Project Report for Neural Computing

Sports Image Classification

Author Lombardia Giuseppe
Neural Computing course
Prof. Sebastiano Battiato

MSc in Data Science for management

Contents

1 Introduction

Image Classification constitutes a pivotal task within the realms of computer vision and image processing. Its primary objective is to assign a categorical label to a given input image. The typical procedure entails training a model on an extensive data-set, which is then leveraged to forecast the classification of novel, previously unseen images.

Numerous techniques and algorithms exist for image classification, yet the prevailing approach involves employing Convolutional Neural Networks (CNNs). CNNs are a class of deep learning models thoughtfully engineered for processing data structured in a grid-like topology, such as images. They consist of multiple layers, each of which is designed to extract progressively intricate features from the input image. In CNNs, the concluding layers are often fully connected, serving the purpose of making the ultimate class prediction.

The overarching objective of this project is to utilize a data-set comprising images to accurately classify playing cards, harnessing the power of a CNN. The data-set can be easily reached by clicking here.

Figure 1: various types of sports

2 Background and Data

The objective of this image classification task will be tackled using Matlab, employing the specialized ***Deep Learning Toolbox***. This toolbox provides the capabilities to construct deep neural networks from the ground up or utilize pre-existing ones.

To be more specific, we will train a Convolutional Neural Network (CNN) on both the training and validation sets, subsequently making predictions on the test set. In accordance with the project's requirements, I have chosen to initiate the training process with two distinct pre-trained models: ***GoogLeNet*** and ***ResNet-18***. This approach aims to explore disparities in terms of time efficiency and classification accuracy.

The data-set, is already divided in train set, validation set and test set. The images are in different sizes and in jpg format.

The images had already been set to the same dimensions, **224x224**, but during the initial training phase, an error occurred, which I promptly resolved by manipulating images using ***'augmentedImageDatastore'***. In short, an ***'imagesize'*** of 224x224x3 was set for all the images. Subsequently, by using ***'ColorPreprocessing'*** and specifying a type of image processing set to ***'gray2rgb,'*** the original grayscale images were transformed into RGB images (with 3 channels).

As previously mentioned, the data-set is organized into 100 classes. As described earlier, we already have the 'train,' 'valid,' and 'test' folders. The following will be presented graphically to provide a clearer and more detailed understanding (Figure 2).

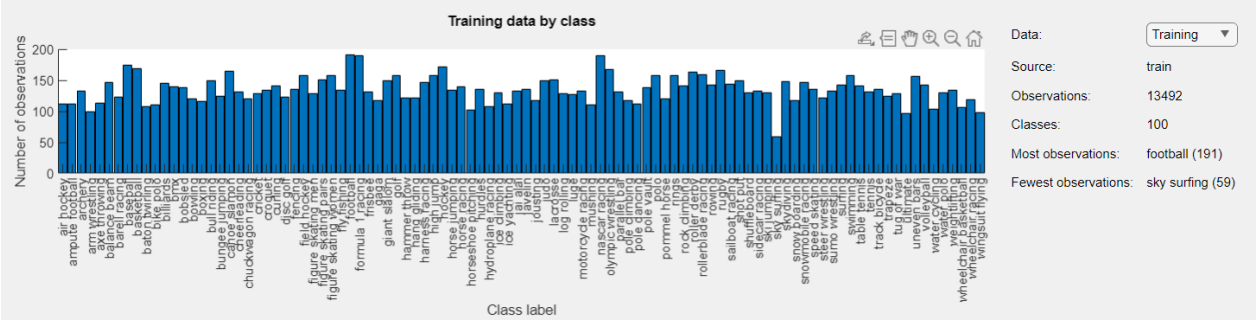

Figure 2: Train data observations

The folder containing the training images consists of 13,492 photos divided, as previously mentioned, into 100 classes. The most numerous class is ***"football"***, with nearly 200 images. The class with fewer observations, on the other hand, is ***"sky surfing"***. If we refer to validation, however, we have 500 images divided into 100 classes, 5 images per class (Figure 3-4).

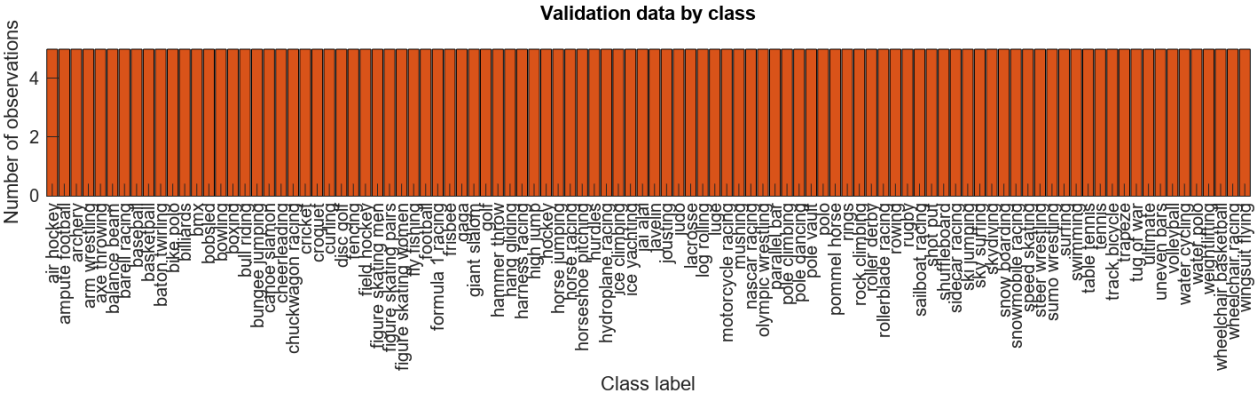

Figure 3: Validation data observations

3 Problem designing

The problem I am addressing is a classification task, as mentioned earlier. I will be working within the ***MATLAB*** environment. The objective is to classify images containing various sports into their respective sports categories. To achieve this goal, I will employ a transfer learning technique. This entails commencing with a pre-trained model from the selection available in ***MATLAB*** and subsequently fine-tuning the neural network by retraining it with a new output layer while preserving the pre-trained model's weights. It's important to note that all the neural networks used in this context have been previously trained on the ***ImageNet*** data-set. I have decided to use two pre-trained networks, GoogLeNet and ResNet-18.

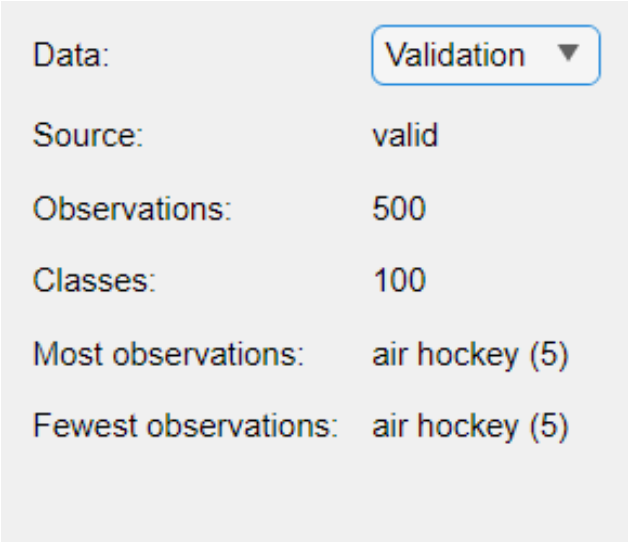

Figure 4: Validation data observations

4 GoogLeNet

GoogLeNet, also known as Inception-v1, is a deep convolutional neural network (CNN) architecture developed by Google. It gained prominence when it won the 2014 Image-Net Large Scale Visual Recognition Challenge with a significantly lower error rate compared to previous models. GoogLeNet introduced several innovative architectural features:

1. **Inception Modules**: One of the most notable features of GoogLeNet is its use of "Inception modules." These modules incorporate multiple convolutional and pooling layers within a single module. This allows the network to capture a wide range of features at different scales simultaneously, enhancing its ability to recognize objects in various sizes and orientations.

2. **Factorization**: GoogLeNet employs a "factorization" strategy to reduce the number of parameters and computational cost. This is achieved by using 1x1 convolutions to reduce the depth of the network before applying larger convolutions. Factorization helps maintain computational efficiency while preserving model capacity.

3. **Auxiliary Classifiers**: Inception modules include auxiliary classifiers at intermediate layers of the network. These classifiers are used for training purposes and help the network to better understand the intermediate representations of images. This approach aids in mitigating the vanishing gradient problem during training and improves the overall accuracy.

4. **Global Average Pooling**: Instead of fully connected layers at the end, GoogLeNet uses global average pooling to reduce over fitting and the number of parameters in the model. This is a form of spatial average pooling where each feature map is averaged independently, resulting in a compact representation.

5. **Deep Network**: GoogLeNet is a deep network with 22 layers, and it efficiently captures hierarchical features from low-level edges to high-level object parts and categories.

The combination of these architectural elements makes GoogLeNet an effective model for image classification tasks. It showcases the advantages of parallel and multi-scale feature extraction, which allows it to excel in recognizing complex patterns and objects within images.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 5: GoogLeNet

4.1 Training Options

Within a Convolutional Neural Network (CNN), the training settings encompass critical components like the optimization method, batch size, learning rate, and more. The optimizer is instrumental in adjusting the model's weight values during the training process, with options that include stochastic gradient descent (SGD) or Adam, among others.

The batch size dictates the quantity of data samples processed in a single forward and backward pass. It holds influence over both the model's performance and computational efficiency. Additional factors, such as the learning rate, the duration of training (number of epochs), and regularization techniques, are also pivotal in the successful training of a CNN.

Now let's focus on my training options of my ***GoogLeNet***.

1. **sgdm**: This step sets the optimizer to Stochastic Gradient Descent with Momentum (**SGDM**). **SGDM** is a popular optimization algorithm used for updating the model's weights during training. It incorporates momentum to accelerate convergence and navigate through local minima.

2. **ExecutionEnvironment**, **auto**: The execution environment is configured to "auto." This means that MATLAB will automatically determine whether to utilize the CPU or GPU for training based on the available hardware resources.

3. **InitialLearnRate**, **0.001**: The initial learning rate is set to 0.001. The learning rate controls the size of the steps the model takes during weight updates. A lower value like 0.001 indicates smaller steps, which can help convergence but may be slower.

4. **MaxEpochs**, **20**: The training process is limited to a maximum of 20 epochs. An epoch represents one complete pass through the entire training dataset. In this case, training will stop after 20 such passes.

5. **Shuffle**, **every-epoch**: Data shuffling is configured to occur "every-epoch." This means that at the start of each epoch (full pass through the data-set), the training data is shuffled. Shuffling helps prevent the model from learning patterns related to the order of the data.

```
opts = trainingOptions("sgdm",...
    "ExecutionEnvironment","auto",...
    "InitialLearnRate",0.001,...
    "MaxEpochs",20,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",30,...
    "Plots","training-progress",...
    "ValidationData",dsValidation);
```

Figure 6: Training options GoogLeNet

6. **ValidationFrequency**, **30**: Validation is performed every 30 mini-batches. A mini-batch is a subset of the training data. By validating every 30 mini-batches, you monitor the model's performance more frequently.

7. **Plots**, **training-progress**: This setting enables the generation of a training progress plot. The plot displays the training loss and accuracy over time (epochs). It provides a visual representation of the model's learning progress.

8. **ValidationData**, **dsValidation**: The validation data source is set to **'dsValidation'**. This data-set is used for evaluating the model's performance during training. It helps ensure that the model generalizes well to data it hasn't seen during training.

These steps collectively define the training options for my neural network. They control various aspects of the training process, such as the optimizer, learning rate, training duration, data shuffling, validation frequency, and progress visualization. These settings are vital for configuring the training process to achieve the desired model performance.

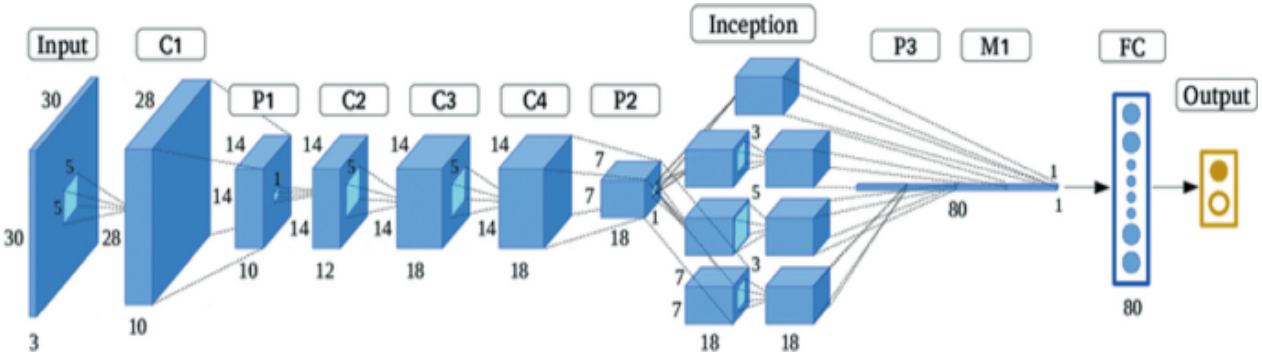

Figure 7: GoogLeNet configuration

4.2 Final results using GoogLeNet

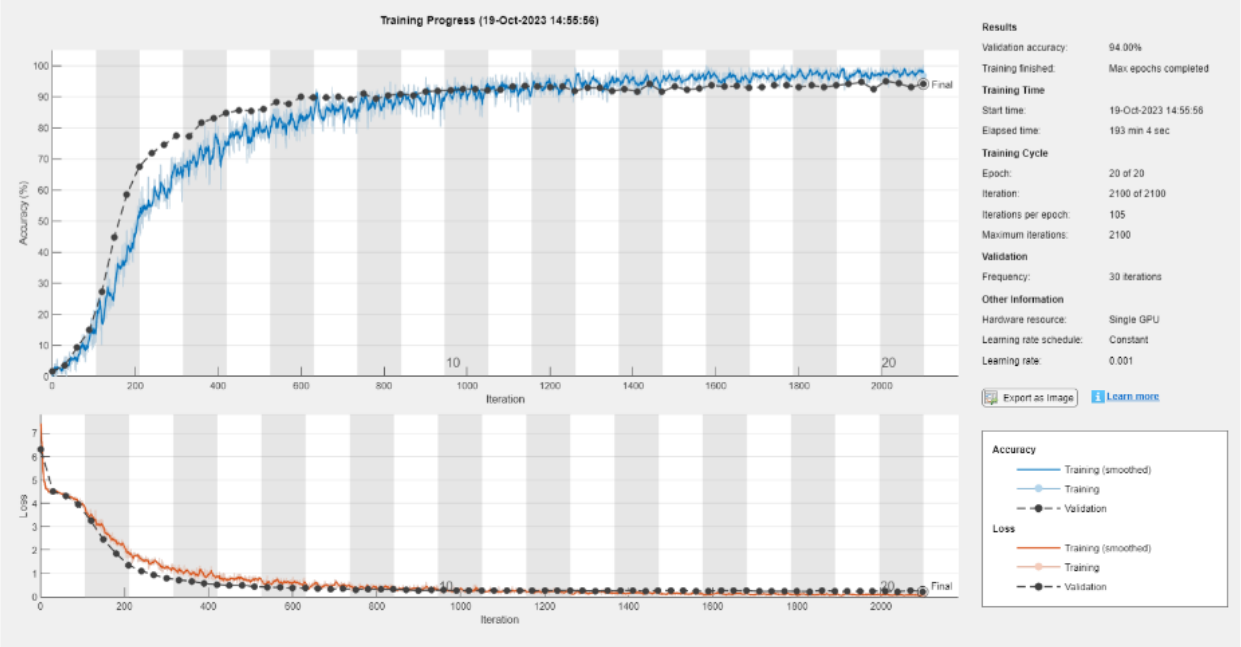

Figure 8: Training GoogLeNet

In figure 8 is possible to observe that the final accuracy in the validation set is around 94% achieved in 3 hours. Also in this case I will provide you the confusion matrix for both validation and test set. We start with the confusion matrix for validation. A confusion matrix with 100 classes is used to evaluate the performance of a classification model when there are 100 different classes or categories to predict. This is common in multiclass classification problems where the goal is to assign each instance to one of 100 possible classes.

Here are some key aspects to consider when dealing with a confusion matrix for 100 classes:

1. **Matrix Size**: A confusion matrix for 100 classes will be a 100x100 matrix, with rows and columns corresponding to the actual and predicted classes, respectively.

2. **Diagonal Elements (True Positives)**: The elements on the main diagonal of the matrix represent the true positives for each class. These are instances correctly classified within their respective classes.

3. **Off-Diagonal Elements**: The elements off the main diagonal represent misclassifications. For example, if an instance from class A is predicted to belong to class B, it will contribute to the element at the intersection of row A and column B.

4. **Analysis**: To assess the model's performance, you can examine the diagonal elements and compute various metrics for each class. Common metrics include precision, recall, and F1-score for each class.

5. **Overall Metrics**: You can calculate overall performance metrics for the entire classification task, such as micro-averaging and macro-averaging. Micro-averaging aggregates the counts across all classes to compute a single metric, while macro-averaging computes metrics for each class and then averages them.

6. **Visual Representation**: Representing a 100x100 confusion matrix can be challenging due to its size. Heatmaps are often used to visualize the matrix, with color intensity indicating the number of misclassifications.

7. **Handling Imbalanced Data**: In real-world applications, some classes may have more instances than others (class imbalance). When working with imbalanced data, it's important to consider metrics that account for this, such as class-specific metrics or the use of balanced accuracy.

A confusion matrix for 100 classes is a powerful tool for evaluating the strengths and weaknesses of a classification model with a high number of categories. It provides a detailed breakdown of the model's performance and helps identify which classes may require further attention or model improvement.

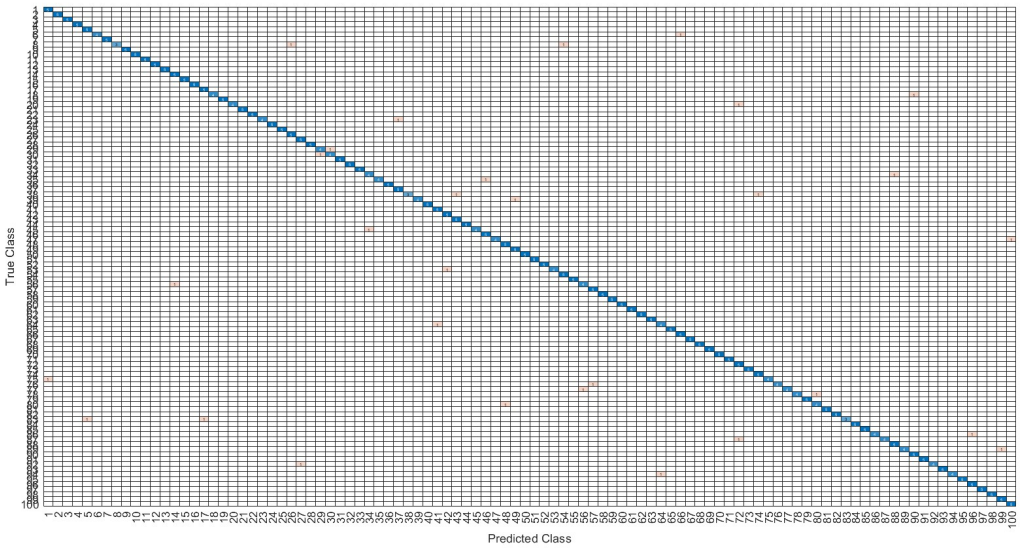

Figure 9: Confusion Matrix Validation

By analyzing the Figure 9 is possible to see that only 30 images have been misclassified on a total of 500. Now let's look at the confusion matrix for the test and the accurary test.

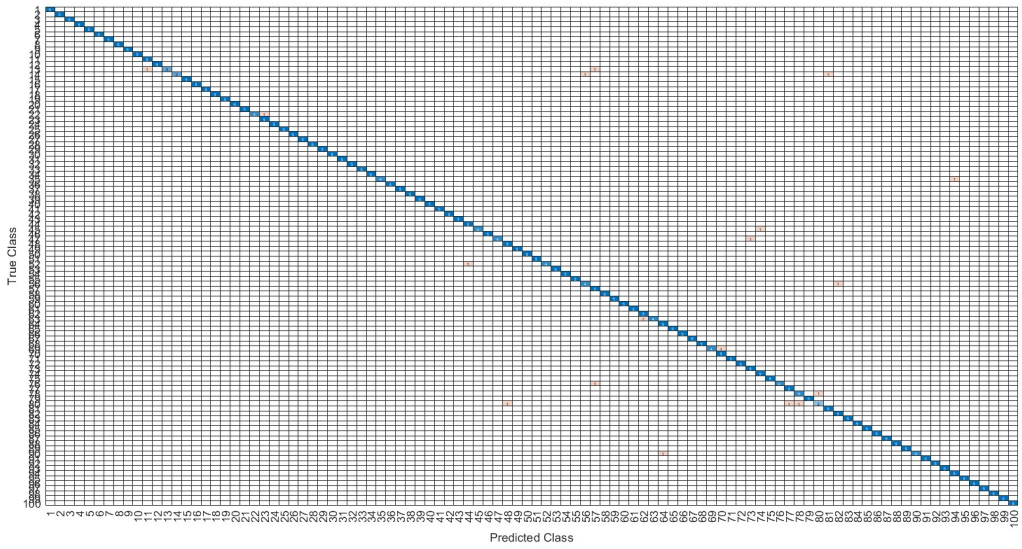

Figure 10: Confusion Matrix Test

By analyzing the Figure 10 is possible to see that only 18 images have been misclassified on a total of 500. The accuracy test is 0.964.

```
accuracy_test
```

```
accuracy_test = 0.9640
```

Figure 11: Accuracy test

Some misclassified images by GoogLeNet are (Figure12):

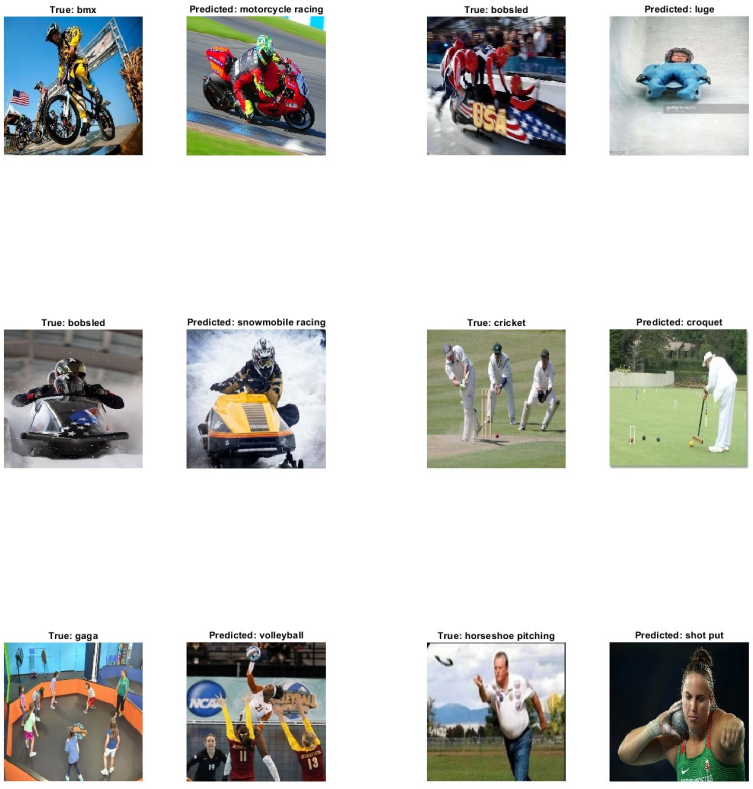

Figure 12: Some misclassified images

4.3 Applications

After focusing on the misclassifications made by GoogLeNet, it is interesting to see if the network can recognize other images. I decided to take two images and properly transform them to 224x224 so I could submit them and the result was more than satisfactory.

Figure 13: image to be submitted

Starting from the first image (Figure 13). In Figure 14, the submitted image was classified correctly.

Figure 14: Classification

Figure 15: Second image to be submitted

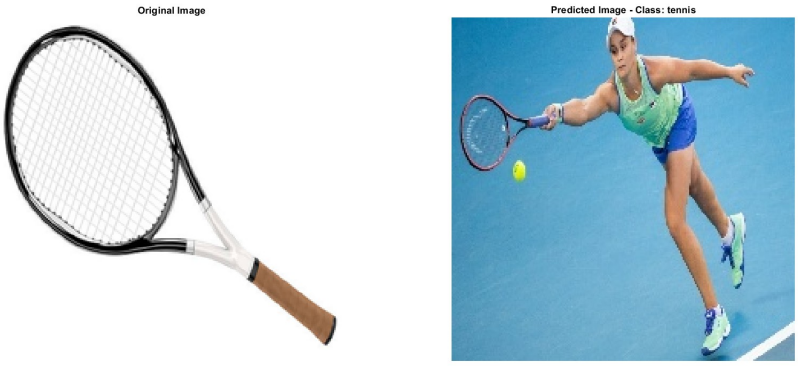

Figure 16: Classification

Regarding the second image submitted, the network was able to recognize the sport by giving a correct classification (figure 16).

5 ResNet-18

ResNet-18 is a convolutional neural network (CNN) architecture that belongs to the family of ResNet (Residual Network) models. It was introduced in 2015 by Kaiming He et al. as part of the paper titled "Deep Residual Learning for Image Recognition." **ResNet-18** is known for its relatively shallow depth compared to some other ResNet models, but it is still highly effective and is used in various computer vision applications. Here are the key details of **ResNet-18**:

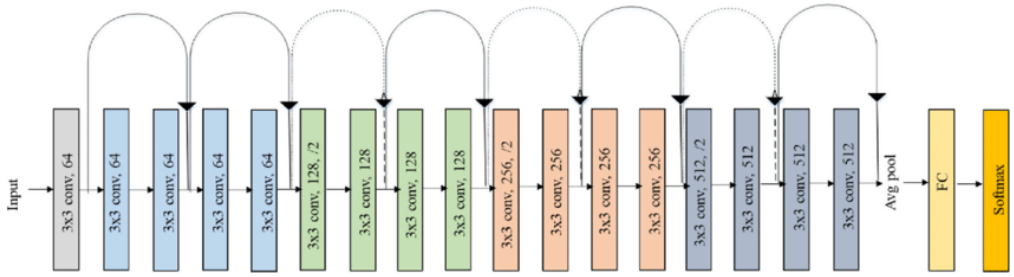

Figure 17: ResNet-18

Residual Architecture: The hallmark of ResNet models, including **ResNet-18**, is the use of residual blocks. These blocks contain shortcut connections that allow the flow of information to "skip" one or more layers. This helps mitigate the vanishing gradient problem in deep neural networks and enables the training of deeper networks with better performance.

Shallow Depth: **ResNet-18** has a relatively shallow depth compared to other ResNet variants like ResNet-50 or ResNet-101. It consists of 18 layers, including 16 convolutional layers and 2 fully connected layers. This reduced depth makes it lighter in terms of parameters compared to deeper variants, making it suitable for applications where model complexity needs to be constrained.

Convolution and Batch Normalization: **ResNet-18** uses 3x3 and 1x1 convolutions along with batch normalization layers to extract features from the input. These convolutions are followed by Rectified Linear Unit (ReLU) activation functions.

Pooling and Classification: The model includes pooling layers, such as max-pooling, to progressively reduce the size of feature maps. Ultimately, one or more fully connected layers are present for classification.

Performance: **ResNet-18** is known for its excellent performance in image classification and other computer vision tasks. It has been trained on large data-sets like ImageNet and has demonstrated top-tier results.

ResNet-18 is a popular choice for transfer learning applications, where you can use a pre-trained model on widely used data-sets and then fine-tune it for a specific task or data-set. Its relatively shallow depth makes it suitable for projects with limited computational resources or real-time applications.

5.1 ResNet-18 in details

ResNet-18 as showed in figure 18 is composed by 18 layers grouped in 6 main groups with about 11 million parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Figure 18: ResNet-18 architecture

5.1.1 First group

In the first group **Input Layer** take in input an RGB image of 224x224 pixels. **Conv1** is a 2D convolutional layer with :

- kernel size 7 x 7
- stride 2 x 2
- padding 3 x 3

Batch Normalization

Normalize data Before ReLU

ReLU activation function

Introduce non linearity

Max Pooling

Reduce size feature maps

5.1.2 Second, Third, Fourth, Fifth Group

These groups follow the same pattern, each consisting of two branches (Figure 18):

1. 1. One branch contains the shortcut connection, which is a direct path between layers.
2. 2. The other branch includes two convolutional layers, two batch normalization layers, and a ReLU activation function.

The settings for stride, kernel size, and padding remain the same for each group. Furthermore, the configuration described above is repeated two times for each group. The only variation between the groups is the number of filters in the convolutional layers:

- In the Second Group, there are 64 filters.
- In the Third Group, there are 128 filters.

- In the Fourth Group, there are 256 filters.
- In the Fifth Group, there are 512 filters.

This architecture is designed to capture features at different levels of abstraction as information flows through the various groups. Each group contributes to creating increasingly complex representations of the images.

5.1.3 Sixth Group

In this section, there is no repetition. Instead, there is a cleansing phase that consists of a ReLU function and a MaxPooling function. The output of this phase is then fed into a Fully Connected layer, which, in the original scenario, had an output for 1000 categories. However, I modified it to suit my task, changing the output to accommodate 100 categories. Following that, there is a softmax layer and a classification layer.

6 Final Results using ResNet-18

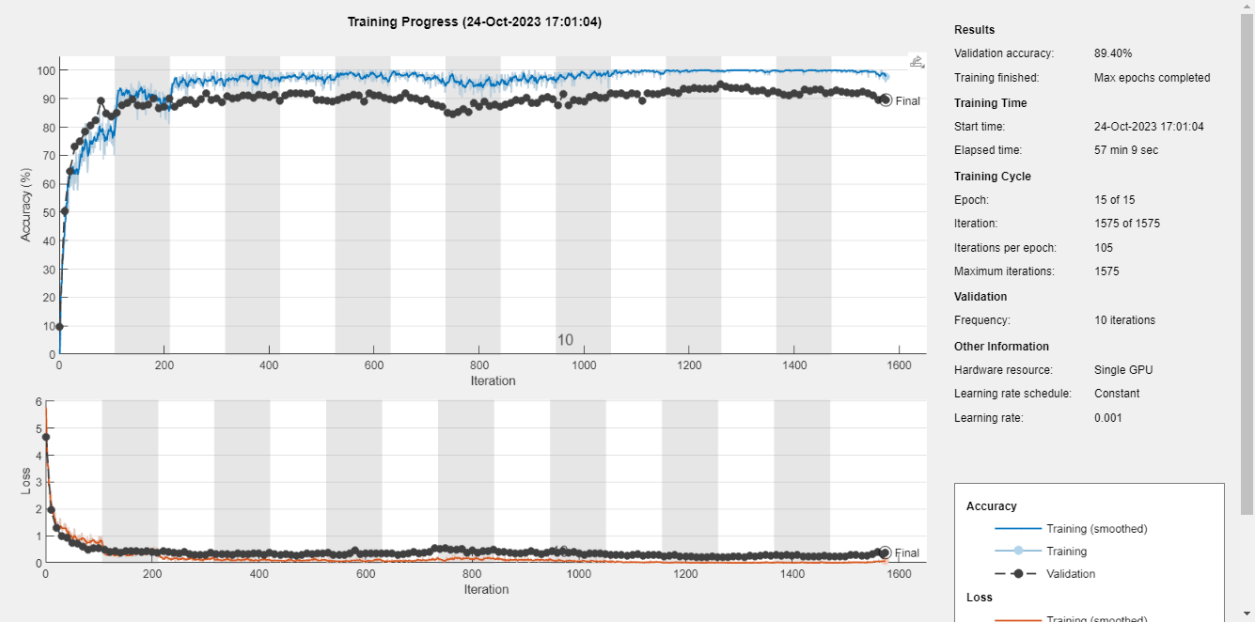

Figure 19: Training using ResNet-18

```
opts = trainingOptions("adam",...
    "ExecutionEnvironment","auto",...
    "InitialLearnRate",0.001,...
    "MaxEpochs",15,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",10,...
    "Plots","training-progress",...
    "ValidationData",dsValidation);
```

Figure 20: Training options

In figure 19 is possible to observe that the final accuracy in the validation set is around 89,4% achieved in 1 hour hours. In figure 20 is possible to see how the various configuration topics were set up for training a model, in this case, the ResNet-18. I have previously discussed all the various configurations. The only difference is the use of Adam as opposed to SGD. Adam is a widely used optimization algorithm for updating model weights during training. It is known for its efficiency and adaptability to changes in gradients. Also in this case I will provide you the confusion matrix for both validation and test set. We start with the confusion matrix for validation and then matrix for test.

6.1 Confusion Matrix

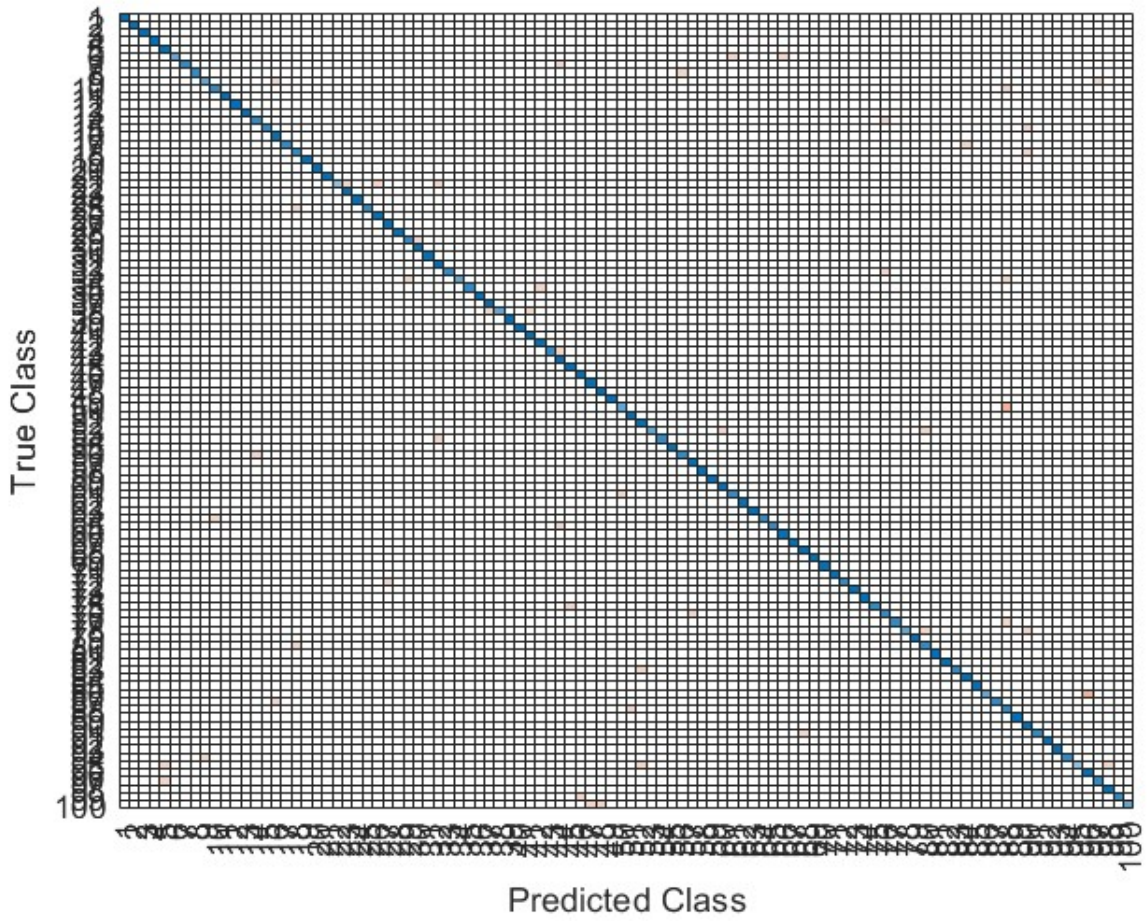

Figure 21: Confusion Matrix of validation set

Analyzing the confusion matrix of the validation set (Figure 21) we can see that misclassifications are greater than in the previous model, about 53 out of 500 images were misclassified. Looking at Figure 22 there is the confusion matrix of the test set where we can see that 37 images were misclassified. The test accuracy is 0.926.

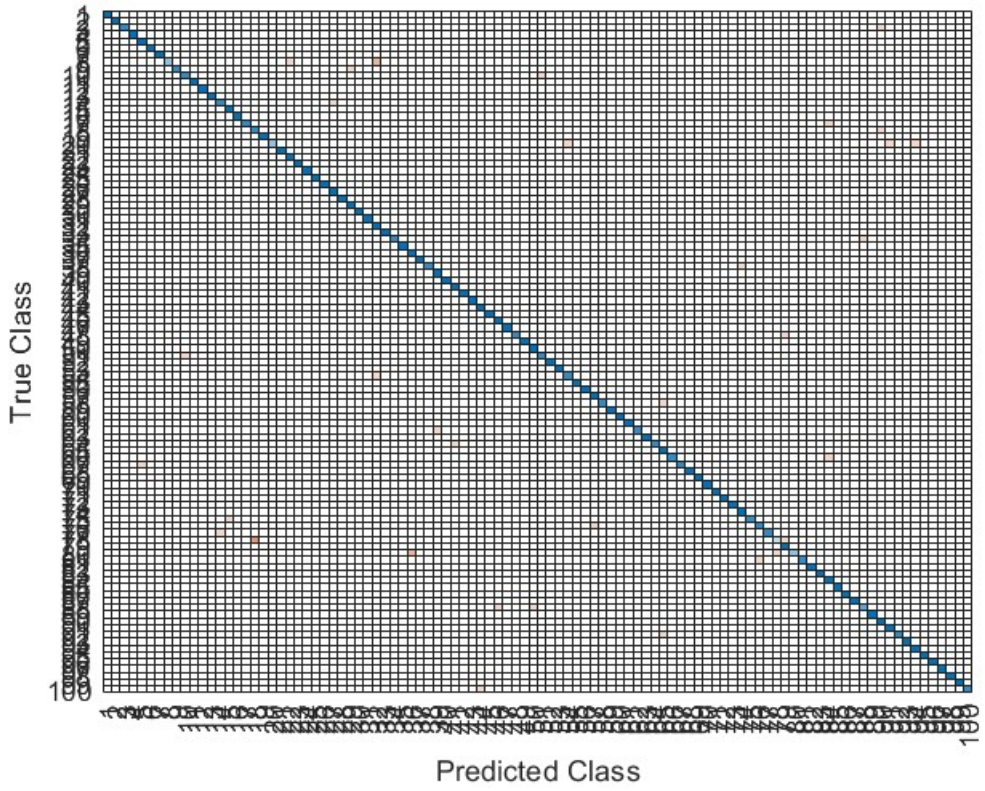

Figure 22: Confusion matrix of test set

6.2 Some tests to submit to the network

I submitted the same images, submitted earlier in GoogLeNet. The first image (Figure 13) is predicted the class correctly (Figure 23), however, for the image (Figure 15) there is an incorrect classification (Figure 24).

Figure 23: Classification

Figure 24: Misclassification

Figure 25: two tests

7 Appendix

Later I will paste the codes used on Matlab by network. The same code was used for both networks, first on GoogLeNet and after ResNet-18

```
    imagesize = [224,224,3];
train = imageDatastore("train/","IncludeSubfolders",true,"LabelSource","foldernames");
valid = imageDatastore("valid/","IncludeSubfolders",true,"LabelSource","foldernames");
test = imageDatastore("test/","IncludeSubfolders",true,"LabelSource","foldernames");
resizedTrain = augmentedImageDatastore(imagesize,train,"ColorPreprocessing","gray2rgb");
resizedValid = augmentedImageDatastore(imagesize,valid,"ColorPreprocessing","gray2rgb");
resizedTest = augmentedImageDatastore(imagesize,test,"ColorPreprocessing","gray2rgb");

pred_valid_labels = classify(net, resizedValid);
true_valid_labels = valid.Labels;
accuracy_valid = mean(true_valid_labels == pred_valid_labels);
D = confusionmat(true_valid_labels, pred_valid_labels);
confusionchart(D)
totalMisclassifications = sum(D, 'all') - trace(D);
totalMisclassifications
accuracy_valid

pred_test_labels = classify(net, resizedTest);
true_test_labels = test.Labels;
accuracy_test = mean(true_test_labels == pred_test_labels);
C = confusionmat(true_test_labels, pred_test_labels);
confusionchart(C)
accuracy_test
totalMisclassifications1 = sum(C, 'all') - trace(C);
totalMisclassifications1

immagine = imread('Image1.jpg');
```

```
imshow("Image1.jpg");
dimension_image=[224,224];
imgResized=imresize(immagine, dimension_image);
imshow(imgResized);
[predictedLabel, score] = classify(net, imgResized);
subplot(1,2,1);
imshow(imgResized);
title("Original Image");
predictedImage = readimage(test,find(pred_test_labels == predictedLabel,1));
subplot(1,2,2);
imshow(predictedImage);
title(sprintf('Predicted Image - Class: %s', predictedLabel));



immagine2 = imread('tennis.jpg');
imshow("tennis.jpg");
dimension_image=[224,224];
imgResized2=imresize(immagine2, dimension_image);
imshow(imgResized2);
[predictedLabel, score] = classify(net, imgResized2);
subplot(1,2,1);
imshow(imgResized2);
title("Original Image");
predictedImage2 = readimage(test,find(pred_test_labels == predictedLabel,1));
subplot(1,2,2);
imshow(predictedImage2);
title(sprintf('Predicted Image - Class: %s', predictedLabel));

misclass_indices = find(true_test_labels~= pred_test_labels);
for i = 1:numel(misclass_indices)
    subplot(1,2,1)
    imshow(readimage(test,misclass_indices(i)));
    true_label = true_test_labels(misclass_indices(i));
    title(sprintf("True: %s",true_label));

    pred_label= pred_test_labels(misclass_indices(i));
    pred_idx = find(pred_test_labels == pred_label);
    subplot(1,2,2)
    imshow(readimage(test,pred_idx(2)));
    title(sprintf("Predicted: %s", pred_label));

    input("Press Enter to continue...");
end
```