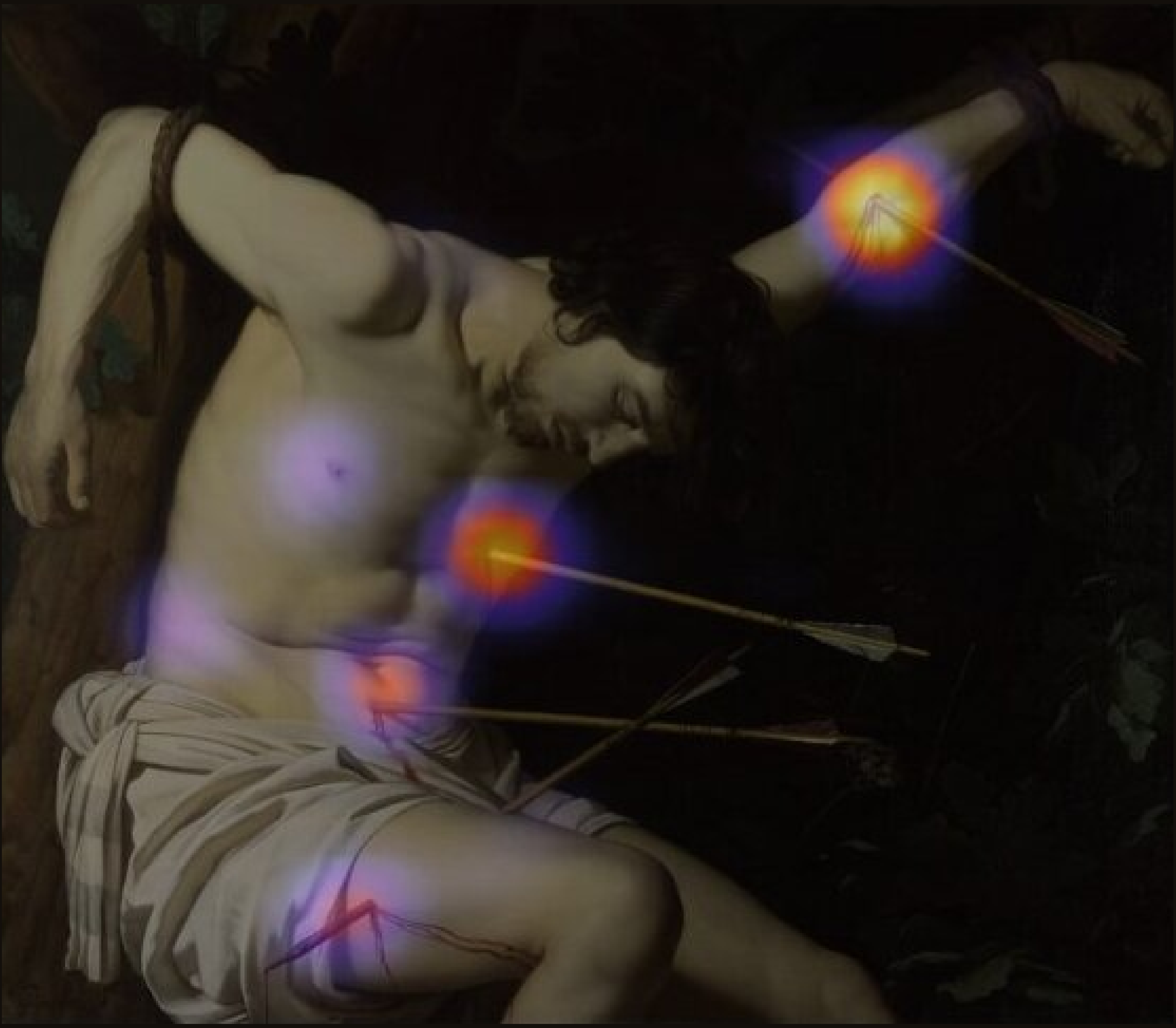
An Explainable Deep Learning Baseline for Iconography Research in Artworks



Christopher Buch Madsen

An Explainable Deep Learning Baseline for Iconography Research in Artworks

Christopher Buch Madsen
11306394

Bachelor Thesis
Credits: 18 ECTS

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. Giovanni Colavizza

Departement Mediastudies
Faculty of Humanities
University of Amsterdam
Turfdraagsterpad 9
1090 GN Amsterdam

January 29th 2021

Abstract

This thesis presents a fully convolutional VGG-16 model with batch normalization, as a baseline for deep learning in iconography research. The model was trained with several strategies on the ArtDL data set through the use of transfer learning, with two strategies thereof yielding promising results. With the addition of class activation mapping, a method from explainable AI, it was possible to explain the bases of inference for the models. The class activation mappings showed that the two most promising iterations of the model are capable of distinguishing saints based on mid-level features extracted in the layers of the model. The model also exhibited promising results as a baseline for automatic image tagging. All code used in the experiments can be found at: https://github.com/christophermadsen/iconography_dl_baseline

Acknowledgements

I would like to thank my thesis supervisor, Dr Giovanni Colavizza, for the useful weekly guidance and feedback sessions, for quick responses to any concerns I had through e-mail, for helping me find a direction when things got overly complicated and methods wouldn't work and for generally inspiring a positive atmosphere. I would like to express my gratefulness to my sweet girlfriend, Lai, for putting up with me and supporting me through these months of stress, hardships and nights of sleeping on the couch to keep me company - thank you! A thank you to my loving family for taking care of me while I was programming in the living room during the Christmas holidays. Furthermore, I'd like to extend my gratitude to Federico Milani who responded and acted immediately when I inquired upon help to apply his convolutional model. Without these people this thesis would not have been possible.

Contents

1 Introduction

In recent years research has made rapid progress towards solving the problem of object recognition in natural world images, yet extreme domain shifts still pose a problem, one example of such domains are historical artworks. An important branch within art history is that of iconography which studies the connection between the visual content of artworks and that which it characterizes. Iconography is an important study for interpreting the meaning of the artworks including investigating the origin and diffusion over time and space of the representations within. Furthermore, it also gives insight into influences across artists and their works. Several artwork data sets have been created for the purpose of academic research, but none with a specific focus upon the iconography. However, iconography has not been neglected and data sets such as Iconclass [3] and IconArt [8] do provide sections dedicated to icons, such as saints. Recently The ArtDL project [16] has supplemented researchers with extensive means to apply deep learning techniques and support specifically iconography research through a data set of artworks with image-level iconography tags. This data set was constructed by collecting artworks containing tags for characters in Christian art from various sources. The ArtDL project also provides a convolutional neural network classifier trained on ImageNet [5] and applied to the ArtDL data through transfer learning.

1.1 Research Questions

Although the ArtDL project provides a trained model, it is at this time the only published model trained specifically for the purpose of recognising characterizations of saints in artworks through their respective iconography. To aid the development of deep learning models for iconography research, the addition of a baseline model will support the state-of-the-art of Milani et al [16] (ArtDL project). A baseline will grant perspective of the criteria of a model for iconography focused deep learning, with respect to Occam's Razor. This thesis will develop such a baseline through transfer learning with a focus on the following research questions:

1. To what extend will the provided baseline be capable of classifying the saints of the ArtDL data set correctly.

2. Through the utilization of methods from explainable AI, will the bases of inference be the distinct visual features from the respective iconography of the saints?

1.2 Related Works

Transfer learning has in recent years proven to be a successful method for shifting from real world data to the artwork domain for tasks such as real world object detection and tagging in artworks [4, 31], detecting people in artworks [30], and detecting artwork style and genre [2]. In 2018, Gonthier et al. [8] applied fine-tuned residual networks [11] and were able to present bounding boxes of two icons, Saint Sebastian and the Crucifixion of Jesus. In 2020, Gonthier et al. continued their work with multiple instance learning and an extremity increase in the domain shift [9]. In 2019, Shen et al. [24] trained a residual network on near duplicate patterns of the Brueghel data set [26], extracted with an original variation of traditional computer vision methods. These duplicate patterns are similar in nature to the distinctive features in the iconography of the saints in the ArtDL data. As mentioned in Section 1.1 Milani et al. [16] provide the state-of-the-art for iconography specific deep learning. They do this with a fully convolutional ResNet50 [11] pretrained on the ImageNet data set and fine-tuned on the ArtDL data set. Furthermore, they investigate the results through the means of *class activation mapping* [33] which shows that the classifier is specifically activated by visual features important to the iconography of the artwork tag, e.g. the arrows in a painting depicting Saint Sebastian. This is the work most closely related to that of this thesis.

2 Methods

The main methods consists of two convolutional neural network models, therefore this section will first give a theoretical foundation of this type of model and afterwards iterate upon the more specific methods. Further information on the architecture of these models, approach to model training and the chosen method from explainable AI, class activation mapping, will be given in 3.

2.1 Artificial Neural Networks

An *artificial neural network* consists of a circuit of neurons that have learnable weights. Each neuron has an input and generally performs a dot product and is followed by a non-linearity typically in the form of an activation function. One such network is the simple feedforward network which is structured as an input layer, several or no hidden layers and an output layer. See Figure 1. Layers where every neuron has weights connected to each neuron in the following layer is called a *densely connected* layer.

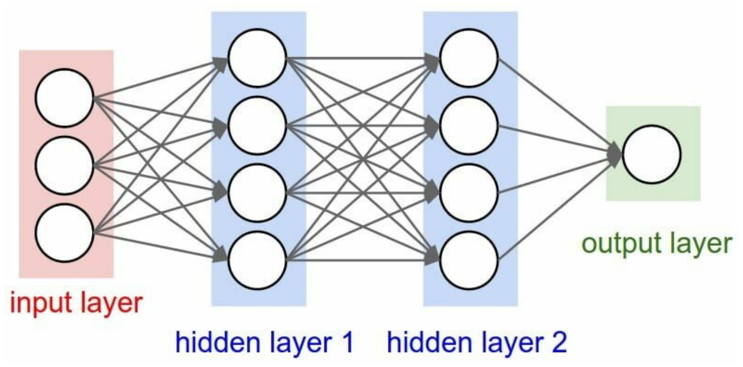


Figure 1: A regular 2-layer feedforward neural network where all layers are densely connected. [6]

2.1.1 Forward pass

Getting the output of a neural network is known as the *forward pass*. In this pass, the dot product of neurons and weights are calculated sequentially from the input, through the hidden layers, to the output. If an activation function is present in a layer, it is calculated in between the respective neuron dot products. In an input with multiple features and layers with several neurons, these dot products are calculated as matrix multiplications. An example of the forward pass of a single layered, densely connected network can be seen in Equation 1.

$$y = f(X \cdot W_1) \cdot W_2 \tag{1}$$

Where y is the output, X is a single sample input with $X_1 ... X_n$ as features and W_i are the weight matrices of the layers. Here f denotes an activation function. In this case we have two weight matrices, one between the input and the hidden layer, and one between the hidden layer and the output. In many cases where a linear value is desired as output, no activation function is applied to the last matrix multiplication with the output weights. However, in classification type problems the output is often transformed with an activation function to produce values between 0 and 1 to denote probabilities the input belongs to that class.

2.1.2 Backward pass

To update the weights and thus train the neural network, an algorithm known as backpropagation is applied in what is called the *backward pass*. Backpropagation computes the *gradient* of the weights with respect to a *loss function*, by finding the derivative (dL/dW) and updates the weights by subtracting/adding the gradients. Here L is the loss function and W are the weights. This derivative is also called the *gradient*. For each layer the derivative and gradient is calculated and the weights of the layer are updated. Which loss function is used is dependent on the given task. For a classification task with multiple classes typically *cross entropy loss*. Both convolutional models in this thesis employ cross entropy looss, see Equation 2.

$$loss = -\sum_{c=1}^{M} y_{i,c} * log(p_{i,c}) \tag{2}$$

Here M is the number of classes, e.g. Mary, Jesus or Noah. log is the natural logarithm. y is a binary indicator of whether the class label, c, is the correct classification for sample i. p is the prediction or predicted probability that sample i is of class c.

The process of updating the weights by adding/subtracting the gradient to the weights is called *gradient descent* and is used for optimising neural networks. The models in this thesis utilize *stochastic gradient descent* (SGD), see Equation 3.

$$W = W_{old} - \eta \frac{dL}{dW} \tag{3}$$

In this equation, η denotes the learning rate, which helps converge to a local minimum in the gradient descent. A good visualisation of gradient descent is that of a landscape made up of the values of the loss function. Our current loss is that of a ball starting on a hill and gradient descent helps us move the ball to a valley of minimum loss, see Figure 2. The step size is computed by the multiplication between the learning rate and the derivative.

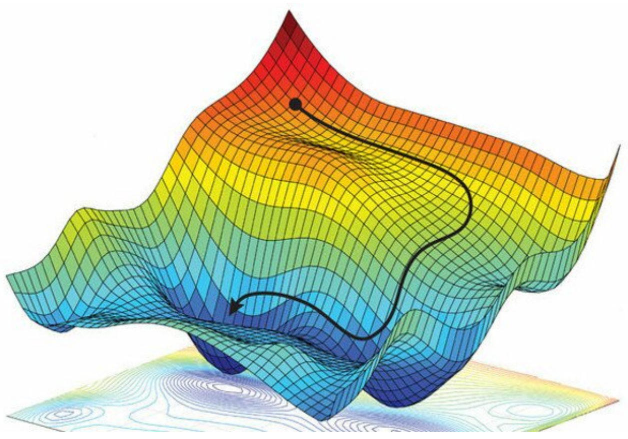


Figure 2: A landscape made up of loss values. The movement of the ball denotes gradient descent. [1]

2.1.3 Activation functions

As mentioned in Section 2.1.1 the output after a multiplication with a weight matrix is oftentimes transformed with an activation function. The activation function introduces non-linearity which in turn introduces a higher degree of complexity and is desirable when attempting to find non-linear patterns in data. The activation function also makes the layer differentiable, which is important in the use of gradient descent, see Equation 3. Furthermore, some activation functions restrict values within a given range. This saves computational expense and helps prevent *vanishing gradients*, a problem where gradients become negligibly small in the backpropagation algorithm from Section 2.1.2. The models of this thesis use *rectified linear unit* (ReLU) as activation function for the hidden layers. ReLU is written:

$$f(x) = x^+ = max(0, x) \tag{4}$$

For the output, the models use the softmax function to restrict the predictions within the range $(0, 1)$ and sum to 1. Softmax is applied during training and the output is used for measuring model *accuracy*. The function is written in Equation 5. Here x_i is an n-dimensional model output and x_j is a value in x_i.

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{5}$$

2.2 Convolutional Models

A *convolutional neural network* (CNN) shares most of the methodology with that of the regular densely connected, feedforward neural networks, described in Section 2.1.1. However, it sets itself apart in several aspects, particularly in that it utilizes convolutions in convolutional layers. Would one attempt to build a densely connected network for image classification with images of 500x500 pixels in RGB space, the number of input variables per sample would be $500*500*3 = 750000$. This is an extreme feature space generated from an image with a relatively low pixel count. Instead of attempting to learn an extreme amount of weights for each neuron, the convolutional layer instead sequentially highlights regions of the image and convolves it with the *filters* of the layer. The weights to the next layer is connected to these filters rather than all pixels in the image. If the filters of the layer is of 7x7x3, then rather than having 750000 weights per neuron like a densely connected layer, the size would be $7 * 7 * 3 = 147$ for a single filter. Modern convolutional network frameworks allow many filters to be applied and trained seamlessly in parallel.

2.2.1 The Convolutional Layer

The key layer of the CNN is the convolutional layer which convolves regions of an image with the trainable weights of the filters in the layer. The equation of a 2D convolution between an image and a filter, can be written:

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h[m,n] \cdot x[i-m, j-n] \tag{6}$$

Here x represents the input image to be convolved with filter h giving image y. Note that the indices i and j are flipped in the input image. If the filter is of size 3x3 then the range of i and j is (-1, 1). In traditional computer vision, filters were handcrafted by the engineer, this is however not the case in modern methods where the filter is a set of trainable weights, updated through the means of backpropagation. See Figure 3 for an intuitive depiction of how a filter is convolved with an image.

Through convolutions with the filters, the convolutional layer is capable of extracting both primitive and abstract features, depending on the position of the layer in the network. Convolving an image with the *Sobel filter* is an example of extracting such primitive features through the means of convolution. The Sobel filter is a combination of two 3x3 filters G_x and G_y, see Table 1.

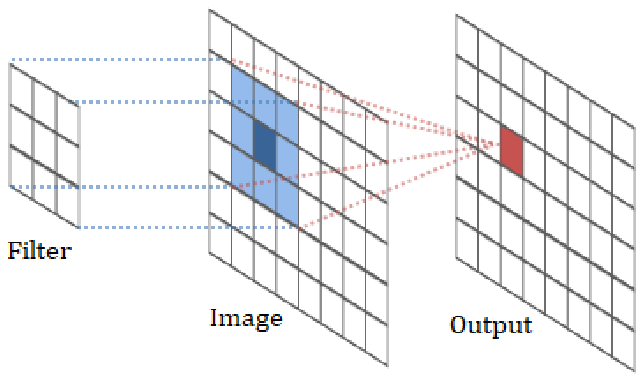


Figure 3: An image being convolved with a filter. Note that the coordinates have already been flipped. [12]

Convolving an image with filters G_x and G_y calculates an approximation of the vertical and horizontal derivative, respectively. Combining the two convolutions with $G = \sqrt{G_x^2 + G_y^2}$ gives us the Sobel filter, which in turn represents an approximation of the gradient of the image intensity, thus it can be utilized as an image edge detector, see Figure 4 for an example of the output of a convolution with these filters.

Gx

-1	0	1
-2	0	2
-1	0	1

Gy

1	2	1
0	0	0
-1	-2	-1

Table 1: The two 3x3 filters used in the Sobel filter.

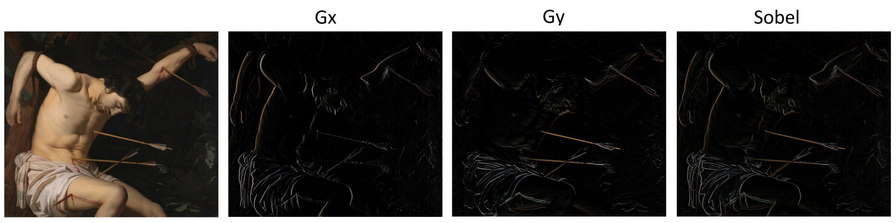


Figure 4: An image of Saint Sebastian being convolved with the Sobel filter.

2.2.2 Pooling and Batch Normalization

As convolutional layers extract features, they in essence summarize an image as a feature map. However, such feature maps are sensitive to the location of the features in an image. One method decreasing this sensitivity and thus increasing the degree of *local translation invariance* is the addition of *pooling layers*. A pooling layer performs a pooling operation over windows of a given size in the feature map. A widely used layer of this type is *max pooling* [19], which is also used in the models of this thesis. Max pooling creates a down sampling of the feature map by extracting the maximum value in windows over the map. The most common usage involves a window size of 2x2, effectively halving the size of the feature map, see Figure 5. This kind of down sampling keeps the larger, important structural elements while shedding finer details, which may not be particularly important to the given task. With the introduction of translation invariance, the model is thus less sensitive to features in an image undergoing transformations such as cropping, shifting and rotation.

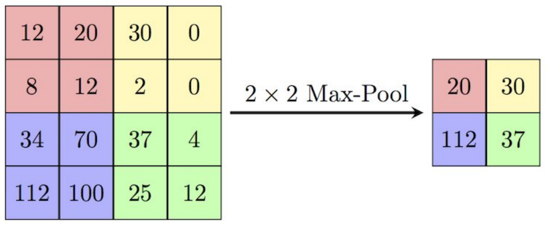


Figure 5: Max pooling from 4x4 to 2x2 with a stride of 2. [7]

A *global average pooling layer* (GAP) [15] is a pooling layer not unlike the max pooling layer. However, instead of performing the max filter in windows of the feature map, an entire feature map is summarized as its average value, see Figure 6. This layer is not only useful in the context of CAMs, but like max pooling, introduces translation invariance. Note that pooling layers are applied after the introduction of non-linearity, see Section 2.1.3.

As mentioned in Section 2.1.1, the network input is fed from layer to layer until the output is reached. As the weights of each layer is continuously updated through the training procedure of gradient descent and backpropagation (Section 2.1.2), the inputs to layers following the input layer are changed in what is described as *internal covariate shift*. This phenomenon makes it harder and slower to train deeper networks of many layers. To address this problem

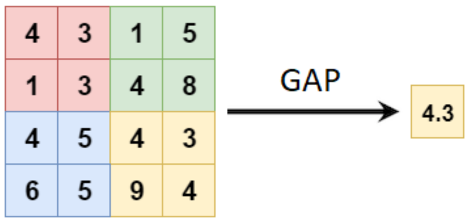


Figure 6: Global average pooling from 4x4 to 1x1. [28]

a trainable layer known as *batch normalization* [13] is often utilized. Batch normalization is applied to the output of an activation function (ReLU) and is generally expressed through three steps, see Equation 7.

$$z = \left(\frac{x^{+} * \mu}{\sigma^2}\right) * \gamma + \beta \tag{7}$$

First the output is normalized according to the mean, μ and standard deviation σ^2. Secondly, it is multiplied with an arbitrary variable, γ. Finally, another arbitrary variable, β, is added to the output. These 4 parameters are all trainable, meaning they too will be optimized in the backward pass of the network. The batch normalization process occurs per batch of data. Because the parameters are included in the gradient process of backpropagation, it prevents outlying large weights from overinfluencing the training process. It is also important to note that some literature suggests that batch normalization does not deal with internal covariate shift and instead is effective because it makes the training landscape smoother (See Figure 2), which induces predictability and stability of the gradients [22]. Batch normalization also ensures increased stability when higher learning rates are introduced in SGD, see Section 2.1.2.

2.2.3 Depth and Residual Blocks

As previously mentioned in Section 2.2.1 the depth of a convolutional neural network is important to the type of features extracted by a convolutional layer. Layers placed early in the network extract primitive, low-level features, such as lines and edges. Layers in the middle of a network typically extract features such as textures or patterns. Deeper layers of the network extract high-level features such as parts of objects or objects in the image. If a network is too shallow it is not able to extract and thus learn from higher level features. However, constructing a deep neural network poses its own caveats. The deeper a network is constructed, the more difficult it becomes training due to the vanishing gradient problem, described in section 2.1.3. In addition to rectified linear units for mitigation of vanishing gradients, an extremely important milestone in computer vision and deep learning was the introduction of the deep residual network in 2015 by He et al [11]. This type of network utilizes what is known as residual blocks (see Figure 7) and achieved a state-of-the-art results on ImageNet in 2016 with deep convolutional models of up to 152 layers deep.

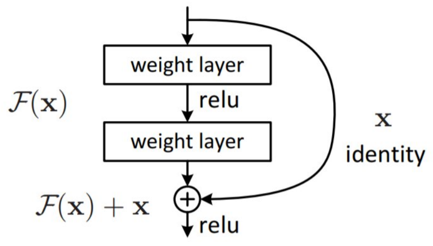

weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 7: The residual block introduced by He et al.

The residual block utilizes outputs from the previous layer and the layer before that. This operation can be formulated as Equation 8.

$$H(x) = f(F(x) + x) \tag{8}$$

Where f is the ReLU activation function. Given two layers, Layer 1 and Layer 2, x is then the output of the layer previous to Layer 1. $F(x)$ is the result of x going through Layer 1, ReLU and then Layer 2. The procedure is then: x is added to $F(x)$ in what is known as the *identity connection* and ReLU is applied. Although deceivingly simple, this method is powerful and strongly mitigates the vanishing gradient problem and allows for training of much deeper neural networks, especially in conjunction with other methods such as ReLU and batch normalization.

3 Approach

This section describes the outline of the experimental setup. First an overview of the data is presented. Secondly, an elaboration of the data pre-processing will be given. Thirdly, the architecture of the models and the application of transfer learning will be explained. Lastly, the approach to explanation of model outputs will be introduced.

3.1 Data

The data used in this thesis is a subset of the ArtDL paintings data set presented by Milani et al. [16] The ArtDL data set is a collection of 42.479 paintings depicting Christian figures, collected with the means of Iconclass. The large majority of the images date back to the Renaissance of Europe. In the data set a myriad of various art styles and mediums can be found, for instance murals, frescoes, canvas paintings and polyptychs. The only constraint is that the artwork is a painting and not a three-dimensional artwork, i.e sculptures. Roughly 60% of the paintings are in colour and the remaining 40% in grayscale. The subset of the data set used in this thesis consists of paintings depicting 10 classes in the form of 10 saints from Christianity, see Figure 8. The 10 saints and thus annotated classes are: **Virgin Mary, Anthony of Padua, Saint Dominic, Francis of Assisi, Saint Jerome, John the Baptist, Paul the Apostle, Saint Peter, Saint Sebastian and Mary Magdalene.** In each painting at least one of these saints are depicted, but may contain multiple. To avoid complicating the task, if a painting depicts several saints, only a single saint is chosen as the annotation for the painting. The annotations are labels denoting the presence of a given saint in the painting, but not a pixel-wise annotation such as bounding boxes or other methods denoting the spatial location of the saint in the painting.

3.2 Pre-processing

The data is prepared in three manners. First the painting is colour normalized in each channel (RGB) by subtracting each channel by the mean of the channel in the data set and dividing the result with the standard deviation of the channel in the data set. This effectively standardizes the data to have a mean of 0 and a standard deviation of 1 and generally helps training. The means and standard deviations used are that of ImageNet, due to the utilization of layers pretrained on ImageNet, see Section 3.7. The paintings are then padded to a square with a constant value. Finally, they are resized to a fixed square size of 224x244. The normalization step is a commonly employed method to center the input data, which helps constraint gradients in gradient descent, see Section 2.1.2. Padding

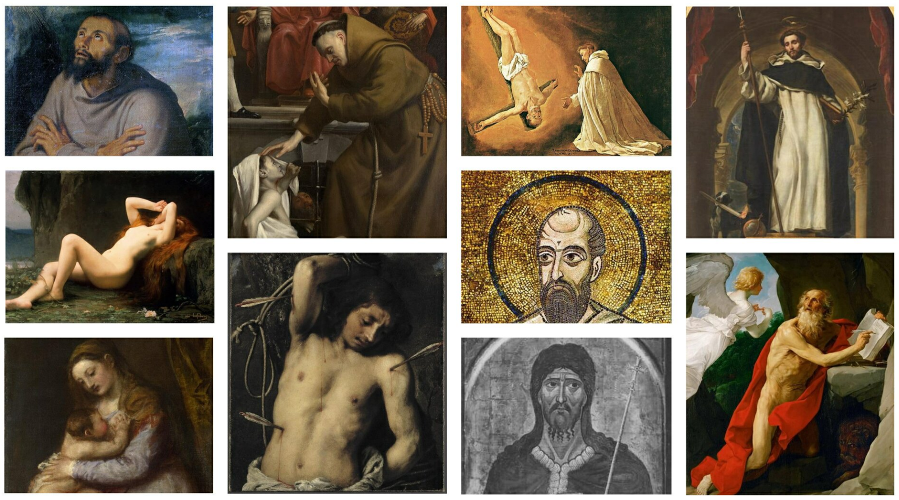


Figure 8: Example images of the ArtDL data set, each is a different saint.

the painting to a square prevents warping when the image is resized and has negligible effects on training. Resizing is important for fitting an image to the dimensions of the input layer of a neural network. In transfer learning (Section 3.7) this is important for properly utilizing potentially frozen layers early in the network. Resizing also saves computational costs and reduces training time, the latter being an especially attractive feature for the many-featured nature of image data. In Figure 9 the pre-processing is visualised.

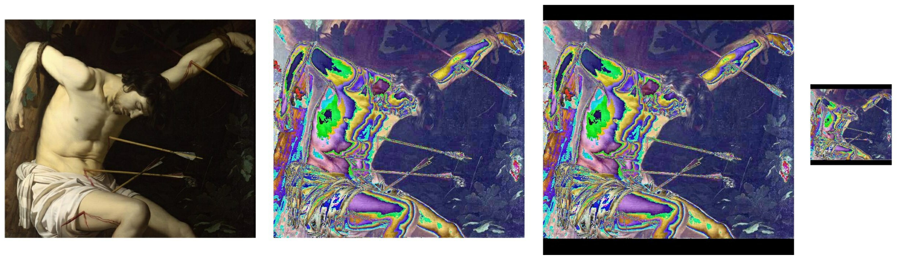


Figure 9: A painting undergoing normalization, square padding and resizing.

3.3 Training Strategies

A good practice is to split the data set in training, validation and test sets. This action is also performed on the data set used for this thesis. In Table 2 the distribution of images in the sets are shown.

Sets	Mary	Ant	Dom	Fran	Jer	John	Paul	Pet	Seb	Mag
Train	9513	115	234	784	939	783	419	949	448	727
Val	1189	14	30	98	117	97	52	118	56	90
Test	1189	14	29	98	118	99	52	119	56	90

Table 2: Data distribution in classes, by training, validation and test sets.

Observe that Virgin Mary has an overwhelming amount of samples compared to that of the other classes in the data set. Ca. 64% of the sets consist of paintings of Virgin Mary. Such an extreme imbalance in class distribution might easily lead to model overfitting on the overrepresented class. To address this problem, three separate strategies were employed. The first strategy involves expanding the data set in such a manner that each class contains an equal amount of paintings, matching the size of the largest class set, Virgin Mary. This means the paintings in each class, aside from Virgin Mary, are duplicated at random until the class set contains 9513 images. This strategy was employed by Milani et al. [16] to train the model they provide and was the most successful strategy for their model. The second strategy does not involve expansion of data, rather it includes *random erasing* [32], a data augmentation technique which at random, given a probability, covers a part of the painting with a rectangle given a random dimension and scale within a range, see Figure 10.

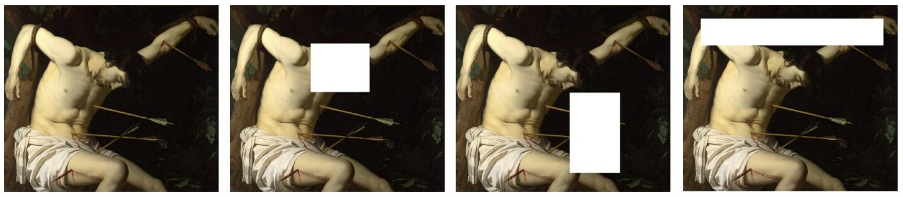


Figure 10: Random erasing data augmentation.

This lowers the risk of the model overfitting on a class as well as a single predominant feature in an image, see Section 3.6 for an example of such a case. Furthermore, the second strategy also involves the addition of weights to the loss function of the network. The loss of each class is divided by a given weight and the results are summed. The weights used in this case is the amount of paintings in the Virgin Mary class set divided by the amount in each other class set, respectively. See Table 3 for the weights. The third strategy employs the methods of strategy 1 and random erasing from strategy 2.

Mary	Ant	Dom	Fran	Jer	John	Paul	Pet	Seb	Mag
1	83	41	12	10	12	23	10	21	13

Table 3: Loss function weights (rounded) per class introduced to cross entropy loss.

For the first and third training strategies an SGD learning rate of, $\eta = 0.01$, is used in contrast to the lower $\eta = 0.001$ used in the second strategy. As mentioned in Section 2.2.2 batch normalization allows for higher learning rates in SGD. This effect is therefore exploited to make sure the model properly fits to the training data. This is because expansion of the data also significantly increases the duration of every pass through the data. The second strategy handles less data and the duration of a single pass is significantly smaller, so it does not need to exploit this effect.

3.4 VGG-16

In 2012, the first convolutional neural network to win the *ImageNet Challenge* [21] was the so called *AlexNet* [14]. With a benchmark result of a top-5 error of 15.3%, which was 10.8% lower than the runner up method. The main results of the paper behind the model displayed the importance of depth, see Section 2.2.3. AlexNet was an 8 layer model employing convolutional layers, different size filters (Section 2.2.1), max pooling (Section 2.2.2) and was trained on a GPU. In 2014 a deeper model of 16 layers, *VGG-16* [25], was introduced. VGG-16 was based upon the architecture of AlexNet. However, it did not use varying sizes of filters and instead constrained it to filters of size 3x3. VGG is a model easily implemented relative to more modern models, for instance the residual network, see Section 3.5. Each layer is only connected to the previous and the next, without detours. This means the forward pass (Section 2.1.1) and backward pass (Section 2.1.2) are also simple and straightforward to implement. VGG-16 achieved a new benchmark of a top-5 error of 9.62% in the ImageNet Challenge. See Figure 11 for an overview of the version of VGG-16 presented in the original paper.

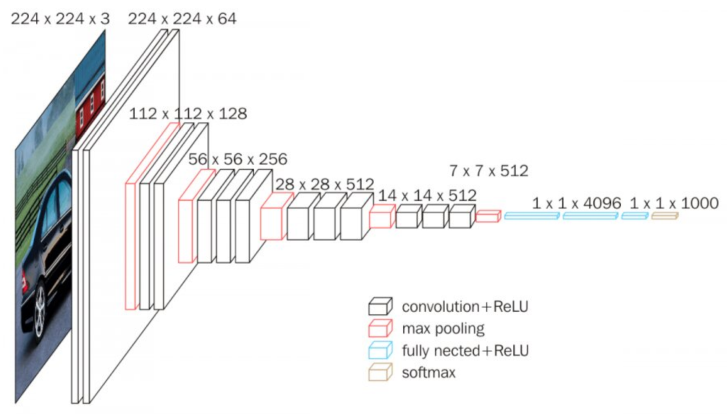


Figure 11: Architecture of VGG-16 presented in the original paper. [17]

The original version of VGG-16 is presented in 5 convolutional blocks as feature extractors and a classifier in the form of the final 3 densely connected layers (Section 2.1). Aside from the first convolutional block, each block begins with a max pooling layer mapping dimensions of the feature maps from the previous block to the current block. The architecture of the original VGG-16 is altered in a few ways to suite the needs of this thesis. The final classifier block consisting of the 3 densely connected layers is replaced by a large convolutional layer with an output size of 10, for the 10 classes in the data set. This effectively makes the modified VGG-16 a fully convolutional model. The dimensions of the layers can be seen in Figure 12.

Densely connected layers are interchangeable with convolutional layers if the dimension are fitted correctly. Densely connected layers are prone to overfitting, especially as the size increases. As described in Section 2.2 the amount of weights in densely connected layers are at the extreme compared to convolutional layers. Removing such a large amount of weights not only saves memory, but also speeds up the training procedure. Furthermore, the convolutional layer also makes it easy to implement class activation mapping, see Section 3.6. These reasons are the bases of the choice of changing the layer for the modified VGG-16. The final convolutional layer in the fifth block is also modified to fit the added convolutional layer which serves as the classifier of the network. Between the fifth block and the classifier a GAP layer is added, see Section 2.2.2. This layer helps prevent overfitting and is a crucial layer in the implementation of class

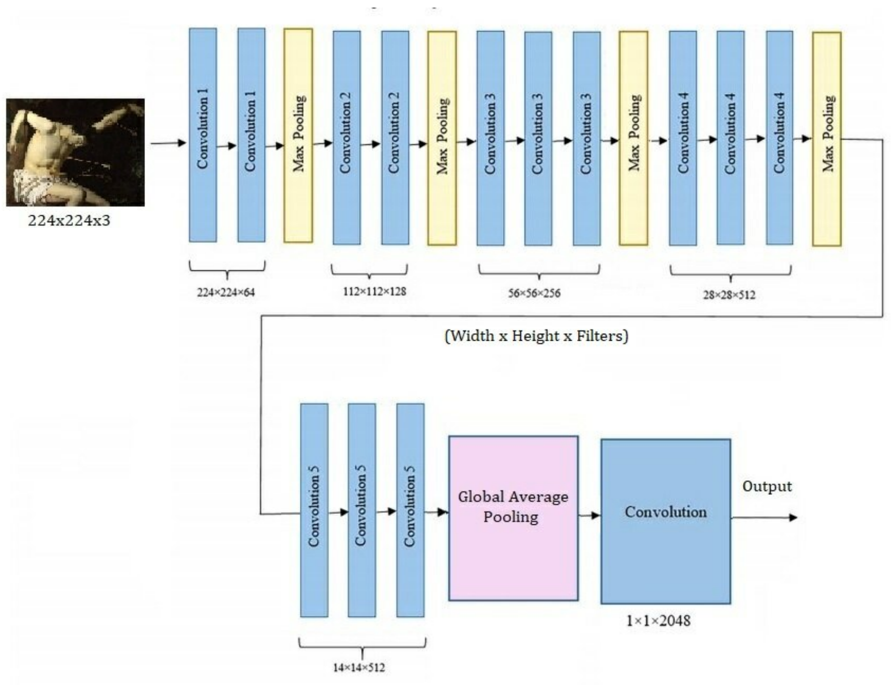


Figure 12: Representation of the fully convolutional VGG-16. Batch normalization is not shown.

activation mapping. The final modification to the network is the implementation of batch normalization layers after each convolutional layer, to allow for easier training and mitigate overfitting, see Section 2.2.2. The final architecture sans batch normalization can be seen in Figure 12.

3.5 Residual Network

The convolutional model employed, modified and fine-tuned by Milani et al [16] is a 50 layer deep residual network (ResNet50), see Section 2.2.3. Like the fully convolutional VGG-16 (Section 3.4) Milani et al. [16] has modified ResNet50 to be fully convolutional and contains a GAP layer before the final classifier layer. ResNet50 is structured in 4 stages where convolutional layers of a stage share the same dimensions. This model also utilizes max pooling layers after each stage, and batch normalization after each convolutional layer. Layers in a stage are all connected as residual blocks with the signature identical connection of the residual block. Before beginning the 4 stages of residual blocks, a convolutional layer and a max pooling layer follows the input layer, which is then connected to the first stage of residual blocks. A visualization of the architecture can be seen in Figure 13

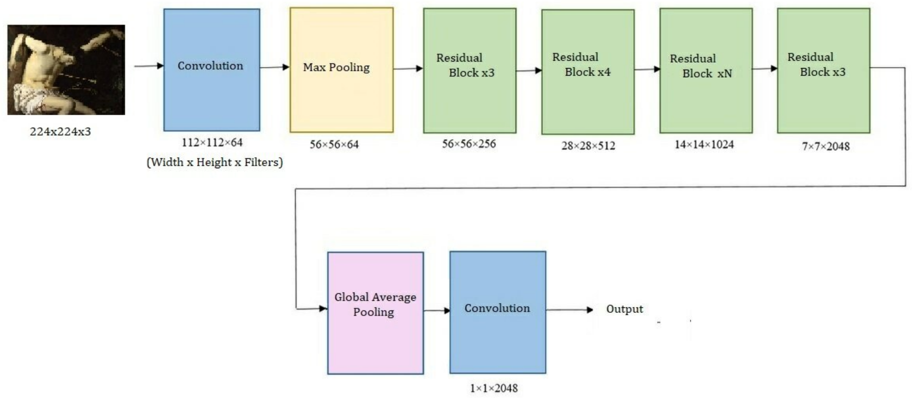


Figure 13: Representation of the ResNet50 architecture used by Milani et al. [16] Note that N, in the third stage denotes N amount of residual blocks. In ResNet50, N = 6.

3.6 Class Activation Mapping

An unfortunate side-effect of the many parameters in a neural network is the loss of inference transparency. By not being able to diagnose the basis of a decision made by the neural network, it is difficult to assess whether or not the network has a wrong basis of inference. An example is if a model has been trained on images of Jesus Christ. Jesus is quite often depicted as crucified. In this case the model may activate based on only the cross and not Jesus. This raises the issue of whether the model is capable of recognising Jesus outside the context of crucifixion. Class activation mapping (CAM) is a method of identifying and inspecting such bases of inference of a model, in the form of an activation heat map. This heat map can be overlapped with the input image for a human decipherable visualisation of input regions that are important for the prediction. To produce CAMs a GAP layer is added to the network before the final output layer and its activation. The weights to the classes of the final layer in the model is then mapped back to feature maps of the final convolutional layer. The output produces a 14x14 grayscale heat map. This heat map is then upscaled to the same width and height of the input image (painting) and overlapped. This method of implementing CAM to a model is based on the Grad-CAM paper [23] and is used in the modified VGG-16 convolutional model constructed in this thesis, see Section 3.4. See the title page and Section 4 for examples of the CAMs produced by the models of this thesis.

3.7 Transfer Learning

Transfer learning is loosely based on transfer learning in psychological literature. In deep learning the embodiment of transfer learning is a methodology involving using the layer weights of a pretrained model in a modified version of the model or an entirely newly constructed model. After loading the weights in the new model, there are generally three ways of proceeding. First approach is using the pretrained layers for feature extraction. In this case the final layer of the network is modified to fit the dimensions of the desired number of output classes. The pretrained layers are then locked in a frozen state where no updates are made to the weights through SGD, except the modified final layer, see Section 2.1.2. The final layer is then fitted to the rest of the model and the new data by running the training procedure over the data set. This is a successful approach in data that is similar to the data of the task at hand. The second approach involves freezing only a part of the pretrained layers and running the training procedure again, like in the first approach. The third approach uses the weights of the pretrained layers as initialization of the weights in the new model, as opposed to random initialization. Each approach may be combined in whatever way is desired for the task. Milani et al. [16] ran experiments which showed freezing only the earlier layers of the ResNet50 model yielded the greatest results. This is because the model is pretrained on ImageNet, which contains real world image data and the ArtDL data set is an extreme domain shift in the form of painted art. The VGG-16 model constructed in this thesis employs two freezing strategies. First strategy is freezing only the layers within the first and half of the second convolutional blocks. The first block is responsible for primitive, low-level features such as lines and edges, see Sections 2.2.1 and 2.2.3 and the second block is responsible for mid-level features, such as textures. Second strategy is identical to the first, but freezes the entirety of the second convolutional block. The second block extracts low-level to mid-level features. Figure 14 shows an example of features extracted by a VGG-16 model. Training strategy 2 from Section 3.3 uses freezing strategy 1 and training strategies 1 and 3 use freezing strategy 2.

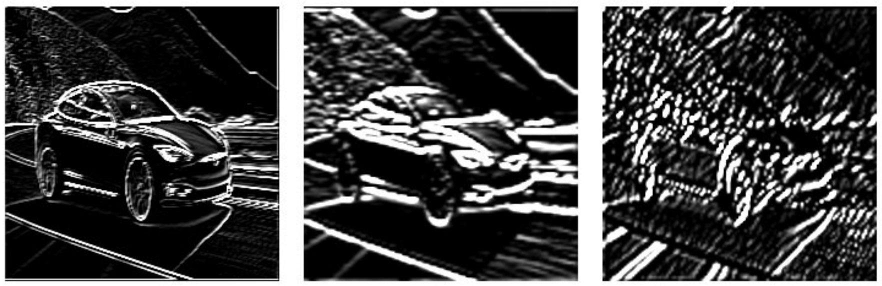


Figure 14: Features extracted from an image of a car by the layers of a trained VGG model. Left: Low-level features from early layers - lines and edges. Middle: Mid-level features from layers in the middle - textures and shapes. Right: High-level features from deeper layers - door, window and wheels.

4 Results

In this section an overview of the machine used for training the models will first be given. The results of the training and validation will then be inspected and evaluated. Finally, an evaluation and comparison of the trained models will be based upon confusion matrices and metrics thereof. Further evaluation will be based on a comparison of the CAMs of the model outputs and the state-of-the-art provided by Milani et al. [16]

4.1 Setup

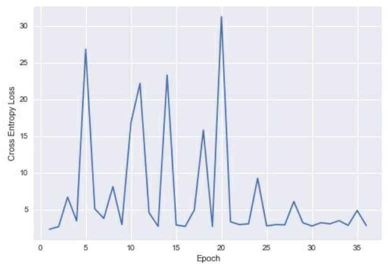
All experiments were run on a Windows 10 (64-bit) PC containing 16GB of RAM, a 6-core AMD Ryzen 5 3600 CPU and a 8GB NVidia GeForce RTX 3070 GPU. All code was written in Python [27]. The deep learning framework PyTorch [18] was used for all experiments and was installed to run on a GPU with NVidia's CUDA and cuDNN. Table 4 is an overview of the version of Python and the essential libraries used.

4.2 Training and Validation

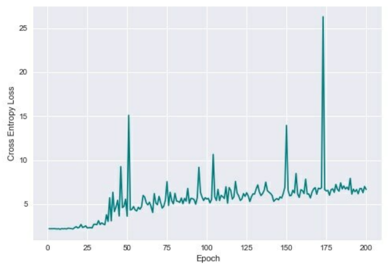
In all experiments an input data batch size was set to a fixed 8 samples and loaded by 3 parallel workers. Each training experiment of the fully convolutional VGG-16 model were allowed ca. 14 hours of run time. For training strategy 1 (Section 3.3) 36 *epochs* (pass through data) of the data were reached at the end

Name	Version
Python	3.9.1
CUDA	11.0
cuDNN	8.0.5
Torch	1.7.1+cu110
Torchvision	0.8.2+cu110
Numpy [10]	1.19.5
Pandas [29]	1.2

Table 4: Software versions



(a)

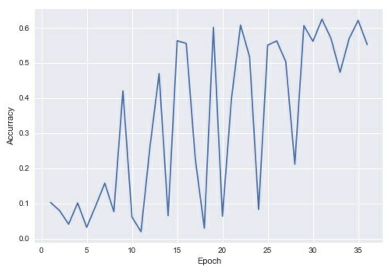


(b)

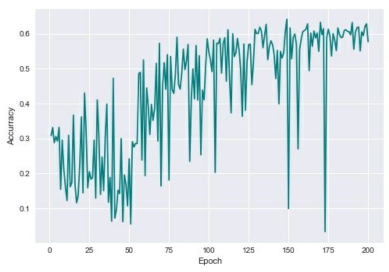
Figure 15: Validation loss per epoch in strategy 1 (a) and strategy 2 (b).

of training, while strategy 2 was significantly faster and reached 200 epochs. After each epoch of training, the models were evaluated on the validation set (Section 3.3) in terms of cross entropy loss and accuracy. In Figure 15 the validation loss per epoch of strategy 1 and 2 are displayed. Note that PyTorch also applies Softmax (Section 2.1.3) to the outputs when calculating the cross entropy loss, resulting in only positive loss values. The ability of the models to generalise to unseen data is prioritised, thus rather than inspecting training loss and accuracy, a focus is made on the validation metrics.

In strategy 1 the validation loss becomes increasingly stable after epoch 20 of 36 and in strategy the loss becomes more stable after an extreme spike at epoch 173. The loss of strategy 2 is relatively stable between epochs 51 and 151 with a few larger spikes. Spiking in loss value is due to the use of batches while training. The batches are a small subset of the data and are chosen at random while loading it, however, all data samples have been loaded without duplicates at the end of an epoch. The minimum loss of the validation phases



(a)

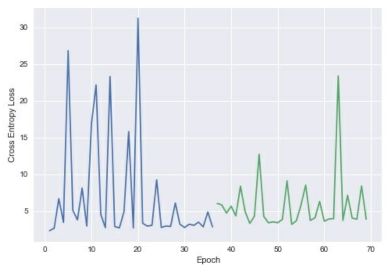


(b)

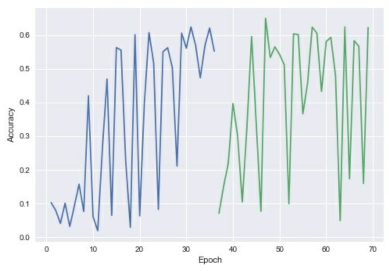
Figure 16: Validation accuracy per epoch in strategy 1 (a) and strategy 2 (b).

is not considered during the backward pass (Section 2.1.2) of the model and is thus only for evaluation post training. Should the loss be extremely high it would indicate overfitting on the training data. This is not the case in either strategy. Note that validation loss is not inversely correlated to training loss, we see this as the validation loss increases per epoch.

Figure 16 depicts the validation accuracies of strategies 1 and 2. Both strategies display an increase in validation accuracy during training. Like the validation losses, the accuracies appear relatively volatile and is again due to the usage of batches during training. The highest accuracy of strategy 1 is found at epoch 31 and for strategy 2 at epoch 149. The second strategy appears to be close to converging to a narrower range of accuracy, whereas strategy 2 does not seem to display this behaviour as of yet. As a model gets increasingly more confident in its prediction, the Softmax transformed output (2.1.3) values, between 0 and 1, will increase towards 1 for the classes with highest activation and fall towards 0 for the rest. An uncertain model might have a correct prediction of [0.6, 0.4] while a confident model predict may wrongly predict [0.1, 0.9]. The loss value in the second case is much higher than the first. This can be observed by comparing the strategy 2 curves of Figures 15 and 16, where both the validation accuracy and loss of the model increases. The minimum loss and maximum accuracy is displayed in Table 5. After inspecting the results of strategy 1 and 2, a third strategy was conceptualized. Strategy 3 uses the weights of strategy 1 as initialization, see Section 3.7. It is therefore an attempt at continuing the training of the strategy 1 model. This experiment too had a run time of ca. 14 hours.



(a)



(b)

Figure 17: Validation loss (a) and accuracy (b) per epoch in strategy 3, displayed as a continuation of strategy 1.

	Strategy 1	Strategy 2	Strategy 3
Max Validation Accuracy	0.624	0.642	0.650

Table 5: Validation accuracy of the 3 different training strategies (VGG-16).

In Figure 17 the validation loss and accuracy may be observed. The loss appears less volatile in strategy 3 compared to strategy 1. Not only is it more stable, but the maximum spiked value is much less than that of 2 out of 5 spikes in strategy 1. The training procedure has thus become more stable even though the learning rates, $\eta = 0.01$ are the same. This suggests that strategy 3 is indeed a continued training of the model in strategy 1. The loss curve appears to move closer towards convergence, but remains less stable than that of strategy 2. The maximum accuracy of the strategy 3 model is found at epoch 11 of 33 (47 of 69 in continuation of strategy 1). It appears the model may be trained further for better accuracy with strategy 3, although at the risk of overfitting. Observe in 5 that strategy 3 yielded the highest validation accuracy. For the final model selection, the models with the highest validation accuracy during training was chosen for each strategy, resulting in 3 trained VGG-16 models.

4.3 Test Evaluation

To properly evaluate the trained VGG-16 models, the models were applied to the test set, then a confusion matrix for each class was computed. From the confusion matrices the *precision*, *recall* and *F1 score* were also computed. For a benchmark comparison, the ResNet50 was also applied on the same machine

Class	Precision	Recall	F1-score
Mary	0.92	0.91	0.92
Anthony	0.64	0.50	0.56
Dominic	0.54	0.76	0.63
Francis	0.68	0.80	0.74
Jerome	0.79	0.79	0.79
John	0.67	0.83	0.74
Paul	0.56	0.37	0.44
Peter	0.73	0.76	0.75
Sebastian	0.96	0.77	0.75
Magdalene	0.84	0.73	0.78
Macro Avg	0.73	0.72	0.72

Table 6: Precision, recall and f1 score of the ResNet50 from Milani et al [16] applied to the test set on the machine from Section 4.1.

	Sebastian	Jerome	John	Anthony
Original Precision	0.91	0.71	0.58	0.73
Reapplication Precision	0.96	0.79	0.67	0.64
Original Recall	0.73	0.78	0.80	0.57
Reapplication Recall	0.77	0.79	0.83	0.50

Table 7: Significant differences between the reapplication results and original results of the ResNet50 model from Milani et al. [16]

as described in Section 4.1. The evaluation of the model was slightly different when applied on this machine. With slightly better results on some classes and slightly worse on others. This can possibly be attributed to machine setup and method of normalization. These results are displayed in Table 6. The most significant differences in the reapplication results versus the reported results from the paper of Milani et al [16] can be seen in Table 7. Here we see an improvement on Saint Sebastian, Saint Jerome and John the Baptist, but a setback on Anthony of Padua. Furthermore, the reapplication had a macro average precision of 0.73 while the original paper reported 0.7117.

The precision, recall and F1-scores of the trained VGG-16 models are presented in Tables 8, 9 and 10. Similar traits of the three models include a macro average recall of 25 and a high recall for the Virgin Mary class. The precision for Mary is also high, but roughly 20 percentage points lower than the recall.

Class	Precision	Recall	F1-score
Mary	0.71	0.88	0.79
Anthony	0.33	0.07	0.12
Dominic	0.32	0.21	0.25
Francis	0.17	0.14	0.16
Jerome	0.47	0.32	0.38
John	0.47	0.08	0.14
Paul	0.10	0.06	0.07
Peter	0.32	0.19	0.24
Sebastian	0.41	0.21	0.28
Magdalene	0.47	0.37	0.41
Macro Avg	0.38	0.25	0.28

Table 8: Precision, Recall and F1-score for the model of strategy 1

Strategy 1 had a macro precision of 38 percentage points, strategy 2 had 48 and strategy 3 had 61. The test set is imbalanced to the same degree as the training and validation sets. It can be observed that as the macro average precision of the models increase so does the recall value of class Virgin Mary, while the precision of Virgin Mary is slightly lower in strategy 2 and 3 than in strategy 1. The migration of the class recalls towards Virgin Mary in strategy 2 and 3, and the lower precision values of said class, suggests the models tend to predict Virgin Mary incorrectly, but becomes more certain in other classes. We can observe this in the fact that strategy 2 and 3 have a higher average macro precision than strategy 1. Furthermore, in relation to the other classes, strategy 1 exhibits prominence in the Saint Jerome, John the Baptiste, Saint Sebastian and Mary Magdalene classes. For strategy 2 this is true not for John the Baptiste, but the same classes listed for strategy 1, with the addition of Anthony of Padua. Strategy 3 shows the same for the dominant classes in strategy 2 and 3, but also Saint Peter. Strategy 3 predicts with a relatively high precision but tends to predominantly predict the Virgin Mary class, and the same can be said for strategy 2. Strategy 1 is imprecise, but predicts other classes than Virgin Mary more often than strategy 2 and 3.

4.4 CAM Inspection

For every image in the test set, the trained VGG-16 models and the ResNet50 model of Milani et al. [16] were applied and the CAMs were extracted from

Class	Precision	Recall	F1-score
Mary	0.69	0.92	0.79
Anthony	0.75	0.21	0.33
Dominic	0.33	0.14	0.20
Francis	0.26	0..08	0.12
Jerome	0.64	0.18	0.28
John	0.28	0.15	0.20
Paul	0.36	0.08	0.13
Peter	0.48	0.21	0.29
Sebastian	0.45	0.25	0.32
Magdalene	0.52	0.27	0.35
Macro Avg	0.48	0.25	0.30

Table 9: Precision, Recall and F1-score for the model of strategy 2

Class	Precision	Recall	F1-score
Mary	0.69	0.94	0.80
Anthony	1.00	0.14	0.25
Dominic	0.21	0.14	0.17
Francis	0.26	0.14	0.19
Jerome	0.62	0.25	0.36
John	0.46	0.23	0.31
Paul	0.21	0.06	0.09
Peter	0.79	0.19	0.31
Sebastian	0.91	0.18	0.30
Magdalene	0.94	0.18	0.30
Macro Avg	0.61	0.25	0.31

Table 10: Precision, Recall and F1-score for the model of strategy 3

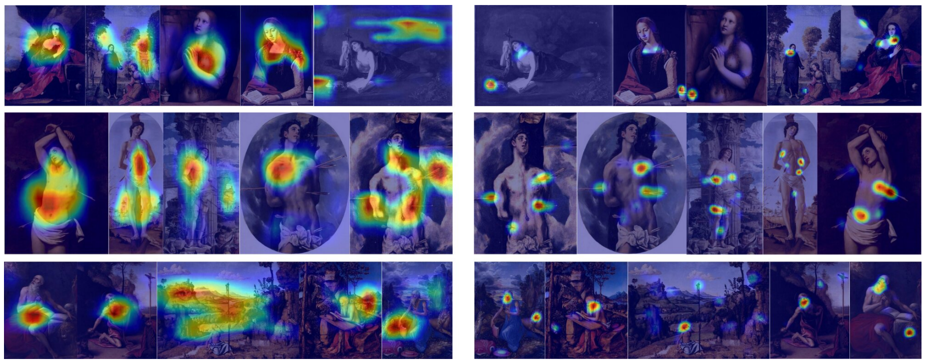


Figure 18: CAMs of correct prediction of model from strategy 2. Left is the VGG-16 model, right is the ResNet50. Mary Magdalene, Saint Sebastian and Saint Jerome are depicted in descending order.

the activation for the predicted classes. In the case of a wrong prediction, the CAM for the ground truth class was also extracted. The CAMs of three classes from the data will be highlighted and inspected, namely Saint Sebastian, Mary Magdalene and Saint Jerome. The selected CAMs for the models are chosen based on the distinctive elements which describe the iconography of the class. The most distinctive iconographic feature of Mary Magdalena depictions is her ointment jar. Saint Sebastian is recognisable as tied to an erect object and pierced by arrows. For Saint Jerome one of the most unique traits of his iconography is the depiction of a lion alongside him in the desert. Models from strategy 2 and 3 were the most promising and have thus been prioritized, based on their higher macro average precision and the similarity of strategy 3 to strategy 1.

In Figure 18 are displayed CAM instances drawn from correct predictions made by the model from strategy 2. In the case of Mary Magdalene, the model did not base its prediction on the activation of the ointment jar. In one instance it can be seen that the ointment jar is partially highlighted, however, this is likely due to activation of the textures around it, as can be seen in the upper part of the particular painting. The model instead seems to base its inference on the combination of two textures; The curly hair of Mary Magdalene and her skin. Activating based on the curly hair, is not an particular incorrect. Mary Magdalene is indeed more often than not depicted with long curly hair. For Saint Sebastian the model activates on regions around his stomach, and areas near arrows, this seems to suggest the combination of a stomach texture and

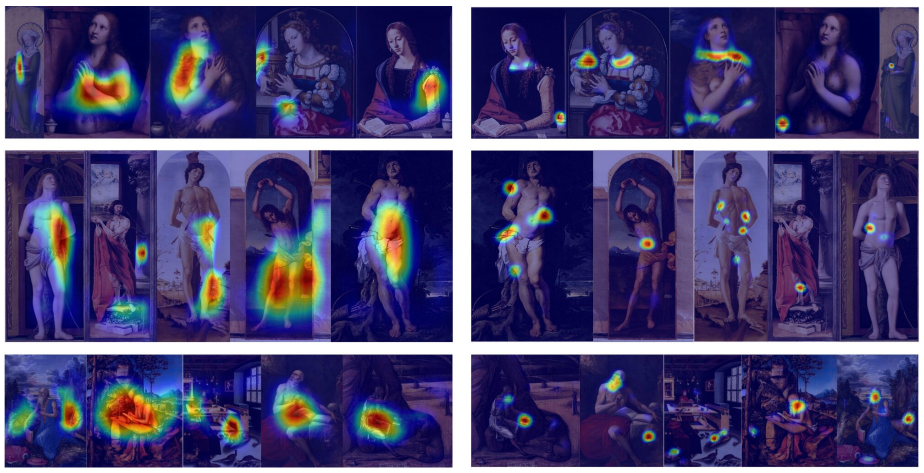


Figure 19: CAMs of correct prediction of the model from strategy 3. Left is the VGG-16 model, right is the ResNet50. Mary Magdalene, Saint Sebastian and Saint Jerome are depicted in descending order.

the lines from the arrows activates this model. The model did not manage to learn any features of the Lion depicted alongside Saint Jerome. It however, did manage to find and activate on features of the desert landscape, which Saint Jerome is often depicted in. Another feature that activated the model appears to be his hand grabbing a book. Saint Jerome is often depicted with an open book, as well.

Figure 19 displays the CAM instances for the model of strategy 3. On paintings of Mary Magdalene this model found and based some of its correct predictions on features of the ointment jar. Two images are shown where the model activates based on the ointment jar. On the left-most one the ointment jar is highlighted in its entirety, however it is quite likely that this coincides with the textures of the robe. In the other image activation is shown on the edge of the lid of the ointment jar, and part of the robe worn by Mary Magdalene. In other instances this model also activates on the texture of her hair and the robe she dons. On Saint Sebastian the model appears to activate based on the cloth wrapped around his loin, in combination with the lines of arrows nearby. In a single activation, the model activated based on the rope on the pillar Saint Sebastian would be tied to, were his hands not free in the depiction. In the depictions of Saint Jerome the model activates based on the lines in conjunction with the textures in the mane of the lion depicted alongside him. Additionally it also activates based on the desk and book, which Saint Jerome is often depicted with. In the top-right image of Figure 19 the state-of-the-art did not predict Mary Magdalene correctly. It instead predicted the class Virgin Mary, based on characters headdress. The CAM however belongs to the correct class, Mary Magdalene.

4.5 Discussion

Sections 4.2 and 4.3 presented the quantitative results of the experiments. The models do not appear to have been overfit to the training data and are able to generalise to unseen data in the validation and test sets. Although the models were chosen on a basis of their validation accuracy, the data set imbalance has had a significant effect. By predicting the Virgin Mary class more frequently, the models achieve a higher accuracy, since the data in the validation and test sets consist of roughly 63.7% paintings of Virgin Mary. The models do not blindly predict Virgin Mary to achieve this, as reflected in the max accuracies of the model in Table 5. However, these facts in conjunction with the relatively low recall in other classes, reflect a bias towards predicting the class Virgin Mary. Furthermore, we see that model of strategy 3 is the most successful in terms quantitative results on the test set, with a precision, recall and f1 score of **0.61**, **0.25** and **0.31** as opposed to the state-of-the-art results of **0.73**, **0.72** and **0.72**.

The curve of validation accuracy during training, Figure 17, for strategy 3 suggests the accuracy did not yet converge. It is possible to further train this model, at the risk of overfitting. However, a trend of increasing recall for Virgin Mary and decreasing recall for other classes, in Tables, 8, 9 and 10 as training continues, also indicates an increasing bias towards predicting class Virgin Mary.

As mentioned in Section 3.1, the imbalance of the data sets are proportional to the other. This naturally means the data for Virgin Mary has a greater variation of data samples. This is another factor that may increase recall for Virgin Mary. As the models attempt to generalise on unseen data samples, many of the otherwise unseen features may have appeared in some quantity of the Virgin Mary samples in the training set. Many of these features are likely not unique to Virgin Mary. When encountered in the validation and test phase, these features have then only been observed in the Virgin Mary context, thus the model predicts that class.

In section 4.4 qualitative results were displayed in the form of CAMs taken from correctly predicted test samples. The models exhibited a minor ability to infer the classes based on the most distinct symbols from their respective iconography. Although, these inferences were not based on the entire part of the symbol. Instead, parts of the symbols such as lines, edges and textures were highlighted, often in conjunction with an unrelated texture in nearby in the painting. In the cases where the model did not infer based on the most distinct symbol, the model had learnt to distinguish the class based on relatively consistent features, such as the curly hair of Mary Magdalene, Saint Jerome grabbing a book and loincloth or stomach of Saint Sebastian.

Although, the inferences are not based exclusively on the most distinct symbols in the iconography of the saints, the CAMs showed that the models were consistent in the features they extracted for inference. These features were also common in the iconography of the respective classes. The bases of inference were consistently a conjunction of low-level and mid-level features such as lines, edges and textures. This indicates that the models lack the depth needed to distinguish entire abstract objects like the entirety of the ointment jar of Mary Magdalene, the face of the lion accompanying Saint Jerome, and the arrows piercing Saint Sebastian. This is reflected in the fact that the over thrice deeper state-of-the-art ResNet50 model was able to consistently base its inference on the most distinctive symbols. Therefore, a convolutional baseline with a similar depth to the VGG-16 model does not appear to be able to retrieve the most distinctive visual elements of the class iconography. However, the model is capable of consistently retrieving other common elements in the class iconography.

The bases of the models were easily investigated with the addition of CAMs and have thus shown good results in terms of model transparency and explainable AI. The f1-scores shown in tables 9 and 10 reflect the capability of information retrieval by the models. The model of strategy 2 showed the greatest performance on Saint Sebastian and Mary Magdalene, while the model of strategy 3 showed the greatest performance on Saint Jerome, but was close to model of strategy 2 in the other two classes.

5 Conclusion

This thesis has presented a baseline model in the form of a VGG-16 modified to be fully convolutional, utilize batch normalization and output class activation mappings through the use of a global average pooling layer. The model was trained through three strategies, where strategy 2 and strategy 3 yielded the greater quantitative and qualitative results. The outputs of both models were explainable by the class activation mappings. These mappings showed that the models lacked the necessary depth of the model used in the state-of-the-art, for distinguishing the classes based on the most distinctive visual elements in their respective iconography. However, the two baseline models were capable of distinguishing other relatively common visual elements used in the iconography of these classes, including hair-style, nudity, clothing and books. Both models displayed promising f1-scores and can be used as a baseline for automatic tagging.

Future work may include the following objectives;

- For the training strategies, different data augmentation methods, like random affine transformations or random cropping, to attempt decrease in bias towards the Virgin Mary class.

- Increases in model depth. While the state-of-the-art is a 50 layer residual network, the increase of model depth can be instances of lower layer residual networks such as ResNet18 or ResNet34 [11] or the 19 layer variant of VGG [25].

- Further investigation of the model inferences through other methods from explainable AI, such as LIME [20].

References

[1] A. Amini, A. Soleimany, S. Karaman, and D. Rus. Spatial uncertainty sampling for end-to-end control, 05 2018.

[2] E. Cetinic, T. Lipic, and S. Grgic. Fine-tuning convolutional neural networks for fine art classification. *Expert Systems with Applications*, 114:107–118, 2018.

[3] L. Couprie. Iconclass: an iconographic classification system. *Art Libraries Journal*, 8(2):32–49, 1983.

[4] E. J. Crowley and A. Zisserman. The art of detection. In *European Conference on Computer Vision*, pages 721–737. Springer, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] L. Dormehl. https://icdn6.digitaltrends.com/image/digitaltrends/artificial_neural_network_1-791x388.jpg.

[7] FirelordPhoenix. https://computersciencewiki.org/index.php/File:MaxpoolSample2.png.

[8] N. Gonthier, Y. Gousseau, S. Ladjal, and O. Bonfait. Weakly supervised object detection in artworks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[9] N. Gonthier, S. Ladjal, and Y. Gousseau. Multiple instance learning on deep features for weakly supervised object detection with extreme domain shifts, 2020.

[10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Intellabs. http://intellabs.github.io/RiverTrail/tutorial/images/convolution2.png.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. 60(6):84–90, May 2017.

[15] M. Lin, Q. Chen, and S. Yan. Network in network, 2014.

[16] F. Milani and P. Fraternali. A data set and a convolutional model for iconography classification in paintings, 2020.

[17] Neurohive.io. https://neurohive.io/wp-content/uploads/2018/11/vgg16-1-e1542731207177.png.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[20] M. T. Ribeiro, S. Singh, and C. Guestrin. ”why should i trust you?”: Explaining the predictions of any classifier, 2016.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[22] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization?, 2019.

[23] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[24] X. Shen, A. A. Efros, and M. Aubry. Discovering visual patterns in art collections with spatially-consistent feature learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[26] B. University of California. Brueghel family: Jan brueghel the elder." the brueghel family database. http://www.janbrueghel.net/.

[27] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[28] C. Versloot. https://www.machinecurve.com/wp-content/uploads/2020/01/Global-Average-Pooling-3.png.

[29] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[30] N. Westlake, H. Cai, and P. Hall. Detecting people in artwork with cnns. In *European Conference on Computer Vision*, pages 825–841. Springer, 2016.

[31] R. Yin, E. Monson, E. Honig, I. Daubechies, and M. Maggioni. Object recognition in art drawings: Transfer of a neural network. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2299–2303, 2016.

[32] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. *CoRR*, abs/1708.04896, 2017.

[33] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

Appendix A: Learning Resources.

https://cs231n.github.io/convolutional-networks/

https://paperswithcode.com/

https://pytorch.org/tutorials/

https://machinelearningmastery.com/

https://scikit-learn.org/stable/

https://stackoverflow.com/

https://stats.stackexchange.com/

https://www.christianiconography.info/

Notes taken while attending:
The 2020 Beeldverwerken (Image processing) course taught by Leo Dorst at the University of Amsterdam.

Notes taken while attending:
The 2018 Leren (Learning) course taught at the University of Amsterdam.