniquely renamed and replaced in two classes: 'fresh' and 'defect'.
```

```
[9] #load the dataset with tensorflow
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    # Data generators for training and validation
    # Preprocess the image to match the input shape expected by the model

    data_datagen = ImageDataGenerator(rescale=1./255)

    batch_size = 32
    img_height = 180
    img_width = 180
```

CNN model and Adam optimizer

Convolutional Neural Networks (CNNs) are well-suited for fruit visual inspection due to their ability to handle spatial relationships, learn features automatically, extract features at multiple levels, and learn from large datasets. While CNNs are a popular choice, other models like RNNs, SVMs, decision trees, and random forests can also be considered depending on the specific task. For tasks like freshness and defect detection, linear regression and K-means clustering might be useful.

Adam optimizer is a good choice for computer vision tasks like defect apple detection due to its efficiency, effectiveness, and ability to handle complex models and noisy data. Its adaptive learning rate, momentum, and computational efficiency make it suitable for this specific task. Adam's popularity, ease of use, and specific benefits for defect apple detection further solidify its position as a preferred optimizer.

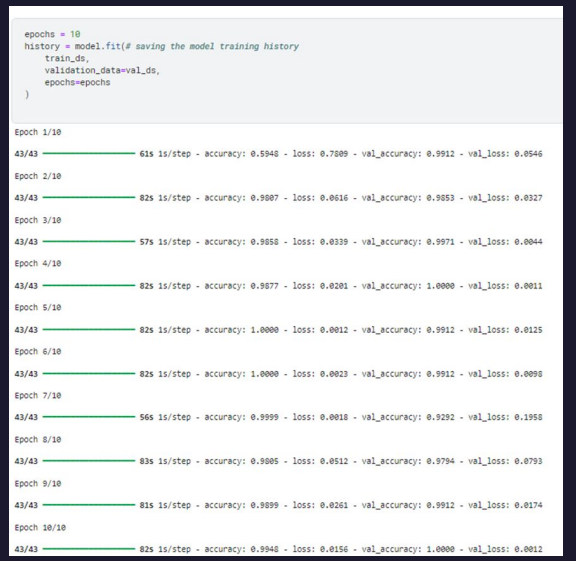
Model Training - check overfit

The training dataset for fresh apples initially consisted of only ‘apple red 1’ images. The model training output suggest overfitting.

Then Apple red 1, 2 and 3 were added into the apple fresh class.

There was an improvement on the val accuracy loss. However, results were not consistent.

```
Epoch 1/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 32s 736ms/step - accuracy: 0.8222 - loss: 0.3977 - val_accuracy: 0.8348 - val_loss: 0.3690
Epoch 2/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 42s 768ms/step - accuracy: 0.9268 - loss: 0.1986 - val_accuracy: 0.9823 - val_loss: 0.0397
Epoch 3/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 33s 754ms/step - accuracy: 0.9801 - loss: 0.0715 - val_accuracy: 0.9941 - val_loss: 0.0200
Epoch 4/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 40s 741ms/step - accuracy: 0.9903 - loss: 0.0386 - val_accuracy: 0.9646 - val_loss: 0.1217
Epoch 5/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 42s 773ms/step - accuracy: 0.9909 - loss: 0.0259 - val_accuracy: 0.9794 - val_loss: 0.0569
Epoch 6/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 41s 770ms/step - accuracy: 0.9866 - loss: 0.0364 - val_accuracy: 0.9971 - val_loss: 0.0071
Epoch 7/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 40s 749ms/step - accuracy: 0.9899 - loss: 0.0333 - val_accuracy: 0.9971 - val_loss: 0.0220
Epoch 8/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 32s 742ms/step - accuracy: 0.9933 - loss: 0.0184 - val_accuracy: 0.9705 - val_loss: 0.0670
Epoch 9/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 32s 736ms/step - accuracy: 0.9981 - loss: 0.0067 - val_accuracy: 0.9912 - val_loss: 0.0161
Epoch 10/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 42s 767ms/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 1.4907e-04
```



```
epochs = 10
history = model.fit(# saving the model training history
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 61s 1s/step - accuracy: 0.5948 - loss: 0.7809 - val_accuracy: 0.9912 - val_loss: 0.0546
Epoch 2/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9807 - loss: 0.0616 - val_accuracy: 0.9853 - val_loss: 0.0327
Epoch 3/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 57s 1s/step - accuracy: 0.9858 - loss: 0.0339 - val_accuracy: 0.9971 - val_loss: 0.0044
Epoch 4/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9877 - loss: 0.0201 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 5/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9912 - val_loss: 0.0125
Epoch 6/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.9912 - val_loss: 0.0098
Epoch 7/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 56s 1s/step - accuracy: 0.9999 - loss: 0.0018 - val_accuracy: 0.9292 - val_loss: 0.1958
Epoch 8/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.9805 - loss: 0.0512 - val_accuracy: 0.9794 - val_loss: 0.0793
Epoch 9/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.9899 - loss: 0.0261 - val_accuracy: 0.9912 - val_loss: 0.0174
Epoch 10/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9948 - loss: 0.0156 - val_accuracy: 1.0000 - val_loss: 0.0012
```

Model Training - check overfit

Model is performing well.

However, suggest overfitting.

Based on the screenshot, starting at epoch 4/10.

We implemented early stopping epoch =3, and there was an improvement in the accuracy and loss. refer to the next slide.

```
epochs = 10
history = model.fit(# saving the model training history
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 61s 1s/step - accuracy: 0.5948 - loss: 0.7809 - val_accuracy: 0.9912 - val_loss: 0.0546
Epoch 2/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9807 - loss: 0.0616 - val_accuracy: 0.9853 - val_loss: 0.0327
Epoch 3/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 57s 1s/step - accuracy: 0.9858 - loss: 0.0339 - val_accuracy: 0.9971 - val_loss: 0.0044
Epoch 4/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9877 - loss: 0.0201 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 5/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9912 - val_loss: 0.0125
Epoch 6/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.9912 - val_loss: 0.0098
Epoch 7/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 56s 1s/step - accuracy: 0.9999 - loss: 0.0018 - val_accuracy: 0.9292 - val_loss: 0.1958
Epoch 8/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.9805 - loss: 0.0512 - val_accuracy: 0.9794 - val_loss: 0.0793
Epoch 9/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.9899 - loss: 0.0261 - val_accuracy: 0.9912 - val_loss: 0.0174
Epoch 10/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9948 - loss: 0.0156 - val_accuracy: 1.0000 - val_loss: 0.0012
```

Accuracy / Loss - check overfit

1. Early Stopping Prevented Overfitting:

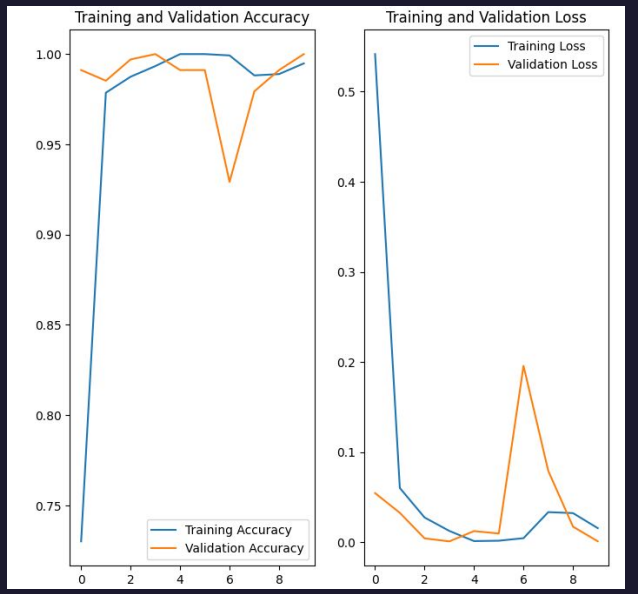
- The validation accuracy curve starts to plateau or even decline after epoch 3. This indicates that the model is starting to overfit the training data, learning the noise and idiosyncrasies rather than the underlying patterns.
- Early stopping at epoch 3 prevents the model from further training and overfitting, preserving its generalization ability.

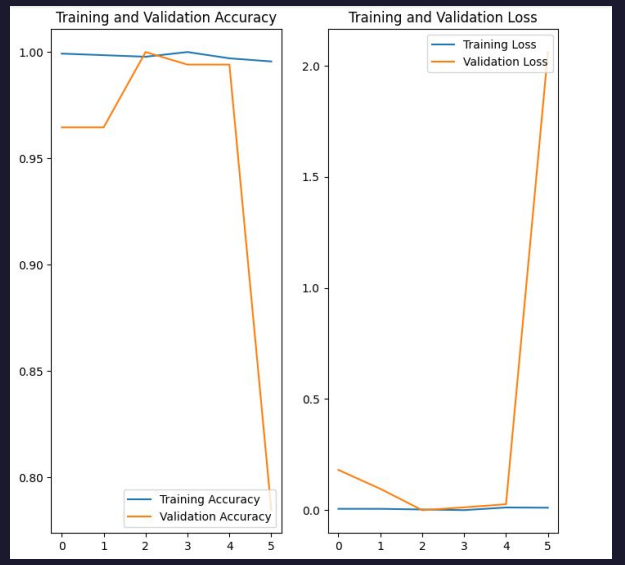
2. Model Performance at Epoch 3:

- The model achieved a training accuracy of around 0.95 and a validation accuracy of around 0.95 at epoch 3. These values suggest that the model has learned the underlying patterns effectively and has reasonable generalization performance.

3. Potential for Improvement:

- While early stopping at epoch 3 prevented overfitting, it's possible that further training with careful regularization techniques (e.g., L1 or L2 regularization) could have improved the model's performance.
- Exploring different hyperparameter settings or architectural changes might also lead to better results.





Accuracy / Loss - recommendations

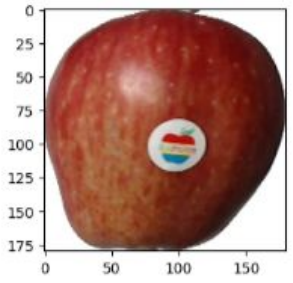
Recommendations:
1. Regularization Techniques: Consider using additional regularization methods, such as Dropout, L2 regularization, or data augmentation, to improve generalization and reduce overfitting.
2. Reduce Model Complexity: If overfitting persists, you might want to reduce the complexity of the model (e.g., reducing the number of layers or units in the layers).
3. Cross-Validation: If feasible, try using cross-validation to assess the model's performance across different validation sets. This can provide more stable estimates of performance and mitigate issues caused by a single noisy validation set.
4. Hyperparameter Tuning: Experiment with hyperparameters like learning rate, batch size, and regularization strength to find a better fit that generalizes well across validation data.

Model Test Validation

3.1 Predicting a sample image

```
# Fetching model predictions for sample image in validation test dataset
plt.figure(figsize=(3, 3))
for images_batch, labels_batch in val_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()
    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict

actual label: apple_fresh

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 385ms/step

predicted label: apple_fresh
```




```
# Defining prediction function for validation testing images
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

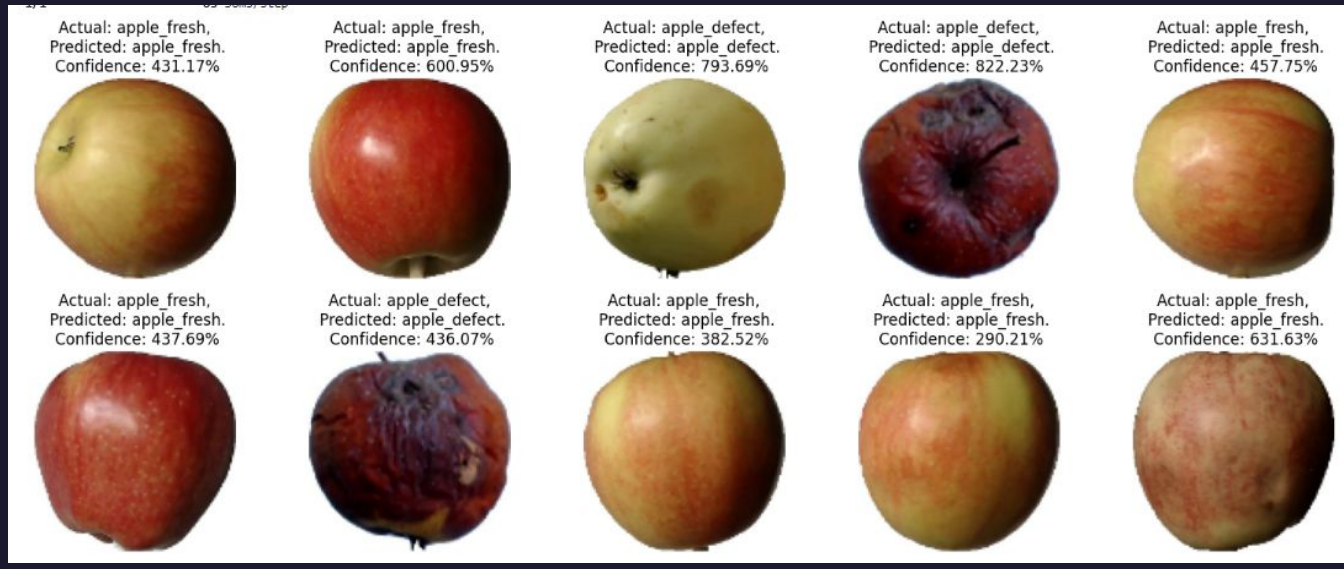
We will do a batch prediction and visualize the results to have a better idea about how our model is performing.

Prediction set

```
plt.figure(figsize=(15, 15))

# Iterate over the batches and then the images to display their predictions
batch_size = 20
for images, labels in val_ds.take(12):
    for i in range(batch_size):
        if i >= len(images):
            break

        ax = plt.subplot(5, 5, i + 1)
        image = tf.image.resize(images[i], (100, 100))
        plt.imshow(image.numpy().astype("uint8"))
        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]
        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%", fontsize=8)
        plt.axis("off")

    # If there are more than batch size images, break out of the loop
    if i >= batch_size - 1:
        break

# Hide any empty subplots
for i in range(i + 1, batch_size):
    plt.subplot(5,5, i + 1)
    plt.axis("off")

plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
```

Using Model to make Predictions



Key Observations:

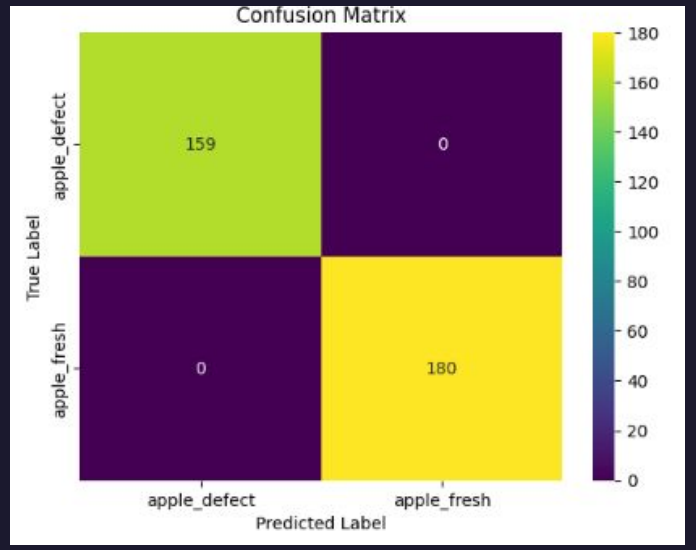
1. **Overall Accuracy:** The model seems to be performing well, with most predictions aligning with the actual classes.
2. **Class-Specific Performance:**
   - **apple_fresh:** The model appears to be more confident in predicting fresh apples, with higher confidence scores.
   - **apple_defect:** The model's confidence in predicting defective apples is generally lower, suggesting potential challenges in distinguishing them from fresh apples.
3. **Incorrect Predictions:** A few predictions are incorrect, particularly for defective apples. This could be due to factors such as the complexity of distinguishing defects, variations in lighting or image quality, or limitations in the model's training data.
4. **Confidence Scores:** The confidence scores provide additional information about the model's certainty in its predictions. Higher confidence scores generally indicate more reliable predictions.

Potential Improvements:

- **Data Augmentation:** Expanding the training dataset with additional images and variations (e.g., different lighting conditions, angles, and defect types) could improve the model's generalization ability.
- **Feature Engineering:** Exploring new features or combinations of existing features might enhance the model's ability to differentiate between fresh and defective apples.
- **Model Architecture:** Experimenting with different model architectures or hyperparameters could potentially improve performance, especially if the current model is underperforming.
- **Error Analysis:** A deeper analysis of the incorrect predictions could help identify patterns or biases in the model's behavior.

In conclusion, the provided image suggests that the model is capable of accurately classifying apples as fresh or defective. However, there is room for improvement, particularly in predicting defective apples with high confidence. Further analysis and experimentation could help refine the model's performance and make it more reliable in real-world applications.

Confusion Matrix



Key Components:
True Label: The actual class of the apple.
Predicted Label: The class predicted by the model.
Color Scale: The color intensity represents the frequency of occurrences in each cell.
\

Interpreting the Results:
Based on the matrix, we can draw the following insights:
Accuracy: The model appears to be highly accurate. The majority of predictions align with the true labels.
Class-Specific Performance:
apple_defect: The model has perfect accuracy in predicting defective apples. None were misclassified as fresh.
apple_fresh: Similarly, the model has perfect accuracy in predicting fresh apples. None were misclassified as defective.
No Misclassifications: The off-diagonal elements (representing incorrect predictions) are all zero, indicating that the model did not confuse any apples between the two classes.
Overall, the model exhibits excellent performance in classifying apples as defective or fresh.
Additional Considerations:
Data Balance: If the dataset is imbalanced (e.g., with a significantly larger number of one class), the accuracy metric might not be sufficient. Precision, recall, and F1-score could provide more comprehensive insights.
Model Complexity: While the model appears to be performing well, it's important to consider the complexity of the model. Overly complex models might overfit the training data and perform poorly on unseen data.
Real-World Applications: Before deploying the model in a real-world setting, it's crucial to evaluate its performance on a diverse test dataset to ensure its generalizability.
In conclusion, the confusion matrix indicates that the model is highly effective in classifying apples as defective or fresh. However, further analysis and considerations are necessary to ensure its suitability for specific applications.

Model conclusion

We created a fairly accurate classifier in Keras.

- The model accurately predicted most of the considered apple fresh or defect categories.
- The model seems to incorrectly label where the apples appear similar in colors and shapes, they can be challengi for the model to classify. This can be improved by training the model with additional data using data augmentation in Keras. Experimenting with adding Batch normalization to the CNN layers to improve and stabilize the learning process, Adding more layers to the neural network, Adding L2 regularization, Experimenting with the dropout rat learning rate

Apart from the above points, the model does a good job of classifying.

Fruit image classification has numerous practical applications, from sorting ripe fruits to detecting diseases. It offers efficiency, accuracy, and potential for optimizing inventory management in various industries. Similarly, Deep Learning can also be extended to broader plant species detection, benefiting agricultural industries. Exploring these use cases c create an awareness about potential AI projects in agricultural domain.

7. Challenges Faced and Problem-Solving Approach

Key Takeaways

Problem statement

PROBLEM STATEMENT DEFINITION VS DATASET

- The dataset was the focus, our first project milestone was to secure the dataset, and I was able to quickly select a popular dataset
- Business objective lack of a clear business objective hindered progress

PROBLEM SOLVING TECHNIQUE

- Refocused the project by defining the problem statement first
- Use case: fruit sorting / grading, defect fruit
- Then aligning dataset usage and model development to meet the defect inspection goal

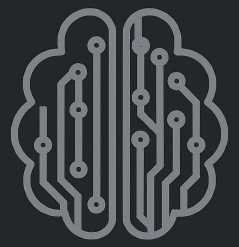
Ethics in AI use

BIASED DATASETS AND JOB DISPLACEMENT

- Unethical job displacement for manual visual inspection
- Biased datasets leading to unfair classification
- Inconsistent labeling across the dataset due subjective interpretations
- if the dataset includes mostly apples from a particular farm, the model might not perform well on apples from other farms with different growing conditions or varieties.
- Overrepresentation of certain classes
- Bias in data collection methods

PROBLEM SOLVING TECHNIQUE

- Collaborating with stakeholders to balance automation with human oversight for fair, accurate, and ethical outcomes
- Low cost implementation for farmers
- Curating diverse training datasets, ensuring transparency in model decisions



Understanding ML

BAD ALGORITHM OR BAD DATA

- •In machine learning, distinguishing between a bad algorithm and bad data involves evaluating model performance and data quality.
- •Unsure what needed tweaking to improve performance

PROBLEM SOLVING TECHNIQUE

•If the algorithm underperforms despite good data, it may be the issue. Conversely, if the model performs poorly due to inconsistent or insufficient data, the data quality is likely the problem

•Learning through research and mentorship

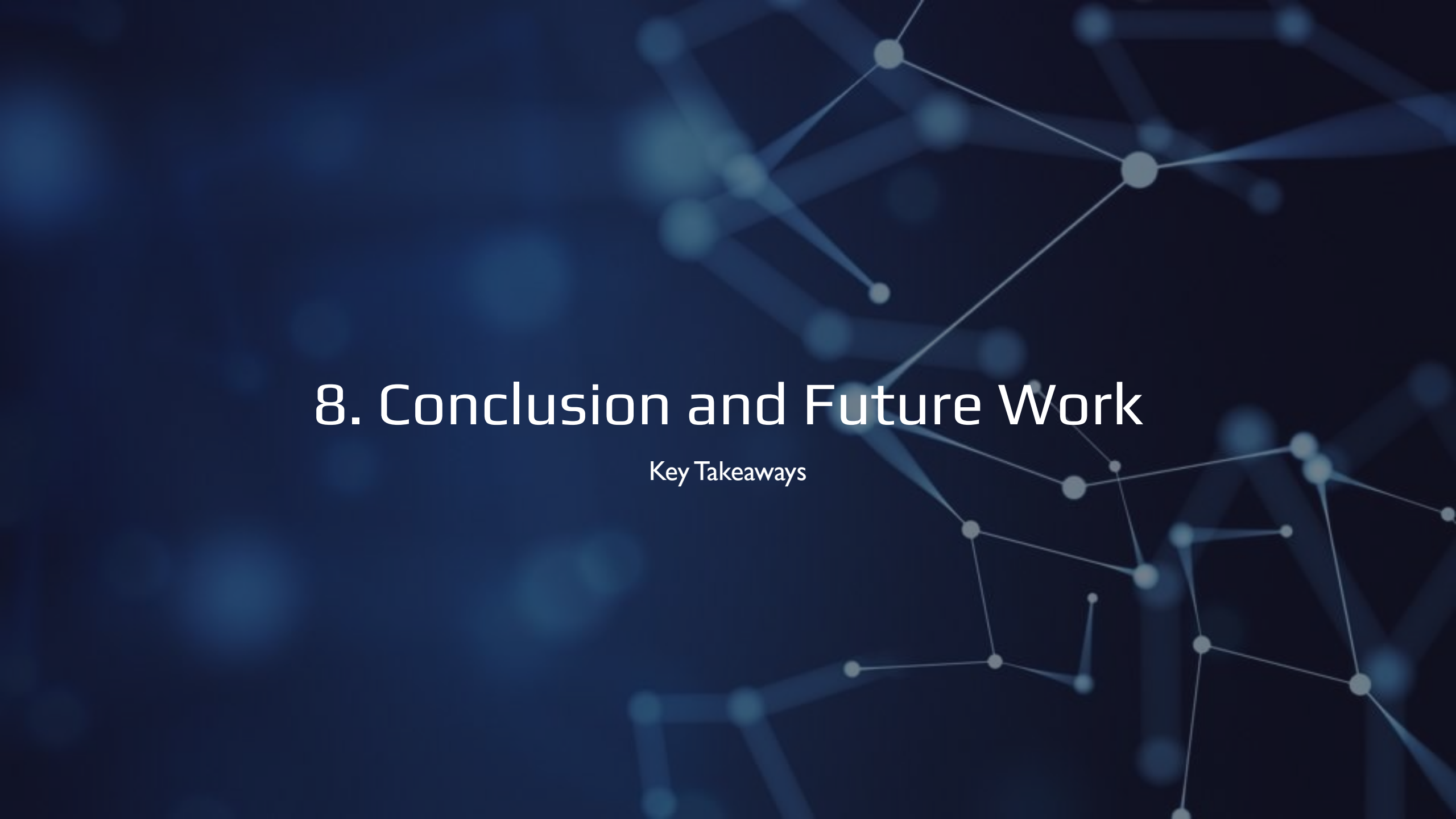
Learning Curve

LEARNING RATE

- To make reference with understanding.
- Fostering critical thinking, hands-on practice, and personalized problem-solving approaches.
- Imposter syndrome
- Unsure what needed tweaking to improve performance

PROBLEM SOLVING TECHNIQUE

- Gain expertise through research, mentorship, and detailed project planning.
- Prioritize, align goals, manage resources, optimize capabilities, and set timelines.
- Prioritize tasks, delegate effectively, and streamline processes for efficiency.
- Focus on recognizing achievements, challenging negative thoughts, and seeking support from peers or mentors.
-



8. Conclusion and Future Work

Key Takeaways

Conclusion

The project idea initially emerged from a potential future work interest for a warehouse system specifically for sorting and grading fruit automation. The project demonstrates the potential of automating apple defect inspection but requires further research for better accuracy.

Recommendations include expanding the dataset, adding more defect types, using OpenCV, and introducing a diseased apple category.

Future Work

The project has the potential to reduce labor costs and improve efficiency in the apple processing industry.

The project could be further developed to inspect other types of fruit or vegetables.

The project could be integrated with other quality control systems in the apple processing industry.

9. Q&A Interaction

9. References

References

1. TensorFlow/Keras documentation: https://www.tensorflow.org/
2. Multilabel Fruits Classification | CNN | Keras' kaggle.com Devashree Madhugiri https://www.kaggle.com/code/devsubhash/multilabel-fruits-classification-cnn-keras
3. SVMClassification_OnAppledatasetOpenCV https://github.com/sarathbabu123/SVMClassification_OnAppledatasetOpenCV
4. Real-time visual inspection system for grading fruits using computer vision and deep learning techniques https://www.sciencedirect.com/science/article/pii/S2214317321000056
5. On line detection of defective apples using computer vision system combined with deep learning methods, https://www.sciencedirect.com/science/article/abs/pii/S0260877420302004
6. Classify disease in apples https://www.kaggle.com/code/yashvi/classify-diseases-in-apple-trees-beginner
7. Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville http://www.deeplearningbook.org/
8. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, Aurélien Géron
9. Deep Learning for Computer Vision, Dr Stephen Moore, 006.3 SHA-COM
10. Artificial Intelligence Programming with Python from Zero to Hero, Perry Xiao, 005.133 XIA-COM

Thank you

Zubaidah binte Sallehuddin

GitHub

SCAN ME

https://github.com/sallez9/SCTPAIMLcapstone

LinkedIn

SCAN ME

https://www.linkedin.com/in/zubaidah-sallehuddin/