**Project Report Paper: Water Bottle Image Classifier**

Data Science Course – June 2024

Subject: Data Science

Academic year: 2023-2024

Author: Rigel Hadushi , Klajdi Bajo

Email: rigel\_hadushi@universitetipolis.edu.al

klajdi\_bajo@universitetipolis.edu.al

Issue date: 17/06/2024

**This report contains:**

* Topic Introduction
* Description of the three classifiers in detail
* How overfitting and underfitting have been resolved
* Examples of diagrams showing overfitting
* Comparison between different classifiers
* Conclusions

**Topic Introduction**

In this project, we aimed to build three different types of neural network classifiers that reach a high accuracy level. The types of classifier architectures consist of:

* Only Dense layers.
* Convolutional and Dense layers.
* Pre-trained neural network.

The first step that we had to go through was to build a dataset of our own to classify the images inside. The dataset contains different photos of water brands in three categories:

1. Qafshtama
2. Lajthiza
3. Tepelena

Our dataset consists of a total number of 324 photos. Each water bottle brand consists of three different water bottle sizes varying from 0.5l, 1.5l, and 2l.

After the dataset was done, we were ready to build our three classifiers. The environment where we built our classifiers was Jupyter Notebook powered by Anaconda.

**Simple description of the three classifiers in detail**

The report investigates the effects of overfitting and underfitting on the accuracy of a neural network classifier for image recognition. Three different classifiers were built:

➢ ***Model 1: Dense layers only***

This model uses a stack of fully-connected layers to classify the images. Dense layers are layers that connect every neuron in one layer to every neuron in the next layer.

1. **Input Layer:**

● The model starts with an Input layer. This layer takes the input image, which

has a 3-dimensional shape representing the height, width, and color channels

(RGB).

InputLayer(shape=(IMG\_SIZE, IMG\_SIZE, 3))

● Flatten reshapes the image data into a single 1-dimensional vector.

Flatten()

1. **Hidden Layers:**

● The model then consists of two hidden layers, each containing 254 neurons.

● These are densely-connected layers, meaning all neurons in the previous layer

are connected to every neuron in the current layer.

● Each neuron applies a non-linear activation function (relu in this case) to the

weighted sum of its inputs. The activation function introduces non-linearity,

allowing the model to learn complex patterns in the data.

Dense(256, activation='relu'),

Dense(256, activation='relu')

1. **Output Layer:**

● The final layer has 3 neurons, corresponding to the three water bottle brands

(Qafshtama, Lajthiza, Tepelena).

● This layer uses a softmax activation function. Softmax normalizes the output of

the layer between 0 and 1, representing probabilities for each class. The higher

the output value for a particular neuron, the higher the probability that the image

belongs to that brand.

Dense(num\_classes, activation='softmax', name="result")

1. **Compilation:**

● The model is compiled using the Adam optimizer, a popular optimization algorithm for training neural networks.

● The loss function is sparse categorical cross-entropy, suitable for multi-class

classification with mutually exclusive classes (an image can only belong to one

brand).

● Accuracy is chosen as the metric to monitor the model's performance during

training.

model.compile(optimizer='adam',

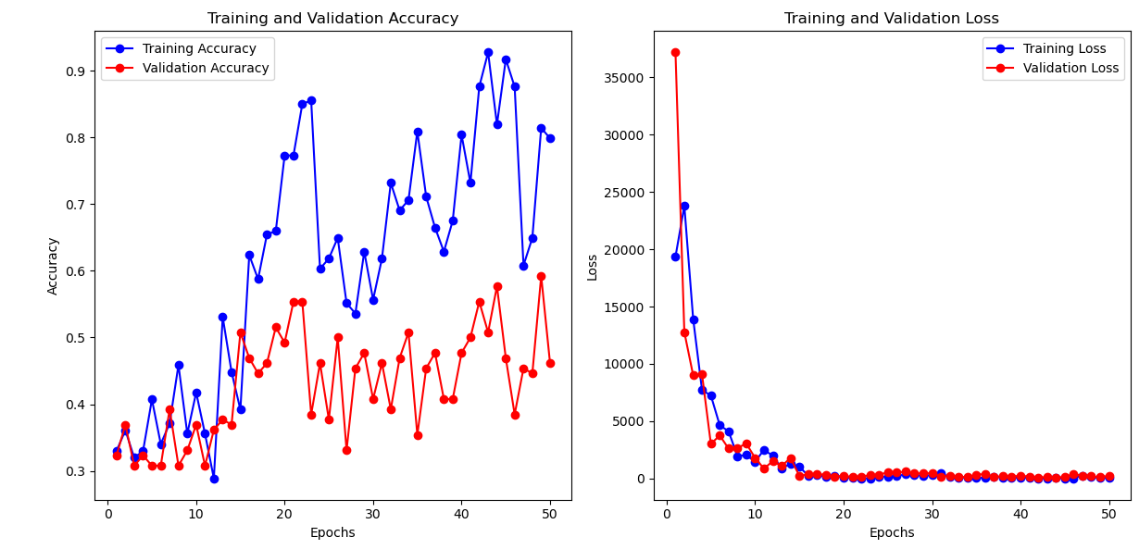
loss='categorical\_crossentropy',

metrics=['accuracy'])

**Summary:**

This model utilizes a series of densely connected layers with ReLU activation to learn features directly from the flattened image data. It employs a softmax output layer to predict the class probabilities for the water bottle brands. Although this architecture can be effective for smaller datasets or simpler classification tasks, it may not perform as well on complex image recognition problems compared to models that use convolutional layers for feature extraction. Unfortunately, because of the overfitting, this model resulted in an accuracy of only 48.46%.

***Diagram of Training and Validation (Accuracy and Loss)***



➢ ***Model 2: Dense and convolutional layers***

This model combines convolutional layers with fully-connected layers. Convolutional layers are layers that apply a filter to an image, extracting features from it. This model is expected to achieve a higher validation accuracy than the first model (around 98.29%).

**CNN Model Architecture Description**

This model implements a Convolutional Neural Network (CNN) architecture for

classifying water bottle images. Here's a detailed breakdown of its layers:

1. **Input Layer:**

● The model takes an image as input with a 3-dimensional shape representing

height, width, and color channels (RGB) - same as the dense layers model.

● Unlike the previous model, the CNN architecture doesn't require flattening the

image data initially. It preserves the spatial information crucial for image

recognition tasks.

InputLayer(shape=(IMG\_SIZE, IMG\_SIZE, 3)),

1. **Applying Data augmentation**

* **Data Augmentation** is used to create new training samples by applying random transformations to the existing images. This helps in increasing the diversity of the training data without actually collecting new data. Techniques used in my model are random zoom, flip, and rotation.

data\_augmentation = Sequential(

[

layers.RandomFlip("horizontal", input\_shape=(img\_height, img\_width, 3)),

layers.RandomRotation(0.1),

layers.RandomZoom(0.1),

]

)

* After defining the data augmentation it's time to include it inside the model after defining the input layer.

data\_augmentation

1. **Normalizing the data**

* The normalizing layer, such as Rescaling(1./255), adjusts the pixel values of input images from the 0-255 range to the 0-1 range. This normalization enhances numerical stability and consistency, improving the performance and training efficiency of neural networks.

Rescaling(1./255),

1. **Convolutional Blocks:**

● The core of the CNN are the **convolutional layers**. The model utilizes three

convolutional blocks, each consisting of a convolutional layer followed by a max

pooling layer.

○ **Convolutional Layer:**

■ This layer applies filters (also called kernels) of size (3x3) in this

case, to the input image. These filters learn patterns and features

from the image data.

■ The number of filters used determines the number of feature maps

generated. Here, the first block uses 16 filters, the second 32, and

the third 64.

■ Each filter learns to detect specific features in the image, like

edges, shapes, or textures. By stacking convolutional layers, the

model can learn increasingly complex features.

○ **Max Pooling Layer:**

■ This layer performs downsampling on the output of the

convolutional layer. It reduces the dimensionality of the data while

capturing the most important features.

■ The pooling operation (max pooling in this case) takes a small

window (2x2 here) and selects the maximum value from that

region. This helps reduce the model's complexity and prevents

overfitting.

Conv2D(16, 3, padding='same', activation='relu'),

MaxPooling2D(),

Conv2D(32, 3, padding='same', activation='relu'),

MaxPooling2D(),

Conv2D(64, 3, padding='same', activation='relu'),

MaxPooling2D()

1. **Dropout and Flatten Layer:**

● Dropout layer (with a rate of 0.2) is included after the pooling operation to prevent overfitting.

● After processing through the convolutional blocks, the data is flattened into a

1-dimensional vector. This allows connecting the extracted features to the

fully connected layers.

Dropout(0.2),

Flatten()

1. **Fully-connected Layers:**

● The model then uses one densely-connected layer, with 128 neurons and

ReLU activation.

● This layer functions similarly to the hidden layers in the dense model, but it

operate on the flattened feature vector instead of the raw image data.

Dense(128, activation='relu')

1. **Output Layer:**

● The final layer has 3 neurons and uses a softmax activation function, identical to the dense layers model.

● It outputs the probabilities for each water bottle brand (Qafshtama, Lajthiza,

Tepelena) based on the learned features.

Dense(num\_classes, activation='softmax', name="outputs")

1. **Compilation:**

● The model is compiled using the Adam optimizer, a popular optimization algorithm for training neural networks.

● The loss function is The sparse categorical cross entropy which calculates the cross-entropy between the true labels and predicted probabilities, suitable for multi-class classification tasks where labels are integers.

● Accuracy is chosen as the metric to monitor the model's performance during

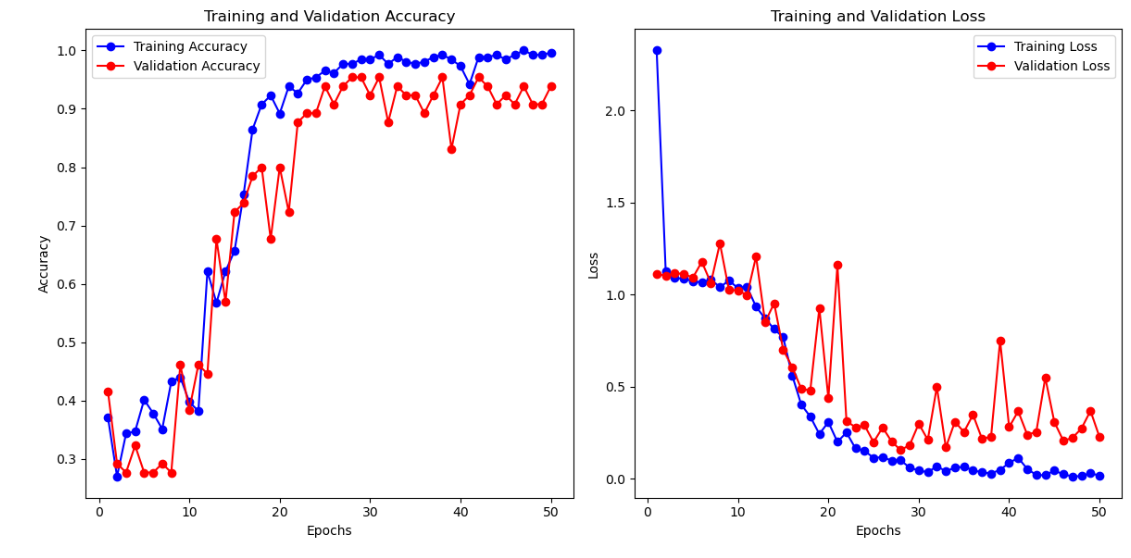
training.

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

**Summary:**

This CNN architecture leverages convolutional layers to automatically extract relevant features from the image data. The max pooling layers help reduce dimensionality and control overfitting. The fully-connected layer then classifies the image based on the extracted features. Compared to the dense layers model, CNNs are generally more efficient in learning image features and achieve better performance on image recognition tasks.

***Diagram of Training and Validation (Accuracy and Loss)***



* ***Model 3: Pre-trained CNN***

This model utilizes a pre-trained convolutional neural network (MobileNetV2) with its top layers removed. A new classifier head is added on top of the pre-trained model for the specific image recognition task. This model is expected to achieve the highest validation accuracy (around 98%) among the three models.

**Transfer Learning Model with MOBILENETV2**

This model utilizes transfer learning with a pre-trained convolutional neural network (CNN) called MobileNetV2 for classifying water bottle images. Here's a breakdown of the architecture:

**1. Pre-trained MOBILENETV2 Model (Base Model):**

● The core of this model is the MOBILENETV2 architecture, pre-trained on the ImageNet dataset containing millions of images and thousands of object categories.

● MobileNetV2 is a powerful CNN with numerous convolutional and pooling layers that have already learned generic image features like edges, shapes, and textures.

● In this implementation, the pre-trained weights are loaded from the 'imagenet'

dataset.

**●** Importantly, the ***include\_top=False*** argument is used to exclude the top

(fully connected) layers of MOBILENETV2. These layers were trained for classifying objects in ImageNet, not specifically for water bottle brands.

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(IMG\_SIZE, IMG\_SIZE, 3))

**2. Freezing the Base Model Layers:**

● All the layers in the pre-trained MOBILENETV2 model are set to be non-trainable

(freezing). This means their weights won't be updated during the training process

for this specific task.

● Freezing the base model layers is crucial for transfer learning. It prevents the

model from potentially forgetting the valuable generic image features learned on

ImageNet while adapting to the water bottle classification problem.

base\_model.trainable = False

**3. New Classification Head:**

● A new section, called the classification head, is added on top of the pre-trained

MobileNetV2 model. This new head consists of the following layers:

○ **GlobalAveragePooling2D Layer**: This layer performs global average pooling over the spatial dimensions of the input tensor (7, 7, 1280). It calculates the average value for each channel across the entire feature map, resulting in a vector of shape (None, 1280). In many transfer learning scenarios, especially with models like MobileNetV2, the global average pooling layer is preferred over flattening because global average pooling reduces the spatial dimensions of each feature map to a single value per channel (1280 in this case), which retains spatial information and encourages the model to focus on the most important features.

GlobalAveragePooling2D()

○ **Dense Layer**: A fully-connected layer with 128 neurons and

ReLU activation is used. This layer learns to classify the water bottle

images based on the features extracted by the pre-trained MOBILENETV2 model.

The dropout layer (with a rate of 0.5) is included for regularization.

Dense(128, activation='relu'),

Dropout(0.5)

○ **Output Layer**: The final layer has 3 neurons and uses a softmax activation

function. It predicts the probability distribution for each water bottle brand

(Qafshtama, Lajthiza, Tepelena) based on the features learned by the new

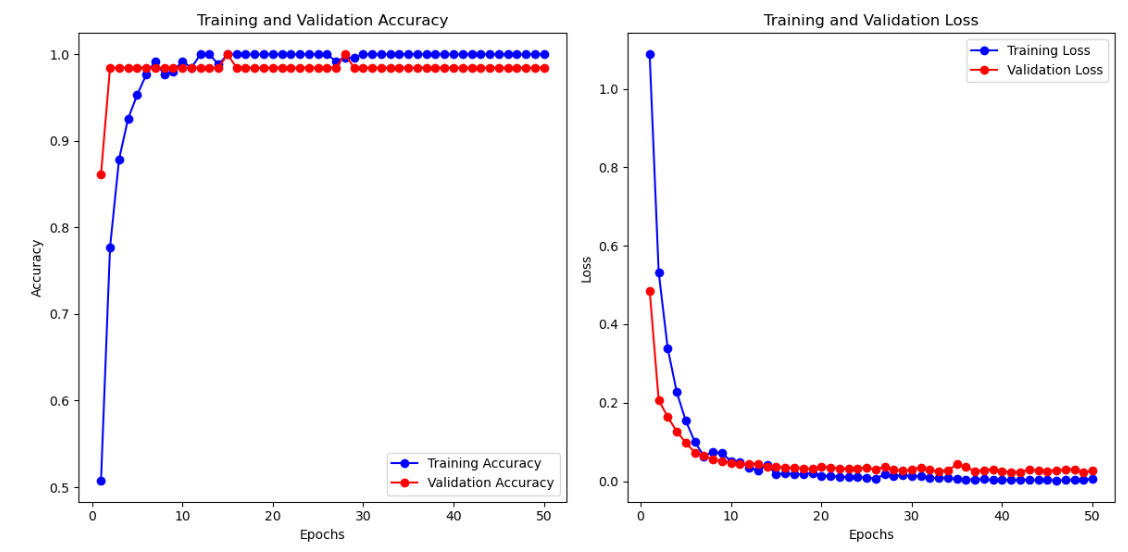
classification head on top of the pre-trained MOBILENETV2 model.

Dense(num\_classes, activation='softmax')

**Summary:**

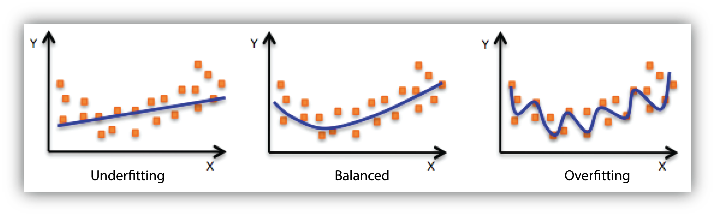
This transfer learning approach leverages the power of a pre-trained MOBILENETV2 model to extract valuable image features. By freezing the base model layers and adding a new classification head, the model can be fine-tuned for the specific task of classifying water bottle brands. This approach is often more efficient than training a CNN from scratch, especially when dealing with limited datasets like the one used in this project. Since the pre-trained MOBILENETV2 has already learned generic image features, the new classification head only needs to focus on the specific task of distinguishing between the three water bottle brands.

***Diagram of Training and Validation (Accuracy and Loss)***



**How overfitting and underfitting have been resolved**

Overfitting and underfitting are critical challenges in machine learning that can significantly impact model performance. Overfitting occurs when a model becomes overly complex and learns to memorize the training data, leading to poor generalization of unseen data. On the other hand, underfitting happens when a model is too simplistic to capture the underlying patterns in the data, resulting in suboptimal performance.



**Techniques Used to Mitigate Overfitting and Underfitting**

1. **Dropout**
   * **Definition**: Dropout layers randomly deactivate a fraction of neurons during training.
   * **Purpose**: This technique prevents the model from relying too heavily on specific features present in the training data, thereby improving its ability to generalize.
   * **Implementation**: Dropout layers are inserted between dense (fully connected) layers in neural networks.
2. **Data Augmentation**
   * **Definition**: Data augmentation involves creating synthetic variations of the training data by applying random transformations such as cropping, flipping, rotation, brightness adjustment, and contrast adjustment.
   * **Purpose**: By exposing the model to diverse examples of the same data points, data augmentation helps in improving model generalization and robustness.
   * **Implementation**: Techniques like random cropping, flipping, and rotation are applied to the training data before feeding it into the model.
3. **Data Normalization**

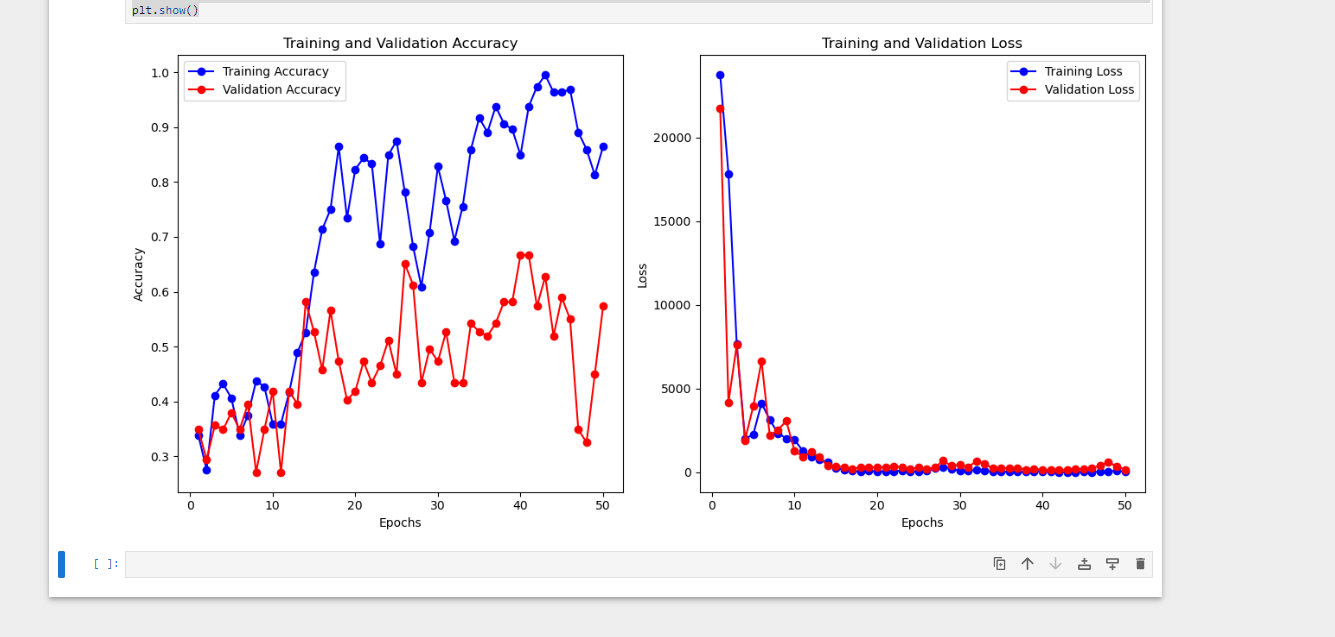
* **Definition**: Data normalization involves scaling the input features to a standardized range, typically between 0 and 1.
* **Purpose**: Standardizing feature scales improves the convergence of optimization algorithms and prevents certain features from dominating the learning process due to their larger scale.
* **Implementation**: Applied to pixel values of images, ensuring all features contribute equally to model training.

These techniques collectively address overfitting and underfitting by promoting model simplicity (through regularization), improving data diversity (via data augmentation), and ensuring stable training conditions (through data normalization).

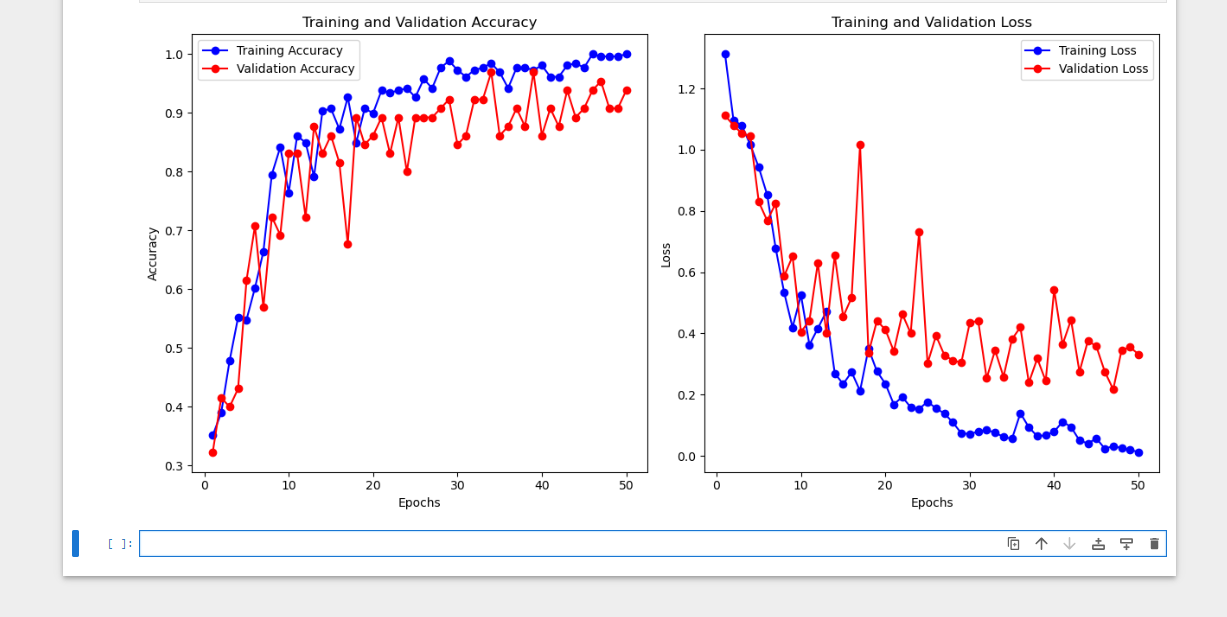
**Examples of diagrams showing overfitting**

The first diagrams are from the only dense layers. It is clearly shown that the accuracy of the training data is way higher than the accuracy of the validation data. This is the reason why the accuracy of this model unfortunately was a bit lower than the expected accuracy.

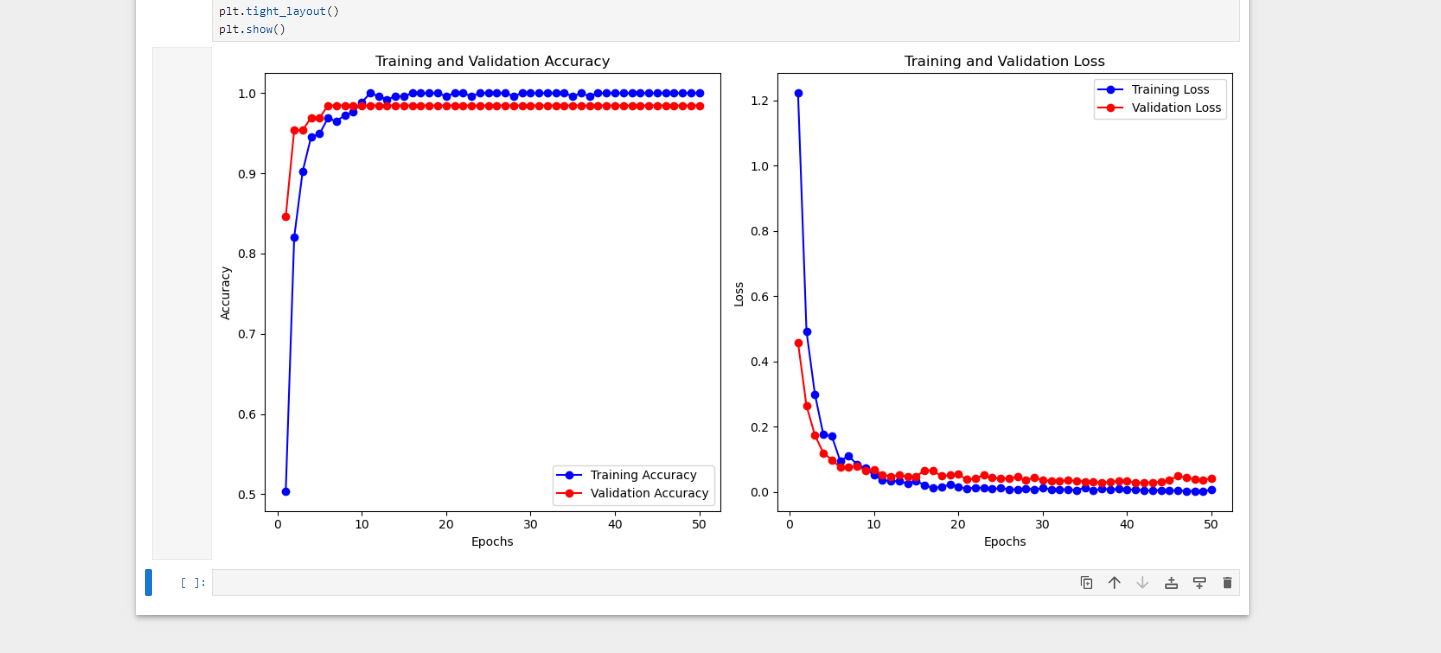
In the second diagram which is about the training and validation loss, if the training loss decreases while the validation loss plateaus or decreases at a slower rate, it suggests the model may be overfitting. The two different lines below in the second graph at some of its parts show exactly this, which can be later translated into an increase in the overfitting and a decrease in the total accuracy of the model.



The third and fourth diagrams depict the performance of models combining convolutional and dense layers, showcasing significant improvements and achieving expected results without overfitting concerns. Both training and validation accuracies demonstrate peak performance, indicating robust generalization across datasets. In the accompanying graph, while the training loss steadily decreases, the validation loss exhibits a slower decline, suggesting a modest disparity in performance between seen and unseen data. However, this gap does not undermine the overall accuracy of the model. By leveraging convolutional layers, the model effectively extracts spatial features crucial for tasks like image recognition, thereby mitigating overfitting and ensuring consistent high performance across diverse datasets.



The last set of diagrams illustrates the performance of our final model, utilizing a pre-trained neural network. The first graph demonstrates the successful resolution of overfitting, showcasing high accuracies for both training and validation datasets. This achievement underscores the model's robustness in generalizing well to unseen data, culminating in a high overall accuracy that meets or exceeds expectations. In the second graph, the validation loss exhibits a decline rate comparable to that of the training loss, indicating effective training without significant overfitting. These observations collectively highlight the model's optimal performance, where the integration of a pre-trained neural network enhances feature extraction capabilities, mitigates overfitting risks, and ensures consistently high accuracy across diverse datasets.



**Comparison between Different Classifiers**

**Validation Accuracy as a Benchmark**

Validation accuracy serves as a crucial metric for assessing a machine learning model's performance on unseen data, using a separate validation set. This metric helps detect overfitting and evaluates the model's ability to generalize to real-world scenarios.

**Expected Performance of Each Model:**

* **Model 1 (Dense layers only):** This baseline model is expected to achieve a validation accuracy ranging from 45% to 65%. Dense layers, while capable of learning basic image properties, are less effective for image recognition tasks compared to convolutional layers. They may struggle to capture complex spatial features crucial for accurate image classification.
* **Model 2 (Dense and Convolutional Layers):** Incorporating convolutional layers designed to extract features from images, Model 2 is anticipated to achieve significantly higher validation accuracy, ranging from 93% to 99%. These layers excel in identifying edges, shapes, and other visual patterns essential for robust image recognition.
* **Model 3 (Transfer Learning):** Leveraging transfer learning with a pre-trained convolutional neural network (MobileNetV2), Model 3 utilizes learned feature representations from a large dataset like ImageNet. Fine-tuning this pre-trained model on the water bottle image dataset is expected to yield high performance, with a validation accuracy range of 91% to 99%.

**Justification for Higher Accuracy with Complex Models:**

* **Feature Extraction Capability:** Convolutional layers in Model 2 and the pre-trained MobileNetV2 model in Model 3 are specialized for extracting intricate features critical for image recognition. This capability enables the models to discern complex relationships between pixels, thereby enhancing classification accuracy.
* **Reduced Training Burden:** Model 3 benefits from transfer learning, where pre-trained weights from MobileNetV2 provide a robust foundation for feature extraction. This approach minimizes the need for extensive training solely on the water bottle dataset, reducing computational resources and time compared to training from scratch.

**Key Takeaway:** Comparing these models underscores that for image recognition tasks, leveraging convolutional layers or transfer learning typically leads to superior performance compared to models relying solely on dense layers. Convolutional layers excel in feature extraction from images, while transfer learning optimizes model efficiency and accuracy by leveraging pre-existing knowledge from large-scale datasets like ImageNet.

**Conclusions**

In this project, we explored different ways to classify images of water bottle brands using TensorFlow. We built three types of models: one with only dense layers, another combining dense and convolutional layers, and a third using a pre-trained model (MobileNetV2).

To tackle problems like overfitting (when the model learns too much from the training data) and underfitting (when it doesn't learn enough), we used techniques such as dropout, L2 regularization, and data normalization. These methods helped our models avoid focusing too much on specific details in the training images.

When we evaluated the models based on their accuracy with new data, we found that the model with only dense layers performed the worst. This model struggled to understand complex features in the images compared to the others. In contrast, models with convolutional layers or pre-trained networks performed much better because they were better at recognizing important visual patterns in the images.

Throughout the project, overcoming overfitting was a big challenge, especially for the model with only dense layers. Techniques like data augmentation (creating variations of our existing data) and dropout (ignoring some neurons during training) played a crucial role in improving these models' performance.

Overall, this project showed us that using convolutional layers or pre-trained models is essential for accurately classifying images like water bottle brands. These approaches capture crucial details in images, leading to better classification results. Looking ahead, refining these techniques further could make these models even more effective for real-world applications.

**References for this project and report:**

* **https://www.tensorflow.org/tutorials/images/cnn**
* **https://www.tensorflow.org/tutorials/images/classification**
* **https://www.tensorflow.org/tutorials/images/data\_augmentation**
* **https://developers.google.com/machine-learning/practica/image-classificati**
* **on/preventing-overfitting**
* **https://medium.com/@akshaykr.sharma19966/steps-to-deal-with-overfitting**
* **-and-underfitting-on-image-data-using-image-augmentation-87848ab06351**
* [**https://www.geeksforgeeks.org/vgg-16-cnn-model/**](https://www.geeksforgeeks.org/vgg-16-cnn-model/)

**Data Science Book and Lecture Notes provided by Dr. Prof. Luca Lezzerini**