Project Raport

Course:

Biologically Inspired Artificial Intelligence

Topic

Fruit Image Classification using Convolutional Neural Networks

Authors:
Michal Sikora
Kacper Czerniak

1.Introduction

The aim of the project was to build and train a convolutional neural network (CNN) capable of classifying images of fruits into nine predefined categories. The dataset consisted of color images representing different fruit types, each stored in a separate folder corresponding to its class label. The model was developed and trained using the TensorFlow library in a Google Colab environment, with additional use of data augmentation techniques to improve generalization and robustness.

2. Analysis of the Task

The task consists in developing a classification model based on convolutional neural networks (CNNs) to recognize fruit types from image data. This problem represents a typical supervised learning scenario, where the model is trained on labeled examples to learn patterns and make predictions on unseen data.

To approach this challenge, we experimented with multiple CNN architectures, varying the number of convolutional layers, the size of dense layers, and the presence of regularization techniques such as dropout. Additionally, we tested different training strategies and data preprocessing parameters, including data augmentation, in order to improve the model's performance and reduce the risk of overfitting. These variations allowed us to analyze which configurations led to better generalization and more accurate classification results.

3. Dataset

The dataset used in this project consists of images of nine different fruit categories. Each class contains exactly 40 RGB images, resulting in a total of 360 images. The data is organized in a folder structure, where each subfolder corresponds to a specific class label:

- **apple fruit**
- **banana fruit**
- **cherry fruit**
- **chickoo fruit**
- **grapes fruit**

- **kiwi fruit**
- **mango fruit**
- **orange fruit**
- **strawberry fruit**

All images were manually placed into these folders and stored in Google Drive. During training, the dataset was automatically split into 80% training data (288 images) and 20% validation data (72 images) using the `validation_split` parameter of Keras's `ImageDataGenerator`.

To improve generalization and help the model learn from limited data, several augmentation techniques were applied to the training set. These included random rotations, width and height shifts, zooming, and horizontal flips. These transformations were generated dynamically at training time, so the number of actual image files remained the same.

4. Technologies Used

The following technologies and tools were used throughout the project:

- **Google Colab** – a cloud-based Python environment used for developing, training, and testing the model. It provides free access to GPUs, which significantly accelerates the training process.

- **TensorFlow & Keras** – open-source machine learning libraries used to build and train the convolutional neural networks (CNNs). Keras, as a high-level API, simplifies model creation and training.

- **Google Drive** – used for storing the dataset and saving trained models. Integration with Colab allowed for easy access to image files and persistent storage across sessions.

- **NumPy** – used for efficient numerical operations and data manipulation.

- **Matplotlib & Seaborn** – visualization libraries used for plotting training metrics (accuracy, loss) and displaying the confusion matrix.

5. Data Preprocessing

In order to prepare the dataset for training and improve model performance, a set of preprocessing steps was applied using the `ImageDataGenerator` class from Keras. This component allowed for on-the-fly image loading, normalization, and augmentation, which are essential in small-scale image classification tasks.

All images were first rescaled by dividing pixel values by 255, converting them from the standard range [0, 255] to [0.0, 1.0]. This normalization step speeds up convergence and stabilizes the learning process. The images were also resized to a fixed shape of **128 × 128 pixels**, which matches the input shape expected by the convolutional neural network.

The dataset was then split into two parts:

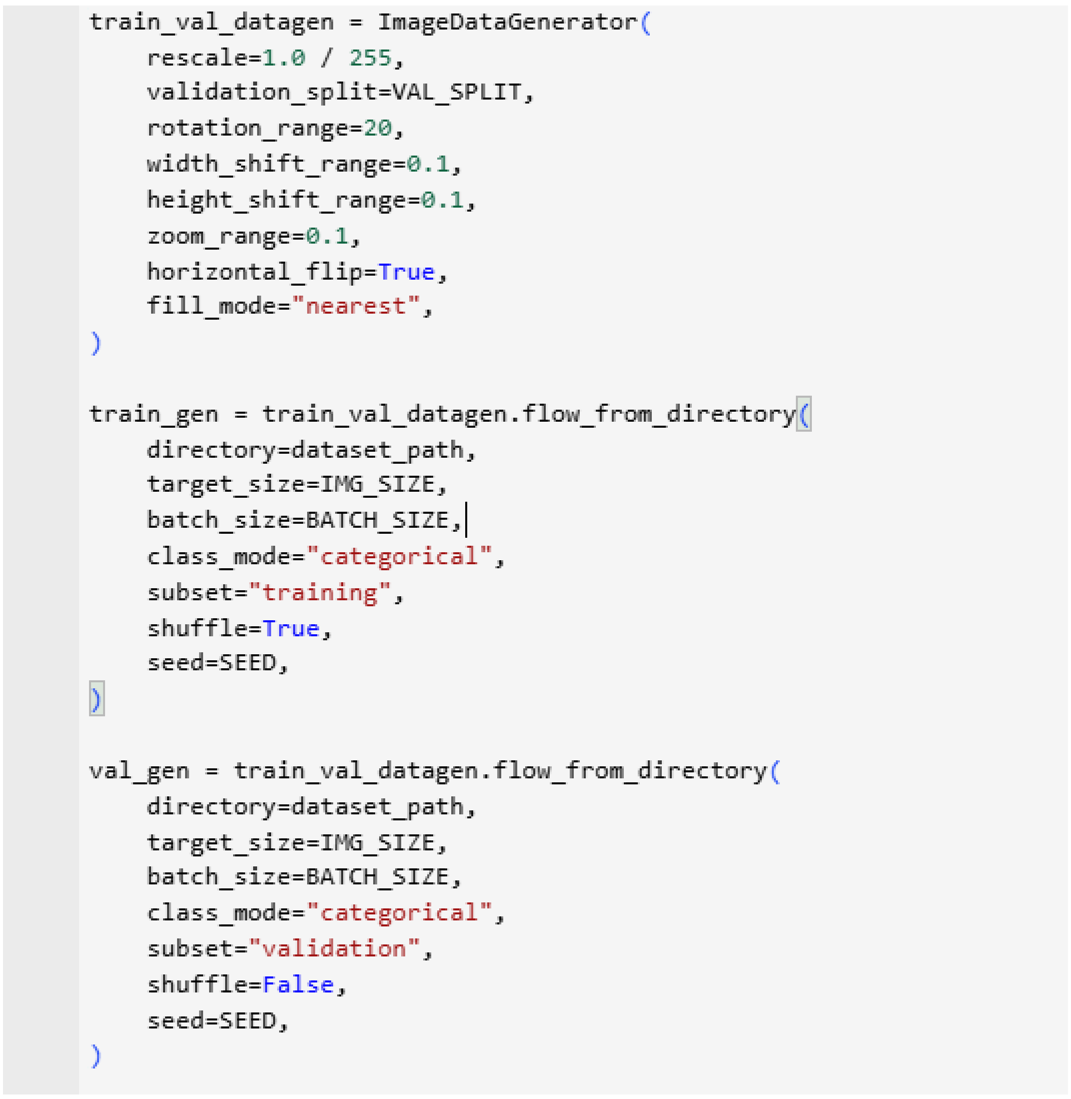
- **80% for training**, where data is shuffled to increase randomness

- **20% for validation**, where the order of data remains fixed for consistency in evaluation

To address the limited number of samples (only 40 images per class), real-time **data augmentation** was applied to the training images. The following random transformations were included:

- **Rotation**: up to 20 degrees, to simulate different angles
- **Width and height shift**: up to 10% of the image size
- **Zoom**: in or out up to 10%
- **Horizontal flip**: simulating mirror images
- **Fill mode**: filling in empty pixels with the value of the nearest neighbor

These transformations do not increase the number of files on disk, but instead create slightly modified versions of the training images during each epoch. This increases variability in training data and helps the model better generalize to unseen inputs.



```
train_val_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    validation_split=VAL_SPLIT,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest",
)

train_gen = train_val_datagen.flow_from_directory(
    directory=dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="training",
    shuffle=True,
    seed=SEED,
)

val_gen = train_val_datagen.flow_from_directory(
    directory=dataset_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    subset="validation",
    shuffle=False,
    seed=SEED,
)
```

6. Model Architecture and Comparison

As part of this project, we designed and evaluated three different convolutional neural network architectures. The aim was to assess how changes in model complexity, depth, and regularization affect classification performance. The visual summaries below show the architecture and number of trainable parameters in each case.

Model 1 – Final Model (Best Performance)

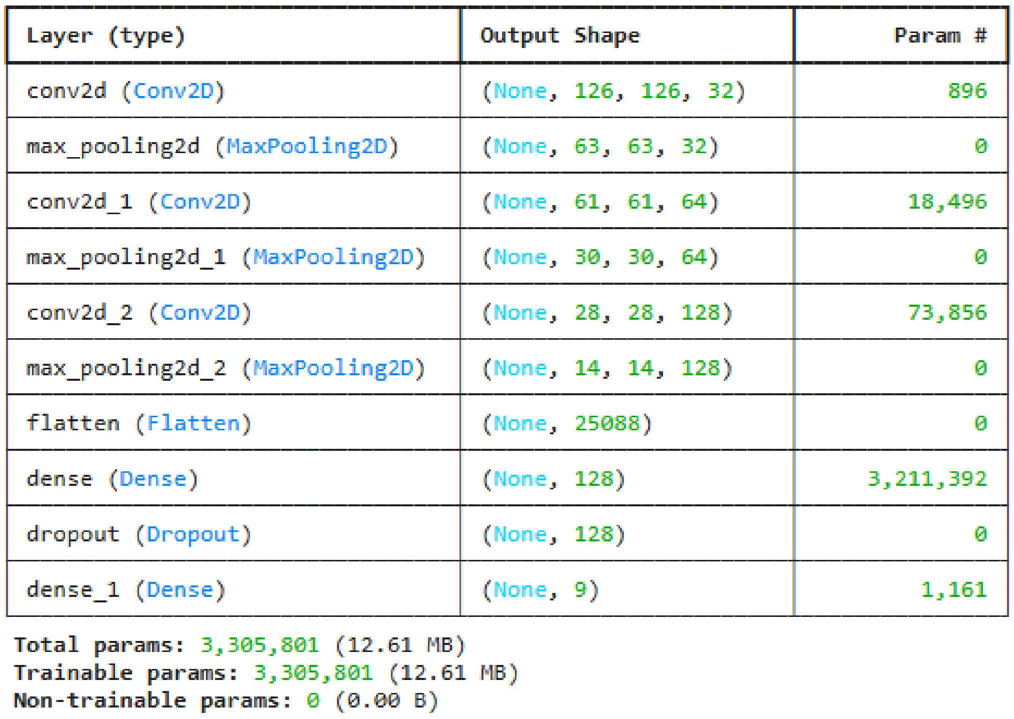
This model showed the **best overall performance** in terms of accuracy and generalization. It consists of three convolutional layers with increasing filter sizes (32 → 64 → 128), each followed by max pooling to reduce dimensionality. The flattening layer prepares the data for two dense layers: one with 128 neurons, followed by a dropout of 50%, and the final classification layer with softmax activation.

Key features:

- Deep feature extraction: 3 Conv2D layers learn progressively abstract patterns.
- Dropout layer (0.5) helps reduce overfitting.
- Dense(128) + Dense(9) creates a strong decision layer.
- Balanced in terms of depth, performance, and training time.

Total parameters: 3,305,801
Model size: ~12.6 MB
Output shape before classification: (None, 128)

This model proved to be optimal for the dataset, combining good performance and robustness without being overly large.



```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                 │ (None, 126, 126, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 63, 63, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 61, 61, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 30, 30, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 28, 28, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 14, 14, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 25088)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │     3,211,392 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 9)              │         1,161 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 3,305,801 (12.61 MB)
 Trainable params: 3,305,801 (12.61 MB)
 Non-trainable params: 0 (0.00 B)
```

Model 2 – Small/Initial Prototype

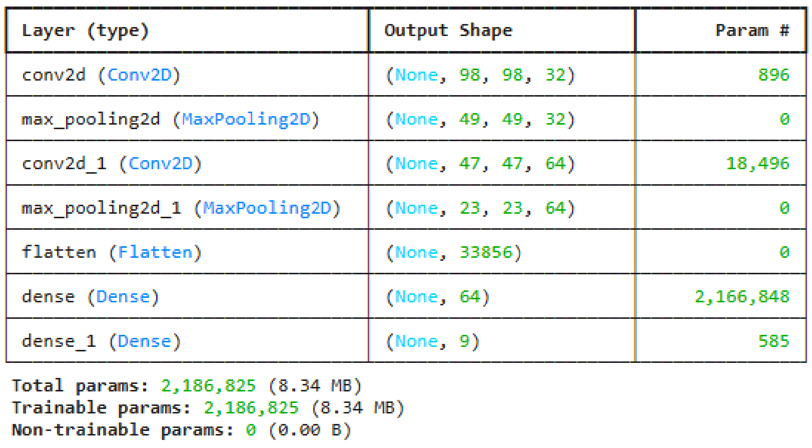
This was the simplest architecture, created at the early stages of development. It has only two convolutional layers and a smaller dense layer (64 neurons). Although it trained quickly and required fewer resources, its validation accuracy was significantly lower than the other models.

Key features:

- Only 2 Conv2D layers (32 → 64)
- No dropout, which increased overfitting
- Dense(64) as the main decision layer

Total parameters: 2,186,825
Model size: ~8.3 MB
Output shape before classification: (None, 64)

This model served as a baseline for verifying data loading and training functionality but was insufficient for final use.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten (Flatten)	(None, 33856)	0
dense (Dense)	(None, 64)	2,166,848
dense_1 (Dense)	(None, 9)	585

```
Total params: 2,186,825 (8.34 MB)
Trainable params: 2,186,825 (8.34 MB)
Non-trainable params: 0 (0.00 B)
```

Model 3 – Deepest Model (Most Parameters)

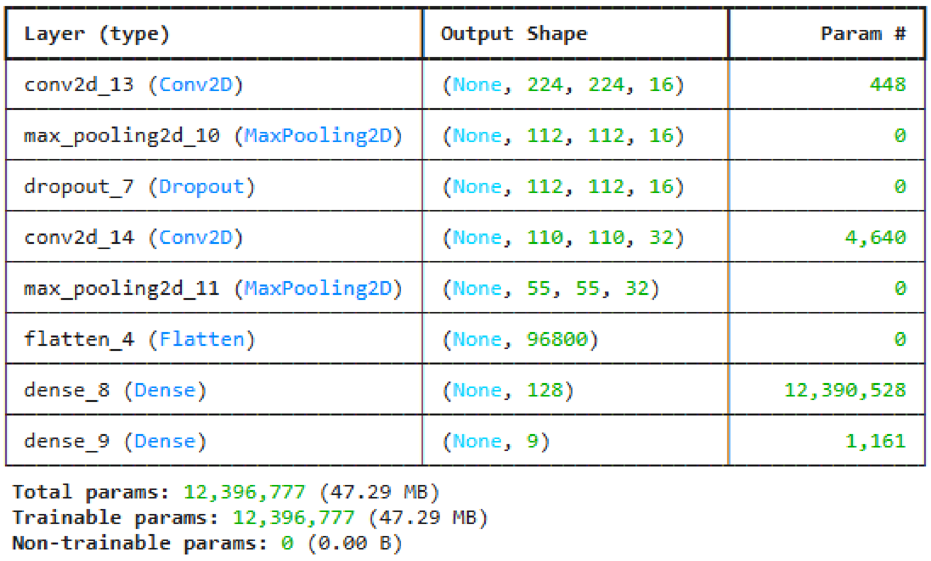
This architecture included Dropout earlier in the model and worked with a larger input size (224×224), resulting in a very large flattened layer (96,800 units!). While conceptually interesting, the model became too heavy and overparameterized, leading to unnecessary complexity.

Key features:

- Conv2D layers: (16 → 32), with padding and dropout early
- Massive Dense(128) after Flatten(96800)
- Uses the most memory and training time

Total parameters: 12,396,777
Model size: ~47.3 MB
Output shape before classification: (None, 128)

Despite its size, this model didn't outperform the more efficient final version, making it less practical for this use case.



Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 224, 224, 16)	448
max_pooling2d_10 (MaxPooling2D)	(None, 112, 112, 16)	0
dropout_7 (Dropout)	(None, 112, 112, 16)	0
conv2d_14 (Conv2D)	(None, 110, 110, 32)	4,640
max_pooling2d_11 (MaxPooling2D)	(None, 55, 55, 32)	0
flatten_4 (Flatten)	(None, 96800)	0
dense_8 (Dense)	(None, 128)	12,390,528
dense_9 (Dense)	(None, 9)	1,161

```
Total params: 12,396,777 (47.29 MB)
Trainable params: 12,396,777 (47.29 MB)
Non-trainable params: 0 (0.00 B)
```

7. Model Training and Evaluation

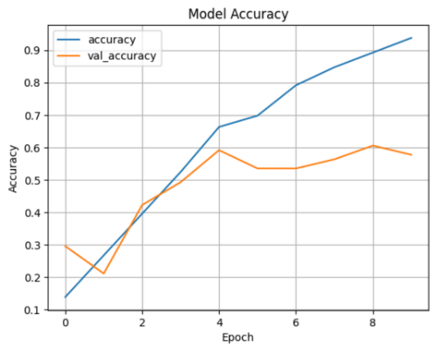
7.1 Training and Evaluation of the Smallest Model

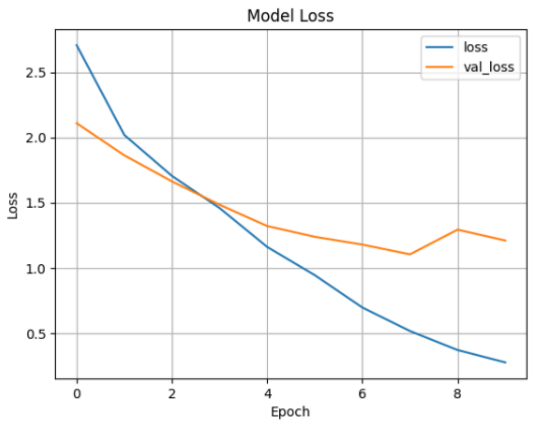
The first model tested was the simplest architecture in this project. It consisted of only two convolutional layers followed by a small dense layer with 64 neurons. Due to its shallow structure and lack of regularization (e.g., dropout), the model trained quickly but showed limited ability to generalize to unseen data.

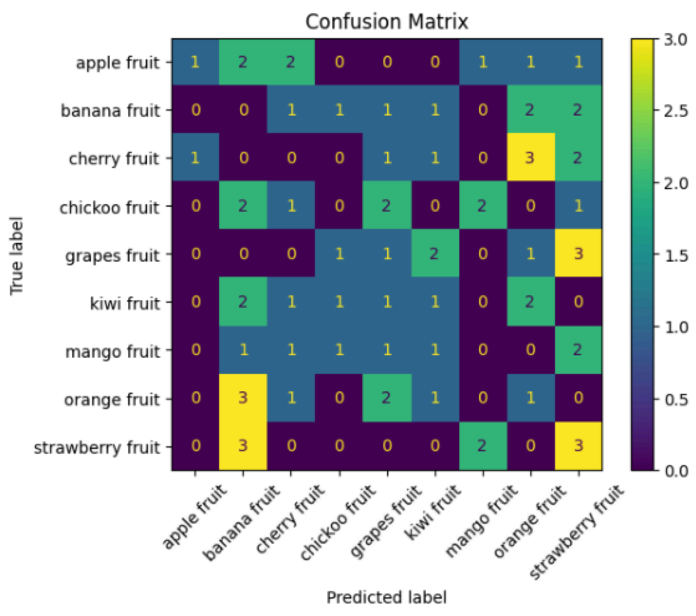
After training, the model reached a **validation accuracy of 57.75%** and a **validation loss of 1.2098**. The training and validation accuracy curves show that the model was able to learn some basic features from the dataset, but started to overfit after a few epochs. The loss graph confirms that the validation loss remained relatively high compared to the training loss.

While the accuracy value may initially suggest that the model performed reasonably well, the **confusion matrix reveals a completely different picture**. The model frequently misclassified fruits, and its predictions lacked consistency across several classes. Many categories were confused with one another, especially those that shared similar visual features.

> **Conclusion:** Although the validation accuracy of **57.75%** might seem acceptable at first glance, the confusion matrix clearly shows that the model fails to properly distinguish between classes. This highlights its inability to extract meaningful high-level features, making it unsuitable for practical use.





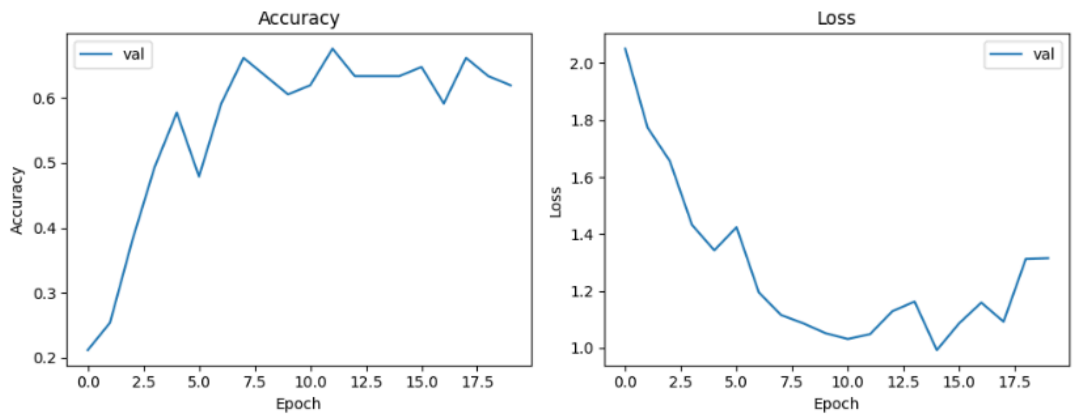


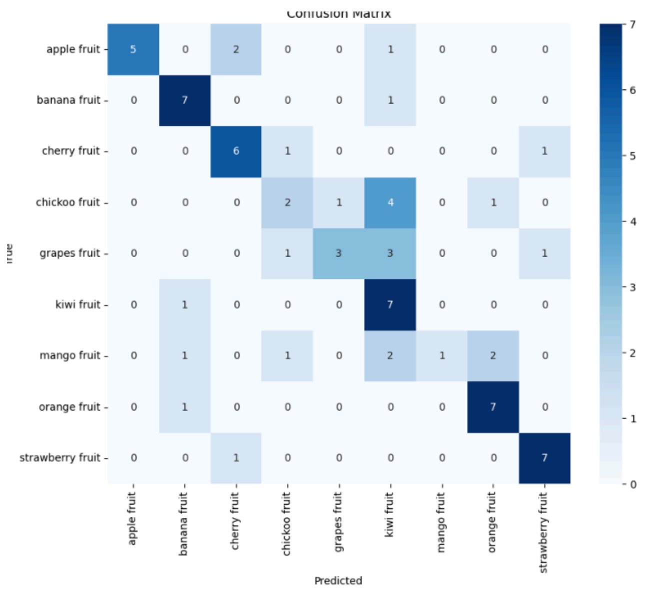
7.2 Training and Evaluation of the Largest Model

The third model tested was the most complex architecture among all variants. It featured a high-resolution input (224×224 pixels), multiple convolutional and pooling layers, and an extremely large flattening layer of 96,800 elements. Despite the use of dropout and strided pooling, the model had over **12.3 million trainable parameters**, making it significantly heavier than other models.

After training, the model achieved a **validation accuracy of 60.56%** and a **validation loss of 1.2016**. Although slightly better than the smallest model in terms of accuracy, the learning curve showed early signs of stagnation. The accuracy plot indicates that the performance fluctuated across epochs, and the loss curve shows that the model did not continue improving beyond a certain point.

Despite its size, the model did not significantly outperform the more balanced architecture used in Model 2. The confusion matrix shows that although some classes (e.g., *strawberry fruit*, *kiwi fruit*, *orange fruit*) were well classified, others still suffered from frequent misclassification (*chickoo fruit*, *mango fruit*, *grapes fruit*).





Conclusion: The results suggest that increasing model size and complexity does not guarantee better performance. This model required more memory and longer training time, but its improvement over simpler models was minimal. It was ultimately not selected as the final solution.

7.3 Training and Evaluation of the Best Model

The final and most balanced model architecture turned out to be the best-performing network in the project. It combined depth, dropout regularization, and sufficient learning capacity without excessive complexity. The model consisted of three convolutional layers (with 32, 64, and 128 filters), followed by a dense layer with 128 units and a dropout layer set to 0.5. The final layer used softmax activation to classify the input image into one of the nine fruit categories.

To further evaluate the model's robustness, it was trained in **two different configurations**: one **without data augmentation**, and another **with augmentation and early stopping**. The results are summarized below.

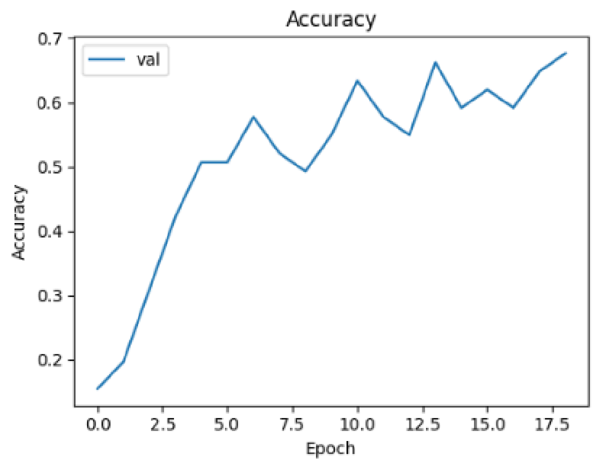
Training 1 – Without Data Augmentation (20 epochs)

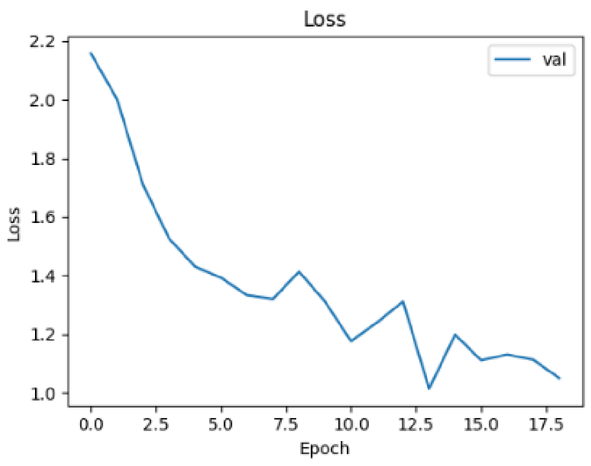
In the first version, the model was trained using only rescaled and resized input images. No rotation, shifting, zooming, or flipping was applied. The training lasted for a fixed 20 epochs.

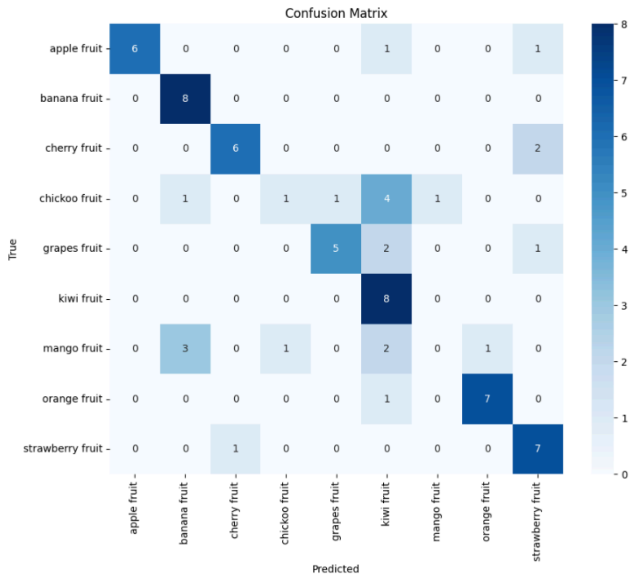
Results:

- **Validation accuracy:** 67.61%
- **Validation loss:** 1.0496

The accuracy curve showed steady improvement over epochs, while the loss decreased consistently. However, the lack of augmentation may have limited the model's ability to generalize to more variable input.







Training 2 – With Data Augmentation + EarlyStopping

In the second version, the model was trained with data augmentation enabled using the following transformations:

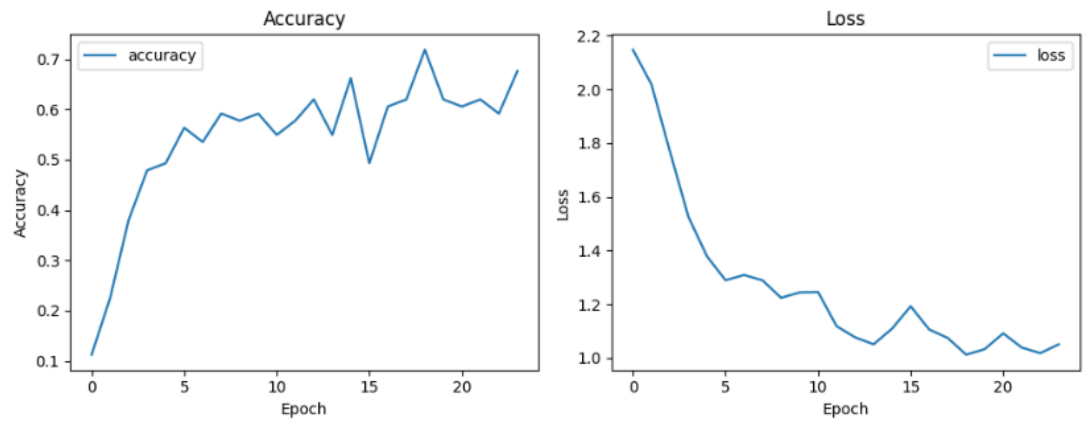
- Rotation (up to 20°)
- Width and height shift (up to 10%)
- Zoom (10%)
- Horizontal flipping

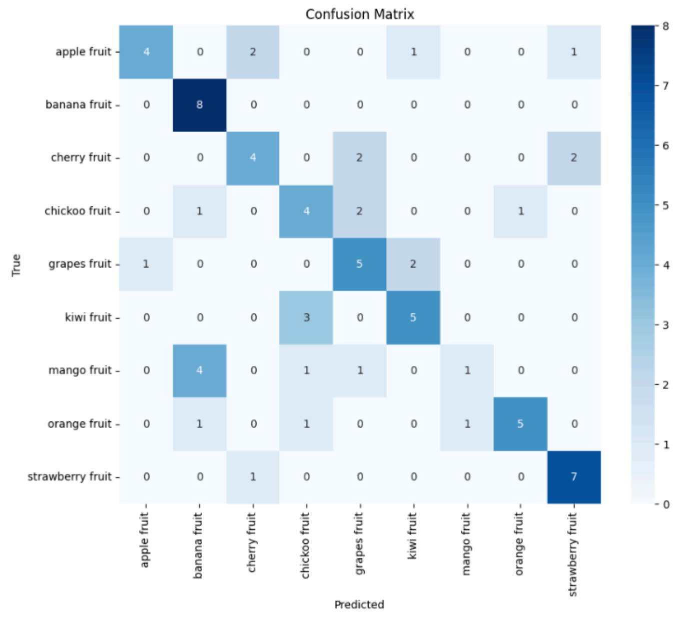
Training was allowed to run for up to **60 epochs**, but the use of `EarlyStopping(patience=5)` interrupted the process when no further improvement in validation loss was detected.

Results:

- **Validation accuracy:** 67.61%
- **Validation loss:** 1.0514

Interestingly, despite the longer training period and data augmentation, the validation accuracy remained exactly the same as in the first version. However, this model may generalize better in real-world scenarios due to exposure to transformed images.





	apple fruit	banana fruit	cherry fruit	chickoo fruit	grapes fruit	kiwi fruit	mango fruit	orange fruit	strawberry fruit
apple fruit	4	0	2	0	0	1	0	0	1
banana fruit	0	8	0	0	0	0	0	0	0
cherry fruit	0	0	4	0	2	0	0	0	2
chickoo fruit	0	1	0	4	2	0	0	1	0
grapes fruit	1	0	0	0	5	2	0	0	0
kiwi fruit	0	0	0	3	0	5	0	0	0
mango fruit	0	4	0	1	1	0	1	0	0
orange fruit	0	1	0	1	0	0	1	5	0
strawberry fruit	0	0	1	0	0	0	0	0	7

8. Final Test on External Image

To further evaluate the generalization capabilities of the best-performing model, an image **not included in the training or validation dataset** was used as input. The image depicts a bunch of bananas on a yellow background and was selected to verify whether the model could correctly classify new, previously unseen data.

The image was preprocessed to match the input shape of the model (128 × 128 pixels, normalized), and passed through the trained network. As shown in the screenshot below, the model predicted the correct class:

Prediction: *banana fruit*
Confidence: 56.58%
Top 3 predictions:

- banana fruit: 56.58%
- mango fruit: 30.37%
- orange fruit: 8.48%

```
Top 3 predictions:
banana fruit: 56.58%
mango fruit: 30.37%
orange fruit: 8.48%
```

Despite the presence of a similar yellow background, which could potentially cause confusion with other classes (e.g., mango or orange), the model correctly identified the fruit with a reasonable confidence margin.

9. Summary of Results

During the course of the project, three different convolutional neural network architectures were designed, trained, and evaluated. Each model varied in depth, complexity, and training strategy. The goal was to determine which configuration would yield the highest classification accuracy while maintaining good generalization.

The final model, trained both with and without data augmentation, consistently outperformed the others. The simplest model served as a useful baseline, while the largest model consumed the most resources without providing a meaningful performance gain.

The table below summarizes the validation accuracy and loss achieved by each configuration:

Model	**Validation Accuracy**	**Validation Loss**
Model 1 – Final (no augmentation, 20 epoch)	**67.61%**	1.0496
Model 1 – Final (with augmentation, 60 epoch)	**67.61%**	1.0514
Model 2 – Smallest	57.75%	1.2098
Model 3 – Largest	60.56%	1.2016

10. Conclusion

The project successfully demonstrated the use of convolutional neural networks for multi-class fruit image classification. Through a series of experiments involving different model architectures and training strategies, it was shown that a well-balanced CNN with regularization and moderate depth delivers the best performance. The final model achieved a validation accuracy of 67.61% and proved capable of correctly classifying unseen images. These results confirm that even with a relatively small dataset, careful preprocessing and model tuning can lead to robust and reliable image classification.

11. Reference

https://www.kaggle.com/datasets/shreyapmaher/fruits-dataset-images/data
https://www.youtube.com/watch?app=desktop&v=zfiSAzpy9NM
https://www.tensorflow.org/api_docs
https://en.wikipedia.org/wiki/Convolutionalneuralnetwork