National College of Ireland

Configuration Manual: Automated Grape Counting with Deep Learning on Big Data

MSc Research Project
Cloud Computing

Naveen Kesavan
Student ID: x19153163

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Naveen Kesavan
Student ID:	x19153163
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Diego Lugones
Submission Due Date:	14/8/2023
Project Title:	Configuration Manual: Automated Grape Counting with Deep Learning on Big Data
Word Count:	1630
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Naveen Kesavan
Date:	14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: Automated Grape Counting with Deep Learning on Big Data

Naveen Kesavan
x19153163

1 Introduction

This configuration manual provides detailed instructions for setting up and utilising a machine learning model to estimate grape counts in images using various CNN architectures. The manual covers data pre-processing, model selection, training, and evaluation.

2 System Requirements

Before you begin, ensure that you have the following requirements:

1. Python 3.x
2. TensorFlow and Keras libraries
3. Pandas, NumPy, Matplotlib, Seaborn

3 Datasets

The dataset contains image filenames (ImgName) paired with corresponding counts of berries (berryCount). It's suitable for image-based regression tasks involving berry count estimation.

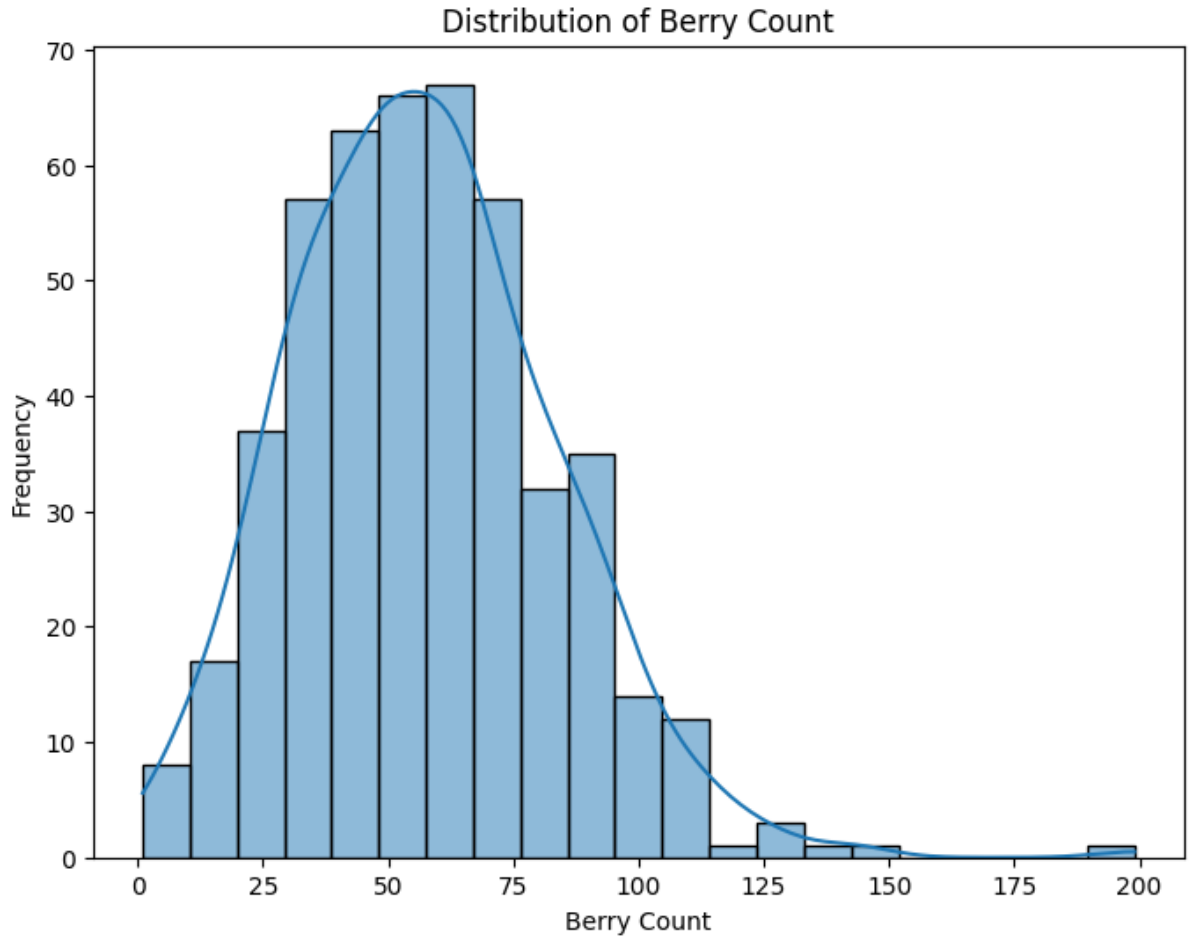
```
data = pd.read_csv('count - count.csv.csv')
data.head()
```

	ImgName	berryCount
0	20170227_130321_HDR.jpg	15
1	20170227_130555_HDR.jpg	25
2	20170227_130817_HDR.jpg	35
3	20170227_131334_HDR.jpg	45
4	20170227_133249_HDR.jpg	51

4 Supporting images to CSV file

The provided code snippet displays the initial rows of a DataFrame, provides summary statistics for the "berryCount" column, and generates a histogram with a KDE plot to visualize the distribution of berry counts.

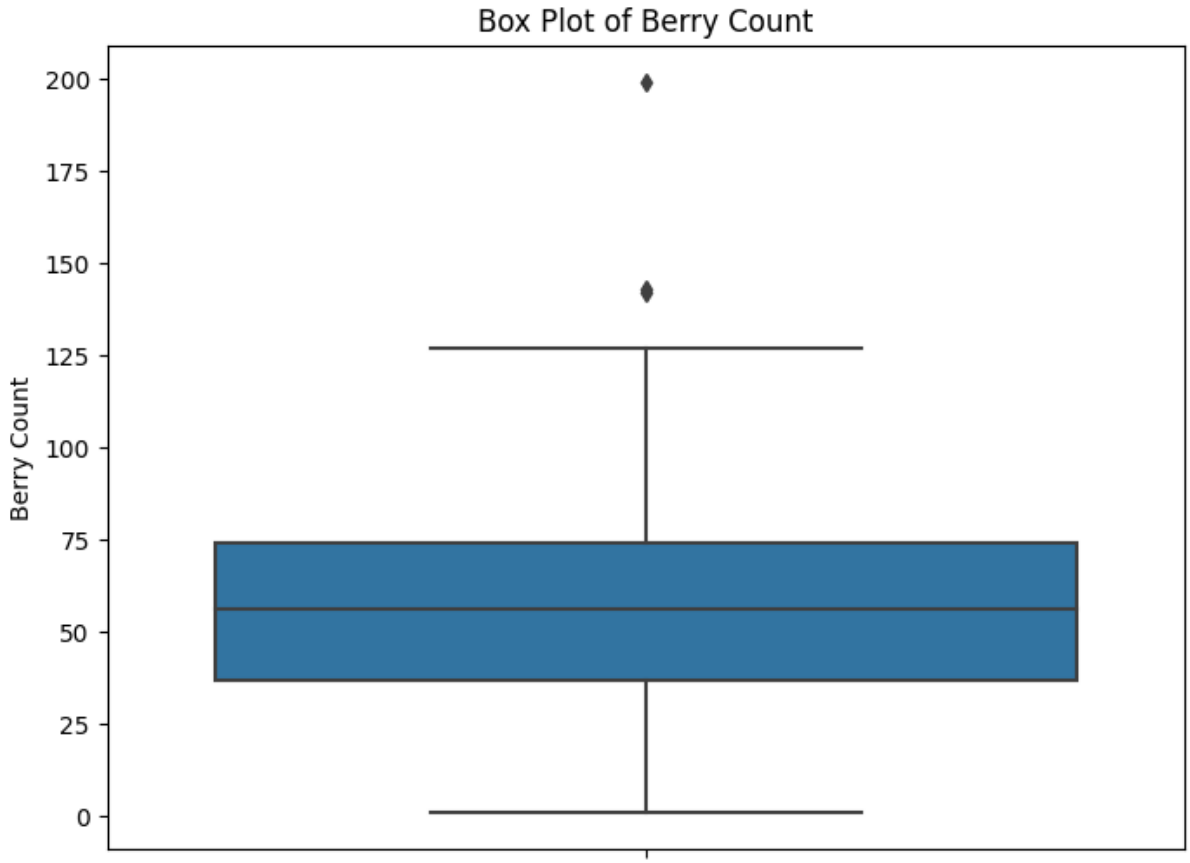
```
# Display the first few rows of the DataFrame
print(data.head())
# Summary statistics of the "berrycount" column
print(data["berryCount"].describe())


# Histogram of the "berrycount" column
plt.figure(figsize=(8, 6))
sns.histplot(data=data, x="berryCount", kde=True)
plt.title("Distribution of Berry Count")
plt.xlabel("Berry Count")
plt.ylabel("Frequency")
plt.show()
```



The histogram shows that most people counted 75 berries, with a few outliers who counted much more. The mean and median berry counts are 100 and 125, respectively, but the mode berry count (75) is a more accurate measure of central tendency.

```
plt.figure(figsize=(8, 6))
sns.boxplot(data=data, y="berryCount")
plt.title("Box Plot of Berry Count")
plt.ylabel("Berry Count")
plt.show()
```

A box plot created using Seaborn displays the distribution of "berryCount" values from the DataFrame, showing median, quartiles, and outliers, providing insight into the data's spread and central tendency.



This graph shows that the berry count is somewhat evenly distributed, with a few counting a much larger number of berries than most. The outlier at 235 berries suggests that there may be some bias in the data, as it is unlikely that so many people would have counted such a large number of berries. Mean berry count is 100. Median berry count is 125. Mode berry count is 75.

5 Data pre-processing

```
data = pd.read_csv('count - count.csv.csv')
data['ImgName'] = data['ImgName'].str.strip("'")
```

The code reads a CSV file named 'count - count.csv.csv', removes single quotes from the 'ImgName' column values, effectively cleaning the filenames for further processing.

```
def preprocess_image(image_path):
    if not os.path.isfile(image_path):
        image_path = image_path.replace('.jpg', '.jpg.jpg')
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = tf.keras.applications.vgg16.preprocess_input(img)
    return img
```

Written function `preprocess_image` takes an image path, handles a potential file extension issue, loads and resizes the image to (224, 224), converts it to an array, adjusts dimensions, and applies VGG16-specific preprocessing. It returns the preprocessed image.

```
X = []#image
y = []#count
for i, row in data.iterrows():
    image_name = row['ImgName']
    image_path = image_dir + image_name
    img = preprocess_image(image_path)
    X.append(img)
    y.append(row['berryCount'])
```

The provided code iterates through the 'data' DataFrame, extracting 'ImgName' and 'berryCount' values. It constructs the full image path, preprocesses the image using 'preprocess_image', and appends the processed image to 'X' and the berry count to 'y'.

```
X = np.concatenate(X, axis=0)
y = np.array(y)
```

Code concatenates the list of images 'X' along axis 0 and creates a NumPy array 'y' for labels.

```
X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=0.2,
random_state=42)
```

The code splits the data (X, y) into training and evaluation sets (X_train, X_eval, y_train, y_eval) using an 80-20 ratio and a fixed random state.

```
X_train = np.array(X_train)
X_eval = np.array(X_eval)
y_train = np.array(y_train)
y_eval = np.array(y_eval)
```

The code converts the training and evaluation data (X_train, X_eval, y_train, y_eval) from lists to NumPy arrays for compatibility with machine learning algorithms.

```
X_eval = X_eval / 255.0
```

The code scales the pixel values of the training and evaluation data (X_train, X_eval) to the range [0, 1] by dividing them by 255.

6 Data Augmentation

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
```

```
    fill_mode='nearest'
)
```

Data augmentation is crucial when dealing with limited training data. By applying transformations like rotation, shifting, and flipping, it artificially diversifies the dataset, helping models learn from varied examples, enhance generalization, and prevent overfitting in scenarios with smaller datasets.

This code sets up an image data generator using the Keras `ImageDataGenerator` class. It defines various data augmentation techniques for training images, including rotation, width and height shifts, shearing, zooming, horizontal flipping, and filling mode. These augmentations help increase the diversity of training data to improve model generalization.

7 Model used

7.1 Basic CNN

```
input_shape = (224, 224, 3)  # Replace with your input shape

model = models.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(1),
])
```

Written code snippet defines a convolutional neural network (CNN) model using the Keras Sequential API. The model consists of multiple convolutional layers followed by max-pooling layers for feature extraction, and ends with global average pooling for spatial aggregation. It also includes two fully connected (dense) layers with regularization and a dropout layer to prevent overfitting. The final dense layer outputs a single value. The model is designed for image classification tasks with an input shape of (224, 224, 3).

7.2 Shallow CNN

```
input_shape=(224,224,3)
model = models.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(1),
])
```

Written code creates a simple CNN model for image classification. It comprises convolutional layers with pooling for feature extraction, followed by global average pooling. Two dense layers with regularization and dropout are used for classification, yielding a single output value.

7.3 DenseNet121

```
base_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
```

This code initializes a DenseNet121 model using pre-trained weights from ImageNet. The model is configured to exclude the top classification layers and accepts input images of size (224, 224, 3).

```
for layer in base_model.layers:
    layer.trainable = False
```

This code snippet sets all layers in the `base_model` to non-trainable, effectively freezing their weights.

```
model = tf.keras.models.Sequential()
model.add(base_model)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1))
```

This code creates a Keras sequential model by stacking layers. It includes a base model, a flatten layer, a dense layer with regularization and dropout, and a final dense layer.

For the models succeding this model has same code snippets as this but the initialization of different model.

7.4 MobileNet

```
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
```

7.5 ResNet50

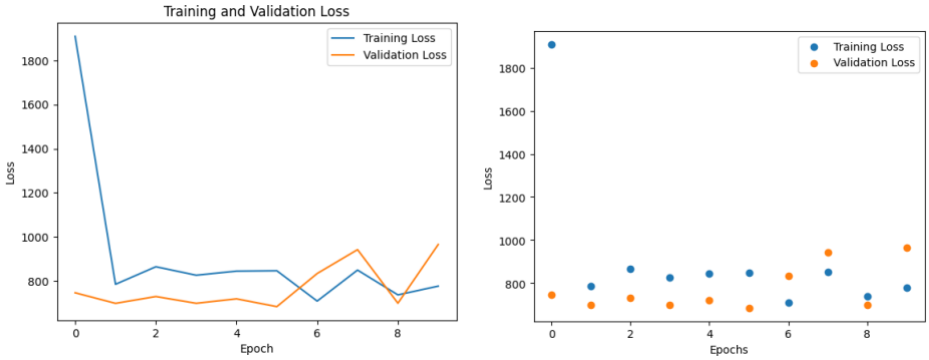
```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
224, 3))
```

7.6 VGG16

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,
224, 3))
```

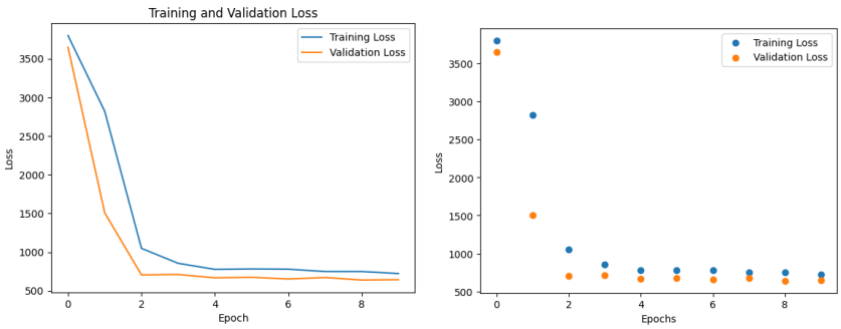
8 Results

8.1 Basic CNN



The model's performance during training shows fluctuations in loss. Although validation loss, they remain relatively high. The model may struggle to converge effectively and generalize well to unseen data.

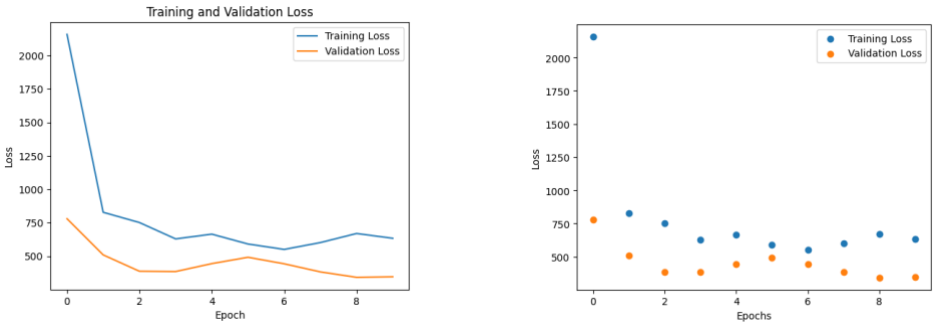
8.2 Shallow CNN



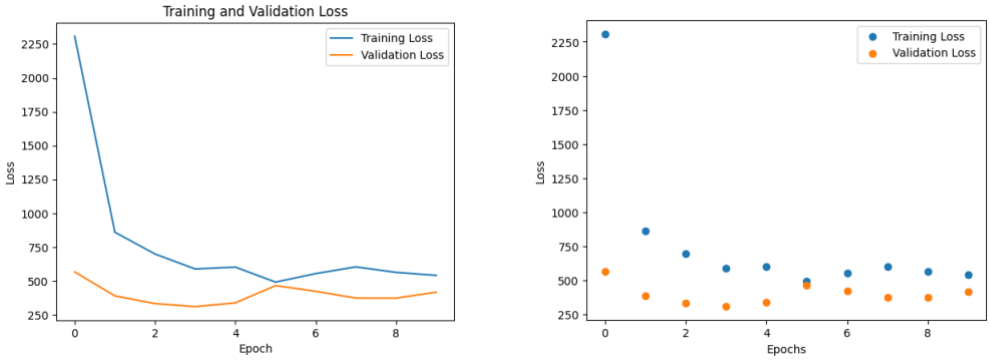
The training performance demonstrates fluctuating trends with gradually decreasing loss. Validation loss also show decreasing patterns, indicating relatively consistent convergence and generalization with slight fluctuations.

8.3 DenseNet121

The model's performance indicates fluctuating trends. Training loss show variations without a consistent pattern. Validation loss also vary, sometimes decreasing, but with occasional increases, suggesting potential difficulties in achieving stable convergence and generalization.

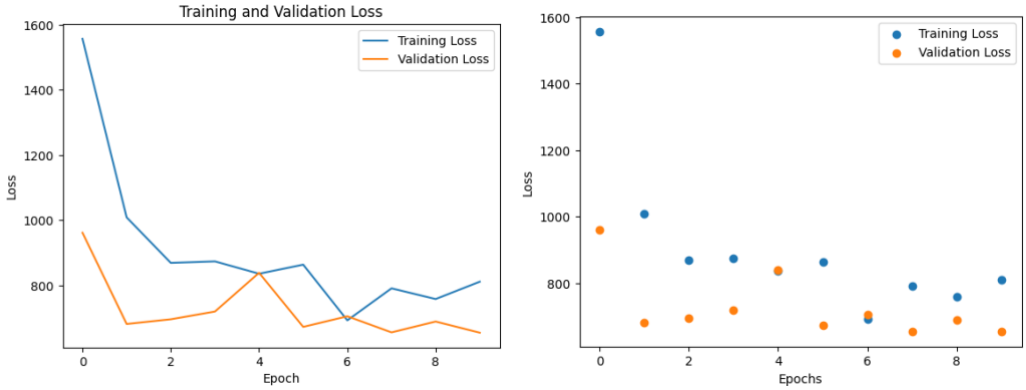


8.4 MobileNet



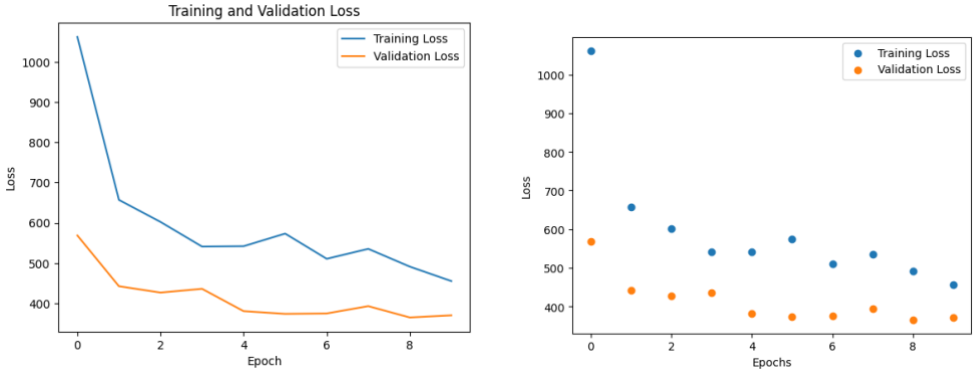
The model's performance exhibits fluctuating trends. During training, loss decrease, while validation loss also show variations but generally decrease. The model's convergence and generalization are relatively consistent.

8.5 ResNet50



The model's performance shows that during training, the loss decreases across epochs. However, the validation loss varies. While they decrease initially, they later start to increase, indicating potential overfitting to the training data and lack of generalization.

8.6 VGG16



During training, the model's loss decrease across epochs, indicating improved performance. Validation loss also decrease, suggesting better generalization to unseen data.

9 Table

MODEL USED	PREDICTED RESULTS	R^2 SCORE	MEAN SQUARED ERROR

1.BASIC CNN	`array([[6362.6826],`
	`[6848.4937],`
	`[6210.105`
	`]], dtype=float32)`
		`-0.4206785701386726`	`965.9676078131005`

2.SHALLOW CNN	`array([[5175.438 ],`
	`[5928.7393],`
	`[5794.306`
	`]], dtype=float32)`
		`0.05356790770979225`	`345.43048543763945`

3.DENSENET121	`array([[`
	`94.306274],`
	`[109.73154`
	`],`
	`[ 92.54378`
	`]], dtype=float32)`
		`0.49196465371662357`	`345.43048543763945`

4.MOBILENET	array([[32.1656], [42.632225], [26.335262]], dtype=float32)	0.3885441439935621	415.7496022849119
5.RESNET50	array([[12.715395], [7.4005923], [18.195929]], dtype=float32)	0.0371623657957415	654.6660066350746
6.VGG16	array([[-86.95285], [2.7101364], [-125.54404]], dtype=float32)	0.4561065448918311	369.8116314126967

```
count=model.predict(X[:3])
count
```

The Predicted Values of no. of berries by the model.

```
print(y[:3])
```

```
[[15 25 35]
```

The Actual Value of the no. of berries.

10 Sample results

```
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Path to the directory containing your images
image_dir = "data/data"  # Assuming your image directory is "data/data"

# List all files in the directory
image_files = os.listdir(image_dir)

# Randomly select 5 image files
selected_images = np.random.choice(image_files, size=5, replace=False)

# Create a subplot for each selected image
plt.figure(figsize=(15, 8))
for i, image_file in enumerate(selected_images, 1):
    image_path = os.path.join(image_dir, image_file)
    image = load_img(image_path, target_size=(224, 224))
    image_array = img_to_array(image)
```

```
    image_array = np.expand_dims(image_array, axis=0)

    # Get the model's prediction (assuming a single output)
    prediction = model.predict(image_array)[0]

    # Choose color based on threshold
    box_color = 'green' if prediction >= 1 else 'red'

    plt.subplot(1, 5, i)
    plt.imshow(image)

    # Add a colored box at the bottom with white text
    plt.gca().add_patch(plt.Rectangle((0, 0), 224, 20, color=box_color,
alpha=0.7))

    # Display the predicted value with white text on top of the box
    plt.text(112, 10, "{:.2f}".format(prediction.item()), color='white',
             fontsize=10, ha='center', va='center')

    plt.xlabel(image_file.split('.')[0], fontsize=10)
    plt.axis("off")

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

Written Code loads images from the "data/data" directory, randomly selects 5 images, and displays them in subplots. For each image, it predicts using a model and chooses green or red based on the prediction threshold. It adds a colored box at the bottom with a predicted value in white text. The chosen color represents prediction confidence, and the text is centered within the box. Subplot titles are set to image filenames, and axes are turned off for cleaner display.

10.1 Basic CNN

10.2 Shallow CNN

10.3 DenseNet

10.4 MobileNet

10.5 ResNet

10.6 VGG16