UNIVERSITÀ degli STUDI di CATANIA | DIPARTIMENTO di ECONOMIA e IMPRESA

Subject: Neural Computing

Year: 2021/2022

Prof. Sebastiano Battiato

Students: Gianluigi Mazzaglia Davide Saitta

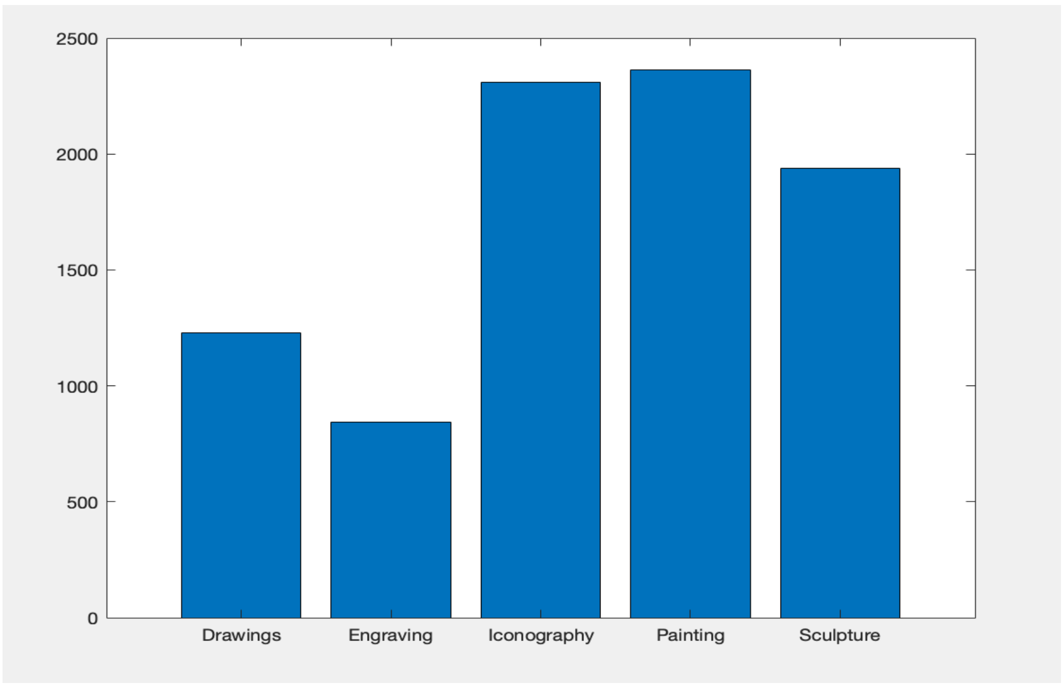
Introduction

Art is a highly diverse range of human activities engaged in creating visual, auditory, or performed artifacts artworks that express the author's imaginative or technical skill and are intended to be appreciated for their beauty or emotional power.
Here we will devote ourselves to the classification of different forms of visual art.

Data Exploration

Our dataset is made up of 8685 images representing different forms of visual art.
the classes are:

- **Drawing,** with 1229 images
- **Engraving,** with 844 images
- **Iconography,** with 2310 images
- **Painting,** with 2364 images
- **Sculpture,** with 1938 images



As we can see from the bar-plot above, the data is unbalanced; specifically, we have the class Engraving which represents the minority class with only the 9.72% of the total number of observations, followed by Drawings with 14.15%, Sculpture with 22.31%, Iconography with 26.60% and Painting which represent the majority class with 27.22% of the total number of observations.

Data Preparation

In this initial phase, while exploring the dataset, we noticed the presence of anomalous images, and of images with different depths.
Since these images are marginal in number, we have decided to eliminate them. specifically, we have removed both, the anomalous images, and the images with depth = 1, therefore in grayscale, in order to keep only RGB images.
In order to do that, we used the following code:

```
rgb = 0;
tot = 0;
grey = 0;
yourfolder = 'sculpture';
theFiles = dir([yourfolder '/*g']);

for k = 1 : length(theFiles)

    baseFileName = theFiles(k).name;
    fullFileName = fullfile(theFiles(k).folder, baseFileName);

    imageArray = imread(fullFileName);
tot = tot + 1;
[rows, columns, numberOfColorChannels] = size(imageArray);
if numberOfColorChannels < 2
fprintf(1, 'Now deleting %s\n', fullFileName);
delete(fullFileName);
grey = grey + 1;

else
rgb = rgb + 1;

end
end
```

First think first, we have built tree counters. Then we give to the dir function the path of our images, so we can create the loop. For each image, in the folder, the function adjusts the right counter and if the number of colour channels is equal to one delete the image.
We did it for all the classes, the resulting dataset is composed of 8536 images specifically:

- **Drawings:** 1229, the only class without variations.

- **Engraving:** 806, from this class we removed 38 between anomalous and grayscale images.

- **Iconography:** 2306, from this class we removed 4 images.

- **Painting:** 2270, from this class we removed 94 images between anomalous and grayscale images.
- **Sculpture:** 1925, from this class we removed 13 images.

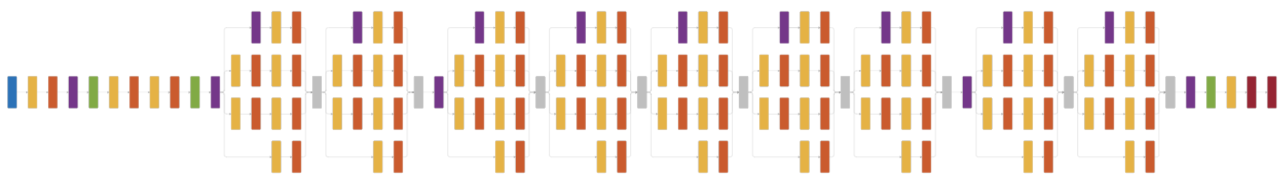
In total we removed 149 images, 40 of which were grayscale and then we split the dataset into training, validation and test set (80-10-10).

Model Designing

As this project has the goal to build an image classification task, we have designed different architectures in order to achieve it.

One of the architectures used is **GoogLeNet.**

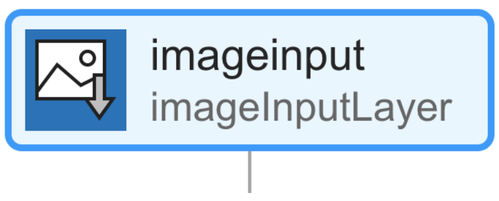
GoogLeNet, is 22 layers deep, (27 if we count the pooling layers).



The main layers that compose it are the following:

Input Layer

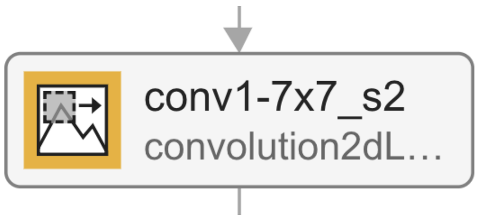
Is from the input-layer that the data come inside the network.



Specifically, using this layer we set the input size. In our case 227,227, 3, that corresponds respectively to width, height, and depth.

Convolutional Layer

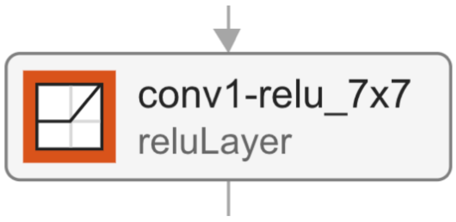
Convolutional layers perform convolutions, which are operations where a filter is moved over an input image, calculating the values in a resulting feature map.



A convolutional layer is usually built up of multiple filters, which will produce multiple feature maps. During training of the CNN, the model will learn what weights to apply to the different feature maps and, hence, be able to recognize which features to extract from the input images.

Activation Function

An activation function is an operation that takes a floating-point number as input and returns a new floating-point number as output.

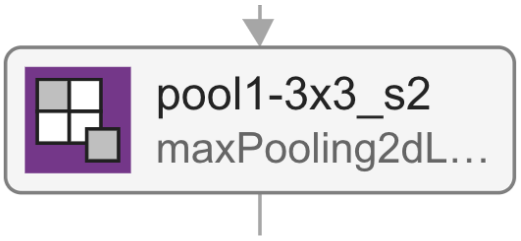


You're applying an activation function onto your feature maps to increase non-linearity in the network.

The activation function used is a piecewise linear function called rectified linear unit, which is abbreviated ReLU.

MaxPooling Layer

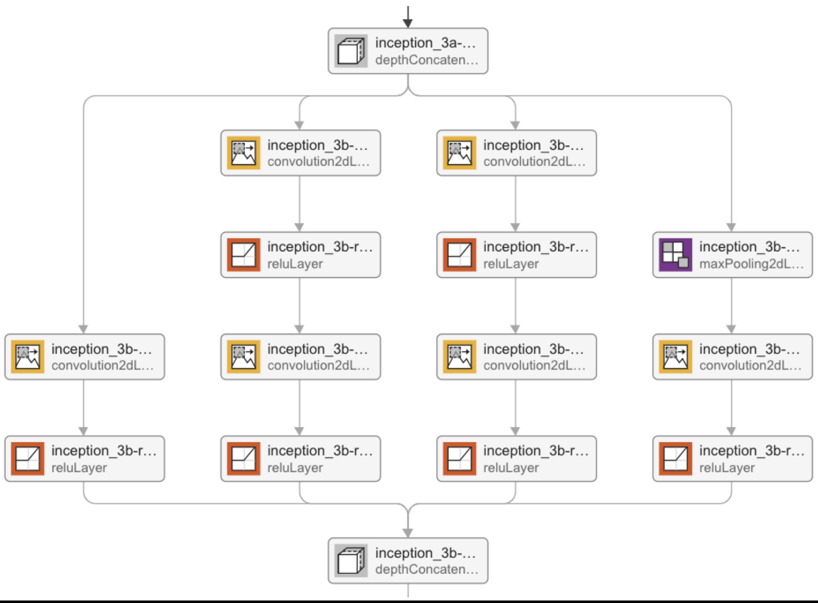
A pooling layer is a layer that reduces the computational cost of the model and helps fight overfitting by reducing the dimensionality of its input.



Specifically, MaxPooling select the largest value in the matrix

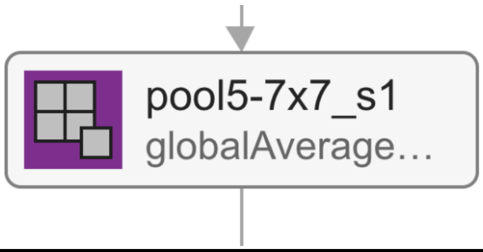
Inception Layer

Salient parts in the image can have extremely large variation in size. Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough.


inception_3a-… depthConcaten…
inception_3b-… convolution2dL…
inception_3b-… convolution2dL…
inception_3b-r… reluLayer
inception_3b-r… reluLayer
inception_3b-… maxPooling2dL…
inception_3b-… convolution2dL…
inception_3b-… convolution2dL…
inception_3b-… convolution2dL…
inception_3b-… convolution2dL…
inception_3b-r… reluLayer
inception_3b-r… reluLayer
inception_3b-r… reluLayer
inception_3b-r… reluLayer
inception_3b-… depthConcaten…

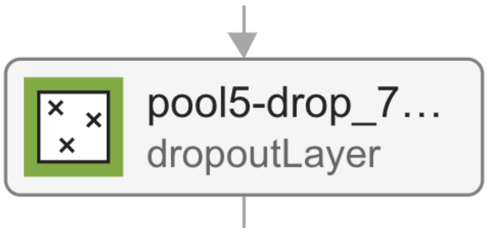
Inception layer is composed by filters with multiple sizes operate on the same level. The network essentially would get a bit "wider" rather than "deeper".
It performs convolution on an input, with 3 different sizes of filters (1x1, 3x3, 5x5). Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.

Global average Layer



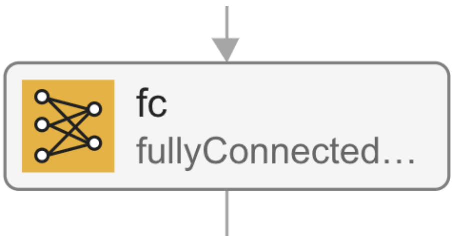
Dropout Layer

A dropout layer involves turning off nodes randomly with a probability *p* during training. Such layers are especially helpful to fight overfitting in models with a lot of complexity.



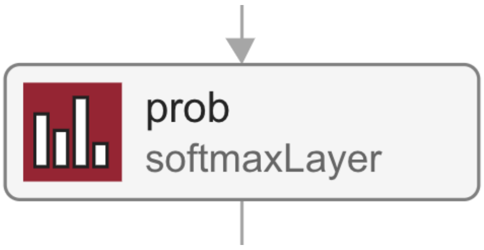
FullyConnected Layer

A fully connected layer transforms its input to the desired output format. In a classification task, this typically includes converting a matrix of image features into a *1xC* vector where *C* is the number of classes.



Softmax Layer

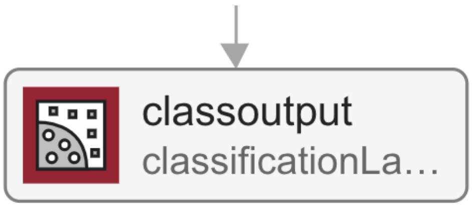
Turns the raw numbers that come out of the network into class probabilities. Each output neuron presents a value, or score, that corresponds to how much the network thinks that particular input belongs to the corresponding category. The category with the highest score is the one that the network considers its top choice.



A softmax layer is most commonly applied after the fully connected layers. This layer takes a vector of size *1xC* as an input, where *C* is the number of classes, and all of the numbers add up to 1.

Classification Layer

It is the last layer of the network. Usually, the layer infers the number of classes from the output size of the previous layer.

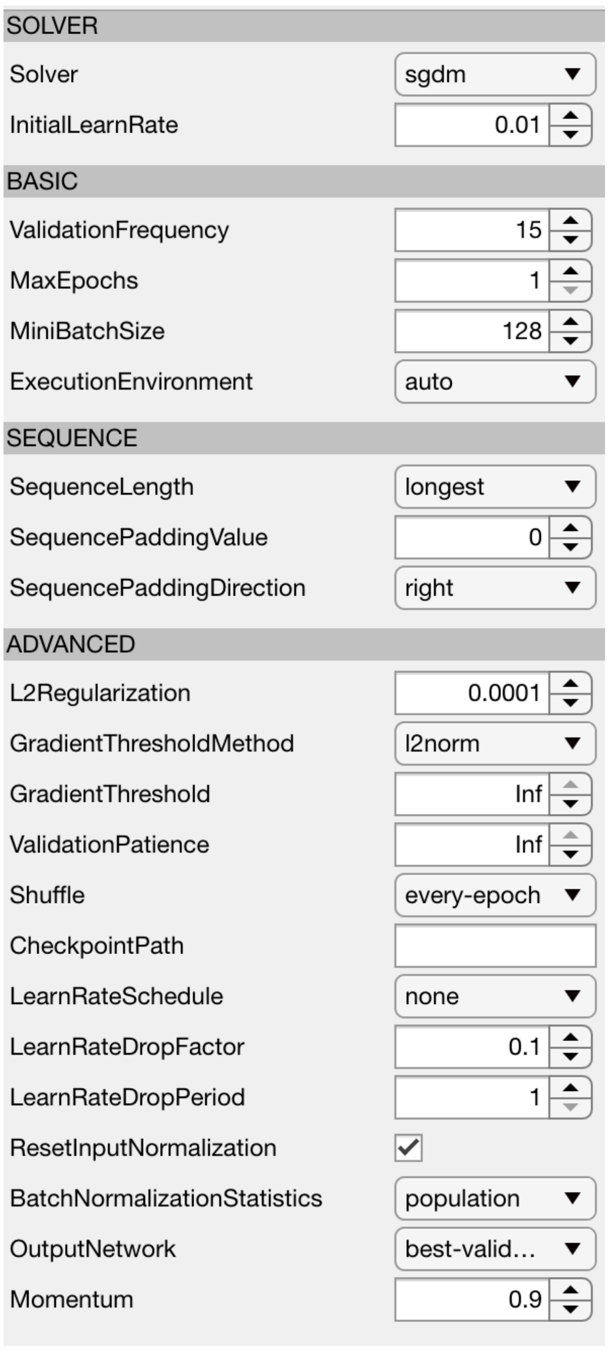


To specify the number of classes *K* of the network, we include a fully connected layer with output size *K* and a softmax layer before the classification layer.

Hyperparameters Optimization and Training

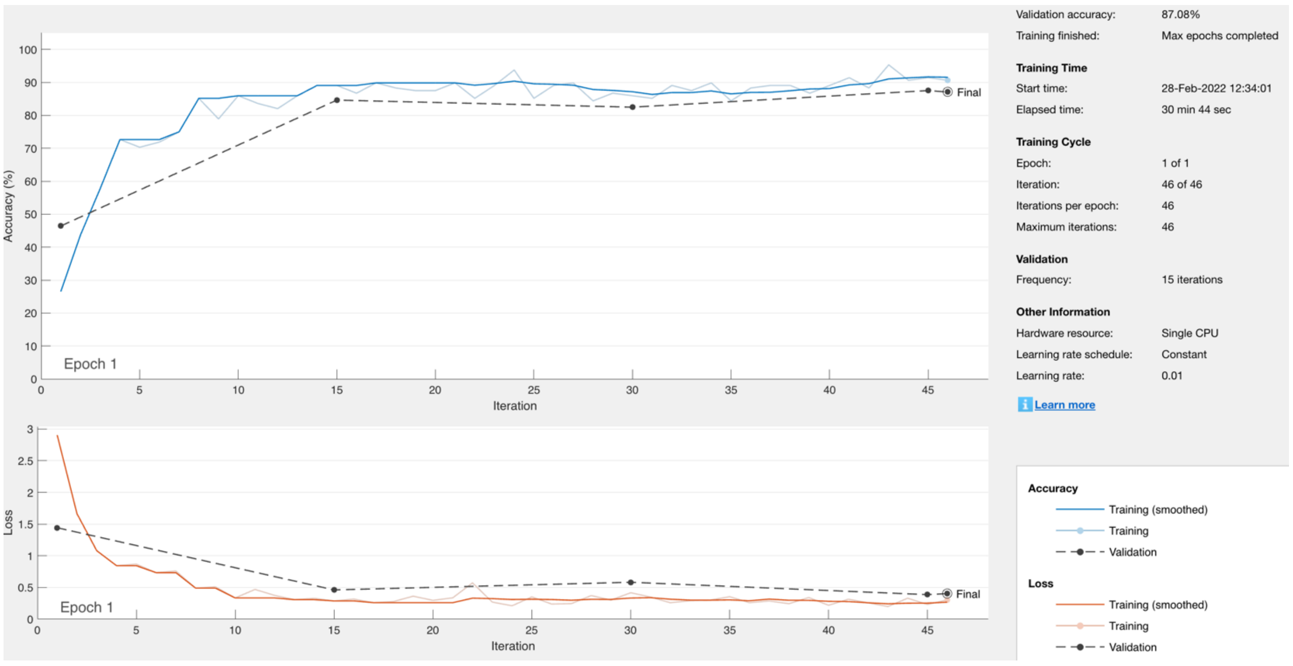
Let’s take a look at the hyperparameters, used to train the data and at the result obtained.

Firstly, has been implemented an architecture with the following hyperparameters:



SOLVER	
Solver	sgdm
InitialLearnRate	0.01
BASIC	
ValidationFrequency	15
MaxEpochs	1
MiniBatchSize	128
ExecutionEnvironment	auto
SEQUENCE	
SequenceLength	longest
SequencePaddingValue	0
SequencePaddingDirection	right
ADVANCED	
L2Regularization	0.0001
GradientThresholdMethod	l2norm
GradientThreshold	Inf
ValidationPatience	Inf
Shuffle	every-epoch
CheckpointPath	
LearnRateSchedule	none
LearnRateDropFactor	0.1
LearnRateDropPeriod	1
ResetInputNormalization	✓
BatchNormalizationStatistics	population
OutputNetwork	best-valid…
Momentum	0.9

The result obtained is the following:



Validation accuracy: 87.08%
Training finished: Max epochs completed

Training Time
Start time: 28-Feb-2022 12:34:01
Elapsed time: 30 min 44 sec

Training Cycle
Epoch: 1 of 1
Iteration: 46 of 46
Iterations per epoch: 46
Maximum iterations: 46

Validation
Frequency: 15 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.01

Learn more

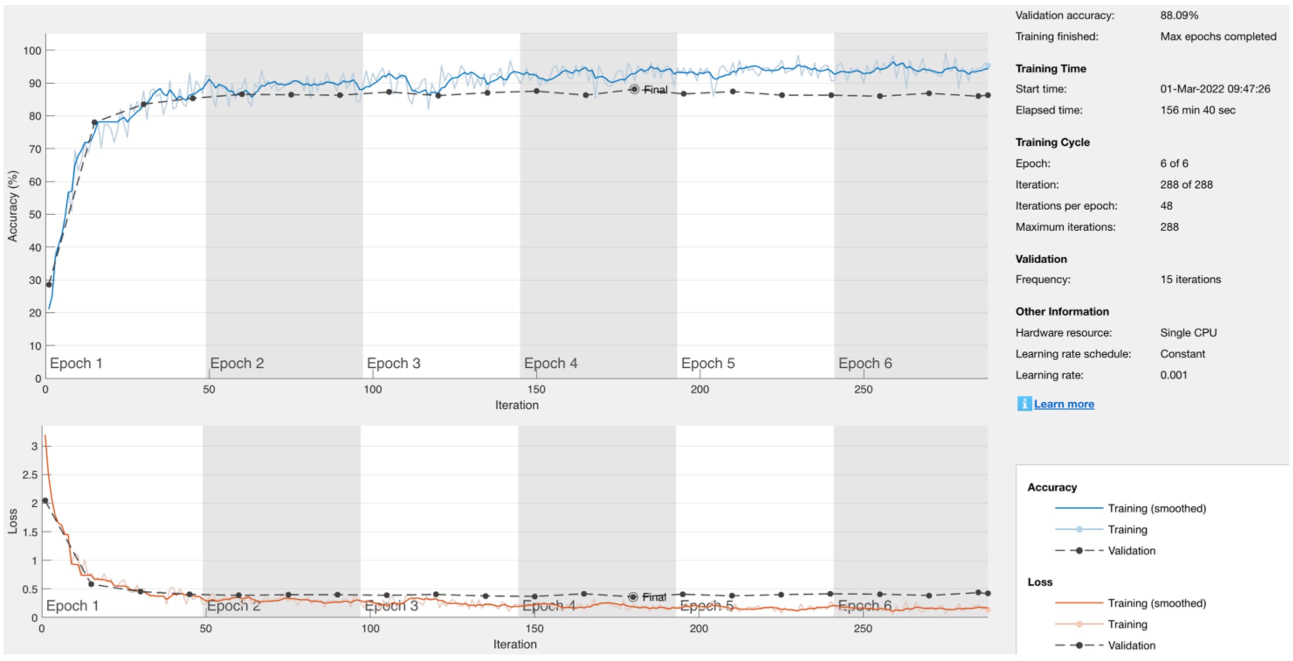
Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

The above result was obtained as a first try to have a quick result and try to understand how to modify the hyperparameters in order to improve our result.

As this result has been obtained using only one epoch, let’s try to increase the number of epochs and modify a bit even other parameter such as:

- Initial learning rate from 0.01 to 0.001.
- Max epochs from 1 to 6.
- Add the shuffle for each epoch.
- Set the output network, as best validation – loss.



Validation accuracy: 88.09%
Training finished: Max epochs completed

Training Time
Start time: 01-Mar-2022 09:47:26
Elapsed time: 156 min 40 sec

Training Cycle
Epoch: 6 of 6
Iteration: 288 of 288
Iterations per epoch: 48
Maximum iterations: 288

Validation
Frequency: 15 iterations

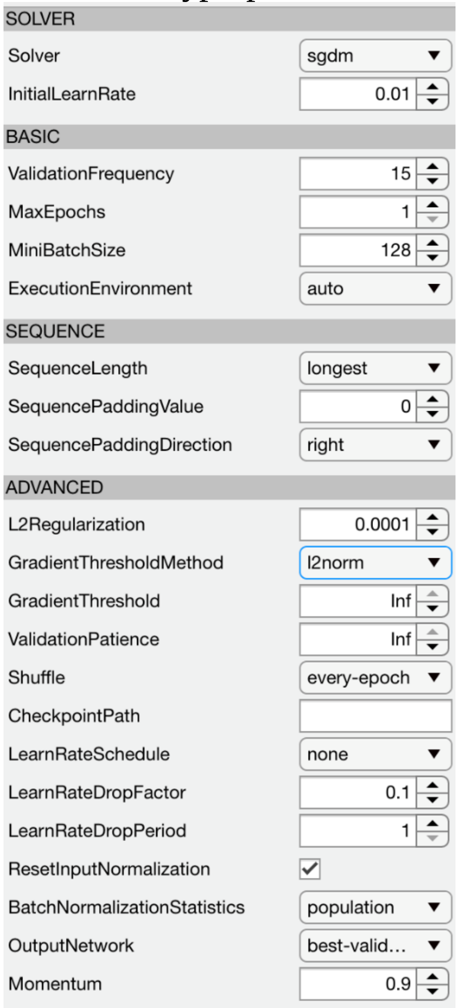
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

Learn more

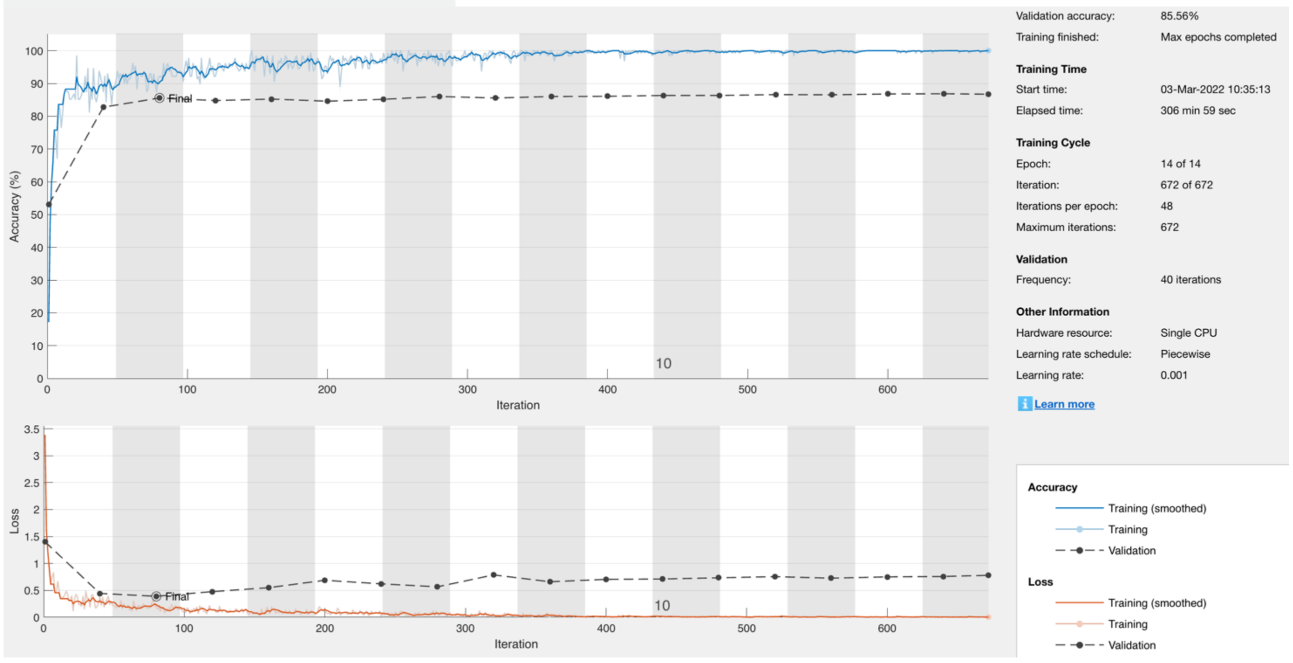
Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

As we can see using this configuration, we obtain an increase of the validation accuracy from 87.08% to 88.09% (+1.01%).
We are going to perform one other try changing again the hyperparameters. This time we set the hyperparameters as follows:



SOLVER	
Solver	sgdm
InitialLearnRate	0.01
BASIC	
ValidationFrequency	15
MaxEpochs	1
MiniBatchSize	128
ExecutionEnvironment	auto
SEQUENCE	
SequenceLength	longest
SequencePaddingValue	0
SequencePaddingDirection	right
ADVANCED	
L2Regularization	0.0001
GradientThresholdMethod	l2norm
GradientThreshold	Inf
ValidationPatience	Inf
Shuffle	every-epoch
CheckpointPath	
LearnRateSchedule	none
LearnRateDropFactor	0.1
LearnRateDropPeriod	1
ResetInputNormalization	✓
BatchNormalizationStatistics	population
OutputNetwork	best-valid...
Momentum	0.9



Validation accuracy:	85.56%
Training finished:	Max epochs completed
Training Time	
Start time:	03-Mar-2022 10:35:13
Elapsed time:	306 min 59 sec
Training Cycle	
Epoch:	14 of 14
Iteration:	672 of 672
Iterations per epoch:	48
Maximum iterations:	672
Validation	
Frequency:	40 iterations
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	0.001

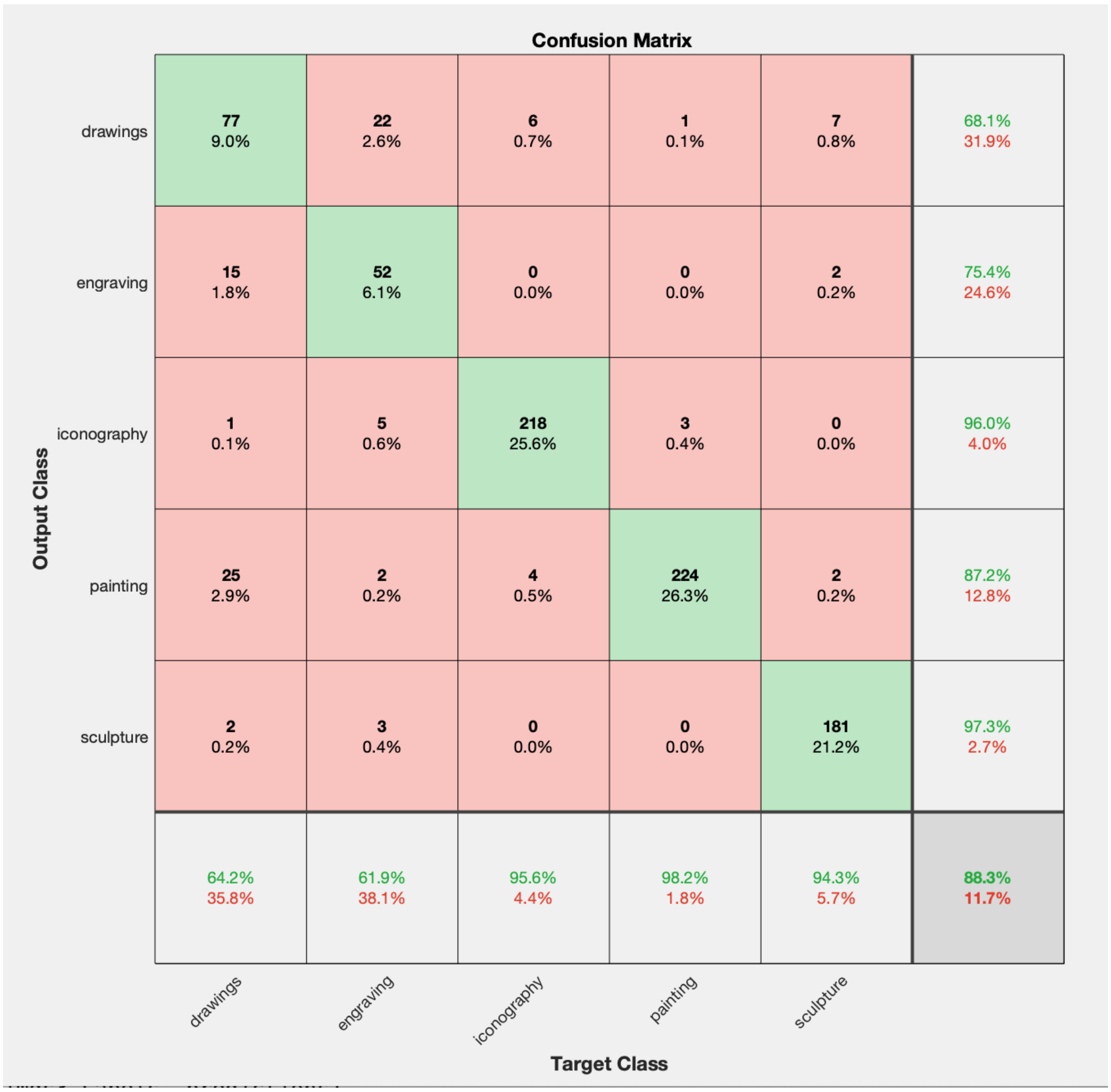
Learn more

Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

This model gives to us an obvious worst result obtaining an accuracy of 85.56%.

Testing

As the second configuration was the best one, we are going to test it.



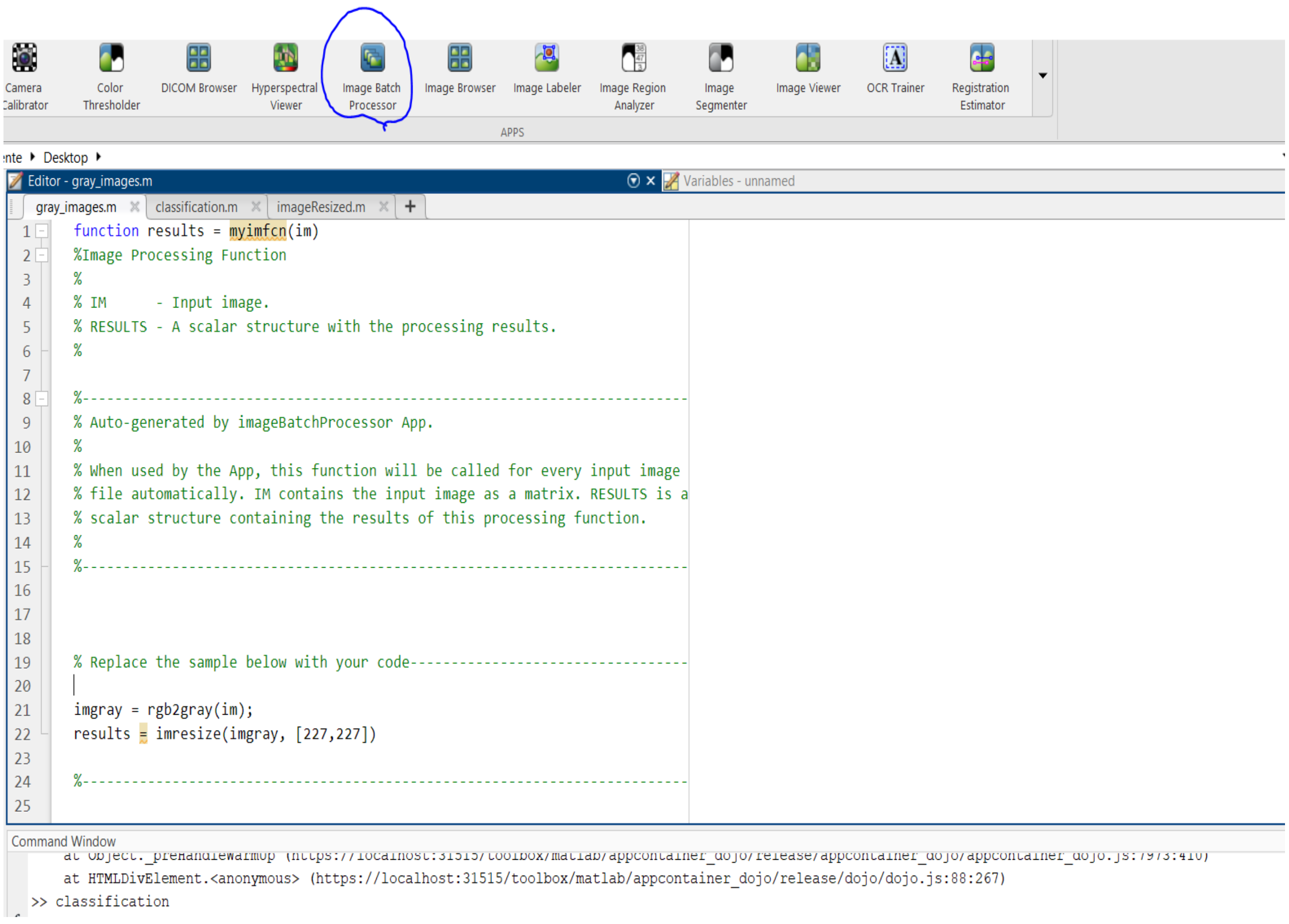
Confusion Matrix

Output Class \ Target Class	drawings	engraving	iconography	painting	sculpture	
drawings	77 9.0%	22 2.6%	6 0.7%	1 0.1%	7 0.8%	68.1% 31.9%
engraving	15 1.8%	52 6.1%	0 0.0%	0 0.0%	2 0.2%	75.4% 24.6%
iconography	1 0.1%	5 0.6%	218 25.6%	3 0.4%	0 0.0%	96.0% 4.0%
painting	25 2.9%	2 0.2%	4 0.5%	224 26.3%	2 0.2%	87.2% 12.8%
sculpture	2 0.2%	3 0.4%	0 0.0%	0 0.0%	181 21.2%	97.3% 2.7%
	64.2% 35.8%	61.9% 38.1%	95.6% 4.4%	98.2% 1.8%	94.3% 5.7%	88.3% 11.7%

We can see from the confusion matrix that the test accuracy is of 88.3%.
As expected, due to the fact that we have an unbalanced data, the classes with the worst results are those with less images, so engraving, followed by drawings. On the other hand, the classes with a higher number of images obtain a better result. From the best to the worst we have, painting, followed by iconography and sculpture.

Resize: Image Batch Processor

The first trial we did was to understand if considering images with just one channel and so in a grey scale or considering images with 3 channels could improve our prediction, and by the time we had almost the total amount of images in rgb we got into consideration how to manage this condition, to transform the whole dataset in a grey scale or eliminate the (few) images with no 3rd channel. Matlab provides using the ”Image Batch Processor” to apply a defined function (in this case we applied a “**rgb2gray()** and then **imresize()**)to a whole set of images:



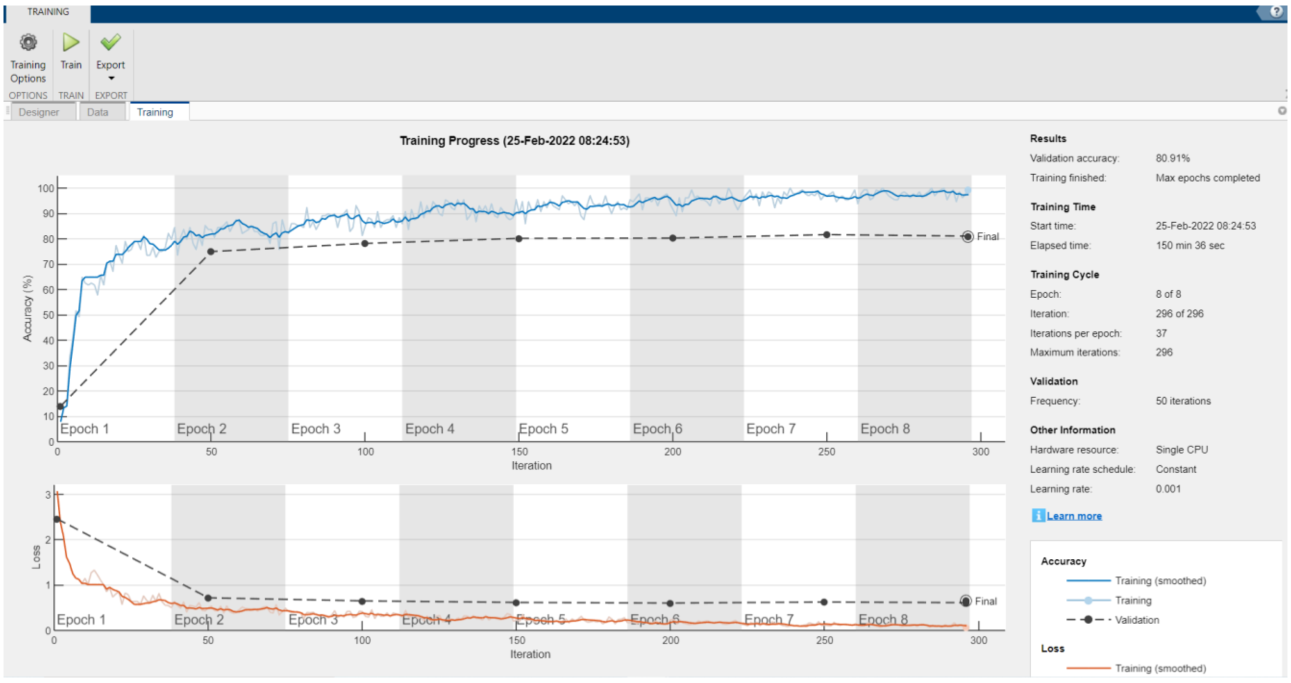
```
function results = myimfcn(im)
%Image Processing Function
%
% IM      - Input image.
% RESULTS - A scalar structure with the processing results.
%

%--------------------------------------------------------------------------
% Auto-generated by imageBatchProcessor App.
%
% When used by the App, this function will be called for every input image
% file automatically. IM contains the input image as a matrix. RESULTS is a
% scalar structure containing the results of this processing function.
%
%--------------------------------------------------------------------------



% Replace the sample below with your code--------------------------------

imgray = rgb2gray(im);
results = imresize(imgray, [227,227])

%--------------------------------------------------------------------------
```

```
    at Object._preHandleWarmup (https://localhost:31515/toolbox/matlab/appcontainer_dojo/release/appcontainer_dojo/appcontainer_dojo.js:7973:410)
    at HTMLDivElement.<anonymous> (https://localhost:31515/toolbox/matlab/appcontainer_dojo/release/dojo/dojo.js:88:267)
>> classification
```

Applying this easy function and resizing the images in a [227,227] dimension we compared the results using the same pretrained network.

ResNet18 model with Grey Images

In this first network we used the pretrained net ResNet18 a network with 71 layers and 77 connections, as we can see in the images below we have a training accuracy that reach a value very close to hundred percent and considering a learning rate equals to 0.001 with a total of 296 iterations we reached a validation accuracy of 80.91%.



Results

Validation accuracy:	80.91%
Training finished:	Max epochs completed

Training Time

Start time:	25-Feb-2022 08:24:53
Elapsed time:	150 min 36 sec

Training Cycle

Epoch:	8 of 8
Iteration:	296 of 296
Iterations per epoch:	37
Maximum iterations:	296

Validation

Frequency:	50 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

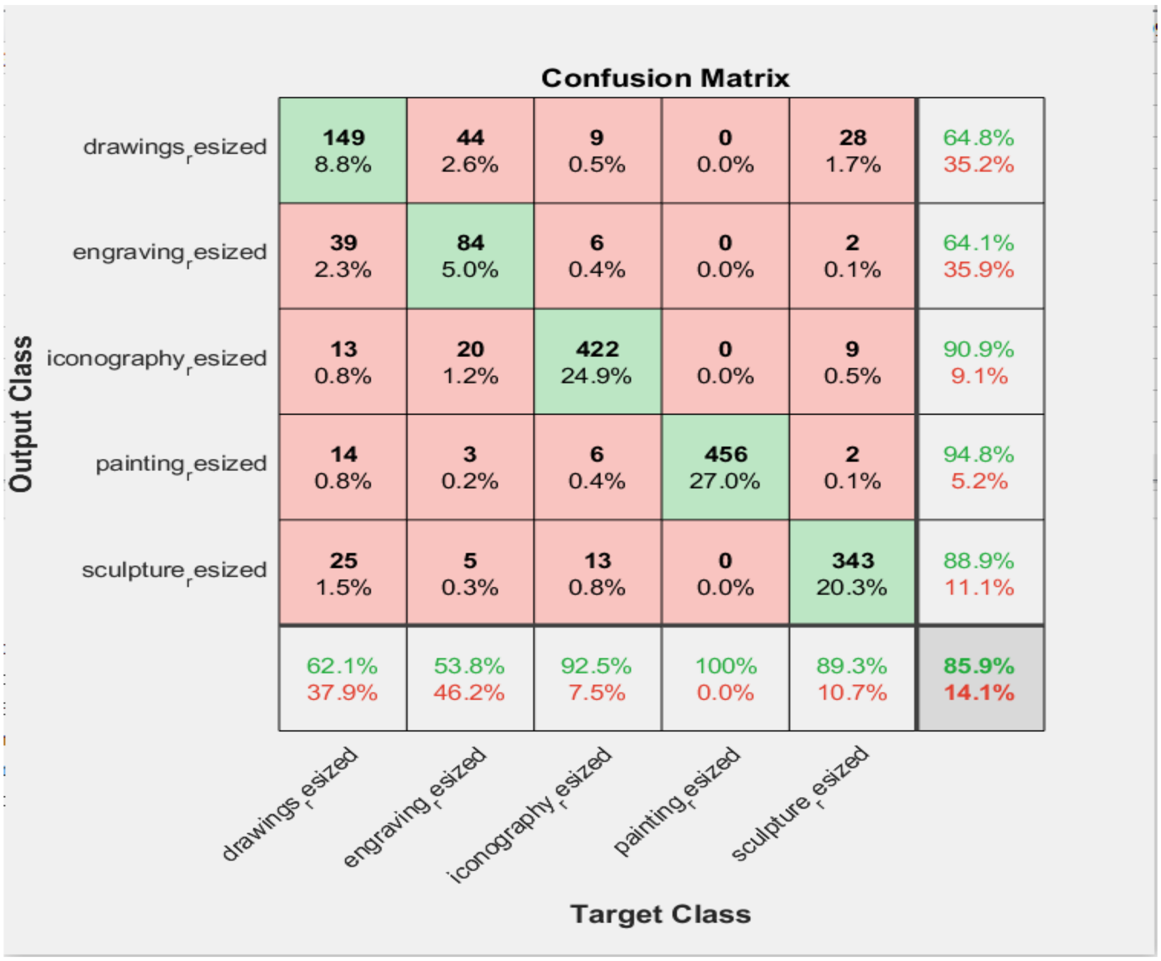
Accuracy

- Training (smoothed)
- Training
- Validation

Loss

- Training (smoothed)

These are the **hyperparameters** we set in the beginning:
Solver : sgdm, that represent the Stochastic gradient descent with momentum
Validation Frequency : we set 50 as a value to check the trend of the validity
We can see the Test Set with an accuracy of 85.9 % that can be considered as acceptable score. We can see from the confusion matrix below that the last 3 classes are correctly classified for a very high percentage and these 3 classes have a number of images close to 2000, while the first 2 classes (drawings and engravings) are not well classified, with just 62% and 51% of images correctly classified respectively.

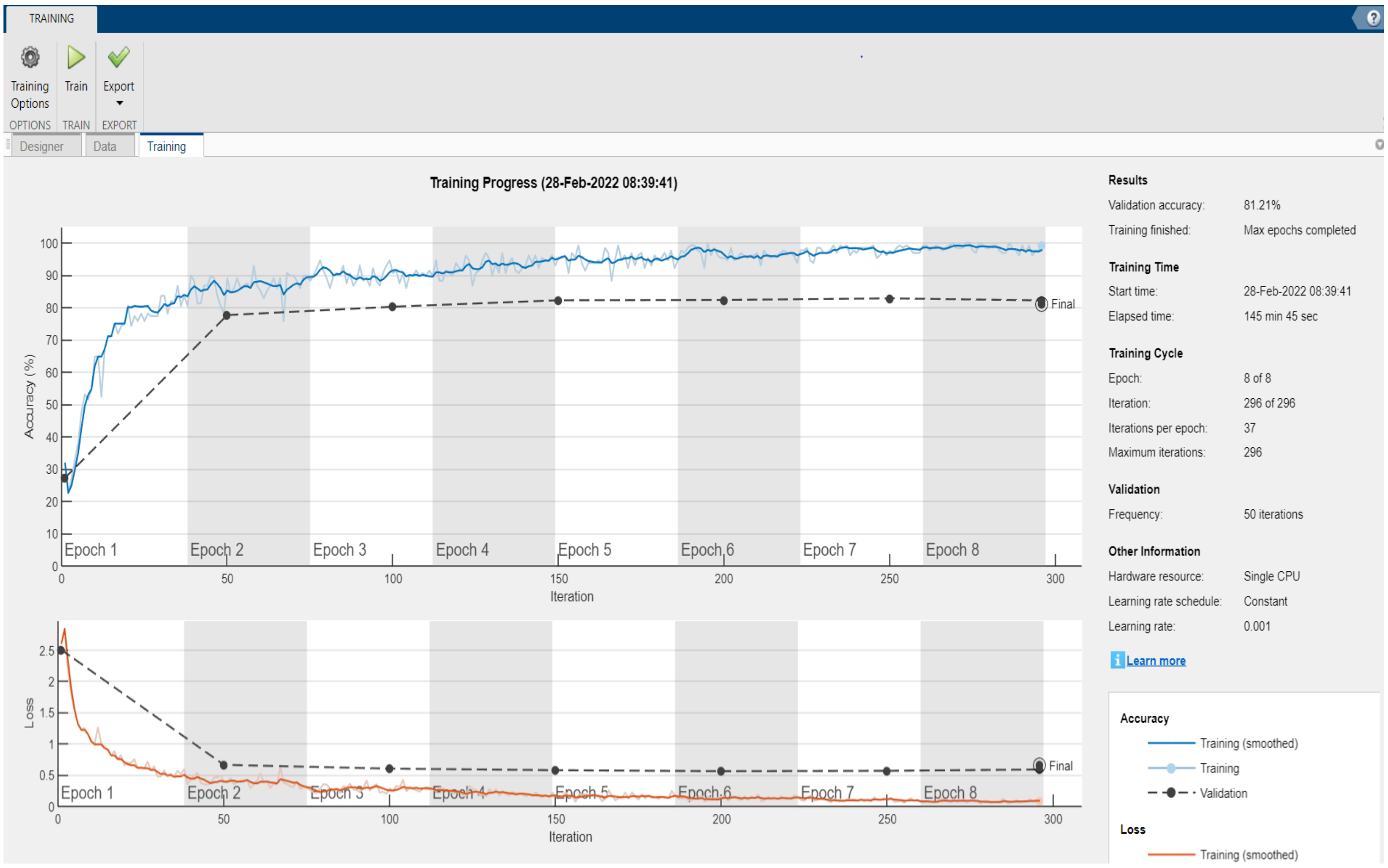


Confusion Matrix

Output Class \ Target Class	drawings$_r$esized	engraving$_r$esized	iconography$_r$esized	painting$_r$esized	sculpture$_r$esized	
drawings$_r$esized	**149** 8.8%	**44** 2.6%	**9** 0.5%	**0** 0.0%	**28** 1.7%	64.8% 35.2%
engraving$_r$esized	**39** 2.3%	**84** 5.0%	**6** 0.4%	**0** 0.0%	**2** 0.1%	64.1% 35.9%
iconography$_r$esized	**13** 0.8%	**20** 1.2%	**422** 24.9%	**0** 0.0%	**9** 0.5%	90.9% 9.1%
painting$_r$esized	**14** 0.8%	**3** 0.2%	**6** 0.4%	**456** 27.0%	**2** 0.1%	94.8% 5.2%
sculpture$_r$esized	**25** 1.5%	**5** 0.3%	**13** 0.8%	**0** 0.0%	**343** 20.3%	88.9% 11.1%
	62.1% 37.9%	53.8% 46.2%	92.5% 7.5%	100% 0.0%	89.3% 10.7%	**85.9% 14.1%**

ResNet18 model with rgb Images

The result using the ResNet18 and considering images with rgb, so considering images with 3 channels, we can see that there is so difference from the previous model that uses the same set training option with a learning rate equals to 0.001. There is a low improvement in terms of validation accuracy that reaches the value of 81.21%. The consideration is that under certain kind of conditions like the size of the data images and the importance of the images in the learning process, considering images with a grey scale or not can be useful in terms of accuracy having a look the difference in terms of cost complexity. But in our case there is no very difference and so from now we are going to use rgb images taking into consideration that we have just few grey scale images.



Results

Validation accuracy:	81.21%
Training finished:	Max epochs completed

Training Time

Start time:	28-Feb-2022 08:39:41
Elapsed time:	145 min 45 sec

Training Cycle

Epoch:	8 of 8
Iteration:	296 of 296
Iterations per epoch:	37
Maximum iterations:	296

Validation

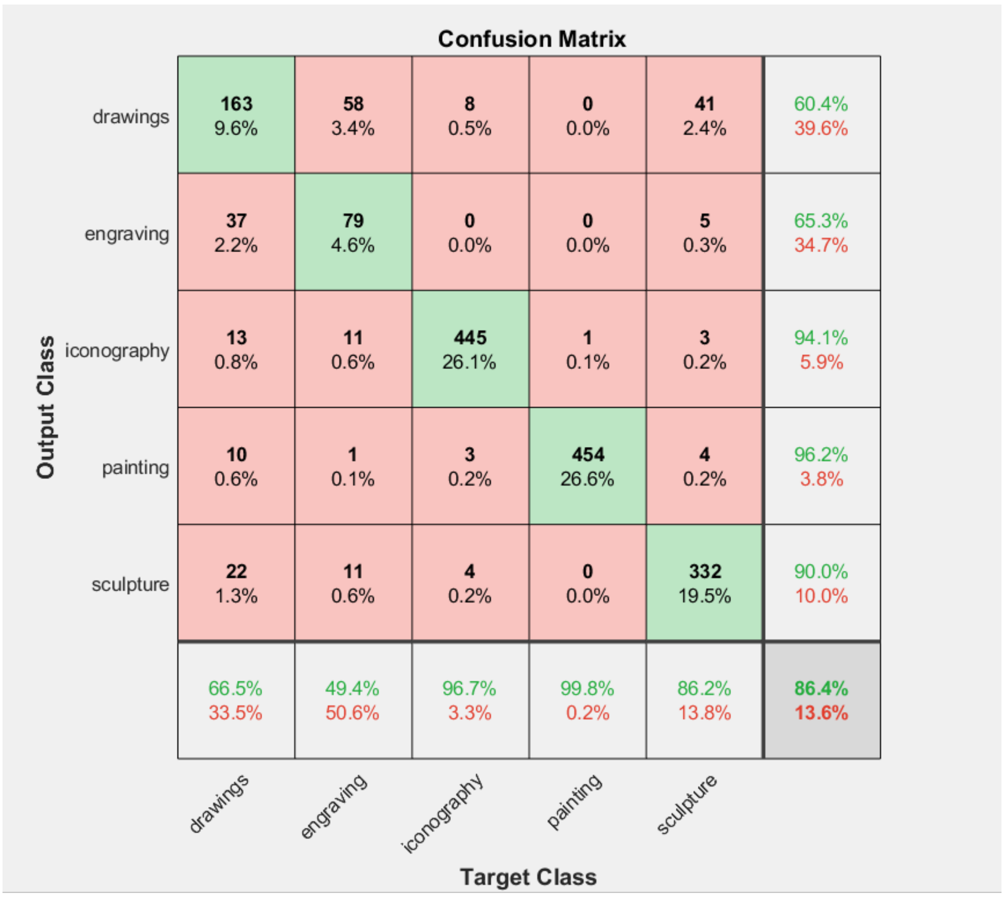
Frequency:	50 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more

Here we have the test accuracy that as we can see is 86.4 %, it's more or less similar to the previous value of 85.9% but we can see for example that in some cases where the importance of the colours is underlined, there an improvement like in the iconographic images where from 422 images correctly classified we move on 445 over 460. Images in the "engraving" class are that one that less are correctly classified, maybe because in general are the images with less that 1000 images in total, and the quantity could be not enough to let the model able to learn (remember that classe are unbalanced and engraving is that one with less elements).



Confusion Matrix

Output Class \ Target Class	drawings	engraving	iconography	painting	sculpture	
drawings	163 9.6%	58 3.4%	8 0.5%	0 0.0%	41 2.4%	60.4% 39.6%
engraving	37 2.2%	79 4.6%	0 0.0%	0 0.0%	5 0.3%	65.3% 34.7%
iconography	13 0.8%	11 0.6%	445 26.1%	1 0.1%	3 0.2%	94.1% 5.9%
painting	10 0.6%	1 0.1%	3 0.2%	454 26.6%	4 0.2%	96.2% 3.8%
sculpture	22 1.3%	11 0.6%	4 0.2%	0 0.0%	332 19.5%	90.0% 10.0%
	66.5% 33.5%	49.4% 50.6%	96.7% 3.3%	99.8% 0.2%	86.2% 13.8%	**86.4%** **13.6%**

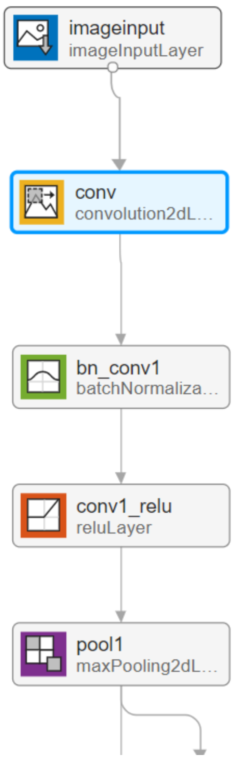
After the first trial just to understand how to move on, we decided to remove the 40 images in the grey scale, expecially to pass more informations to the class with a low quantity of images.

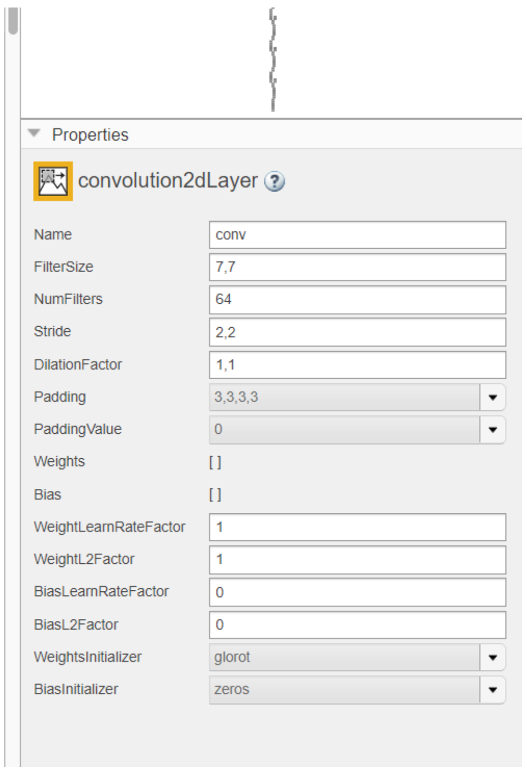
Best Performance achieved: ResNet18

Residual Network (ResNet) is one the famous deep learning models and it is made up with Residual blocks. We are going to see how the model works and which kind of bottleneck is able to manage.

Network Design

Firstly the model has the **imageInputLayer**, in our case by the time we resized the images according to [227,227,3] we set this as InputSize. Even passing images that don't follow the descripted size, the **DeepNetworkDesigner** in Matlab resized the input in before proceeding with the training (of course we resized our test folder following the right dimensions).
Then there is the first block called "**Conv**" composed by a **convolution** + **batch normalization** + **max pooling operation** :
We can see in the image below that the convolution operation use a **kernel size** of 7 and a **feature map size** of 64; the **PaddingValue** equals to zero means that they have padded with zeroes three times on each dimension. The next step is the **batchNormalization**, the goal is to normalizes the mean and the variance of the inputs to the activaction function (we want that the activaction doesn't die). Then we have the activaction function, in this case is used the **ReLu** (Rectified Linear Unit), that will output the input directly if it is positive, otherwise, it will ouptut zero (f(x) = x ^+ = max (0, x)), one of the advantage of the ReLu is the not vanishing gradient (that is common in other activactions like sigmoid or tanh). Then we have the (3 x 3) **MaxPooling** operation (it selects the maximum valued element from the region captured by the filter in any feature map, helping to extract the sharpest feature on the image) with a stride of 2 (applying every s-th spatial location, i.e. subsample the image)

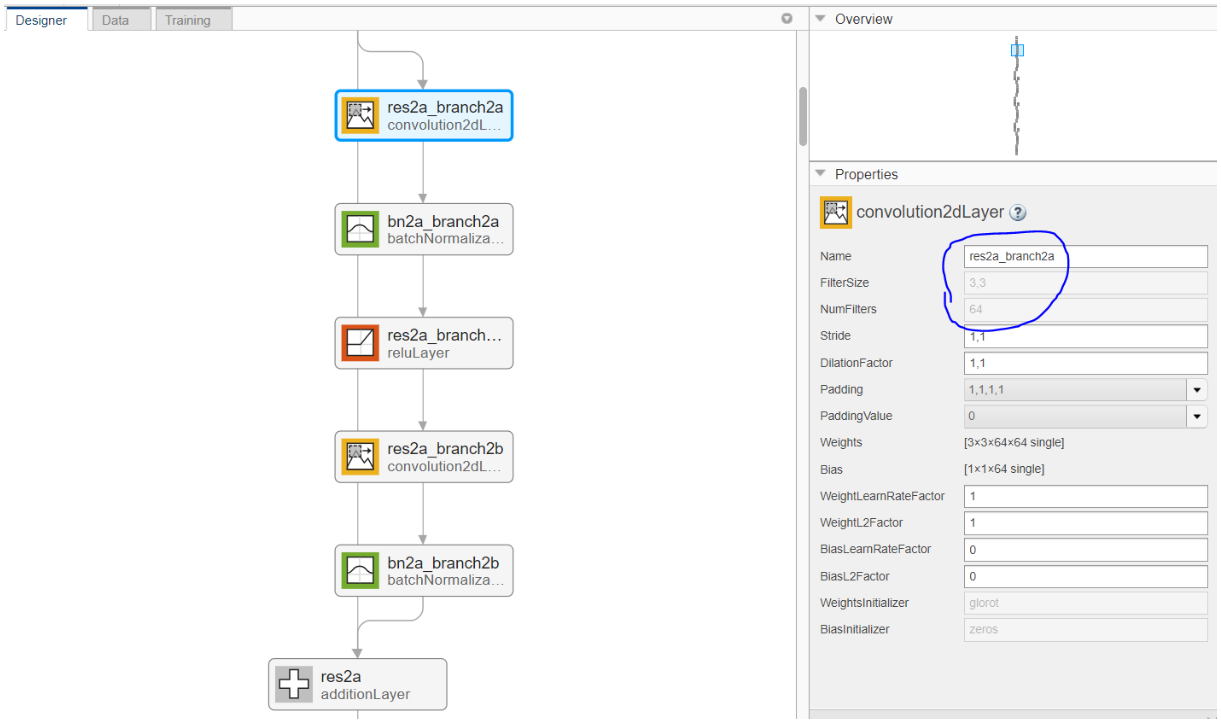




convolution2dLayer

Name	conv
FilterSize	7,7
NumFilters	64
Stride	2,2
DilationFactor	1,1
Padding	3,3,3,3
PaddingValue	0
Weights	[]
Bias	[]
WeightLearnRateFactor	1
WeightL2Factor	1
BiasLearnRateFactor	0
BiasL2Factor	0
WeightsInitializer	glorot
BiasInitializer	zeros

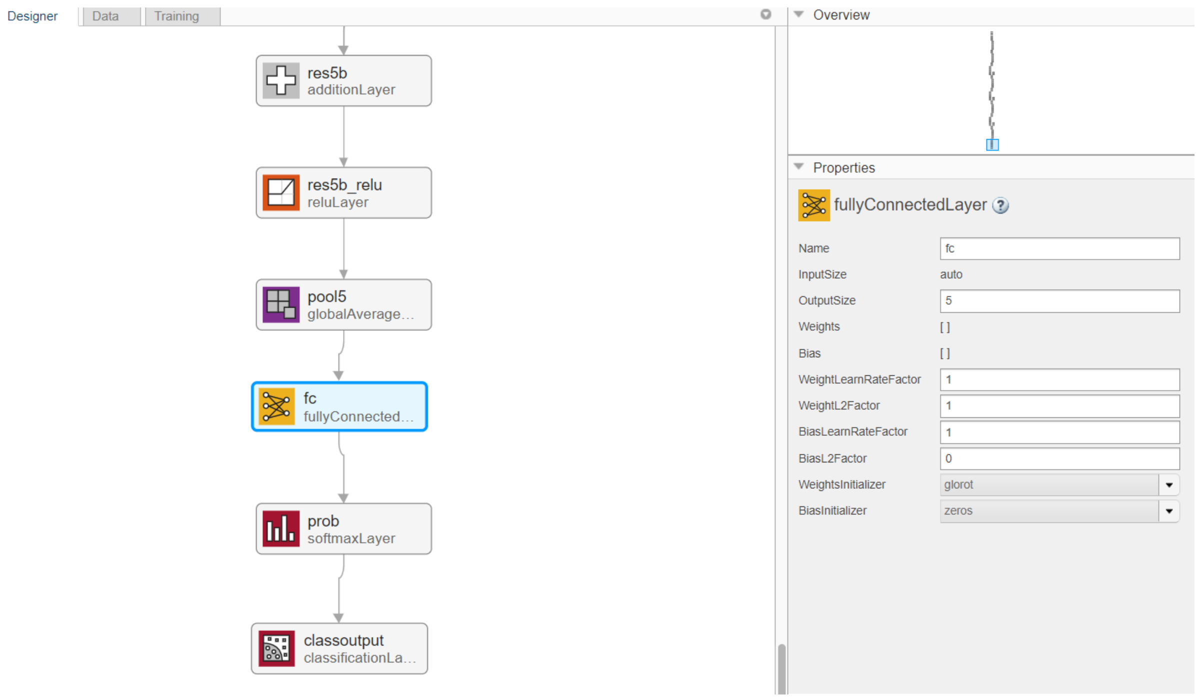
This repeated chain is called **block** and every layer of a ResNet is composed of several blocks. This is because when ResNets go deeper, they normally do it by increasing the number of operations within a block, but the number of total layers remains the same. An operation here refers to a convolution, a batch normalization and a ReLu activaction to an input, except the last operation of a block that does not have the ReLu. The difference within the first operation of each layer is that the kernel uses a filter size of 3 and the **stride** used at the first one is 2 instead of 1 like the rest. This means that the **downsampling** of the volume that the network achieve is done **increasing the stride** instead of a pooling operation, in fact we have **just one max pooling** operation in the first Convolution layer and one average pooling at the end of the ResNet, before the fully connected layer:



convolution2dLayer

Name	res2a_branch2a
FilterSize	3,3
NumFilters	64
Stride	1,1
DilationFactor	1,1
Padding	1,1,1,1
PaddingValue	0
Weights	[3×3×64×64 single]
Bias	[1×1×64 single]
WeightLearnRateFactor	1
WeightL2Factor	1
BiasLearnRateFactor	0
BiasL2Factor	0
WeightsInitializer	glorot
BiasInitializer	zeros

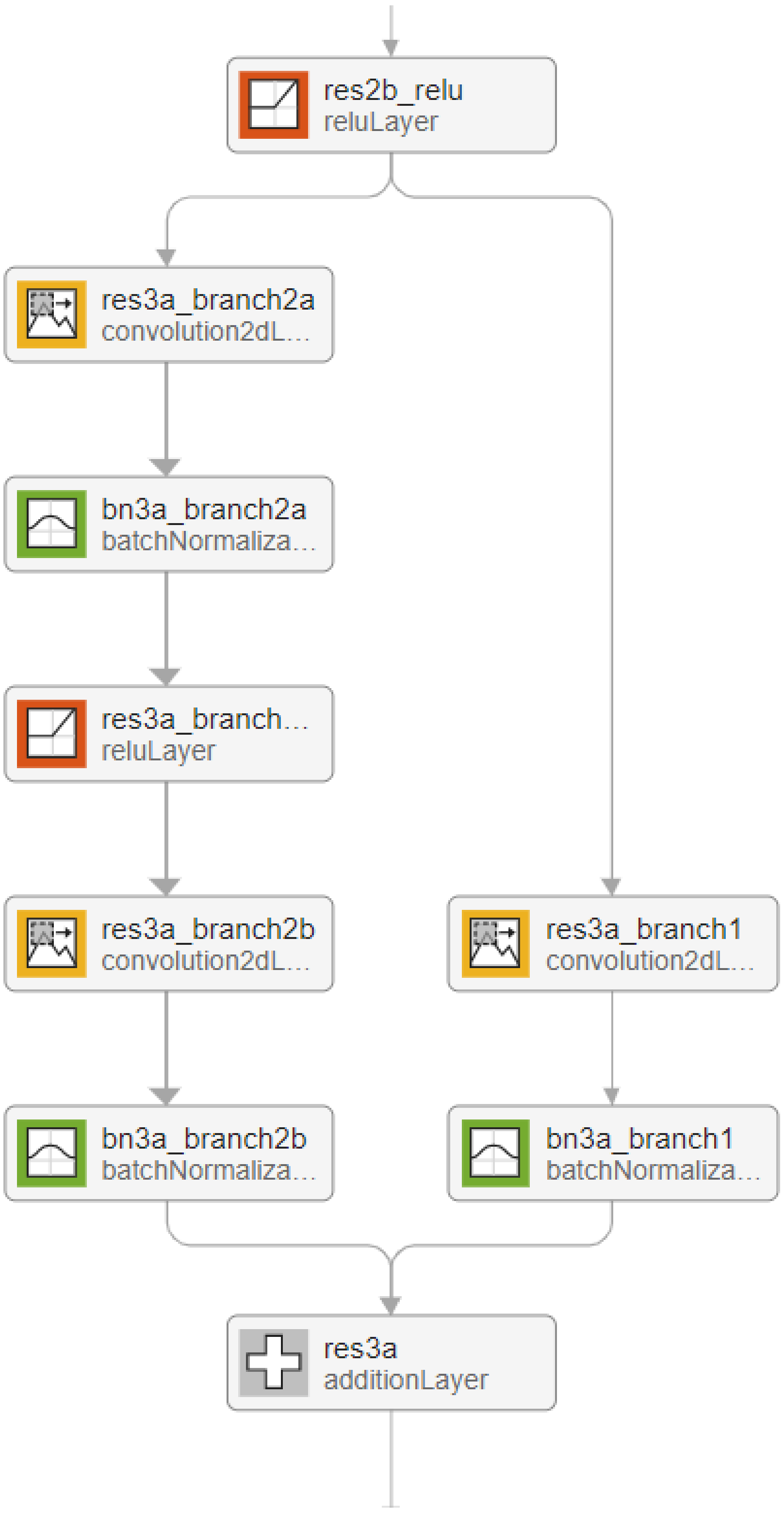
In the image below we have the last block with the fully connected layer (OutputSize equals to 5), then we have a **Softmax layer** (that is a generalization of the sigmoid with multiclass settings and output a probability to each output) and at the end we have the **Classification Layer**.



fullyConnectedLayer

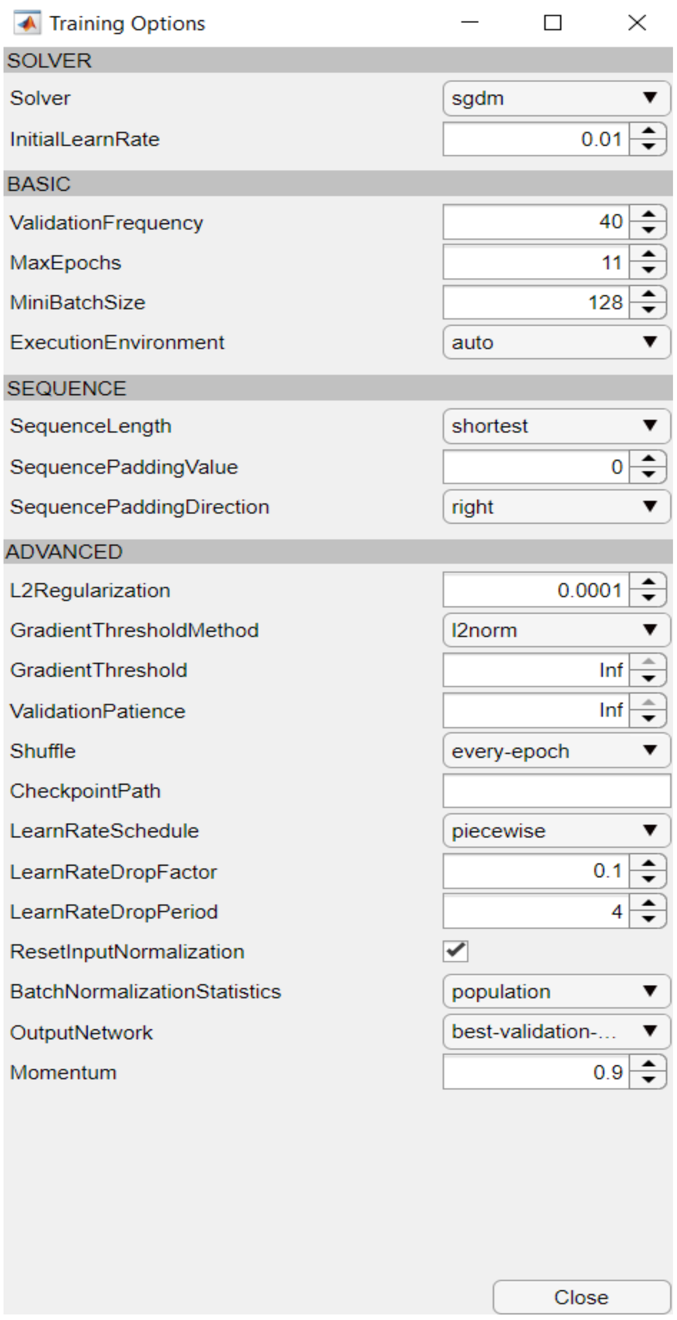
Name	fc
InputSize	auto
OutputSize	5
Weights	[ ]
Bias	[ ]
WeightLearnRateFactor	1
WeightL2Factor	1
BiasLearnRateFactor	1
BiasL2Factor	0
WeightsInitializer	glorot
BiasInitializer	zeros

ResNet is able to overcome the problem of the **Vanishing Gradient** problem, using some particular connections called “Identity shortcut Connections” where as we can see in the image below there is a connection between 2 convolutional layers and the input correspond to the output (so named Identity)


res2b_relu
reluLayer
res3a_branch2a
convolution2dL...
bn3a_branch2a
batchNormaliza...
res3a_branch...
reluLayer
res3a_branch2b
convolution2dL...
res3a_branch1
convolution2dL...
bn3a_branch2b
batchNormaliza...
bn3a_branch1
batchNormaliza...
res3a
additionLayer

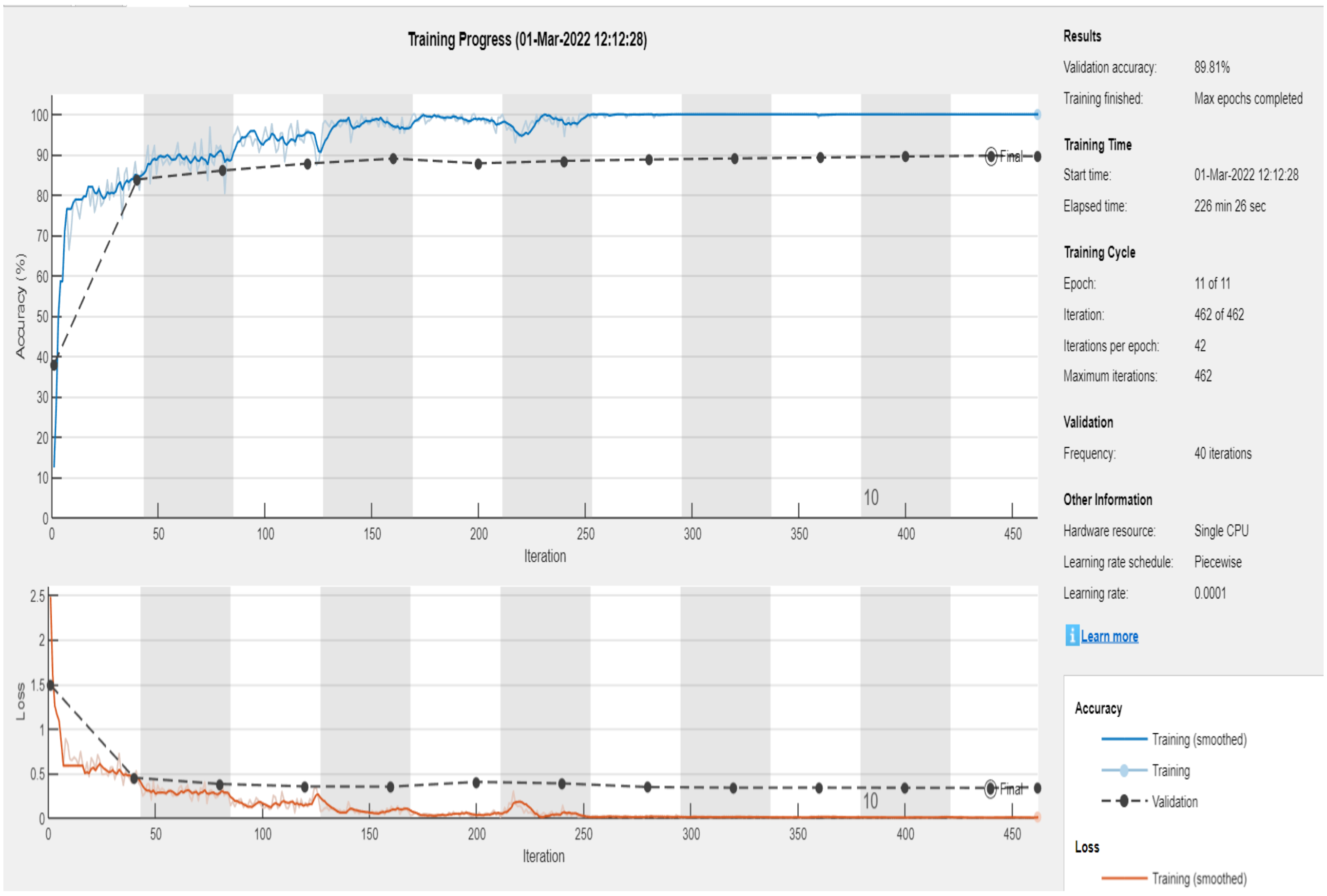
Choosing of Hyperparameters

Now we train our model using a pretrained ResNet18 with 71 layers and 78 connections, here we are going to see the choice of hyperparameters, in particular we have chosen the Stochastic gradient Descent with Momentum as a Solver, we defined the **initialLearnRate** of 0.01 with a **LearnRateDrop** factor equals to 0.01 setting a Period of 4 (this means that every 4 epochs out a total of 11 epochs it will be moltiplied by a value of 10^-1 and it will reach a value of 0.0001). We also set a **miniBatch** size equals to 128 and the dropped line indicates the **ValidationFrequency** every 40 iterations and the **Momentum** equals to 0.9 (the Default value).



SOLVER	
Solver	sgdm
InitialLearnRate	0.01
BASIC	
ValidationFrequency	40
MaxEpochs	11
MiniBatchSize	128
ExecutionEnvironment	auto
SEQUENCE	
SequenceLength	shortest
SequencePaddingValue	0
SequencePaddingDirection	right
ADVANCED	
L2Regularization	0.0001
GradientThresholdMethod	l2norm
GradientThreshold	Inf
ValidationPatience	Inf
Shuffle	every-epoch
CheckpointPath	
LearnRateSchedule	piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	4
ResetInputNormalization	✓
BatchNormalizationStatistics	population
OutputNetwork	best-validation-…
Momentum	0.9

We can see that we reach an accuracy of 89.81% on the Validation:



Results

Validation accuracy:	89.81%
Training finished:	Max epochs completed

Training Time

Start time:	01-Mar-2022 12:12:28
Elapsed time:	226 min 26 sec

Training Cycle

Epoch:	11 of 11
Iteration:	462 of 462
Iterations per epoch:	42
Maximum iterations:	462

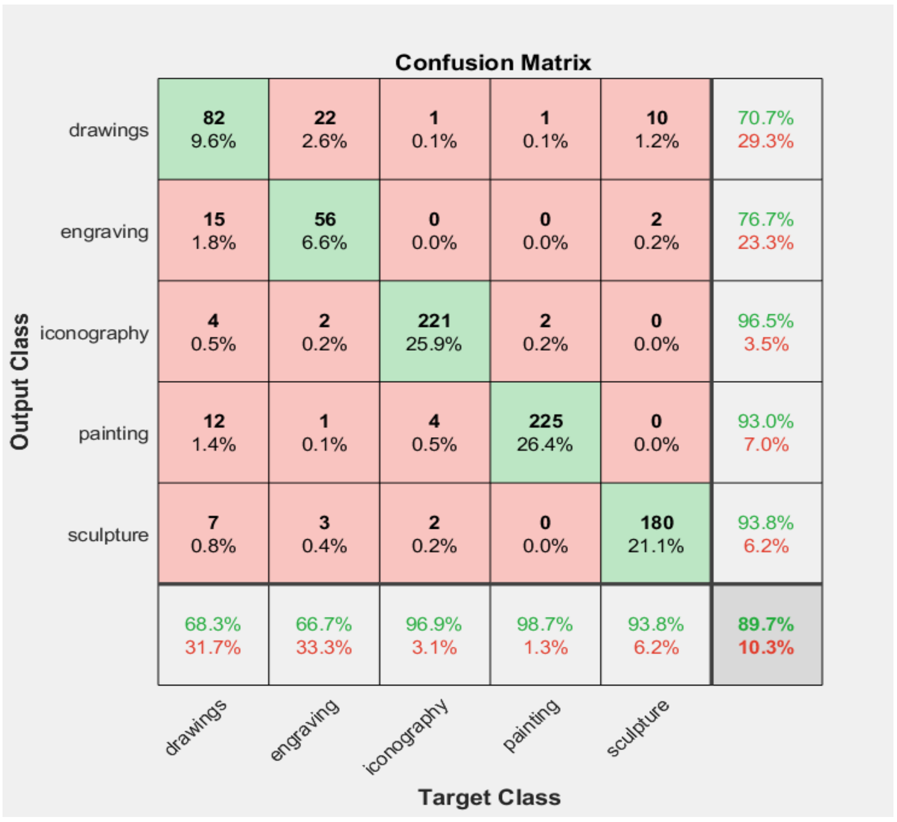
Validation

Frequency:	40 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	0.0001

In accordance to the previous result we can see that the test follow the Validation accuracy as we can see from the confusion matrix below the test accuracy is 89.7%. We set a decay learning rate starting from 0.01 with a drop factor of 0.1 every 5 epochs over a total of 11 epochs. The main issue is connected to the first 2 classes that are not perfectly classified due to the scarcity of images that we have.

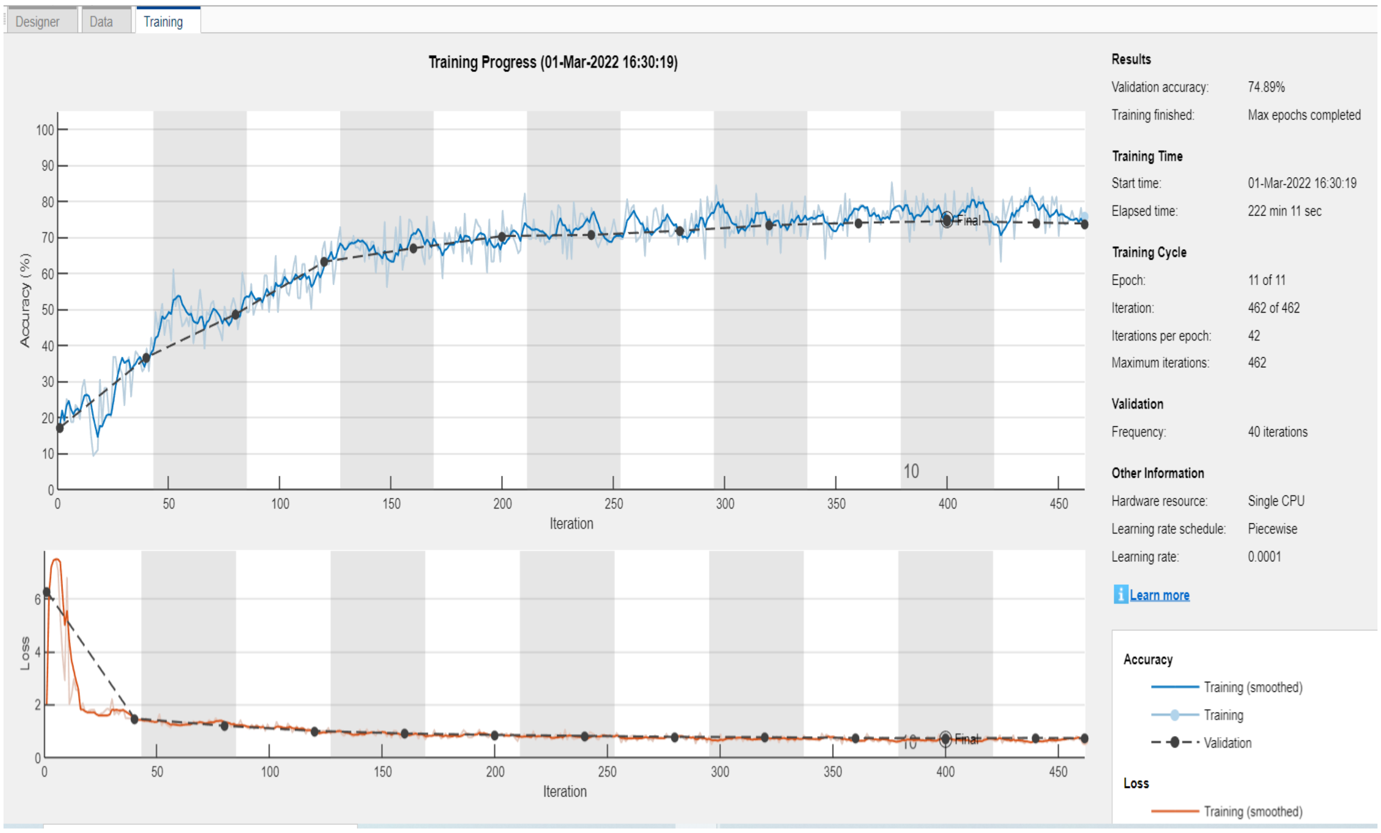


Confusion Matrix

Output Class \ Target Class	drawings	engraving	iconography	painting	sculpture	
drawings	82 9.6%	22 2.6%	1 0.1%	1 0.1%	10 1.2%	70.7% 29.3%
engraving	15 1.8%	56 6.6%	0 0.0%	0 0.0%	2 0.2%	76.7% 23.3%
iconography	4 0.5%	2 0.2%	221 25.9%	2 0.2%	0 0.0%	96.5% 3.5%
painting	12 1.4%	1 0.1%	4 0.5%	225 26.4%	0 0.0%	93.0% 7.0%
sculpture	7 0.8%	3 0.4%	2 0.2%	0 0.0%	180 21.1%	93.8% 6.2%
	68.3% 31.7%	66.7% 33.3%	96.9% 3.1%	98.7% 1.3%	93.8% 6.2%	89.7% 10.3%

Same model – same hyperparameters – used Adam to respect the sgdm

One of the trials we did was to apply the same model and what we changed was just the Solver, instead of using the **StochasticGradientDescentwithMomentum** (if we consider just StochasticGradientDescent it will estimate the derivative on a small Batch and this means that we are not going always in the optimal direction, because the presence of noisy. Here the presence of **Exponentially-weighted average** can provide us a better estimate which is closer to the actual derivate than the noise calculations) we decided to use the **Adam** Solver, that combines the RootMeanSquaredProp (RMSProp that divides the learning rate by an exponentially-decaying average of squared gradients) and the sgdm, this means that it combine the **First Momentum** (mean of gradients) and the **Second Momentum** (variance of gradients) but how we can see the result is very low in terms of accuracy to respect to the previous model, the validation accuracy reaches an accuracy of 74.89% (much lower to respect the previous 89.81%).



Results

Validation accuracy:	74.89%
Training finished:	Max epochs completed

Training Time

Start time:	01-Mar-2022 16:30:19
Elapsed time:	222 min 11 sec

Training Cycle

Epoch:	11 of 11
Iteration:	462 of 462
Iterations per epoch:	42
Maximum iterations:	462

Validation

Frequency:	40 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	0.0001

Accuracy

Training (smoothed)

Training

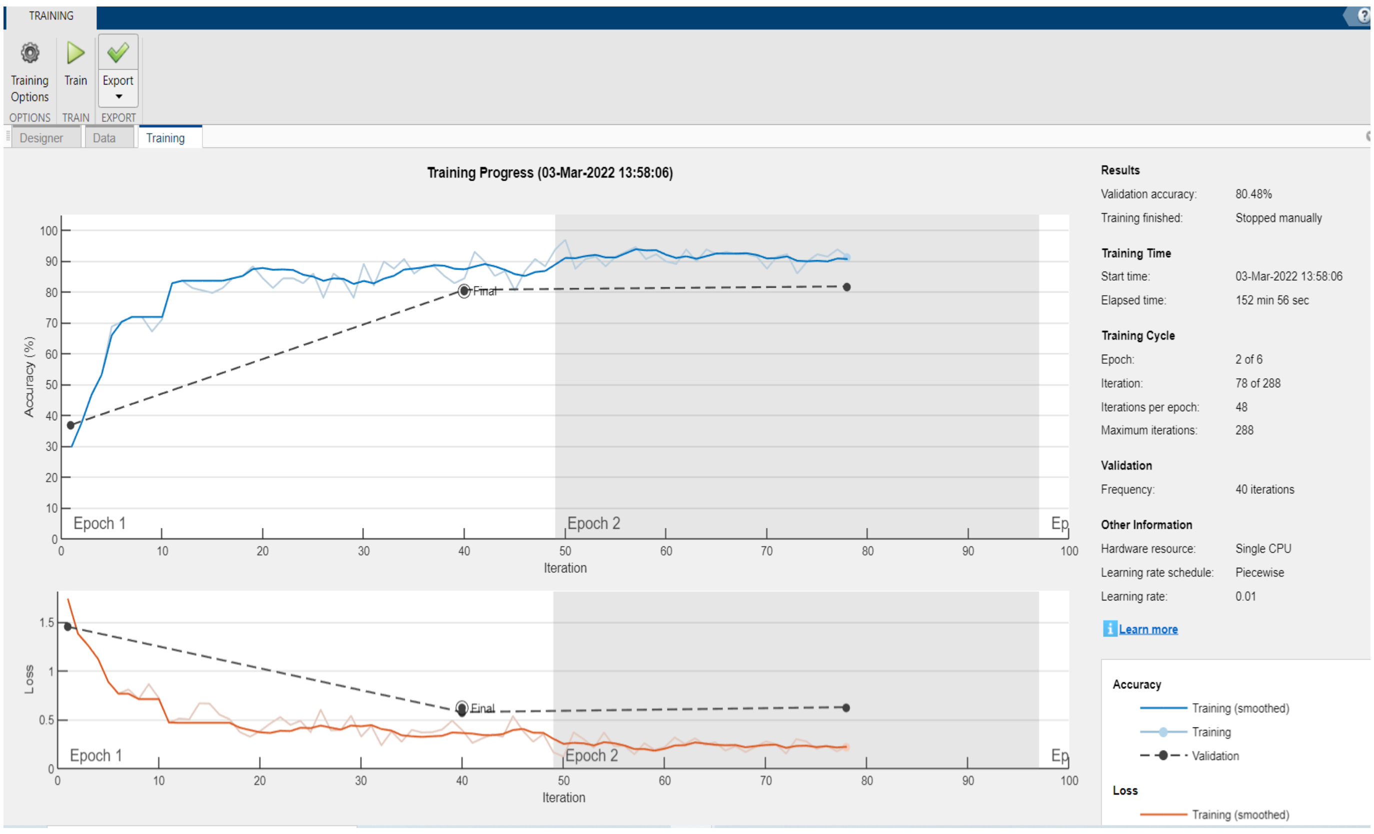
Validation

Loss

Training (smoothed)

ResNet50

Just to understand that the complexity of the net doesn't depends just from how deep is the net, as we can see there is no improvements in terms of accuracy when we use the ResNet50 (the only difference with the ResNet18 is that this contains 50 layers deeper to respect to 18). As we can see from the image below, we stopped the training after almost 3 hours, we didn't finish the 2nd epoch yet and the result was much more worst that the previous one.



Training Progress (03-Mar-2022 13:58:06)

Results

Validation accuracy:	80.48%
Training finished:	Stopped manually

Training Time

Start time:	03-Mar-2022 13:58:06
Elapsed time:	152 min 56 sec

Training Cycle

Epoch:	2 of 6
Iteration:	78 of 288
Iterations per epoch:	48
Maximum iterations:	288

Validation

Frequency:	40 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	0.01

Learn more

Accuracy

Training (smoothed)

Training

Validation

Loss

Training (smoothed)

Conclusion

We can say that after several trials using different architectures and handling the hyperparameters with different initial settings, we found 2 models that better perform and reaches an higher accuracy. In particular we trained the GoogleNet and the ResNet18 that give a good result.
The main problem of the dataset is the unbalanced class in the target variable, and this doesn’t allow the model to understand all the images the same way.

Main network used:

ResNet18: Accuracy: 89.70%

GoogleNet: Accuracy: 88.30%

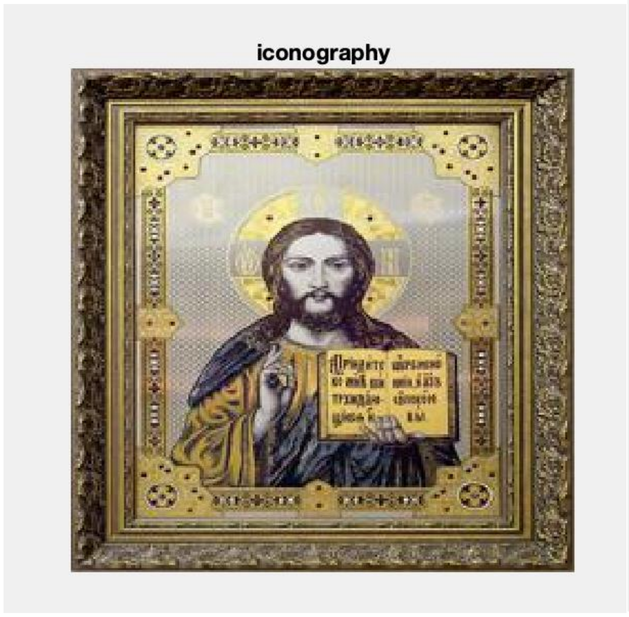
Even if the ResNet18 reach an higher accuracy to respect the GoogleNet we have to underline the difference in terms of computational time (226 min vs 156 min) considering that the GoogleNet has an accuracy not too much far from the ResNet and we could save up more than 1 hour, we can consider the GoogleNet as more efficient considering the variable score – time.

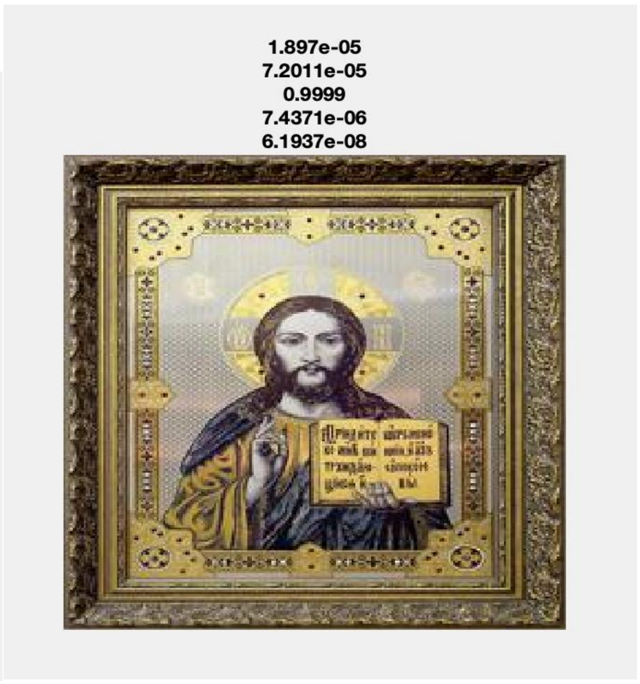
Misclassified Images

We are going to compare some misclassified images and looking at the images below we can see that there are some similarities, in particular focus to the two unbalanced classes (drawings and engravings)

Engravings

Let’s have a look to some of the images that has been wrongly classify.
Specifically, we are going to see the elements of the class that reached the worst score.
The below image belongs to the class engraving but has been classified as iconography. This classification has been done with a probability of 99.99%.



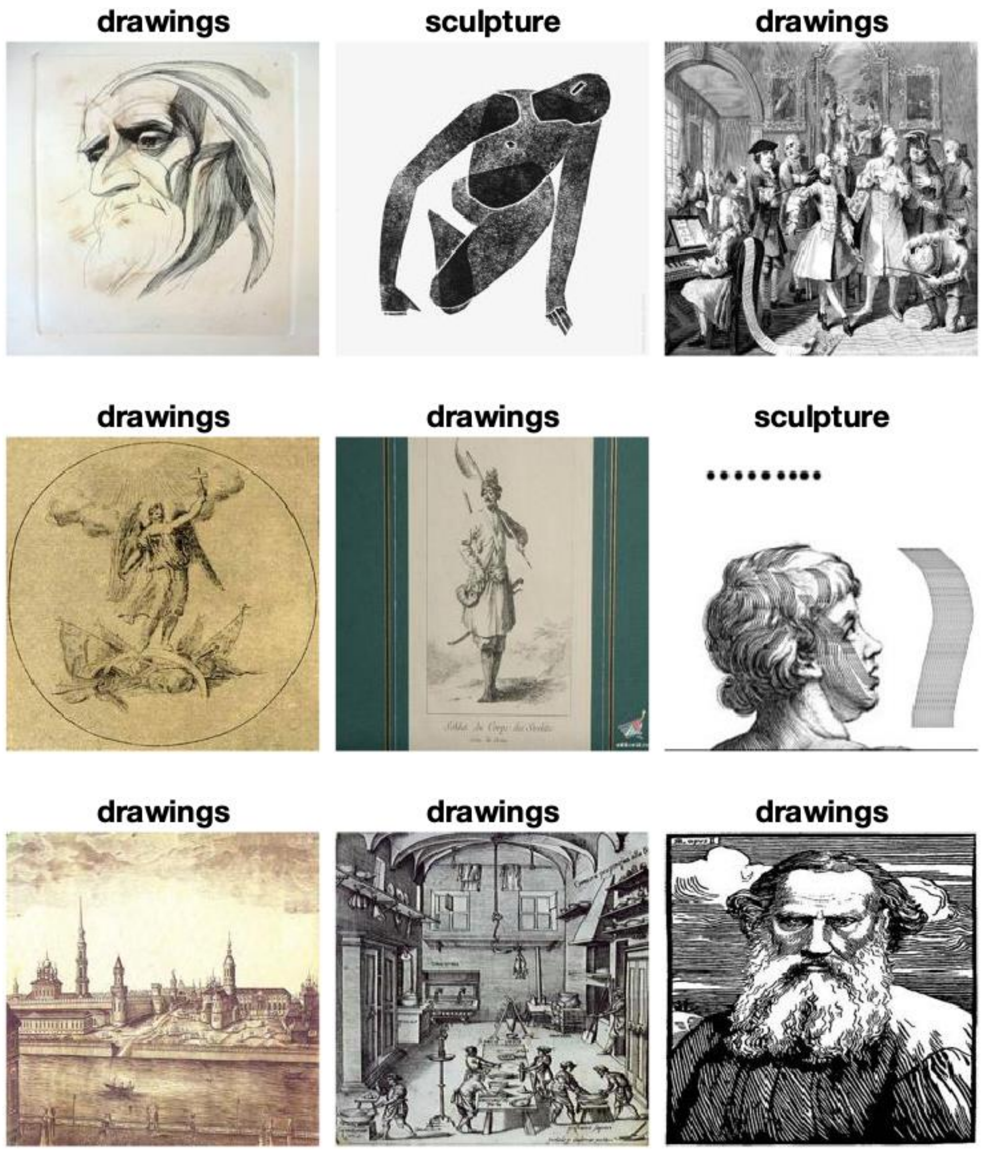


This might seem like we have implemented a completely wrong model, but if we give a look more in depth and compare this image with the images of iconography, come up the actual situation in which we belong.

As mentioned in the introduction, art expresses the imagination and technical skills of an artist, therefore, even if we are able to classify a good part of the types of art present in our dataset, it will always be possible to run into situations in which a single image represents more than one class even if it belongs to only one.

The image above (wrongly classified) belongs to the engraving class, but being an engraving of an iconographic representation, it inevitably makes it difficult to classify.

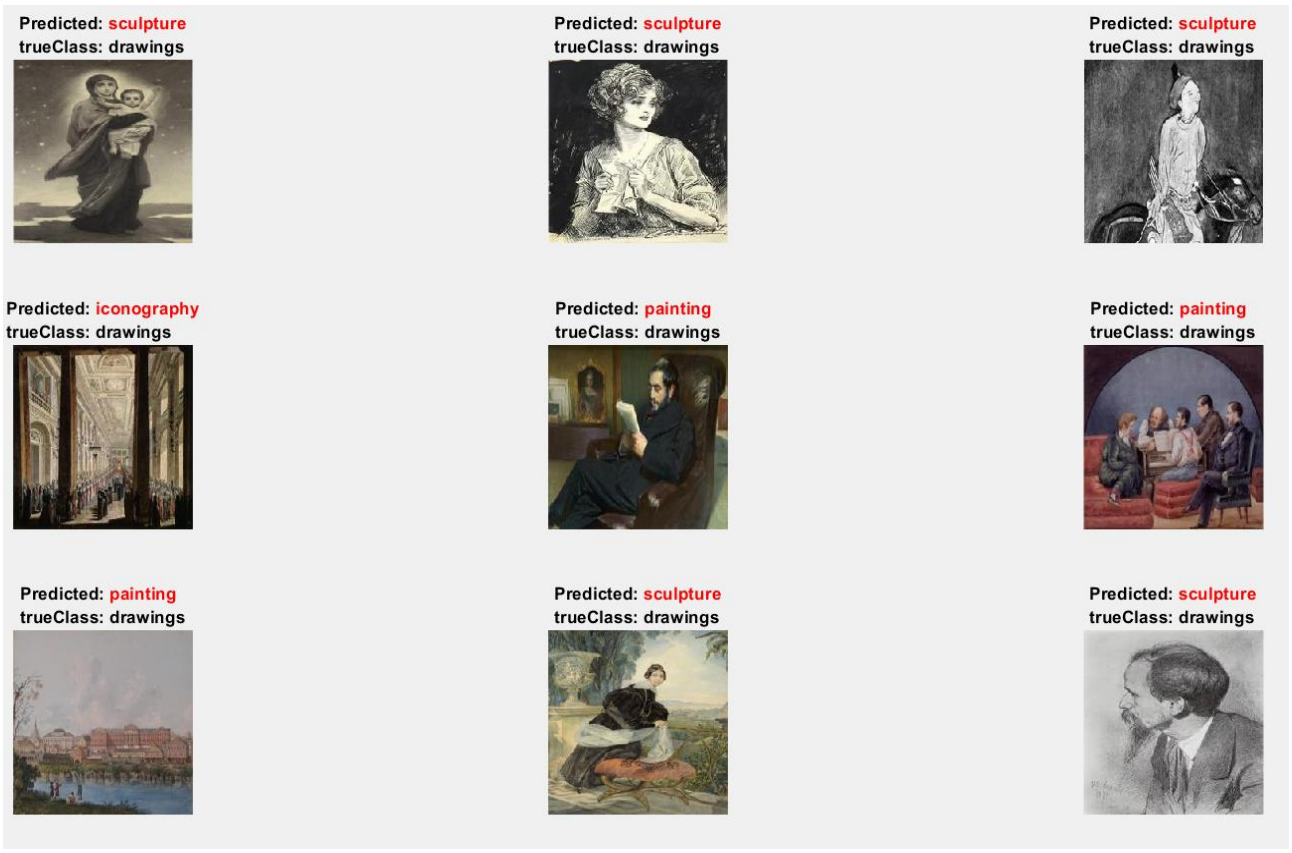
Below is shown other images that belong to the class engraving that has been wrongly classify.



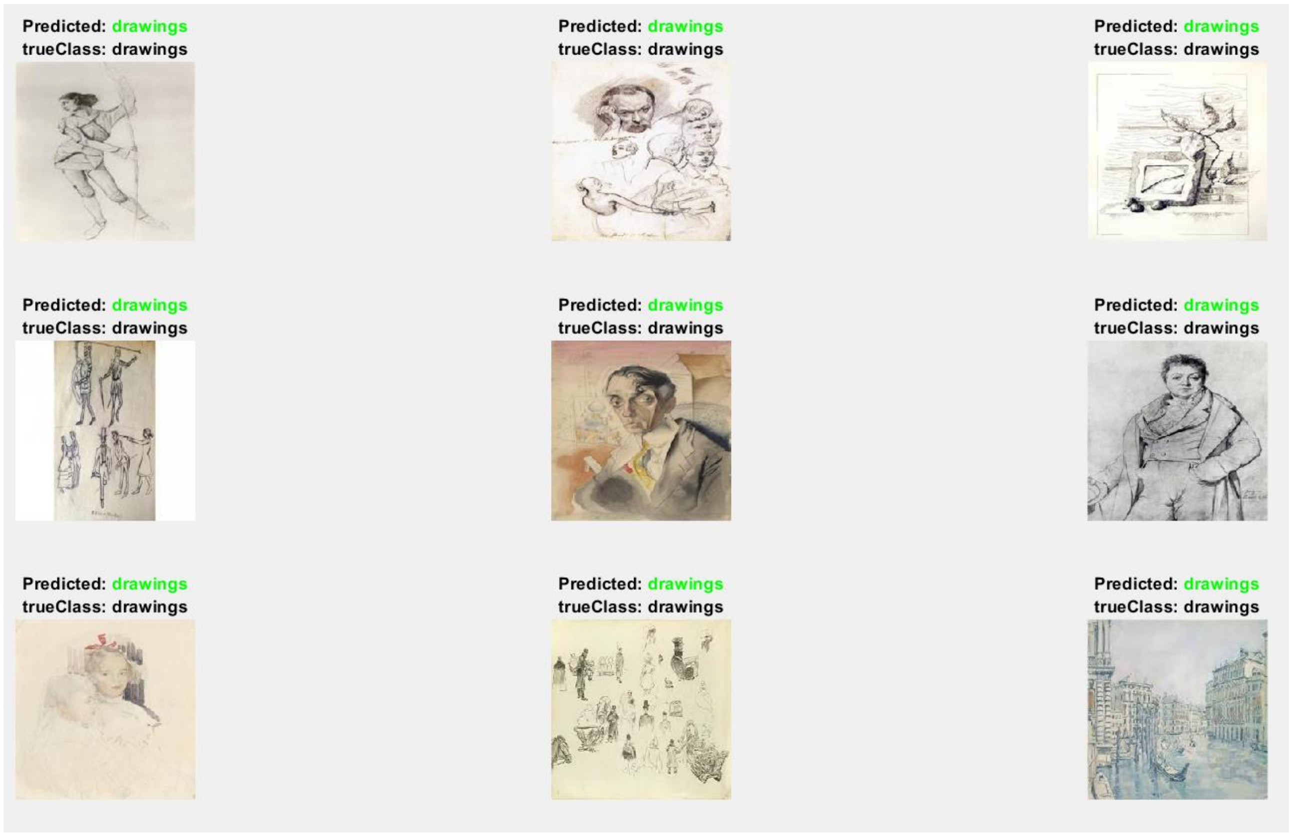
It result quite clear that most of the images that have been wrongly classified have been classified as drawings as expected from the results of the confusion matrix.

Drawing

We selected a subsample of 9 images of misclassified drawing images and we discovered 5 out 9 drawings were classified as sculptures, and what is at first sight clear is that the model when faces images with the presence of "persons", "faces", "body" it makes difficult to distinguish the 2 classes and it tend to classify persons as sculptures. Of course, the fact to have the drawing class unbalanced is relevant to the misclassification. There is one drawing classified as iconographic and it corresponds to that one with the higher presence of brightness, it's likely that the model in order to recognize the iconographic images the presence of colour play a relevant feature. 3 of them are classified as painting, in all of them seems to appear as relevant the presence of a scenario as a first level image, in fact if we have a look to the painting class we will see that most of them have a landscape. If we see the image in the middle even if there is a man, probably the presence of a scenario and the sofa in the background seems to lead the model to consider it as a painting.



Can be interesting to see the drawings correctly classified, the presence of dense line during the “drawing” process, lead the model to understand that the image belongs to that class, and if we see the subsample of correctly classified, the idea seems to be confirmed:


Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings
Predicted: drawings
trueClass: drawings