**Fish Finder**

**TFS and Operation**

|  |  |  |
| --- | --- | --- |
| Document Version |  | Document Date |
| 1.0 |  | 10/06/2024 |
|  |  |  |

SUMMARY

[1 HISTORY OF THE DOCUMENT 3](#_Toc169123979)

[2 INTRODUCTION 4](#_Toc169123980)

[2.1 Definitions, abbreviations, acronyms 4](#_Toc169123981)

[2.1.1 Definitions 4](#_Toc169123982)

[2.1.2 Acronyms 4](#_Toc169123983)

[2.2 Reference documents 4](#_Toc169123984)

[3 GENERAL INFORMATION 5](#_Toc169123985)

[4 Project Setup 5](#_Toc169123986)

[4.1 Technologies used 5](#_Toc169123987)

[4.1.1 Language 5](#_Toc169123988)

[4.1.2 Libraries 5](#_Toc169123989)

[4.1.3 Development Environment 5](#_Toc169123990)

[5 Components 5](#_Toc169123991)

[5.1 Image datasets 5](#_Toc169123992)

[5.2 Image Transformation 5](#_Toc169123993)

[5.3 Convolutional Neural Network (CNN) 6](#_Toc169123994)

[5.4 Model Training 6](#_Toc169123995)

[5.5 Validation & Testing 6](#_Toc169123996)

[5.6 Inference 6](#_Toc169123997)

[6 ResNet18 6](#_Toc169123998)

[6.1 Deep and simple architecture 6](#_Toc169123999)

[6.2 Performance 6](#_Toc169124000)

[6.3 Computational Efficiency 7](#_Toc169124001)

[6.4 Flexibility 7](#_Toc169124002)

[6.5 Activation function 7](#_Toc169124003)

[6.5.1 Features of ReLu 7](#_Toc169124004)

[7 Pseudo-Labeling 8](#_Toc169124005)

[7.1 Pretraining 8](#_Toc169124006)

[7.2 Pseudo-Label Generation 8](#_Toc169124007)

[7.3 Combine data 8](#_Toc169124008)

[7.4 Re-training 8](#_Toc169124009)

[8 Training & Results 8](#_Toc169124010)

[8.1 First Training (labeled images) 8](#_Toc169124011)

[8.1.1 Performance Training (labeled images) 9](#_Toc169124012)

[8.1.2 Performance Validation (Labeled Images) 10](#_Toc169124013)

[8.1.3 Conclusions (Labeled images) 10](#_Toc169124014)

[8.2 Second Training (all images) 10](#_Toc169124015)

[8.2.1 Performance Training (all images) 11](#_Toc169124016)

[8.2.2 Performance Validation (all images) 11](#_Toc169124017)

[8.3 Final Conclusions 12](#_Toc169124018)

[9 Test 12](#_Toc169124019)

[10 POINTS OF ATTENTION 14](#_Toc169124020)

[11 ATTACHMENTS 14](#_Toc169124021)

# HISTORY OF THE DOCUMENT

The information in this document updates and replaces information in previous versions. Any changes to the document will be checked in the appropriate procedure.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description of the Revision** | **Modified by** |
| 1.0 | 10/06 /2024 | First version | Carlo Blatti |

# INTRODUCTION

## Definitions, abbreviations, acronyms

### Definitions

### Acronyms

* CNN: Convolutional Neural Network
* ReLU: Rectified Linear Unit
* TFS: Technical and Functional Specifications

## Reference documents

# GENERAL INFORMATION

The developed project is an **image classifier** based on a convolutional neural network (CNN). This type of model is used to assign a default label (class) to each image in the dataset, learning to recognize the distinctive visual features that define each class. Specifically, the images on which the model has been trained are images of fish.

## Project Links

<https://colab.research.google.com/drive/12yOHtln1mLHBLye5u6tS6Nia5esqnxMP?usp=sharing>

# Project Setup

The application project was developed using Colab as a development environment and runtime.

## Used Technologies

### Language

* **Python**: The primary programming language used to implement the image classifier.

### Libraries

* **Bookshop**:
  + **PyTorch**: Core library for deep learning and neural network building.
  + **Torchvision**: Used to manage datasets of pre-trained images and models.
  + **Pandas**: For the manipulation and analysis of tabular data (CSV files).
  + **Matplotlib**: For visualizing data, such as accuracy and loss curves.
  + **scikit-learn**: For splitting data into training and validation sets.

### Development Environment

* **Google Colab**: Cloud-based notebook environment that offers free access to GPUs, used to run Python code and accelerate model training.

# Components

## Image datasets

* The images are provided in a CSV file that contains the file names and their respective classes.
* The images are divided into a training set and a validation set.

## Image Transformation

Images are preprocessed through transformations that include scaling, cropping, and normalizing to improve the robustness and performance of the model.

## Convolutional Neural Network (CNN)

* **ResNet18**: A pre-trained convolutional neural network model, known for its ability to learn deep representations of images.
* **Fine-tuning**: Modifying the last level of the model to fit the specific number of classes in the dataset.

## Model Training

* Using optimization techniques such as the Adam algorithm to update model weights.
* Mixed Precision Training to speed up training and reduce memory consumption.

## Validation & Testing

* Monitor model performance on a validation set to avoid over-training.
* Visualization of loss and accuracy curves to evaluate the behavior of the model during training.

## Inference

* Testing the model on validation images to verify the accuracy of the predictions.
* Visualize images with model predictions and compare them with the correct labels.

# ResNet18

ResNet18 was chosen for this image classification project for several reasons that make it a good fit.

## Deep and simple architecture

* **Residual Networks**: ResNet stands for "Residual Network". ResNet's architecture introduces residual connections that help mitigate the problem of vanishing gradient in deep networks, allowing for more effective training of deep networks.
* **Moderate Depth**: ResNet18 is a relatively lightweight version with 18 layers. It is less complex than the deeper versions, but still offers enough learning capability for many image classification tasks.

## Performance

* **Pre-training on ImageNet**: ResNet18 is usually pre-trained on ImageNet, a very large and varied dataset. This pre-training allows the model to start with well-defined characteristics, speeding up the fine-tuning process and improving performance even on smaller or more specific datasets.
* **Accuracy**: ResNet18 has been shown to perform well on a wide range of computer vision tasks, offering a good balance between accuracy and model complexity.

## Computational Efficiency

* **Speed**: Compared to more complex and deeper models, ResNet18 is faster at both training and inference, which is especially important when using Google Colab, where computational resources can be limited.
* **Memory Efficiency**: ResNet18 requires less memory than deeper versions of ResNet, making it a practical choice for resource-constrained environments.

## Flexibility

* **Ease of Adaptation**: ResNet18 can be easily modified to fit different output classes. In the project, the last layer of the model was replaced to fit the specific number of classes in the dataset, making the model versatile for various classification tasks.
* **Optimization Techniques Compatibility**: Easily supports optimization techniques such as mixed precision training (torch.cuda.amp), which further improves training efficiency.

## Activation function

The activation function used in the ResNet18 model is **ReLU.** This activation feature is built into network blocks and is critical to its training capabilities, especially in deep networks, due to its simplicity, computational efficiency, and ability to mitigate the vanishing gradient problem.

### Features of ReLu

* **Simplicity**: The ReLU function is very simple to calculate, being linear for all positive values and zero for negative values.
* **Sparsity**: ReLU introduces sparsity into activations, as many of the negative values are mapped to zero. This can improve computational efficiency.
* **Mitigation of the Vanishing Gradient Problem**: ReLU helps mitigate the vanishing gradient problem, which can be prevalent in deep networks, by keeping the gradients meaningful for positive activations.

# Pseudo-Labeling

In addition to the labeled images, whose name and class are contained in the training csv, we wanted to use a set of unlabeled images.

Including unlabeled images through pseudo-labeling allows you to use more data, potentially improving the performance of your model. This approach leverages the model's ability to generalize and learn from new information, leading to improved accuracy and robustness in image recognition.

## Pretraining

The model was initially trained using only labeled data. This provides a solid foundation of learned characteristics.

## Pseudo-Label Generation

The unlabeled images were passed through the trained model to generate predicted labels.

## Combine data

The original labeled data and the pseudo-labeled data were combined to create a larger dataset.

## Re-training

The model was re-trained using this combined dataset, potentially improving its performance due to the increased amount of data available.

# Training & Results

The training lasted 1m and 32s for the dataset containing the labeled images and 35m and 17s for the complete dataset of all the images.

## First Training (labeled images)

At the end of the 10 epochs of the first training, the following results were obtained:

Epoch 0/9

----------

train Loss: 2.5358 Acc: 0.2250

val Loss: 2.3280 Acc: 0.2000

Epoch 1/9

----------

train Loss: 0.9321 Acc: 0.7500

val Loss: 1.9768 Acc: 0.3333

Epoch 2/9

----------

train Loss: 0.5385 Acc: 0.8750

val Loss: 2.2904 Acc: 0.3000

Epoch 3/9

----------

train Loss: 0.3172 Acc: 0.9167

val Loss: 2.8322 Acc: 0.3667

Epoch 4/9

----------

train Loss: 0.3579 Acc: 0.9167

val Loss: 2.4085 Acc: 0.4667

Epoch 5/9

----------

train Loss: 0.2903 Acc: 0.9167

val Loss: 1.3135 Acc: 0.7667

Epoch 6/9

----------

train Loss: 0.2692 Acc: 0.9083

val Loss: 3.0307 Acc: 0.5667

Epoch 7/9

----------

train Loss: 0.3656 Acc: 0.9083

val Loss: 7.3186 Acc: 0.2667

Epoch 8/9

----------

train Loss: 0.2201 Acc: 0.9500

val Loss: 8.5100 Acc: 0.2000

Epoch 9/9

----------

train Loss: 0.2954 Acc: 0.9083

val Loss: 8.5215 Acc: 0.3000

Training complete in 1m 32s

Best val Acc: 0.7667

Image Containing Text, Diagram, Diagram, Line

Auto-generated description

### Performance Training (labeled images)

* The Loss on the training data decreases rapidly from 2.5358 to values around 0.2201-0.3656.
* The accuracy (Acc) on training data rapidly increases from 0.2250 to values between 0.9083 and 0.9500.

### Performance Validation (Labeled Images)

* The loss on the validation data shows a variable behavior, initially decreasing but then increasing significantly, reaching very high values such as 7.3186 and 8.5100.
* The accuracy on the validation data starts at 0.2000 and increases to 0.7667 and then drops dramatically, back to 0.3000.

### Conclusions (Labeled images)

* The loss on the validation data shows a variable behavior, initially decreasing but then increasing significantly, reaching very high values such as 7.3186 and 8.5100.
* The accuracy on the validation data starts at 0.2000 and increases to 0.7667 and then drops dramatically, back to 0.3000.
* There is an initial improvement in performance on the validation data to an accuracy of 0.7667 at the fifth epoch. However, this improvement is not sustained in later eras.

The model shows a clear sign of overfitting, with very good performance on training data but poor ability to generalize to validation data.

## Second Training (all images)

At the end of the 10 epochs of the second training, the following results were obtained:

Epoch 0/9

----------

train Loss: 1.2140 Acc: 0.5841

val Loss: 1.8214 Acc: 0.4333

Epoch 1/9

----------

train Loss: 1.0478 Acc: 0.6344

val Loss: 2.1608 Acc: 0.3667

Epoch 2/9

----------

train Loss: 0.9879 Acc: 0.6522

val Loss: 2.3595 Acc: 0.3000

Epoch 3/9

----------

train Loss: 0.9590 Acc: 0.6637

val Loss: 2.3528 Acc: 0.4667

Epoch 4/9

----------

train Loss: 0.9293 Acc: 0.6710

val Loss: 1.8443 Acc: 0.3667

Epoch 5/9

----------

train Loss: 0.9000 Acc: 0.6782

val Loss: 1.4038 Acc: 0.4667

Epoch 6/9

----------

train Loss: 0.8803 Acc: 0.6874

val Loss: 1.6182 Acc: 0.5333

Epoch 7/9

----------

train Loss: 0.8700 Acc: 0.6899

val Loss: 1.5088 Acc: 0.5000

Epoch 8/9

----------

train Loss: 0.8458 Acc: 0.7010

val Loss: 1.5987 Acc: 0.5333

Epoch 9/9

----------

train Loss: 0.8254 Acc: 0.7074

val Loss: 1.2500 Acc: 0.6333

Training complete in 35m 17s

Best val Acc: 0.6333

Image Containing Diagram, Diagram, Line, Text

Auto-generated description

Validation Loss: 1.2501 Acc: 0.6333

### Performance Training (all images)

* The loss on training data steadily decreases from 1.2140 to 0.8254.
* The accuracy on the training data steadily increases from 0.5841 to 0.7074.

### Performance Validation (all images)

* The loss on the validation data shows fluctuating behavior initially, but with a general downward trend from 1.8214 to 1.2500.
* The accuracy on the validation data varies, with a significant improvement in the last epoch, reaching a final value of 0.6333.

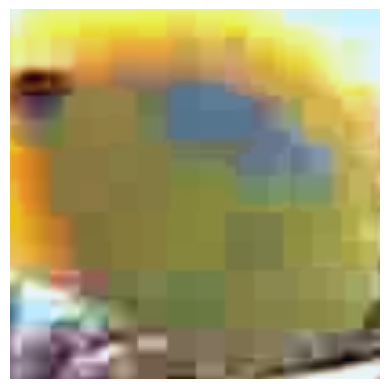
## Final Conclusions

* Initially, there seems to be a tendency towards overfitting. In the early epochs, the loss on validation data increases while the loss on training data decreases, indicating that the model is storing training data rather than generalizing.
* However, in recent epochs, there is an improvement in both loss and accuracy on validation data, suggesting that the model may have begun to generalize better.

The model shows improvement, but there are initial signs of overfitting that are partially mitigated in later eras.

# Test

Following training, the model was tested using the validation dataset, showing that it returned the correct class 65% of the time.

Here are some of the test results:  
  


Predicted: 1, Correct: 1

Image containing aquarium, Organism, Marine biology, underwater

Auto-generated description

Image That Contains Text, Screenshot

Auto-generated description

# POINTS OF ATTENTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | Description | Owner | Decisions | State |
|  |  |  |  |  |
|  |  |  |  |  |

# ATTACHMENTS

|  |  |  |
| --- | --- | --- |
| **ID** | Attachment | Description |
|  |  |  |
|  |  |  |
|  |  |  |