Using ML-based Computer Vision for Arms Control: Insights into Conflict Zones through Social Media Imagery

A thesis presented for the degree of
M.Sc. in Geoinformatics by

BEN LUCA BUCHENAU

Supervision by
Dr. Dominik Drees & Prof. Dr. Benjamin Risse,
Computer Vision & Machine Learning Systems Research Group

Institute for Geoinformatics
University of Münster
March 2024

Acknowledgements

I gratefully acknowledge *Dr. Dominik Drees* for his patient support and useful feedback during the work. I thank him and *Prof. Dr. Benjamin Risse* for supervising my thesis and being open-minded about the envisioned work concept.

Additionally, I want to express my gratitude to *Lars Wirkus* for enabling me to conduct my thesis within the institute, for collaborating in developing the thesis idea and for giving continuous input. I also thank my coworkers in the data & IT team for their thoughts on my work, especially *Louai Haddad* and *Nele Kerndt*.

Lastly, I would like to acknowledge the help of my family, with a special thanks to *Josefine Weiss* for her endless support and understanding, as well as to *Dr. Stefan Weiss* and *Nikolas Buchenau* for sharp proofreading.

Declaration of Academic Integrity

I hereby confirm that this thesis, entitled *"Using ML-based Computer Vision for Arms Control: Insights into Conflict Zones through Social Media Imagery"* is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledges as such and the sources cited. I am aware that plagiarism is considered an act of deception which can result in sanction in accordance with the examination regulations.

[illegible]

(Date, Signature of Student)

I consent to having my thesis cross-checked with other texts to identify possible similarities and to having it stored in a database for this purpose. I hereby confirm that I have not submitted the following thesis in part or whole as an examination paper before.

[illegible]

(Date, Signature of Student)

Abstract

With violent conflicts enduring and emerging across the globe, the need for arms control measures is omnipresent. As many illegal activities are hard to track, analyzing social media images can open a new facet of information, while serving as an open-source intelligence tool for societal verification. For analysis, the application of machine-learning-based Computer Vision methods can be highly useful. But currently, there is a gap between the known possibilities of Computer Vision and previous research on societal verification for arms control. To fill that gap, two Convolutional Neural Networks based on the pretrained YOLOv8 and ResNet50 architectures are finetuned in this work to perform binary and multilabel image classification. These models are designed to detect and classify possibly interesting images from social media, particularly focusing on identifying arms, soldiers and military vehicles. The training resulted in a classifier that is able to identify interesting social media images to a valuable degree, while the recall-specificity trade-off can be influenced with a training bias. At the same time, results suggest that classification on object-level needs further development and that high variance across areas is likely present. Along with ethical considerations regarding problematic training biases, future model refinements are presented. Overall, the findings of this thesis highlight the potential of Computer Vision in generating new insights into conflict regions.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Illegal arms trade and militant activities are a huge threat to global peace. Especially in conflict zones such as the Sahel, the Ukraine and some Middle Eastern and Northern African (MENA) countries, militant groups and other malicious actors amplify the proliferation of arms [77]. Mainly small arms and light weapons (SALW) are distributed uncontrollably, but also large military equipment such as vehicles and heavy arms. Those arms goods are the reason even non-state conflicts quickly turn into uncontrolled violence, and any limitation effort can save lives. Many of the illegal weapons come from third countries with low restriction, but originate in Western countries – meaning those countries also have the duty to develop solutions and implement measures. For example, Germany holds responsibility to some degree, knowing that the country is still among the top five arms exporters worldwide [76]. Because such arms trade activities are extremely hard to track by authorities once they are forwarded to third countries, there is the need for societal verification using open data [22]. Following that need, there are journalistic efforts such as the data-driven *EU Arms* project that tries to follow unofficial trade routes to establish a bottom-up arms control mechanism. In addition to those public efforts, there are also scientific approaches, which will be described in the following.

1.1 Related work and Motivation

A common technique in previous research is to apply data analysis based on machine learning (*ML*) to uncover problematic trade activities and therefore support export control [3, 41, 78]. Recently, there is a lot of scientific focus on applying automated Computer Vision (*CV*) methods to get insights not only from textual and numeric information in documents, but also from imagery [30, 68]. In similar domains that tackle armament, there is already much research on ML-based approaches, for instance in police investigations [28, 32], X-ray baggage control [2] or other surveillance and control systems [18, 33, 46, 49, 51, 80]. For some applications, social media images are used, for example to identify social media weapon posers [71]. Utilizing social media images for scientific purposes is a very promising method, as the information is often very detailed, fine-grained and especially close to real-time [14, 42, 54, 59]. Because social media functions as a tool to attract international attention to many conflict regions and suppressed nations, it is already now an integral part of the political discourse and holds useful information [39, 70]. A popular example is the crash of Malaysian Airlines flight MH17, where journalists uncovered which conflict party shot down the airplane just by analyzing social media images [5]. Investigations like this are often performed by groups like *Bellingcat*, the *Conflict Intelligence Team* and alike, which are mainly data scientists with educational journalistic objectives. As an anti-government group, the latter mainly focus on private military companies (*PMCs*) such as the well-known Russian *Wagner Group* in their research [55]. In line with that, this work is motivated by the mentioned journalistic groups and can add value to investigative efforts.

Also in collective action events such as demonstrations [82], tourism [40] and emergency response [11, 21, 47], social media text and image analysis was applied to gain otherwise hidden information. The main benefit is that this completely new facet of information can be used to make more profound, comprehensive and justified decisions [37]. From the data perspective, those methods are some form of open-source intelligence (OSINT) – a way to gather publicly available but mostly obscure information. In the context of this work, ML-based computer vision applied to social media images functions as an OSINT tool by collecting, identifying and classifying image data from conflict zones that can be relevant for disarmament measures. According to previous research, those measures can be of different nature: For example, trade regulations and agreements on government-level can go hand in hand with usage-related measures within countries [52].

1.2 Objective, Contribution and Use-Cases

Knowing about the strong call for more arms control through societal verification [22] and the extensive applications using social media imagery as information source as stated in section 1.1, there is a clear research gap in combining both fields. Following the current state of science, the thesis will start to fill that gap by using social media images to support arms control. To achieve this, the aim is to create two CV-based models that can detect possibly insightful images (Model I: *is_interesting*) in conflict zones from social media and optimally also classify them (Model II: *is_object_class*).

The output will be beneficial for expert inspections and can unveil information such as confirming the presence of specific illegal arms in conflict locations. This work is attached to the peace and conflict research institute *Bonn International Center for Conflict Studies (BICC)* – so in this context, expert inspection means interested scientific staff working in various projects within the domain of militarization and arms control to achieve disarmament. Two possible use-cases will be described to demonstrate the benefit of this work for the peace and conflict research experts:

Use-case A. A BICC project called *SALW Guide* [20] maintains a database of global arms distribution and trains locals in conflict regions to implement arms control measures. So far, knowledge is created mainly by time and cost expensive field work as well as textual documentation. Reviewing social media images from those areas can add valuable information with little effort. For example, identifying a weapon on an image in a location this weapon previously was not known to occur would be a useful outcome for the project.

Use-case B. BICC holds expertise and advises the German government on PMCs (see section 1.1), which violate and act against international humanitarian law. Therefore, it is important to track and hold those groups accountable for their actions. Currently, mostly Russian PMCs pose a threat to different conflict regions (see section 1.1). Many of their global activities remain unverified, and often it is even unknown if they are present in a conflict area or not. For example, identifying a soldier or vehicle owned by a PMC in a previously unverified location would be a useful outcome for the research community and the political discourse.

While demonstrating that social media imagery holds valuable information for arms control in the first place, the use-cases highlight specific situations where it is useful to pin down images that can be insights into violent conflict areas. This will be evaluated on a fixed social media image set. For this study, images defined as *interesting* (or in other words, possibly insightful) are defined by containing $n \geq 1$ of the following objects:

- **Arms/Weaponry.** Weapons, such as SALW by the UN definition, are the main tool for harm and danger driving global conflicts [77]. Knowing about the presence of specific arms in conflict areas is crucial [71]. There are many efforts to collect data on region-specific spread of weapon types, building knowledge on their global distribution. For example, the *SALW Guide* project (see section 1.1) is part of the *UN Programme to Prevent, Combat and Eradicate Illicit Trade in SALW* and creates a database on collected data. This work will test how and to what extend social media images can contribute by adding information.

- **Military Vehicles.** Although the occurrence of military vehicles such as tanks and self-propelled artillery is not as hard to track as SALW due to their size, there is still no clear picture of global distribution. Also, access of malicious non-state actors to those vehicles has become easier, leaving space for trade actions that cannot be approved and tracked well by governments.

- **Soldiers.** While the record of soldiers in conflict regions is rather well documented, the presence of various militant groups and other malicious actors sometimes remains in the dark. For example, Russian PMCs operate in many African, Middle-Eastern and Latin-American countries – some of them without official confirmation.

The underlying idea for this approach is developing an application based on CNN classifiers. Outcomes of this work will create a theoretical framework for application development by assessing the feasibility. Practical considerations such as database implementation, hosting, continuous social media scraping or front-end development are left as future work.

1.3 Arms Control in Conflict Regions

This project will spatially focus on social media imagery coming from six selected countries that suffer from major conflicts as of September 2023 (see table 1.1 and figure 1.1). The selection is based on the current political relevance as indicated in the latest Peace Report published by four leading German peace research institutes [7], but also on the availability of social media imagery originating from those regions. All

Table 1.1: Overview of selected conflict regions for this work.

	Country	**Conflict**	**Start date**
A	Ukraine	Russian invasion	February 2022
B	Syria	Civil war	March 2011
C	Afghanistan	Taliban takeover	August 2021
D	Venezuela	Socioeconomic crisis	June 2010
E	Yemen	Civil war	September 2014
F	Mali (Sahel)	Civil war	January 2012
F	Niger (Sahel)	Military takeover	July 2023

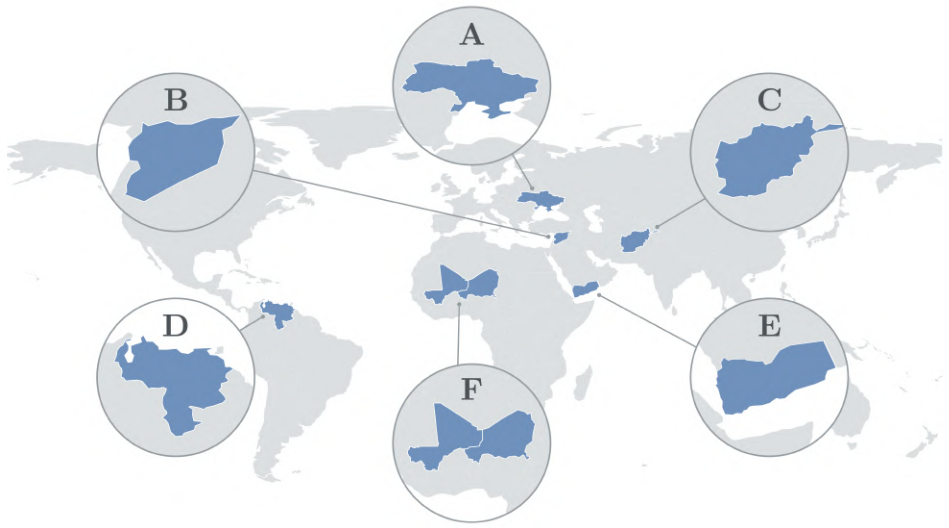


Figure 1.1: Conflict areas that were chosen for the analysis (the countries Mali and Niger will be treated jointly as the Sahel region).

of the countries currently suffer, or lately suffered from violent conflicts due to different actors. While civil wars in Syria, Yemen and Mali are still ongoing, the Russian invasion in the Ukraine led to a inter-state war in Europe. In Afghanistan and Niger, military takeovers took place lately, and Venezuela still suffers from violent instability after a socioeconomic crisis.

Whereas the conflict cause may be varying, all regions have in common that the availability of reliable information on the conflict is sparse – especially when it comes to the presence of various militant actors and specific arms goods. In most other states involved in a violent conflict, the situation is similar. For instance, in the Israel-Hamas war that started just after data collection, reliable information is rare as well. This justifies the need for societal verification through ML-based CV methods in the selected regions. At this point it is important to note that this work does not aim in any way at monitoring, assessing or publishing imagery or actions of the regions' individuals. Interestingly, even the contrary is true: In all of the selected areas, militant actors and accompanying illegal arms goods come from outside or a government regime itself. The application of ML-based CV to develop image classifiers for information therefore targets violent suppressors and advocates for the oppressed. From this perspective, social media can be seen as a magnifier for transparency [39]. Nevertheless, ethical issues raised in previous literature [14, 18, 42, 82] inherent to this work will be addressed (see section 5.3).

Chapter 2: Conceptual Framework

This chapter will explain the most important theoretical concepts of this thesis. First, a closer look at the domain of computer vision and its most basic methods (called *Conventional Computer Vision*) as described in [66] is helpful. Next, the fundamentals of *Deep Learning* will be explained – mostly based on [23] – as it is important to understand for this thesis and significantly enhances the applications of computer vision techniques. To combine both fields, a description of *Machine-Learning-based Computer Vision* will clarify the connection, with a focus on Convolutional Neural Networks, because they build the methodological backbone of this work.

2.1 Conventional Computer Vision

Computer Vision (CV) is a broad field that enables computers to make decisions based on visual data from images or videos (p.440 [23]). It is multidisciplinary in a way that it involves multiple fields such as computer science and mathematics as well as application-related fields such as image processing and optics [66]. In CV, the aim is to replicate and even improve the capabilities of human vision, allowing machines to understand, analyze and interpret image data in a way that is similar to how humans perceive the visual world – gradually being able to recognize colors, edges or textures [68]. The main advantage is that in CV, it is possible to automate and speed up image analysis tasks such as the detection of shapes. Some common applications include human facial or gesture recognition, navigating autonomous vehicles, analyzing medical imagery, environmental monitoring through time-series or as in the case of this study: gathering information from social media for political and social sciences. Those applications are based on techniques such as image recognition (identifying and classifying objects or patterns within an image), object detection (locating objects in an image) and image segmentation (dividing an image into different semantic regions). For this work, different image classification tasks will be most important (see section 3.3).

Image Processing. To perform those tasks, different image processing techniques can be applied, that are the foundation of computer vision. On the technical side, they are all based on operations on the pixel-level, where pixels can be understood as coordinates in a 2D coordinate system and therefore allow precise manipulation. For example, color manipulation involves converting images between different color space representations (e.g., RGB to grayscale) or filtering by color to isolate single channels. Similarly, the contrast or brightness can be manipulated. In this case, for any output pixel value g at position (i, j) a transformation is calculated like

$$g(i, j) = a \cdot f(i, j) + b,$$

where $f(i, j)$ is the input pixel value at the same position and the variables $a > 0$ (gain) and b (bias) control the saturation and brightness. It is also common to compute binary images using a specific threshold for pixel values. Those techniques have in common that they apply some calculation to one pixel and use the result for the same pixel in the output image (called *point operations* [66]). Other operations apply a *kernel* of any given size to calculate the output pixel. For many practical applications, neighborhood filters (often using the Gaussian distribution as a function for pixel weight) can be applied. As a basic denotation, the output pixel value g at position (i, j) is calculated as

$$g(i, j) = \frac{\sum_{k,l} f(k, l) \cdot w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)},$$

where f is the input pixel value at the neighbouring position (k, l) and $w(i, j, k, l)$ is the weighting coefficient based on the specific kernel applied. Note that $w(i, j, k, l)$ is 4-dimensional for neighbourhood filters as the weight is computed not only from the input but also neighbouring pixels. It is also common to combine multiple kernels. For example, in bilateral filtering (see figure 2.1), the coefficient is the product of a Gaussian range kernel $r(i, j, k, l)$ and a data-dependant domain kernel $d(i, j, k, l)$ (p.134 [66]) to account for more than just the spatial dimension. The bilateral filter is used in this work's image preprocessing (see section 3.2.2).

Morphological operations enable modifying contours or object shapes in images. Instead of looking at the pixel distribution in the neighborhood defined by the kernel, a local maximum or minimum is identified and applied to each pixel in the output. This operation will either *dilate* or *erode* shapes in an image. To identify boundaries within an image, which are represented by significant local changes in pixels, gradients can be calculated. Applying a threshold to the gradient images can then extract the

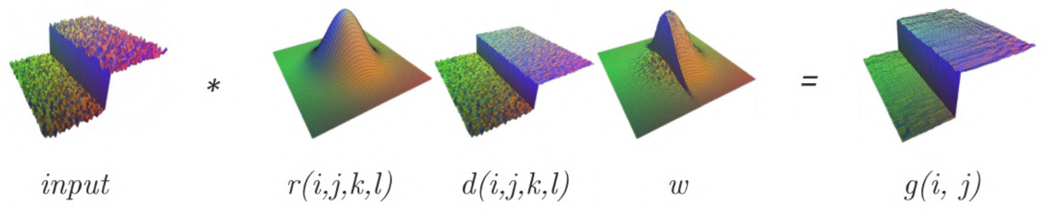


Figure 2.1: Neighbourhood operation (bilateral filtering) as example for important CV methods in image processing (left: input, center: range and domain kernel forming the weight for the central pixel, right: output, taken from [17]).

most significant edges. Those described operations using more than one pixel $f(i, j)$ to compute the output $g(i, j)$ at the same location are called *neighbourhood operations* [66]. Lastly, another important set of operations involves geometric transformation such as scaling, rotation, warping or translation. An important property in image processing tasks is *translation invariance*, which means that the kernels described above produce the same output regardless of input shifts (p.122 [66]).

To sum up, each pixel in the output image of a processing step is either calculated using the same pixel and some constant as input, or a kernel of multiple pixels of any given size. In order to actually *learn* from an image, the mentioned operations build the backbone of higher-level understanding in CV that was enabled through the introduction of deep learning.

CV classification methods. The point and neighbourhood operations on pixel-level are image processing techniques forming the basis for useful classification methods in CV. The most important methods for this study are mentioned in the following. For example in image preprocessing, a combination of filters were applied to classify images based on their color intensity and gradient. During training, image classification (p.349f [66]) based on feature recognition for object detection is used, allowing to map object classes to images (p.455 [23]). This is the integral part of the thesis and is also applied in most research focusing on social media imagery [14]. Being able to recognize content features is also a good example for the advantages of applying CV compared to traditional image content analysis [**?**]. For deeper understanding of the image data, background segmentation (p.227f [66]) was applied, as well as manifold learning methods for dimensionality reduction (p.265f [66]). The latter methods are mostly applied in research as a form of exploratory CV applications [54].

2.2 Deep Learning

Deep learning is one of many methods used in machine learning, a subfield of data and computer science that applies statistical algorithms in order to generate information from data. While other machine learning techniques such as classic linear or logistic regression models, decision trees or support vector machines (SVM) predate the emergence of deep learning methods [41], it evolved significantly in the last two decades as a comparatively new field in the artificial intelligence (AI) domain, while having theoretical roots back in the 1940s (p.13 [23]). The foundation of deep learning are artificial neural networks, which mimic the structure of the human brain. A deep neural network always consists of the following setup: An input layer receives the raw data that is forwarded to the network. Each node (or analog to neuroscience also called *neuron*) in the input layer represents a feature or attribute of the input data, which can be images, text or audio. Practically speaking, this input data is always denoted as numerical information in the form of *tensors*, which are multidimensional mathematical objects that can store data (e.g., 1D vectors or 2D matrices). The last neural network layer is called the output layer. It produces the final result (i.e., the *prediction*) of the model based on outcome probabilities often in the form of relative *confidence* scores in range $(0, 1)$. Each node in the output layer represents a possible outcome category, which is dependent on the specific use case. For instance, in a *binary classification* task the output layer consists of two classes representing the two possible outcomes. If each input can belong to more than one output class, the task is called *multilabel classification*. This is especially relevant in multi-class models where $n(classes) > 2$. Between input and output there are multiple hidden layers that process the input information through weighted connections and *activation functions*, which are essential to the learning process. The weighted sum of inputs to a neuron is calculated by multiplying all inputs by their corresponding weight and summing up these products. Mathematically, for a neuron j in a neural network layer:

$$z_j = \sum_i (w_{ij} \cdot x_i) + b_j,$$

where z_j is the weighted sum for neuron j, w_{ij} is the weight of the connection between neuron i and neuron j, x_i is the input from neuron i and b_j is a possible bias term for neuron j. The weighted sum z_j is then passed through an activation function f to introduce non-linearity. Without this step, the neural network would be limited to detecting linear relations in the data, not being able to learn complex patterns. The activation function takes the weighted sum z_j as input and produces the output a_j of the neuron, which is then passed to the neurons in the next layer – and so on. The output of neuron a_j is given by:

$$a_j = f(z_j),$$

where $f(z_j)$ is the activation function applied to the weighted sum. The most important activation functions used in this work are visualized in figure 2.2.

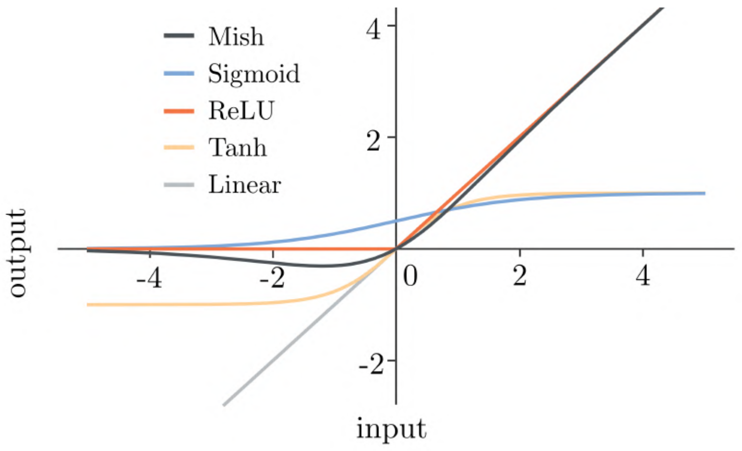


Figure 2.2: Commonly used activation functions (Mish [45], ReLu [1] and Sigmoid functions are applied in this work).

How often this step is repeated correlates with the amount of nodes and will determine the complexity of any network. The quantity of hidden layers and how many parameters form a network highly depends on the specific use-case. The quantity of those layers is what the word *deep* refers to (p.168 [23]). In simple terms: the more layers, the deeper the network. Network width is mainly defined by the number of nodes building the layers and commonly set in alignment with model depth (p.197 [23]). The basic NN structure is visualised in figure 2.3.

Depending on the task, machine learning methods can be *unsupervised* or *supervised*. In unsupervised learning, unlabeled input data is passed into the model, which is then trained to discover relationships, clusters or reduce the data dimensionality without explicit human guidance. In contrast, for most use-cases (and also for this work), the input data is labeled, therefore pairing it with the corresponding output – also called *ground truth* information. The goal of the model in this case is to learn the mapping from inputs to outputs that enable it to make predictions on unseen

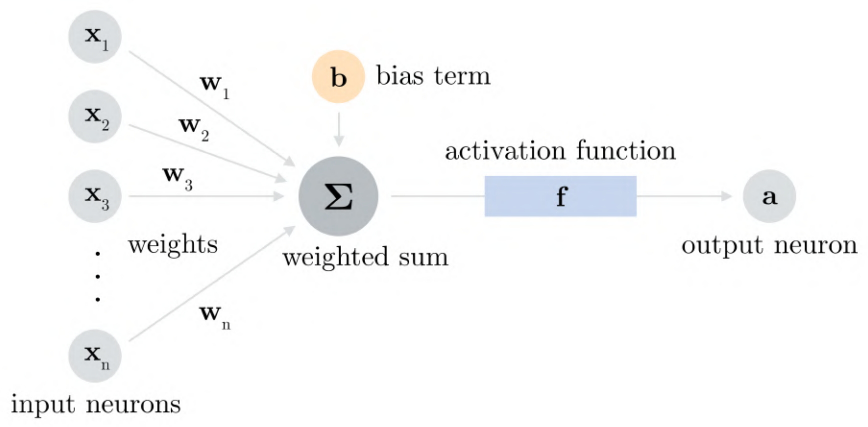


Figure 2.3: Calculation of one NN node (recreation of [30]).

data. Some specific cases require a combination of both methods, which is called semi-supervised learning.

Knowing the actual output from labeled data allows us to measure how well a model's predictions match the ground truth. This is done by quantifying the error between the predicted and true model output, which is also called the *cost* or *loss*. There are various loss functions for different use cases, such as the Root Mean Squared Error (*RMSE*) or Cross-Entropy Loss. For example, in classification problems where Binary Cross-Entropy (*BCE*) loss is used, the loss function can be expressed as:

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

where n is the number of data points, y_i is the true label (0 or 1) for the i-th data point and $\hat{y}_i$ is the predicted probability of class 1 for the i-th data point.

In deep learning, the goal of each training is always to minimize the model's loss (p.280 [66]) based on the chosen loss function. As the data in NN development is split into training, validation and test sets, different losses can be calculated: The *training loss* expresses the discrepancy between the predicted output of the model and the actual ground truth in the training data. The validation set is used to estimate generalization error during training (p.121 [23]), therefore *validation loss* can be seen as a proxy for the network's generalization ability on unseen data. Neural networks are commonly trained in *minibatches* of input data $\mathbb{B} = \{x^1, x^2, \dots, x^m\}$, where m

can be as low as 1 (p.151 [23]). It is therefore possible to calculate the training loss after each batch, enabling the model to adjust its weighted neuron connections and therefore optimize the prediction. This technique is called *backpropagation* [38]. The adjustment is performed by applying one of many *optimization* algorithms, which try to find the optimal way of adapting input parameters to receive the desired prediction. An important parameter in this context is the *learning rate*, which determines the size of the steps taken in the parameter space during each iteration of the optimization – or simply how strong the neural network adjusts its weights. As in many statistical methods, it is also common to normalize the data at some point during the process to avoid the weights adjusting to an internal co-variate shift epoch by epoch. This is also called *batch normalization*. In visualization, the training and validation loss is often displayed in *epochs*, which indicate the number of iterations over the input dataset (p.246 [23]).

Lastly, the test set is a completely independent dataset that the network has not seen during the training phase. It is used to evaluate the final performance of the model and evaluate how well it can generalize to new, unseen data. This is important, because often the networks learn the training data and its specific details too well, performing promisingly on the training set, but poorly on the new test data. This behaviour is called *overfitting* (p.199 [66]). In contrast, models can also be *underfitted*, which occurs when the network is too simplistic to capture the underlying patterns in the training data, resulting in poor prediction performance on all sets.

The evaluation is therefore an integral part following each NN trainig. Plotting the loss for the validation set by epochs gives insights into the model's learning process. A confusion matrix plotting the predicted classes against the ground truth in the test set visualizes NN performance. In binary classification, this matrix shows true positive (TP), true negative (TN), false positive (FP) and false negative (FN) predictions (p.442 [66]), which are commonly normalized in order to have proportional values in range $(0, 1)$. In a multilabel task with multiple classes those scores can be calculated per class to highlight performance differences. In this case, an image can belong to a class (positive) or not (negative). For a model evaluated on test images, where

$$\begin{aligned}
\text{accuracy} &= \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{FN} + \text{TN})} \\
\text{precision} &= \frac{\text{TP}}{(\text{TP} + \text{FP})} \\
\text{recall} &= \frac{\text{TP}}{(\text{TP} + \text{FN})} \\
\text{specificity} &= \frac{\text{TN}}{(\text{TN} + \text{FP})}
\end{aligned}$$

the accuracy measures the total proportion of correct predictions, the precision measures the share of true positive predictions across all positive predictions, the recall (or sensitivity) measures the share of true positive predictions across all actually positive images and the specificity vice versa measures the share of true negative predictions across all actually negative images. To reduce the number of evaluation metrics, often also the *f1 score* is calculated using

$$\text{f1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

to balance precision and recall, as they are often inversely related – as one increases, the other may decrease. In classification, a common issue is therefore to find a satisfying *trade-off* between those metrics. The problem can also be described as finding a trade-off between network complexity and its ability to generalize [41].

2.3 Machine-learning-based Computer Vision

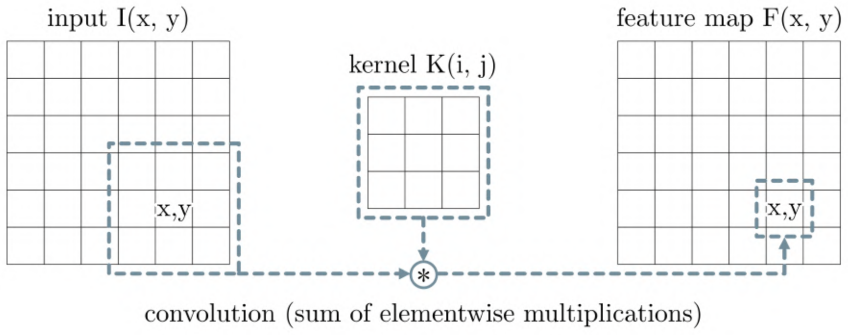
As already indicated in section 2.2, there are many different machine-learning methods, deep neural networks being one of them. Machine-learning-based computer vision therefore is a very wide field, and for the purpose of this study it is sufficient to look into deep neural networks applied for CV. The two most important types of neural networks for image-related training that emerged in the last decade are Convolutional Neural Networks (CNN) and Transformers optimized for vision tasks. Both will be described below, with a focus on CNNs as they are also used for this study.

2.3.1 Convolutional Neural Networks (CNN)

A specific type of network developed to process images as input are CNNs. This is because they are focused on finding local patterns and therefore capture spatial features in data. While they can also be used for other input, this characteristic makes them well-suited for images due to their grid-based pixel structure (p.330f [23]). In most but not all setups, the hidden CNN layers consist of *convolutional*, *pooling* and *fully-connected* layers – which will be described in the following.

Convolutional layers. The convolutional layers are fundamental to the learning process: They produce a set of feature maps as output, corresponding to specific *filters* applied to the input. The filters consist of small matrices forming *kernels* (typically either 3x3 or 5x5) as introduced in 2.1, which will be applied to all pixels within the image successively on their surrounding neighbourhood (see figure 2.4). More practically, the convolution operation involves sliding the filter over the input image and computing the dot product for each position, storing the results in the output feature map [68]. This setup enables the network to learn basic local features in early, and complex patterns in later convolutional layers. For example, the filters in the first convolutional layer might focus on detecting different kinds of edges and textures that will be used to identify different shapes in a second layer, and infer object parts in a third layer.

Pooling layers. In between the convolutional layers, pooling layers are implemented to down-sample the spatial dimensions of the feature maps, effectively reducing the computational load in the network. Another advantage of pooling is that it preserves dominant features, while removing some of the noise from the image [38]. Different pooling methods can be applied depending on the use case. It is common



$$\sum_{i=1}^{h}\sum_{j=1}^{w} I(x+i-1,\ y+j-1)\cdot K(i,j)$$

Figure 2.4: CNN computation in convolutional layer producing feature maps based on a kernel as in 2.1 (recreation of [30]).

to either use the maximum value for each local region (*max pooling*), which captures the most dominant features and generally sharpens edges, or to compute the mean value (*average pooling*), which generally smoothes the sampling-down effect.
Fully-connected layers. After the convolutional and pooling layers enabled the CNN to learn regional features, one or more fully-connected layers (also called *dense layers*) allow the network to capture global relationships. To do so, the output of the previous layers is first flattened into a one-dimensional vector, which ensures that each neuron in the fully-connected layer receives input from all neurons in the preceding layer – following the described NN node calculation as in section 2.2. Due to the many neuron connections, fully-connected layers usually have a large number of parameters, allowing them to provide high-level representations of the learned features. For most classification tasks, the output layer with predictions for classes is also fully-connected.

As all of the described layers in CNNs so far perform linear operations, different activation functions (see section 2.2) are commonly applied – directly after the convolution, in the fully-connected layer or both. The basic CNN architecture is visualised in figure 2.5.

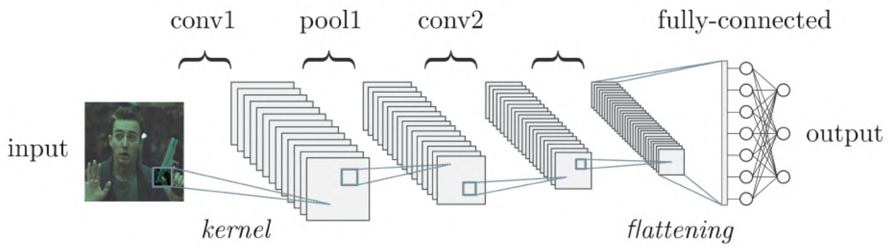


Figure 2.5: CNN architecture consisting of convolutional, pooling and fully-connected layers.

Local vs. global patterns. Convolutional layers have been described as recognizing local image patterns and fully-connected layers as rather capturing global image patterns – which is a simplified explanation. Theoretically, the property of translation invariance (see section 2.1) allows both convolutional and fully-connected layers to detect local patterns in the data. Also, a neural network does not have to contain both types of layers: For instance, there are also fully convolutional networks (FCN) (p.296f [66]) and nets with only fully-connected layers (p.372 [23]).

2.3.2 Vision Transformer (ViT)

The NN architecture of Transformer is mainly applied in natural language processing (NLP), often performing better than LSTM models [62]. After also attaining excellent results in CV, many models processing images are also based on Vision Transformer (ViT) lately [16]. Coming from the NLP domain, their main advantage is that they are better at detecting global relationships than CNNs and can therefore learn more complex contexts in data [13]. The architecture design is based on weighting importance through a self-attention mechanism of data patches – which are pixel grids in images. In the ViT setup, all patches are connected and ranked against each other, allowing to identify dominant features and also to understand the 'bigger picture'. To give an overview of ML-based CV it is important to mention ViTs because of their recent prominence, but for this work CNNs have been applied due to a broader range of scientific references and documentation, especially in arms control.

2.4 Finetuning a pretrained CNN

Training a new network is cost expensive in terms of computational power, especially for image data. Commercial and scientific CNNs are usually trained on powerful computing clusters bundling extensive GPU capabilities. Once a network is trained, the architecture including all weights between the node connections (*parameters*) can be saved to a file. This is why there are some high-performance open-source CNNs available online that outperform others on standardised benchmarks. For custom models, it is therefore common to build upon those powerful, pretrained model architectures and *finetune* them according to the individual use case. This way, we can benefit from pretrained low-level features of the architecture and build high-level understanding on top of that. As they are already trained extensively, prediction results are often much higher compared to building a model from scratch [78]. The finetuning involves applying custom hyperparameters such as the type of loss function, batch size, learning rate or optimizer (see sections 2.2 and 2.3). It is also possible to add a *dropout*, which randomly deactivates a faction of neurons, forcing the network to learn more robust and generalized features by preventing excessive reliance on specific neurons [65]. Another technique is to apply random geometric transformations (see section 2.1) called *augmentation*, in order to enrich the training data (p.445 [23]), creating a more diverse and solid dataset [63]. When working with pretrained models it is important to choose, if and which layers should be *unfrozen*, meaning that their pretrained parameters can be adjusted during training. As indicated above, this allows the network to learn new high-level features based on the pretrained low-level features (see section 2.3.1). When using a pretrained architecture for a specific task that needs a different output values, the activation function in the last unfrozen layer can also be adjusted. For instance, many pretrained CNNs are producing output class probabilities for multi-class problems (see section 2.2) as

$$\sum_{i=1}^{n} y_i = 1 \quad with \quad 0 \leq y_i \leq 1,$$

where y_i represents the output probability for class i and n is the number of classes. Changing the last-layer activation function can then enable the network to perform multilabel classification as in this work, where still $0 \leq y_i \leq 1$ but the sum of all probabilities does not have to add up to 1.

In general, finetuning hyperparameters are mostly chosen based on their ability to increase model capacity (p.432 [23]). In this work, common pretrained CNN architectures were finetuned as well, which will be described in detail in the following section 3.3.

Chapter 3: Methodology

This chapter will elaborate on the methods applied for this work. After describing *training* and *testing* datasets and their sources, all *preprocessing* steps performed will be explained alongside issues that aroused. Lastly, the actual two *classification models*, Model I (*is_interesting*) and Model II (*is_object_class*), and their tasks will be presented. This includes highlighting relevant technical specifications such as chosen hyperparameter settings and finetuning actions as defined in 2.4.

3.1 Test set: Social media images of interest

As previously mentioned, this work aims to assess whether and to what degree ML-based computer vision can assist in gathering information from classifying social media imagery for arms control. Therefore it is essential to have representative social media images. To test (and partially also to train) the models, data from the image-based social media platform INSTAGRAM have been used (see figure 3.1). As requests to their API are very limited even for scientific purposes, a modified version of the Python social

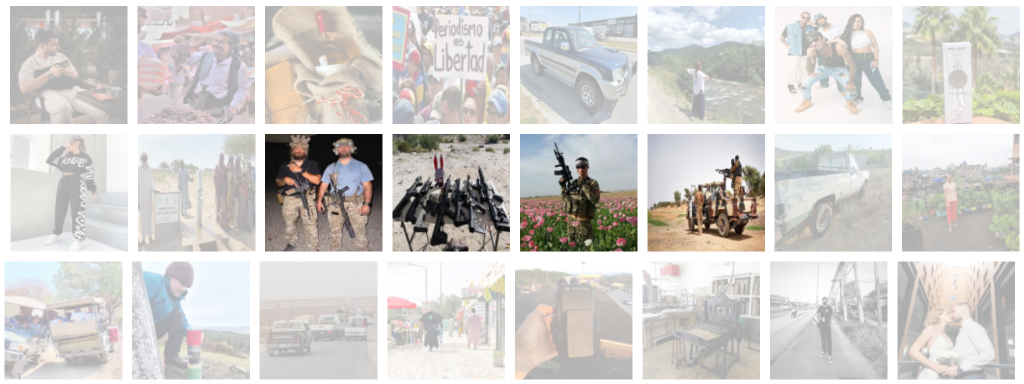


Figure 3.1: Instagram test image dataset (transparent images form the mass of irrelevant, *not_interesting* images, from which *interesting* ones should be identified and classified, as displayed in the center).

network scraper SNSCRAPE [31] was implemented (see algorithm 1). The modification for this work mainly included enabling a location-based search to only get images from the relevant conflict regions defined in 1.3. The different spatial origins of social media images allows to not only test the models on the aggregated dataset, but also to analyse differences in the performance between conflict regions. The platform maps the post location of an image to a *location id* on the country-level, which makes it possible to fetch images from countries through simple HTTPS requests. As this request cannot be combined with content-related tags, the resulting images are not filtered thematically.

Algorithm 1 Social media scraping prodecure

```
$locations \leftarrow dictionary.keys()$
$sleep\_interval \leftarrow t * 3600$
$max\_images \leftarrow y$
$num\_images \leftarrow n$
while $num\_images \leq max\_images$ do
    for $loc$ in locations do
        $metadata \leftarrow$ call load_post_information($loc$)
        $image\_urls \leftarrow metadata.urls$
        call append_to_textfile($metadata$)
        for $url$ in $image\_urls$ do
            if $metadata.media.type(url)$ = image then
                call download_image($url$)
                $num\_images$ += 1
            end if
        end for
    end for
    time.sleep($sleep\_interval$)
end while
```

Filtering keyword classes from all samples such as in [82] after the geolocation in a second process has not shown promising results in early tests. This is mainly due to inconsistent keywords associated with the interesting images – and there is no useful taxonomy as it exists for other domains [4]. This not only means that CV is well suited to use in this case, but also that the test dataset consists mostly of *not_interesting* images and a few *interesting* ones containing *arms/weapons*, *vehicles* or *soldiers* – which will be the base for later classification of *Model I* and *Model II*. Also, only the most recent posts can be fetched. To accumulate the images as a dataset over time, an unattended script ran remotely on a WWU computer that has been used to automatically scrape the latest images after $t = 12$ hours during the period of three weeks. This procedure has gathered around $n = 20k$ media images in total, which will be used mainly as test data. A subset of *not_interesting* social media images was also used for training of the negative case.

3.2 Training set for the classification models

A solid training dataset is crucial in order to train models that are able to identify interesting social media images in the mass of irrelevant posts and classify the ones with the defined arms objects. Although image data on arms and alike is sparse [18], two main image data sources have been used for this work, which will be described in the following. After that, multiple data preprocessing steps were needed in order to clean the dataset.

3.2.1 Data sources

A. Google Open Images Dataset. Google's OIDv4 dataset consists of more than 9 million images with a mix of machine-generated and human-verified labels for thousands of classes [35]. Although automated annotation might raise problems such as inconsistent quality across classes [79], it is useful due to its enormous time-efficiency compared to manual annotation. Using an open-source CLI provided on GitHub [75], roughly $n = 30k$ images with relevant classes as introduced in 1.2 have been extracted. They are a useful source because of the similarity to real-world images potentially posted in social media. Other scientific papers also used images from OIDv4 as training dataset [4, 48, 58].
B. Internet Movie Firearms Database. The IMFDb is a database containing weapon and arms imagery taken from thousands of movie scenes. As there is no collective download functionality provided, a scraping script using the library BeautifulSoup [53] has been created for scraping $n = 250k$ images. Parts of the IMFDb were also used as training dataset in previous literature [6, 32, 44, 74].

3.2.2 Data preprocessing

Multiple preprocessing steps have been performed in order to clean the raw dataset before training (see examples in figure 3.2). First, poor quality images were omitted by considering the height and width dimensions, removing every image below a threshold of $n = 10k$ total pixels (e.g., with lower resolution than 100×100 pixels). While also some OIDv4 images did not meet that requirement, this step was especially important because the IMFDb scraping produced some unwanted by-products such as small-size logos, icons and country flags. They all had similarly low quality and therefore fell below the set threshold. The dataset additionally contained many grayscale images, mostly coming from old movies. The decision to remove them was not technical but rather context-based, as movie images dating back to times before color film technology will most likely not suit modern social media imagery. Although for recognizing arms and military vehicles those images could still contribute to the model's robustness, the choice has been made to ensure precise prediction results. The grayscale images were removed by evaluating the image pixel depths using the Pillow library [15]. The script identified images with only 8-bit pixels containing a single grayscale value between 0 and 255 first, and then searched for images with few unique colors (threshold $\leq$5k). This additional step was done because some actually grayscale images were saved in RGB format and had some minor noise in the color channels, therefore consisting of a 3D pixel shape. The reason for this is probably some form of image conversion before uploading to OIDv4 or IMFDb.

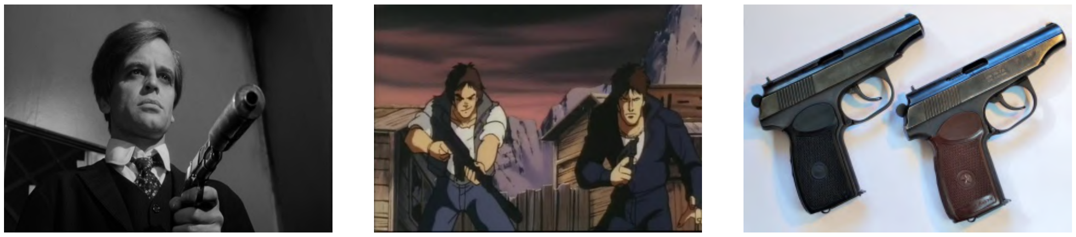


Figure 3.2: Examples of images excluded after preprocessing (left: grayscale image, center: animated/cartoon image, right: duplicate image, $n = 1$ left in training data).

Additionally, the image retrieval through scraping resulted in some duplicates in the dataset, because the n:m-relation (many-to-many) between movies and arms within the database inevitably makes the same images appear multiple times when scraping in a straightforward way (in this case alphabetically by movie title). Mostly, this concerned examples of arms that were used in many movies. To find and omit the duplicates, checksum comparison using the MD5 hash algorithm has been applied (see algorithm 2). Even though there are specific image hash algorithms that can detect very similar images as well [67], the preferred approach was using a conventional encryption hash method to make sure only actual duplicates are removed.

Algorithm 2 Logic to identify and remove duplicate images.

```
$hash\_dictionary \leftarrow \{\}$
for $image$ in images do
    $image\_hash \leftarrow$ hashlib.md5($image$.read()).hexdigest()
    if $image\_hash$ in $hash\_dictionary$ then
        os.remove($image$) ▷ duplicate image
    else
        hash_dictionary[image_hash] = image
    end if
end for
```

Lastly, manual inspection of the data showed that both in Google's OIDv4 and in the IMFDb dataset, many images are not real photographs, but animated graphics such as cartoons, drawings and alike. To tackle that, each image had to run through a detection program (see algorithm 3) that first analyses the amount of colors (threshold $n \leq 120k$). The threshold is way higher as in the previous processing step because it is not about identifying grayscale images but colored animations. Secondly, the script is analysing the color gradient. This is implemented using a bilateral filter (see section 2.1) that applies a Gaussian function in space (diameter of 10px) and another one in intensity differences within the neighbourhood (see figure 3.3).

Algorithm 3 Logic to separate animated images from real photographs.

```
$sim\_threshold \leftarrow s$
$colors\_threshold \leftarrow n$
for $image$ in images do
    $colors \leftarrow$ call calculate_unique_colors($image$)
    $similarity \leftarrow$ cv.compareHist($image$, cv.bilateralFilter($image$))
    if $similarity \geq sim\_threshold$ and $colors \leq colors\_threshold$ then
        os.remove($image$) ▷ image most likely animated
    end if
end for
```

The more similar the histograms of the images before and after bilateral filtering, the smoother the initial gradient. Similarity in this case is calculated as the correlation of both histograms – before and after applying the bilateral filter – averaged across RGB color channels. Manual testing has shown that the best similarity threshold is $s = 0.995$. Based on that, it is possible to sort out images that are most likely not real photographs. The described method is based on common characteristics of cartoons, having fewer colors and smoother gradients within contours [27]. This approach did not work for all images and is further discussed in section 5.2.2.

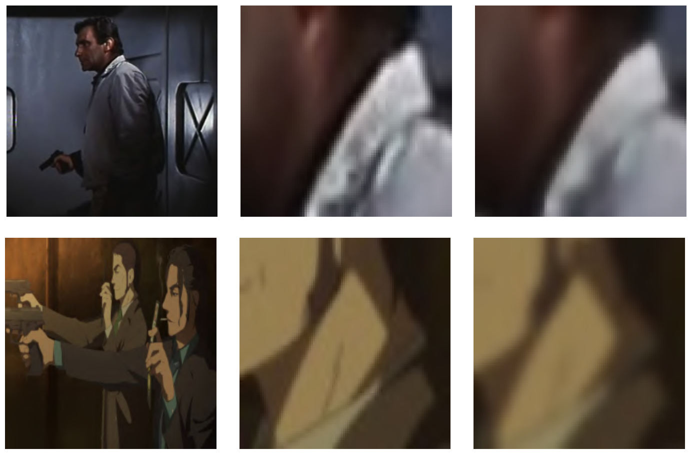


Figure 3.3: Bilateral filter applied to detect animated images (left: original image, center: before filtering, right: after filtering. Similarity $s \leq 0.995$ and unique color count $n \geq 120k$ for real photograph in top and vice versa for cartoon in bottom).

3.2.3 Image annotation tool

While image downloads through the OIDv4 CLI were based on their annotated labels, the IMFDb scraping resulted in a completely unlabeled dataset. To perform supervised training (see section 2.2), an annotation tool was developed, customized for this data. While iterating over and displaying the images, corresponding class labels (namely weapon, vehicle and soldier) were defined using keystrokes. The image names and their mapped classes were appended to a text-file storing all annotation information. An image could also not belong to any class, if none of the classes were present. In this use case it is only necessary to mark images with relevant classes and not to localise them within the image, which is why labels are given on an image-level without any bounding box just like in similar literature [71]. This approach is discussed further in section 5.2. For this work, many hours of manual annotation have been done – covering a good part of multiple $10k$ images, but not all of the available IMFDb samples. The decision to stop labeling at some point was made for time reasons and also reflects considerations such as the logarithmic relationship between network performance and sample size after some decent amount of samples per class [61]. Also, automatically annotating the rest of the unlabeled images based on a model fed with the annotated subset would have been possible. It was not done on purpose because preliminary tests resulted in a recall of $TP = 0.8$ – meaning that every fifth label would have been wrong and forwarded as network input as a consequence. Most likely, this would have resulted in significantly lowered model performances – which is a hypothesis so far and left for future work. The final data distribution within the training image set after all preprocessing steps is visualized in figure 3.4. The two trained models, Model I and Model II, will be described in detail in section 3.3.

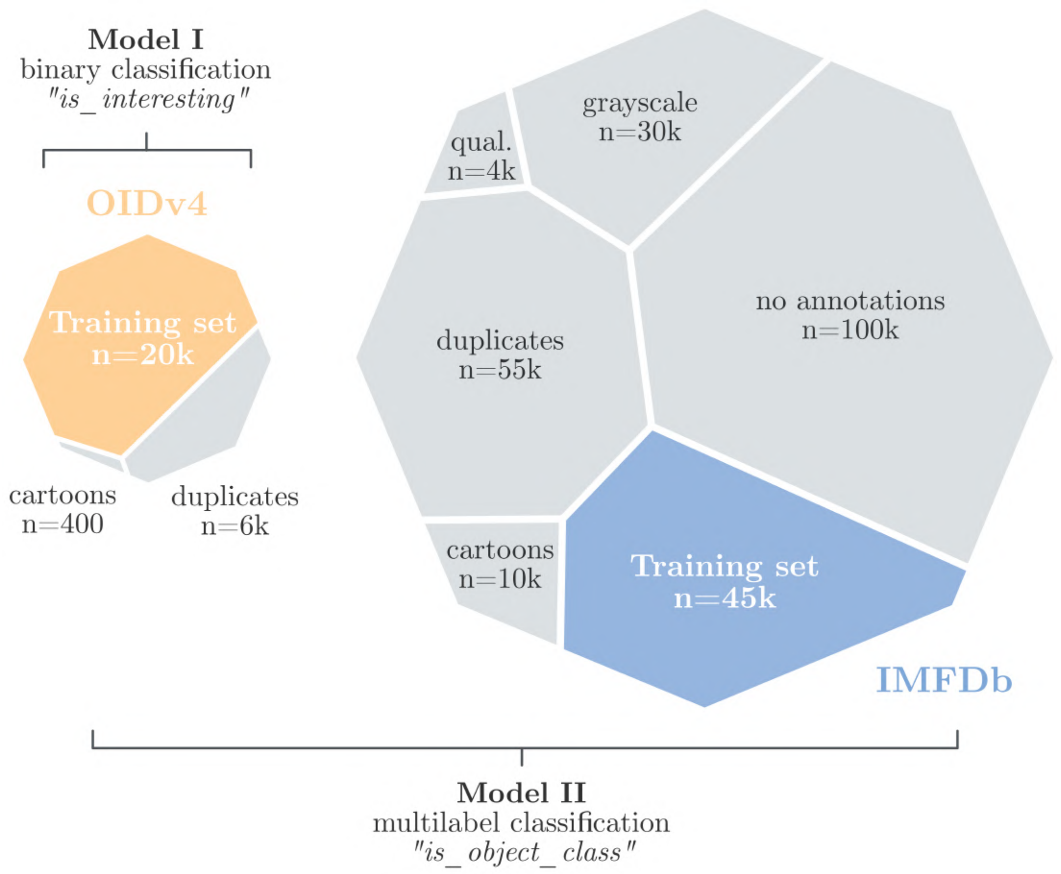


Figure 3.4: Image distribution by filter applied in preprocessing (left: OIDv4 dataset used for Model I & II, right: IMFDb dataset only used for Model II, colored segments are actual training input after preprocessing. Note that even though the IMFDb dataset is way larger than the relevant OIDv4 classes, only a fraction was finally used in training).

3.2.4 The issue of image sequences

As the IMFDb images are movie snapshots, there are numerous image sequences in the training dataset (see figure 3.5). Images taken from different scenes within the same movie are not problematic for the training, but the sequences should be treated carefully. If multiple images from the same sequence were too similar, only one is chosen for training and the rest is discarded. If they were similar in many aspects but showed different angles of objects, all images were forwarded to the final training set. The same is true for geometry: If multiple images inhabited a class in different x, y coordinates, all are used for training. In the latter case, it was necessary to ensure images from the same sequence will be either used in training or validation, not both. This issue is known from other domains such as camera traps in wildlife monitoring, where spatio-temporally correlated sequences [61] (e.g., one image per second) randomly distributed in training and validation may lead to wrongly accurate validation results that the model actually cannot keep up with on new images. It is important to note that those steps have been performed manually during the annotation process and the implementation of a CV-based method could help automate this step in further work.

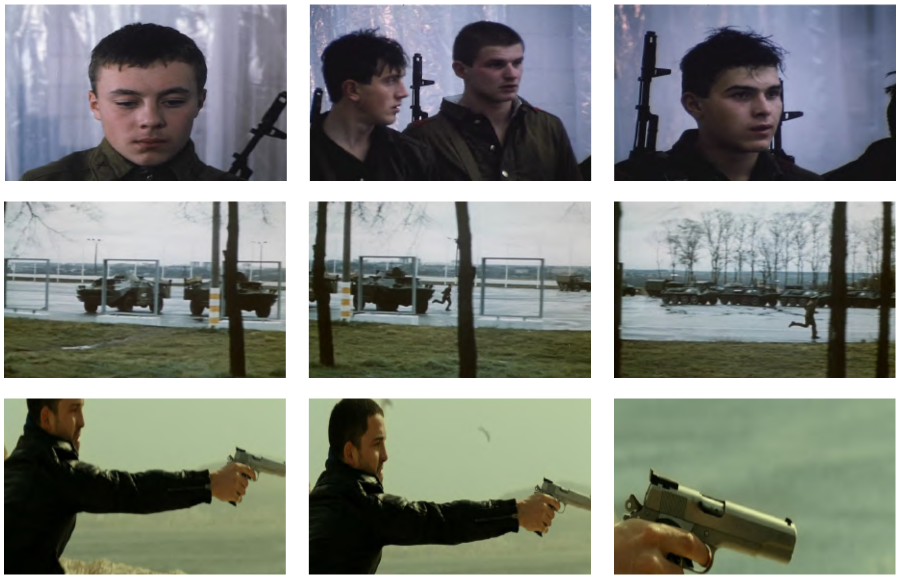


Figure 3.5: Examples for IMFDb image sequences (all three sequences were used for training and kept together in the same set – top & center: different view angles on objects, bottom: different x, y geometry).

3.3 The Classification Models

As introduced in chapter 1.2, the aim of this study is to perform useful image classification to pin down interesting images from conflict zones. Sections 3.1 and 3.2 introduced the relevant training (and validation) as well as test datasets, section 3.2.2 explained all preprocessing measures. In this section, the implementation of two CNN models that were trained to perform the classification tasks will be described.

Two-model approach. The hypothesis for this work is that dividing the classification in two parts will result in a better performance than training a single network. The first model (Model I: *is_interesting*) should be able to differentiate between *interesting* and *not_interesting* images, while the second model (Model II: *is_object_class*) should identify the three mentioned objects (*soldier, vehicle, weapon/arms*) only within the interesting images. This pipeline is supposed to avoid a high misclassification rate (e.g., many false positives), which is discussed in chapter 5.1. In this application, the first model can be viewed as an intermediary element to the second one, forwarding interesting images. Additionally, the two-model pipeline was chosen as for the use-cases, classification of interesting images is most important, and classifying objects within images is useful but not necessary. The complete structure incorporating the training data and the classification models is visualized in figure 3.6.

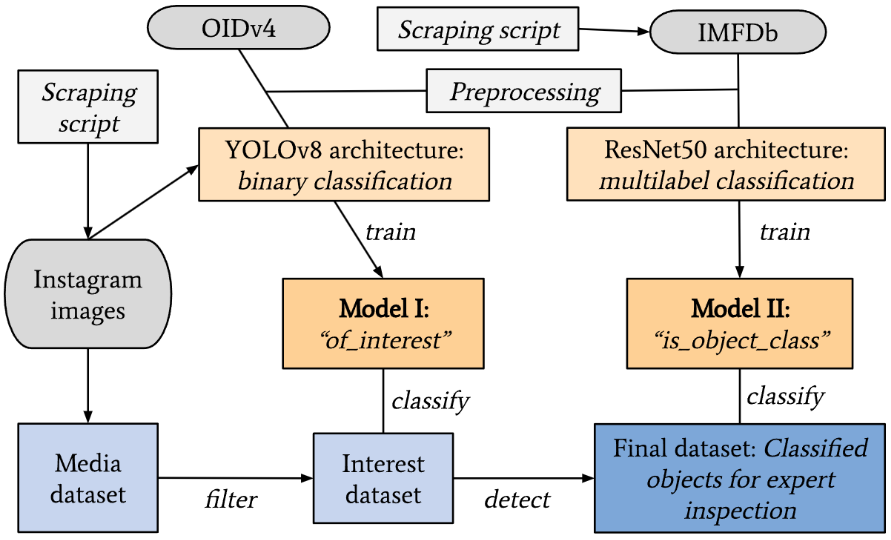

OIDv4
Scraping script
IMFDb
Scraping script
Preprocessing
YOLOv8 architecture: binary classification
ResNet50 architecture: multilabel classification
train
train
Instagram images
Model I: "of_interest"
Model II: "is_object_class"
classify
classify
Media dataset
filter
Interest dataset
detect
Final dataset: Classified objects for expert inspection

Figure 3.6: Process flowchart of the methodology (grey: train/val/test data, orange: CNN-based classification with the two models, blue: envisioned data pipeline similar to [28]).

Both models were trained on an AMD Radeon Pro WX3100 GPU running on a computer with an Intel i9-7900X processor. Although cloud-based training or using the WWU's high performance computing cluster PALMA II would have been more efficient in runtime, the choice was to carry out the whole process locally in order to have full control, implement lots of intermediate tests and generally get a better understanding of how the network training functions when run on a local machine.

3.3.1 Model I – Binary classification task

Model I (*is_interesting*) has been trained to perform a binary classification task directly on the social media dataset that is full of unfiltered images. The prediction on each image will either be *interesting* or *not_interesting*, focusing on the usefulness of images just like in [21]. The network is trained and validated on the full OIDv4 dataset for the positive class (*interesting*) and on a sample of negative social media images for the *not_interesting* class. The training is based on a pretrained YOLOv8m model. A medium-sized network consisting of 25.9 million parameters was chosen because it is a good trade-off between performance and training runtime, with the mAP validated on the COCO 2017 dataset flattening out towards the larger versions while latency increaces [29] (see figure 3.7).

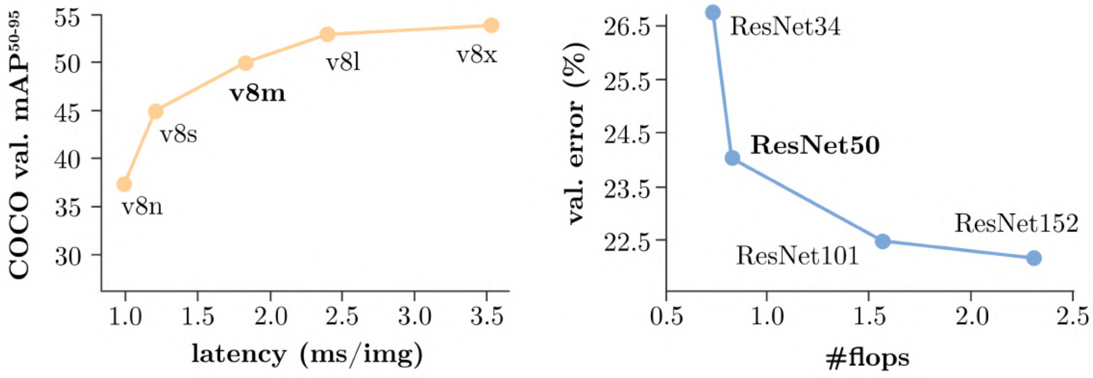


Figure 3.7: Time efficiency of pretrained models (left: YOLOv8 [29] medium compared to other sizes, right: ResNet [26] 50 compared to other sizes).

Data augmentation was used applying horizontal flips and rotations to the images. A dropout probability of $p = 0.2$ was added for robustness to noise (p.266, [23]) just like in similar work [2, 18, 33, 82] and is very suitable for the training data size [65].

The batch size per iteration is chosen to be 16, allowing the model to converge towards better weights quicker, but with higher variance for a duration of $k = 100$ epochs. As for the optimizer, the ULTRALYTICS [29] library enables automatically choosing a method based on dataset size and iterations – Stochastic gradient descent (SGD) was selected over other methods automatically, which is a common method [38]. The learning rate is automatically adjusted for each iteration, in this case starting low at $\lambda(0)$, increasing quickly to $\lambda(4) = 9,8 \cdot 10^{-3}$ and then gradually dropping to $\lambda(100) = 2,9 \cdot 10^{-3}$. This method is inherent to the library's training settings, introducing a short learning rate warm-up period for the first epochs to stabilize training early on [29].

Model I has been trained with four different class biases and $n = 3$ trainings each towards positive (*interesting*) and negative (*not_interesting*) classes (see table 3.1): negative, no bias, positive and strong positive. This was done to evaluate training bias influence on the trade-off between recall and specificity. Those four biases are introduced through the input data in the training set already, and the same bias distribution is also used in the validation set.

Table 3.1: Positive/negative biases introduced in training to evaluate trade-off.

bias	**relation**	**overrepresented class**
negative	3:2	class *not_interesting*
no bias	1:1	equal class distribution
positive	3:2	class *interesting*
strong positive	3:1	class *interesting*

3.3.2 Model II – Multilabel classification task

Model II (*is_object_class*) performs a multilabel classification task, where each image can be labeled as containing a weapon, vehicle and soldier. It was developed using the PYTORCH framework based on the TORCH library. The network is trained and validated on the annotated part of the IMFDb dataset plus the full OIDV4 dataset (see section 3.2.1). It is built on the pretrained ResNet50 [24] architecture and trained for $k = 100$ epochs. Similar to the first model, a medium-sized network with 50 layers and 25.6 million parameters yields optimum trade-off between time efficiency and learning performance [26] (see figure 3.7). Because ResNet is a stand-alone network not embedded in a full development framework such as YOLOv8, the architecture modification and finetuning was done manually. In contrast to the binary classification, batch size was chosen to be 64 to stabilize the learning process and converge smoothly. This choice is based on the assumption that multilabel classification on three classes is a more difficult task than binary classification. For the second model, Adam optimizer [34] has been used as a robust choice (p.309 [23]). A dropout probability of $p = 0.4$ was introduced, purposely greater than in *Model I* to also increase robustness. The ReLU activation functions [1] used in the 16 residual blocks with convolutions of ResNet50 have been left untouched, and a Sigmoid activation function was introduced in the last-layer to fit the output to be in range $(0, 1)$ (p.66 [23]). In contrast to *Model I*, no class bias was introduced in the training and class imbalance was accounted for through weights in the loss function. *Model II* has also been trained for $k = 100$ epochs. All hyperparameters and settings for both models are summarized in table 3.2.

Table 3.2: Training settings and hyperparameters for both models (* automatically set using the ULTRALYTICS library).

setting/hyperparameter type	**Model I**	**Model II**
pretrained architecture	YOLOv8m	ResNet50
type of classifier	multi-class (binary)	multilabel
epochs (patience)	$k = 100(50)$	$k = 100(100)$
batch size	$n = 16$	$n = 64$
dropout	$p = 0.2$	$p = 0.4$
learning rate	$\lambda(0) = 3,3 \cdot 10^{-3}*$	$\lambda = 10^{-2}$
resize input images	$640 \times\ 640$	$224 \times\ 224$
optimizer	SGD*	Adam
activation	Mish	ReLU & Sigmoid (last)
loss function	BCE	BCEWithLogitsLoss

3.3.3 Unfreezing the last convolutional layer

To allow for learning new features, the last CNN layers of the two pretrained architectures have been unfrozen. This is a common technique in finetuning pretrained architectures [2, 68] (see section 2.4). To demonstrate, figure 3.8 shows the network's simplified backbone structures. For Model I based on the YOLOv8m network, this meant unfreezing weights for the last (5th) convolutional layer and the following C2f block, which is basically consisting of a bottleneck with downsampling steps inbetween another two convolution operations [29]. Also, the last pooling layer (SPPF) is unfrozen. The same logic was applied to Model II based on the ResNet50 architecture, where weights of the last residual block of convolutions was unfrozen together with the final pooling and fully-connected layer. As a result, both models are based on low-level feature representations from the pretrained architectures, and enable weight adjustment in high-level CNN layers according to the loss.

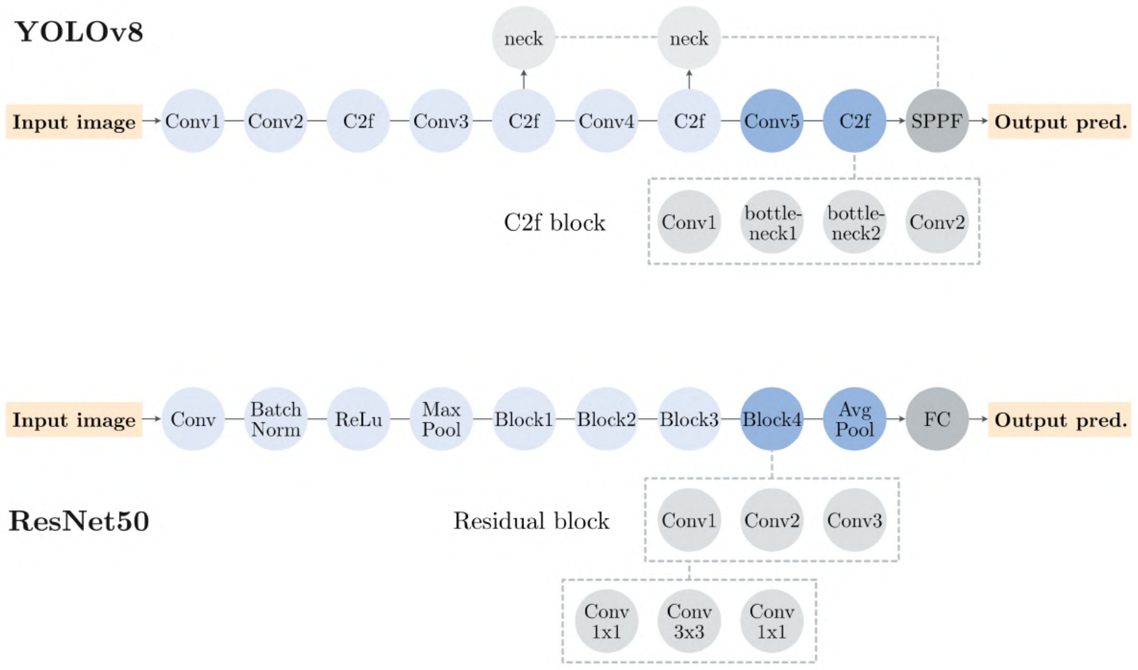


Figure 3.8: Simplified backbone structure of pretrained architectures with highlighted last-layer unfreezing (top: YOLOv8m, bottom: ResNet50).

Chapter 4: Evaluation of Results

This chapter will outline the training results and evaluate them in the context of the defined use-cases and a possible application. Just like in section 3.3, the two models will be assessed successively by looking at common performance metrics in the training process and the prediction performance on the test set. As described in 3.1, the test set is consisting of the scraped social media images. Especially, the trade-off in recall and specificity will be highlighted. After that, a closer look at the different conflict regions (see section 1.3) will reflect on geographic differences.

4.1 Binary classification of Model I

Training. The training process of Model I, the binary classification model named *is_interesting*, is analysed by looking at the confusion matrix, loss and accuracy on the validation set over runtime with epochs $k \longrightarrow 100$. For each bias group towards positive or negative images (as defined in section 3.3.1), multiple training runs ($n = 3$ in this case for each of the four biases) have been performed with different random seeds and were averaged per bias. This ensures that actual patterns can be differentiated from random noise in the training process – which could not be differentiated when just running a single training for each group. Overall, the network's training looks as expected and indicates successful learning. The high TP and TN rates during training demonstrate that the model predicts well on the validation images. Along with the biases introduced towards the two classes, specificity behaves as one might expect and drops with a positive bias – and the recall rises (see figure 4.1) as seen in the confusion matrix plotting predicted against true class labels. Looking at both matrices, it is visible that the TN rate span is slightly more affected by a class distribution bias with $0.96 \leq spec \leq 0.98$ than the TP rate with $0.98 \leq spec \leq 0.99$, but overall both are almost stable.

Those training results are also reflected by the averaged loss function per bias group (see figure 4.2). After a steep decrease, the validation loss flattens out gradually from

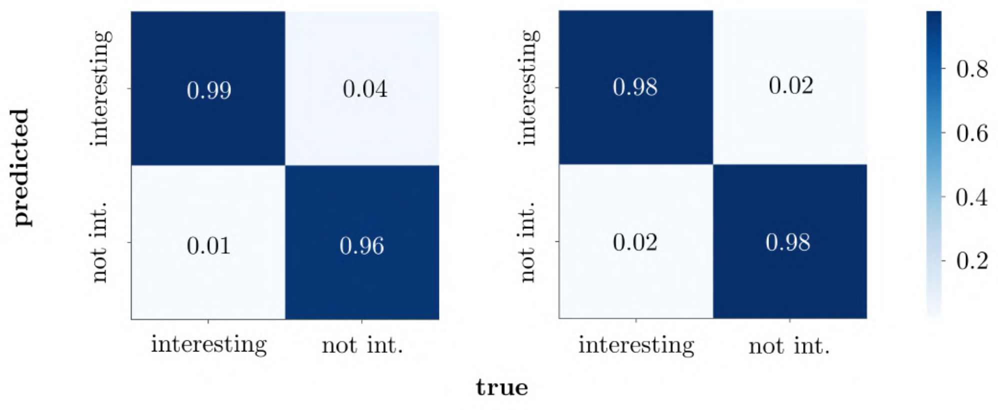


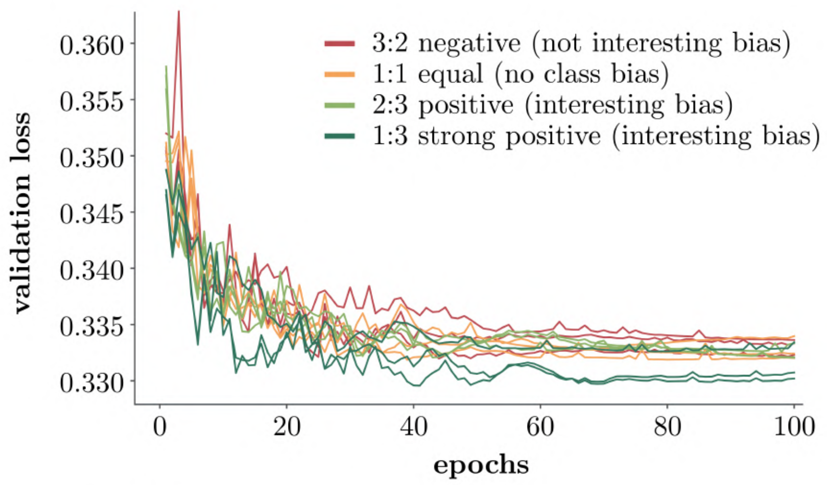
Figure 4.1: Confusion matrix for Model I on validation set (left: strong positive bias, right: negative bias).

epoch $k = 60$ onwards for all biases. Interestingly, the lowest loss (therefore presumably also the best training result) occurs in mid-training for most biases (i.e., loss(min) of 0.33 at epoch $k = 42$ for strong positive bias) like in [33], indicating a slight overfitting behaviour afterwards until training end at epoch $k = 100$. Theoretically, for those biases the training process could even be terminated at epochs $40 \leq k \leq 45$ with the current setup. Only for the (weak) positive bias, $k(best) = k(last)$ is true (see table 4.1).

Even though the learning curve behaves similarly for all biases, there are some differences. Looking at the average loss per bias, it improves with the dominance of positive (i.e., *interesting*) images. Whereas the negative, equal and positive biases all

Table 4.1: Last $k = 100$ and best validation loss for binary classification of Model I by bias group (with best epoch k indicating when the training process reached the lowest validation loss).

bias	**loss (*k=100*)**	**loss k(best)**	**k(best)**
not_interesting	0.33231	0.33197	45
equal_class	0.33202	0.33186	40
interesting	0.33205	0.33205	100
strong_interesting	0.33074	0.32957	42



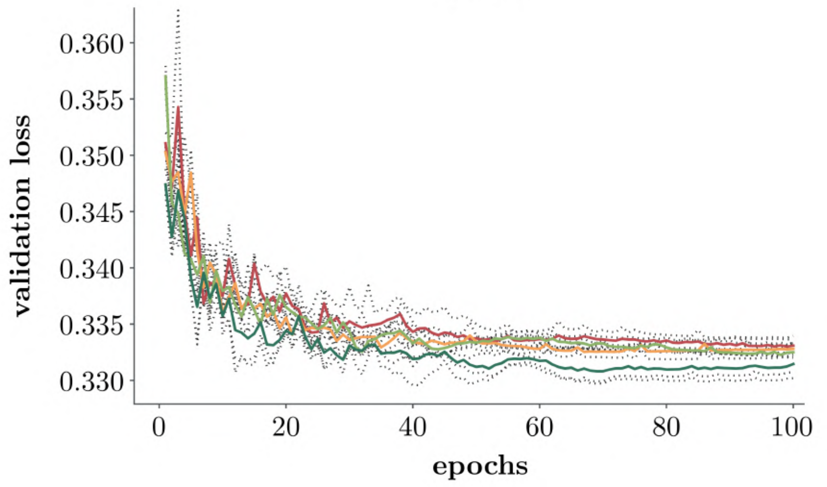


Figure 4.2: Validation loss per bias with n = 3 per group (negative, equal, positive, strong positive) for Model I (top: all training runs for the bias groups, bottom: average validation loss per class bias group, where colored lines indicate the calculated average loss per bias and grey lines represent all actual loss values from top graph, which are the actual training runs).

converge towards a very similar loss from epoch $k = 60$ onwards, the strong positive bias outperforms the others. This finding suggests that introducing a significant bias (with a 3:1-relation) towards interesting images in the training can be beneficial for the training process of Model I. This is also mirrored by the validation accuracy (see figure 4.3), which increases almost inversely to the validation loss in figure 4.2 and shows best results for the strong positive bias, with best accuracy results mid-training at epoch $k = 49$.

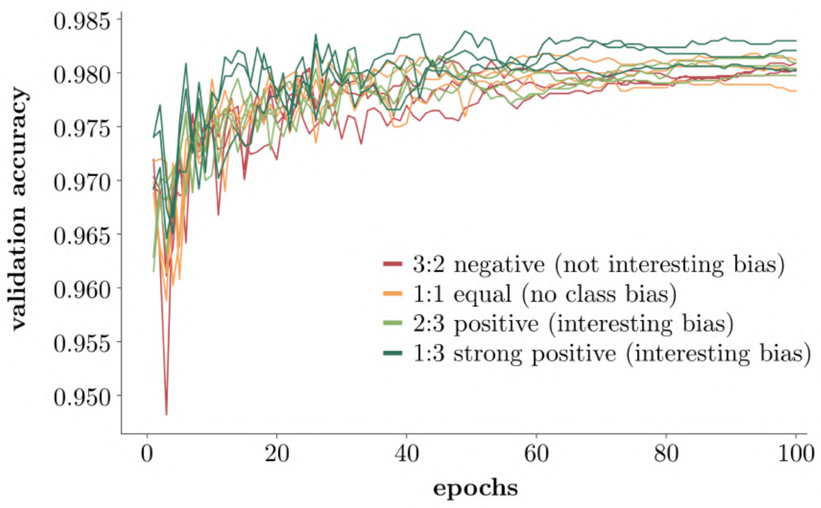


Figure 4.3: Validation accuracy per bias with n = 3 per group (negative, equal, positive, strong positive) for the binary classification task (Model I).

Evaluation on Social Media Images. The evaluation metrics show that Model I has high specificity scores of $0.95 \leq TN \leq 0.98$ across all biases and a moderate recall of $0.71 \leq TP \leq 0.79$ (see table 4.2). The resulting accuracy is rather stable with $0.92 \leq acc \leq 0.94$, as well as the f1-score.

This finding is also represented in the confusion matrix on the test images (see figure 4.4). In other words: The model is better at knowing which images are not interesting than classifying the interesting ones, similar to [28]. In a scenario with way more negative than positive samples, this is not unexpected – high TN rates influencing the overall model accuracy should be analysed carefully (p.411 in [23]). In this context, the test setting can be compared to cases in the medical domain, where low prevalence

Table 4.2: Evaluation metrics on test set for binary classification of Model I.

bias	acc(full)	acc(equal)	rec/sens	spec	prec	f1
not_interesting	0.93	0.85	0.71	0.98	0.87	0.78
equal_class	0.93	0.86	0.72	0.97	0.86	0.78
interesting	0.94	0.88	0.77	0.97	0.84	0.80
strong_interesting	0.92	0.88	0.79	0.95	0.78	0.79

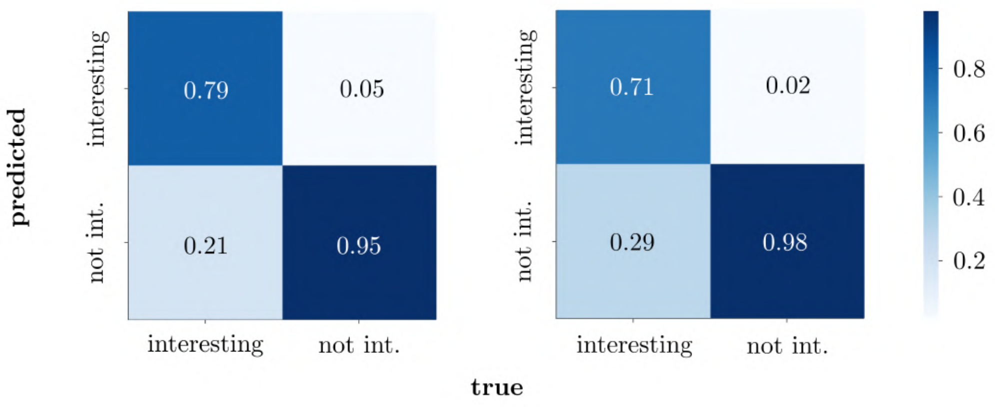
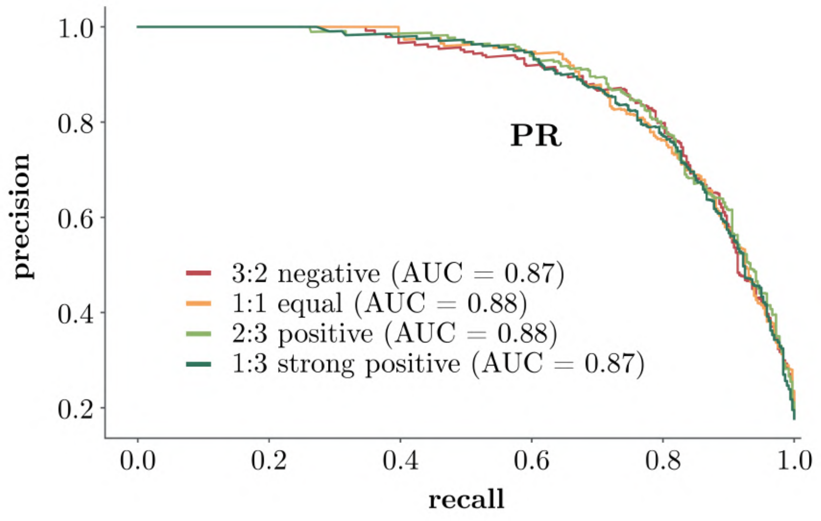


Figure 4.4: Confusion matrices on test images for the two most varying biases of Model I (left: strong positive bias, right: negative bias).

of a condition (that is positive, in this case class *interesting*) in the test set affects performance metrics [69]. This class distribution is also the reason why the accuracy metrics are closer to specificity than to recall values. Also, the recall increases with a bias towards the interesting images, whereas specificity behaves vice versa. So, as in many other studies [18, 69, 74, 82], there is a trade-off between TP and TN that can be influenced with an image bias. The prediction accuracy indicates that the best trade-off for performance is using a (weak) positive bias towards interesting images with $acc = 0.94$ and $f1 = 0.80$ (see figure 4.2).

The same trend can also be seen plotting the TP rate against the FP rate in a ROC curve [10] (see figure 4.5), where the area under the curve is $AUC = 0.96$ for the (weak) positive bias and just 0.01 lower for the other biases. This is in line with similar work [32, 71] and demonstrates that in an application, most of the forwarded positive images from Model I are actually correctly interesting. With $AUC = 0.88$ for the equal and positive bias class, similar results can be seen plotting precision against recall (PR). Nevertheless, the variance in accuracy measurement across classes is low, indicating similarly successful results for all image biases.

It is important to note that in the test dataset, the share of interesting images is very low as indicated – which is also why the accuracy is higher evaluating on the full (or raw) dataset compared to when randomly sub-sampling not-interesting images to the same amount, where $0.85 \leq acc \leq 0.88$ (see table 4.2). The accuracy



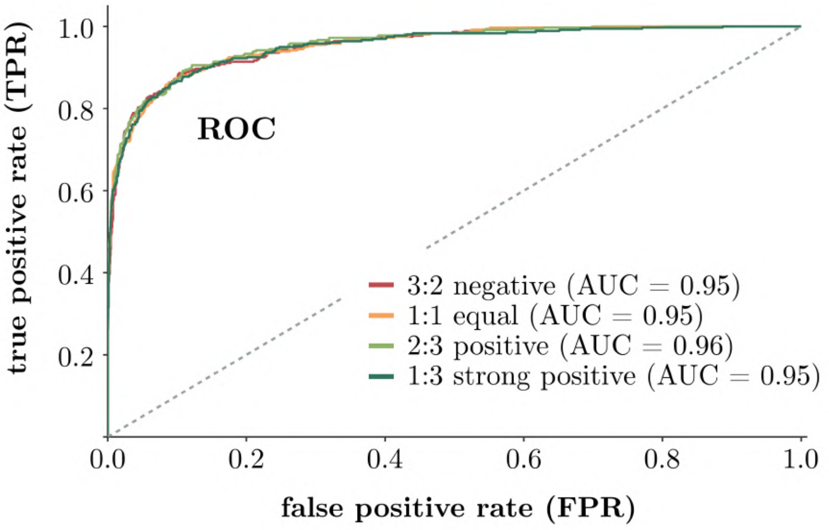


Figure 4.5: PR and ROC curves for all four biases of binary classification model, best training visualized (top: precision/recall, bottom: ROC).

on an equalized test set with images $n(interesting) = n(not\_interesting)$ is given for comparison. For an application of the model, the precision is important as it states how many images will actually be forwarded to Model II and finally to the expert researchers. With $prec = 0.87$, a bias towards not-interesting images is best, dropping towards $prec \longrightarrow 0.78$ for the strong positive bias. This result makes sense because of the model's strong ability to classify TN in comparison to TP images.

Consequently, different training biases should be used when considering the following requirements:

- **Requirement I:** If the model should correctly classify as many images as possible, a weak positive image bias should be used in the training (max. *acc* & *f1-score*).
- **Requirement II:** If the model should classify maximum actually interesting images correctly, no matter how many false positive predictions, a strong bias towards interesting images should be used in the training (max. *rec*).
- **Requirement III:** If the model should classify minimum actually not interesting images falsely, no matter how many false negative predictions, a bias towards not interesting images should be used in the training (max. *prec* & *spec*).

Although in general, low validation loss often also correlates with better prediction results on the test data, this does not have to be the case. The findings from evaluating the first model *is_interesting* demonstrate this: While the training behaviour is clearly best for the strong positive bias, the actual prediction performance on social media images varies for different biases, and which trade-off should be used depends on the use-case requirement of the application.

Testing on IMFDb. As the binary classification model has just been trained on the OIDv4 images (see section 3.3.1), it is also possible to test on the IMFDb data for comparison. This has been done for the best performing model (as described, the one with a weak positive image bias). When replacing the interesting social media images with IMFDb images while leaving all negative instances untouched, the recall is drastically increasing to $TP = 0.94$ (see table 4.3).

The specificity obviously stays unchanged, as the IMFDb only contains positives and negative population is the same as in social media testing. The high TP rate

Table 4.3: Evaluation metrics for binary classification of Model I on IMFDb images.

test images	**acc**	**rec/sens**	**spec**	**prec**	**f1**
IMFDb subset (positive_bias)	0.97	0.94	0.99	0.99	0.96

results in an overall accuracy of $acc = 0.97$ and a precision/recall balance of $f1 = 0.96$ for the weak positive bias. Because the network very accurately predicts both classes, the IMFDb training set is most likely very similar to the OIDv4 set, whereas it is more difficult for the model to generalize on the (interesting) social media images as described above. This finding underlines a strong prediction behaviour of the model on unseen images in general, while also pointing out difficulties with social media images – but this effect can also be influenced by choosing the model with a (weak) positive bias for comparison with IMFDb images. It suggests that including some interesting social media images to the training might be useful for further development in order to also achieve a recall of $rec \geq 0.8$ and precision $prec \geq 0.9$ on new test images.

Prediction confidence levels on test images. For the binary classification model, strong confidence scores (see section 2.2) towards the two classes are noticeable (see figure 4.6). In most cases, the network is very sure about classifying an image with a confidence of $p > 0.95$, often even at $p = 1$. The scores are even visible for false (top-left: FN, bottom-right: FP) predictions. This finding might be the result of some bias in the training data, such as different objects on similar backgrounds [56]. As backgrounds differ in the training images, this can also be a consequence of the classification model itself – because classifiers tend to overestimate confidences to ensure ambiguous images are separated from confident predictions more clearly (p.439ff [23]).

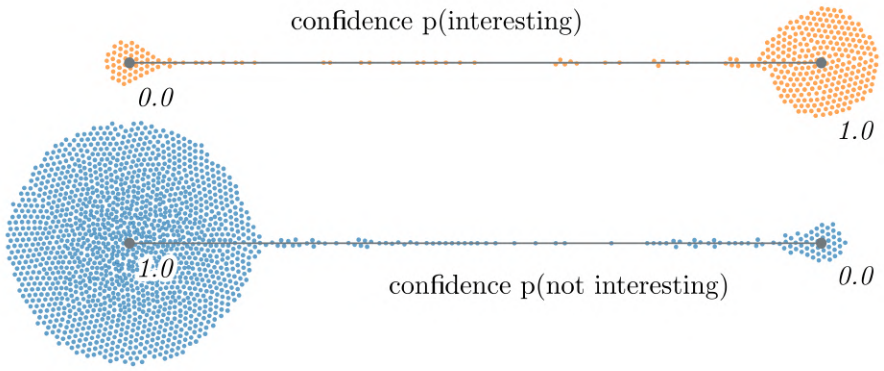


Figure 4.6: Confidence scores for binary classification of Model I, subset of $n = 2000$ (top: class *interesting*, bottom: class *not_interesting*).

4.2 Multilabel classification of Model II

Training. In contrast to the first model performing the binary classification, Model II *is_object_class* is not trained with different biases (see section 3.3.2). Instead, weights (see section 2.2) in the training's loss function accounted for class imbalance and therefore equalized biases due to varying image sample size in the training data. The original image distribution, following the logic $n(arms) > n(soldier) > n(vehicle)$, has been accounted for with compliant weights added to the BCE loss function (see section 3.3.2). Many finetuning efforts have been done (see figure 4.7).

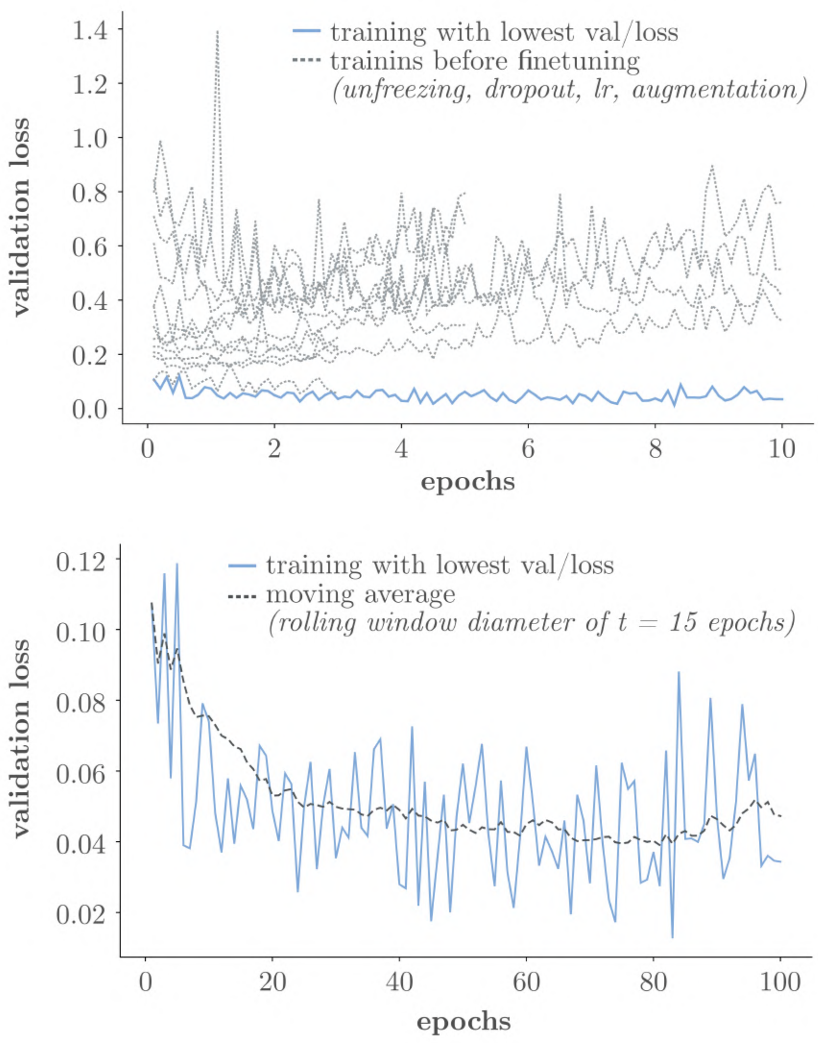


Figure 4.7: Validation loss for multilabel classification of Model II (top: all trainings during hyperparameter adjustment, bottom: training with lowest validation loss).

This included layer unfreezing, dropout introduction, learning rate adjustment and image augmentation of the pretrained ResNet50 architecture – which was needed in order to receive a decreasing validation loss with increasing training time compared to epoch $k = 0$.

For many hyperparameter setups, the loss did not drop and often increased during training. Even for satisfying evaluation metrics on the training data, the validation loss decreases most of the trainings. The best training behaviour could be achieved with a dropout probability of $p = 0.4$ (see 3.3.2) and unfreezing the last three network layers – which is the last residual block, pooling and fully-connected layer of ResNet50 (see section 3.3.3). As epochs $k \longrightarrow 100$, the validation loss drops from 0.107 at $k = 1$ to 0.013 at $k = 84$, before slowly increasing again. As for Model I, $k(best) \neq k(last)$.

For the training with the best loss, the TN and TP rates of the three classes differ (see figure 4.8). In multilabel classification, a confusion matrix as in multiclass classification is not useful [25], but each class can be evaluated individually and treated as a binary prediction. While TN rates are similar across classes with $0.89 \leq spec \leq 0.93$, the TP rate for the class *vehicle* with $rec = 0.89$ is higher than for both other classes with $0.76 \leq rec \leq 0.80$.

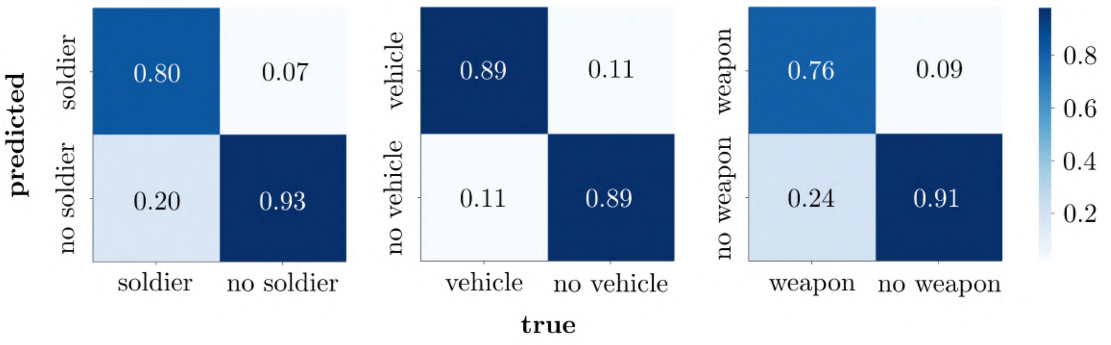


Figure 4.8: Confusion matrix on validation images for multilabel classification of Model II (left: class soldier, center: class vehicle, right: class arms/weapon).

Evaluation on Social Media Images. Model II was tested on all positive social media images filtered by Model I. The prediction of the multilabel classification model does not produce satisfying results in terms of their quality for an application. Even after weighting image classes to account for unequal distribution, results across classes are highly varying (see table 4.4).

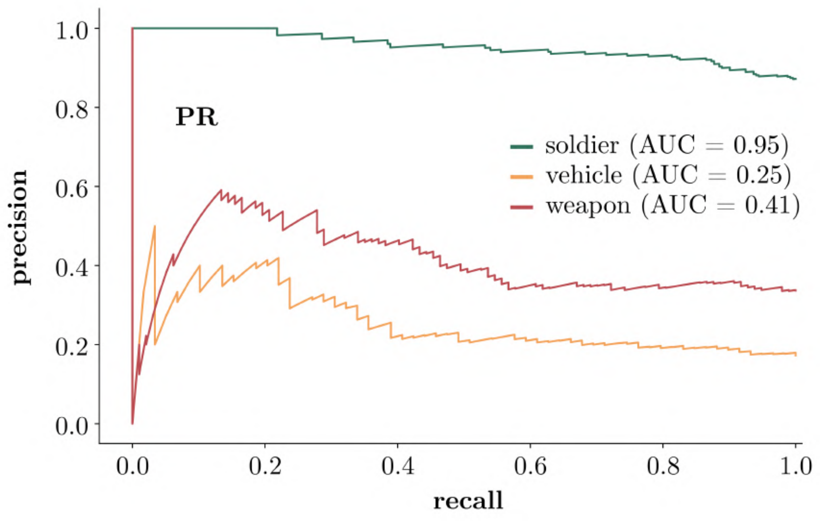
Table 4.4: Evaluation metrics for multilabel classification of Model II.

class	**acc**	**rec/sens**	**spec**	**prec**	**f1**
soldier	0.78	0.81	0.60	0.93	0.87
vehicle	0.82	0.30	0.99	0.79	0.43
arms/weapon	0.61	0.46	0.71	0.42	0.44

Although with $rec = 0.81$, the TP rate for the class *soldier* is high, the network mostly fails to predict the class *vehicle* and also struggles with the class *weapon/arms* with $rec = 0.46$, which is as low as chance (where $rec = 0.5$ could be expected). Logically, the TN rate behaves inversely with a high specificity of $spec = 0.99$ for class *vehicle*, but it is rather a byproduct of almost not predicting that class at all. The overall accuracy of $0.78 \leq acc \leq 0.82$ for the two extremes *soldier* and *vehicle* should not be mistaken for a sufficiently accurate prediction, as this is just the combination of the highly opposite TP and TN rates. A better picture is given by the f1-score, which is below acceptance for vehicles and weapons/arms classes and only satisfying for soldiers with $f1 = 0.87$. The given prediction results on the social media test images pose a significant difference to metrics on the validation set. This finding most likely means that the multilabel classification of Model II is overfitting, because there is a clear gap between training/validation and test errors (p.108ff [23]). A training data bias towards the class *soldier* is likely, which could occur due to their dominant presence in most of the images. In contrast, the high gap between TN and TP rate for vehicles suggest the model did not generate high-level understanding of the class.

Along with this, the ROC area under the curve only shows a reasonable result for the class *soldier* with $AUC = 0.75$ and produces curves just little above random chance classification with $AUC(vehicle) = 0.61$ and $AUC(weapon) = 0.57$ (see figure 4.9). While low FP instances prevent the AUC to be lower for the vehicle class, the combination of low recall and specificity for the weapon class results in worst AUC.

The same class differences are also present in the PR curve: The class *soldier* is close to an ideal curve [64] with $prec > 0.8$ for all recall values and $AUC = 0.95$. But the two other classes show a completely different precision-recall behaviour with significantly lower AUC. In previous research [9], this pattern is likely to occur in bi-normal class distribution when applied to test data, which is a scenario where the normal distributions of the classes (e.g., in this case class *weapon* or not) are not very distant from each other, minimizing the discrimination ability of the model between



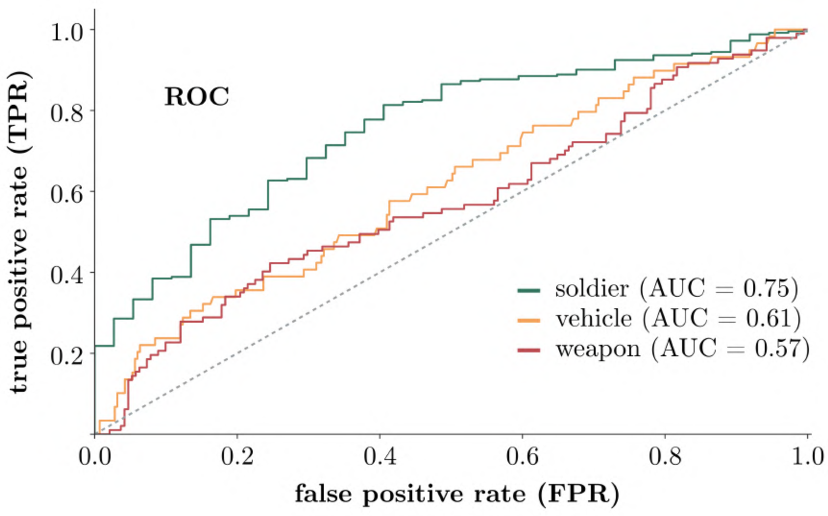


Figure 4.9: ROC curve for multilabel classification of Model II by class.

classes – for example compared to a more separate bi-beta class distribution [9].

A predominant issue for the multilabel classifier for this work is, that the soldier class is actually present in almost all of the images, most likely influencing the other class predictions – even though this should ideally not be the case.

Threshold optimization. The described classification metrics are based on a confidence threshold (see section 2.2) of $p = 0.5$ – so only if the probability is $p > 0.5$, a class is predicted. This is common in research. On application-level, the probability threshold can be modified manually, allowing the network to predict certain classes also with lower confidence. Although this is only necessary if the underlying model struggles to balance the classes correctly, threshold optimization can be a useful and practical mechanism. For example, increasing the confidence to $p = 0.75$ for the class *soldier* can raise its TN rate to $spec = 0.71$, or lowering it to $p = 0.25$ for the class *arms/weapon* will increase the TP rate to $rec = 0.62$. Then again, false predictions in the opposite case will be raised, not solving the issue but calling for a practical trade-off – which is based on application requirements as defined in section 4.1. For instance, an application not requiring highest accuracy but aiming at most possible positive class predictions can be implemented with lowered prediction confidence thresholds of $p < 0.5$. Of course, as $p \to 0$, more and more classes are predicted, which eventually will decrease specificity towards $TN \to 0$.

Layer-wise activation. Knowing about the insufficient performance of Model II, it is possible to take a closer look at the prediction at CNN layer-level for true and false cases. As it is hard to tell how the network predicts a certain class, mapping the activation per layer for each class can give useful insights (see figure 4.10). For the examples in the plot, it can be seen how different activation patterns throughout the network layers lead the model to a correct prediction. In those cases, the activation develops from reacting to low-level edges and shapes in earlier convolutions to high-level features, pointing to the most relevant part of the classes (in the view of the model) but not precisely segmenting it from the rest of the image. This indicates successful prediction behaviour and is also in line with activation heatmaps (GradCAM visualization [60]) from other relevant studies [2, 30].

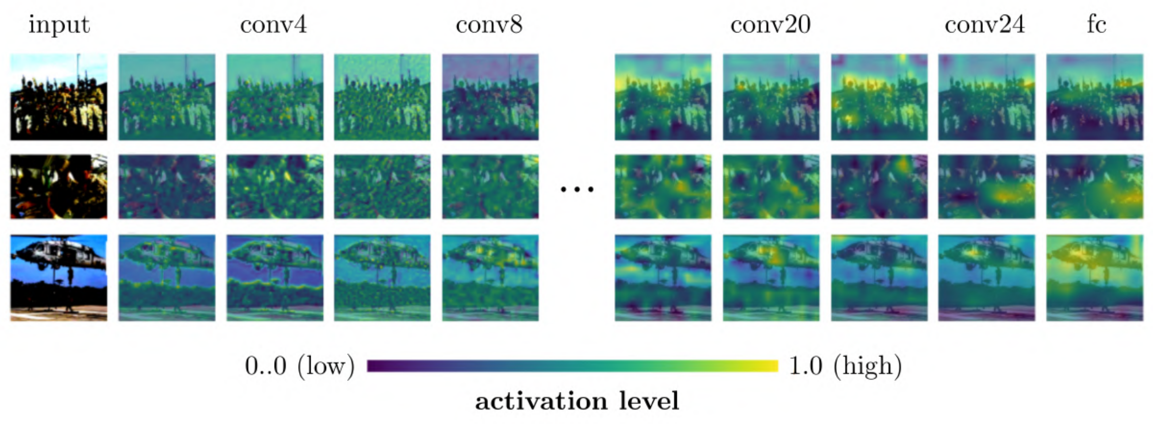


Figure 4.10: Layer-wise GradCAM activation heatmap [60] for every second convolutional layer (top: weapon, center: soldier, bottom: vehicle).

In the social media test dataset, most images contain multiple classes. For those images, $n(classes) > 1$ can be predicted. If $n(classes) = 3$ are visible on an image, the model should also recognize all three of them. For some cases, this works well, as class-wise activation of the last fully-connected CNN layer demonstrates (see figure 4.11). In cases like these, the model did not only predict soldiers and ignore other classes.

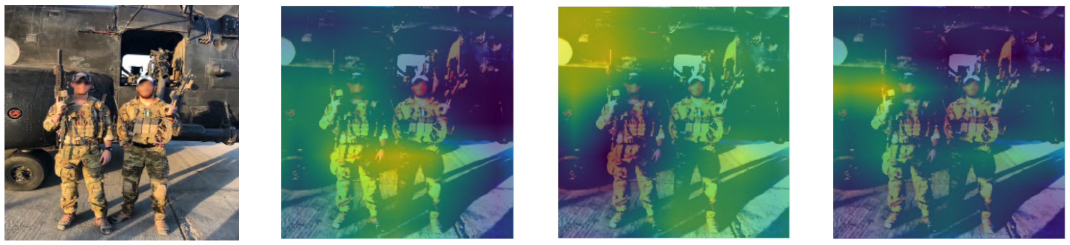


Figure 4.11: Image with all three classes and three TP (left: original image, center-left: last-layer activation for TP class soldier, center-right: last-layer activation for class vehicle, right: last-layer activation for class arms/weapon).

But reflecting on the described evaluation metrics, this recognition of classes failed for many images. One issue is that in images with $n(classes) > 1$, often only one predominant class is recognized. For example, as soldiers (or other uniformed people) are present on most of the images, vehicles are often treated as background and small arms as part of the soldier – possibly because they are occluded by the shape of the body (see figure 4.12). This is also in line with other research, where smaller objects are harder to classify [80]. Whereas in the example, the first activation clearly pinpoints the soldier in the image, activation for the other two classes is rather dispersed, focusing

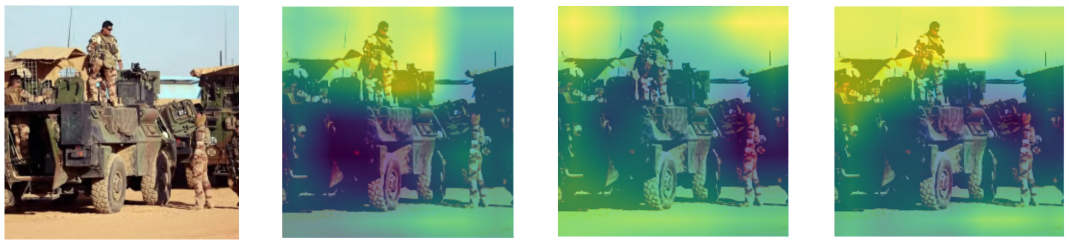


Figure 4.12: Image with all three classes and one TP / two FN (left: original image, center-left: last-layer activation for TP class soldier, center-right: last-layer activation for class vehicle, right: last-layer activation for class arms/weapon).

on the image corners. While this is not uncommon for an 'unsure' network to show high activation at less semantically meaningful areas [12] – especially in corners – this can also be a sign of biases due to insufficiently distributed training data [8]. The decision to include soldiers into the classification to enlarge the detection radius in an actual application on social media images might therefore be obstructive in this context. Overall, the displayed activation distribution rather visualizes the mentioned classification issues, without pointing to their causes.

4.3 Two-model approach

The application of the two-model approach (see section 3.3) has been evaluated in a successive way, where Model I forwards positive images to Model II. Initial tests to also apply Model II directly on the social media test set including all *not-interesting* images showed high error rates. More specifically, in those tests the multilabel classifier has also been trained on *not-interesting* social media images just like Model I and a fourth class (called *none*) was added to the ResNet50 output. The trained network mostly predicted the class *none* on the test set. Therefore, Model I and II cannot be directly compared in their performance on test images. Even after demonstrating insufficient prediction quality of Model II on social media images, the two-model approach resulted in a beneficial outcome: The binary classification (of Model I), which is most important for the use-cases, is producing promising results as presented in section 4.1 – and its performance is not influenced by the multilabel classification issues raised in section 4.2.

4.4 Differences between conflict regions

As introduced in section 1.3, this work also aims to identify differences across the selected conflict regions, which are focus regions of peace research and arms control measurements. Following the finding that the binary classifier produces more satisfying results compared to the multilabel classification as demonstrated in previous sections, this regional analysis is based on Model I only. Major differences can be seen looking at the evaluation metrics by region (see table 4.5). All predictions are based on the (weak) positive biased model, as it performs best (see section 4.1). Whereas the TN rate is almost equally high with $0.95 \leq spec \leq 0.99$, the TP rate shows huge variance across regions with $rec \geq 0.83$ for Afghanistan and Syria, and $rec \leq 0.68$ for Yemen, Ukraine, Venezuela and the Sahel. Accuracy is rather stable as it balances out the

Table 4.5: Share of interesting among all social media images in test set (based on test set prediction from binary classification).

region	n(all)	int/all	acc	rec/sens	spec	prec	f1
Afghanistan	2188	0.0612	0.94	0.88	0.95	0.81	0.84
Syria	2352	0.0489	0.95	0.83	0.97	0.87	0.85
Yemen	1435	0.0335	0.91	0.58	0.99	0.94	0.72
Sahel	1729	0.0237	0.90	0.60	0.98	0.84	0.70
Ukraine	4435	0.0203	0.92	0.68	0.97	0.85	0.76
Venezuela	6154	0.0021	0.92	0.64	0.98	0.88	0.74

TN/TP trade-off and can hardly be evaluated as a consequence. With $f1 \geq 0.84$ for Afghanistan and Syria, recall/precision balance demonstrates satisfying results for both regions, which stands out from the f1-scores of all other conflict areas in the range of $0.70 \leq f1 \leq 0.76$.

For interpretation it is important to take into consideration the amount of samples and the rate of truly interesting images within the scraped data. As test sets from Afghanistan and Syria contain the most interesting images by far (see figure 4.13), a higher recall can be explained. For the other regions, the recall seems to increase with sample size, which is a good sign as more scraping and training can improve further classification. Still, the dilemma in the current test data is the following: When sub-sampling all sample sizes to the lowest incidence ($n = 1435$ for Yemen) to increase comparability across regions, the low ground truth sample size for interesting images and their different share within the full dataset would highly affect the evaluation metrics. Vice versa, maximising the overall samples lowers region-specific comparability. Two main findings can be concluded: First, the share of images $p(interesting) = 0.0612$ for Afghanistan and $p(interesting) = 0.0489$ for Syria is noticeably high for the two regions, and lowest with $p(interesting) = 0.0021$ for Venezuela. This affects the model and influences classification results, so an application will also work more reliably for those first conflict areas with many interesting images. Secondly, for those regions with less interesting images, model performance (especially TP rate) must increase in order to identify the rare positive cases. As a consequence, with $n(interesting)/n(not\_interesting) \longrightarrow 0$, high recall importance increases for the use-cases.

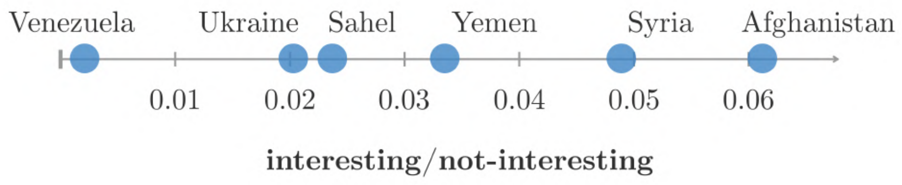


Figure 4.13: Share of interesting images within test dataset by location (e.g., almost every 15^{th} image is interesting in the Afghanistan dataset, and only every 500^{th} image is interesting for Venezuela).

When addressing geographic differences it is noticeable that a trend in interesting image share is visible by type of region – more specifically, MENA regions have the highest share, followed by Sahel countries and the Ukraine, and a tropical country shows the lowest share. This trend should not be mistaken for an actual correlation, but nevertheless it is important to understand that such differences in image distribution across environmental zones can introduce problematic biases (e.g., the model learns that soldiers and weapons are mostly located in images with a dry, desert-like background as in many MENA or Sahel regions).

Chapter 5: Towards an application

This chapter will reflect on two main network evaluation results: First, the present training bias resulting in a recall/specificity trade-off of Model I *is_interesting* described in section 4.1 will be discussed in detail and especially what this means in practical classification, before reviewing the issues rising from insufficient classification of Model II *is_object_class* as stated in section 4.2. Lastly, methodological implications of this study based on CNN training data choice, preprocessing steps and technical characteristics will be addressed.

5.1 Reviewing the trade-off: Misclassified images

The trade-off between recall and specificity (see section 4.1) inversely raises FP and FN cases, which have implications for the application. As the trade-off can be influenced with different training biases, those misclassification examples demonstrate which error cases will end up in an application.

False Positives. Some of the many *not-interesting* social media test images are wrongly predicted as *interesting* by Model I. Also, forwarding all images to Model II raised interesting misclassification. Most of the examples from other research for common misclassified images in the category *weapon* also occur in this work (see figure 5.1). Usual false positives are all sorts of allusions in shape and handling [32], such as mobile phones, pens, handbags, controllers, hair dryers, power-drills or some kinds of sticks [18, 28, 49, 71, 80]. Also (motor)cycle handles occur as FP just like in [32]. New examples resulting from this work are music instruments, cameras (including and excluding tripods), microphones, tools and weights. False positives in the class *vehicle* are mainly medium and large trucks, freight trains, quad and two-wheel motorbikes – which is somehow an expected result due to their similarity in texture and geometry. This observation cannot be compared directly as there is no scientific reference detecting military vehicles in social media imagery, but it goes along with the finding that larger, truck-like vehicles are harder to classify than others [73]. Regarding the

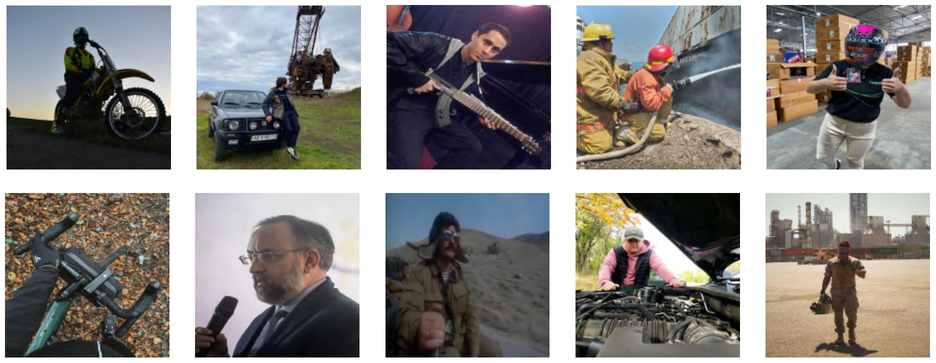


Figure 5.1: Examples for social media images falsely classified as interesting.

class *soldier*, misclassification concerned people wearing different helmets (e.g., when riding bikes), people casually wearing camouflage clothes outside of any military situation or wearing outfits with a very similar pattern. In some special cases, the false positive images contained multiple of the given examples, such as a firefighters (classified as soldiers due to their helmet and uniform plus fire hose classified as weapon) or motorbike riders (classified as soldiers due to their helmet and protection gear plus motorbike classified as vehicle). Also in the binary classification of Model I, those FP cases occured. Interestingly, some of those examples also rarely occurred in the OIDv4 training dataset. This is relevant because it introduces misclassification in the training data already – and it also proves that even GOOGLE AI's powerful classifiers are prone to such errors. Examples in this dataset on top of the named ones were space suits, metal statues (class soldier) as well as chainsaws and binoculars. As the OIDv4 images were forwarded to both model training sets, the FP most likely affected (and even reinforced) this work's outcome.

False Negatives. Most FP results could just be manually sorted out in these use-cases based on an application. More problematic are false negatives (see figure 5.2), as they will not appear in a possible application. A reasonable explanation for some of the falsely classified negatives are perturbations interfering with the interesting objects within the images, just as elaborated in previous research [19]. This work's results demonstrate that this might be objects such as flags, signs and clothing parts located in front of the class items. For images that have been edited before uploading to the platform, it might also be artificial perturbations such as text (labels within image) and icons (e.g., logos and emojis). For the class *vehicle*, there were examples of environmental perturbations such as high grass covering the wheels of a military truck or a dust cloud obscuring large parts of another vehicle. This finding goes along with similar research in arms detection, where the presence of other objects confuses the prediction [2]. A useful counter-mechanism can be an augmentation technique named object-aware random erasing [63], which strengthens the network's ability to recognize object parts by forcing it to focus more on other characteristics than shape.

With the presented misclassifications resulting from this work, it is clear that FP and FN predictions on social media data will occur regardless of the introduced bias of Model I. While the training bias can influence the TP/TN trade-off, it is likely that while one increases, the other decreases. For instance, other research applying a similar classification produces no FN but high FP [49], contrary to this work's results. The different examples of misclassification inherent to Model II are likely to keep occurring even in further training, and just the quantity is determined by the bias of Model I forwarding filtered images.

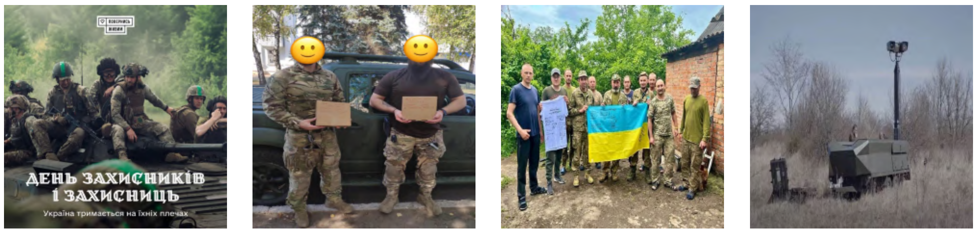


Figure 5.2: Examples for social media images falsely classified as not-interesting due to real or digital perturbations (left & center-left: digital perturbations through labels and icons, center-right & right: real perturbations through flag and grass coverage).

5.2 Methodological implications of this work

Because the results of this work are highly dependant on the methodological choices that have been made, it is necessary to review them in the context of the use-cases and a possible application. This will be done by looking at data-related, preprocessing and technical issues.

5.2.1 Data-related issues: social media, OIDv4 and IMFDb

Social Media. After many past studies primarily utilized images from the platform X (formerly Twitter) [11, 21, 30, 37, 40, 47, 68], it has drastically reduced data availability through its APIs recently. Although scraping is a useful workaround often applied by scientists, even this is hard to realize as the platform constantly modifies their URL structures to avoid unpaid data access. This is why Instagram was used as a test and train data source. As part of the company Meta (which also operates the social media platform Facebook), they recently developed a similar restrictive API access [43] and are known for not being very accessible publicly [70], but scraping is possible due to consistent URLs. This data source choice resulted in some positive and negative consequences: On the pro side, Instagram is an image-centered platform, making it a suiting choice for CV applications [36]. Although descriptive texts and hashtags come as metadata for each image, there is just a minor information loss when ignoring them. In comparison, X is mainly text-centered, often requiring textual analysis such as NLP in order to gather useful information. For instance, to analyse spatial information on images, one study first had to apply a textual geotagging analysis, only to raise the proportion of georeferenced images from 0.4 to 2.5 percent [21]. This work's whole image dataset was already scraped by geolocation (on country level) due to accessible *location ids*. Here it needs to be mentioned that the *id* is just a strong indicator and no proof for an image geotag – manual data inspection suggests that VPN services have been used for some photos taken in other regions than the conflict countries, inevitably leading the platform to record a wrong location.

On the downside, X is generally speaking a more political space than Instagram, which is also why researchers preferred it for studies back when it was still easily accessible. This means that the data for this work contains a mass of trivial images, raising the chance for false positives and the need for a high recall value. But technically, this can also be seen as a better challenge for the trained network. Less scientific focus on the platform additionally leaves a greater knowledge gap, because the presence of conflict-related images on Instagram is an information void from a research perspective.

One issue that results from the social media scraping procedure (see section 3.1) is that the test images are rather snapshots than equally distributed over time like in [21]. Assuming that roughly the same amount of images is uploaded on the platform over time, density of downloaded images differs throughout the scraping (see figure 5.3). This does not have relevant implications for the performance of the models, but for the use-case – where possibly interesting images might be lost. In an application, the data retrieval from Instagram should therefore be on hourly, or even minute basis for full coverage.

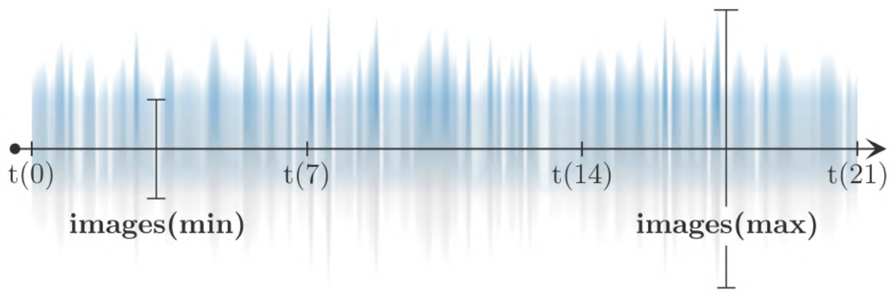


Figure 5.3: Image distribution intensity over time resulting from scraping (for $t = 21$ days on x-axis) shows variance in the amount of scraped images on y-axis.

OIDv4 and IMFDb. To reflect on the training data, it is useful to look at the image sources again. When reducing the dimensionality for subsamples of the training data sources – IMFDb and OIDv4 – and plotting social media images on top using t-SNE [72], there is a clustered data pattern (see figure 5.4) in abstract x, y dimensions. This is true especially for the IMFDb dataset, which was only used for Model II (see section 3.3.2). The social media set is more dispersed, mainly within the scatter extent of the OIDv4 training dataset. When removing more data variance and just looking at the first two principal components of PCA in t-SNE (p.262f [66]), the clustering of IMFDb images is even more dominant. This pattern suggests that a majority of

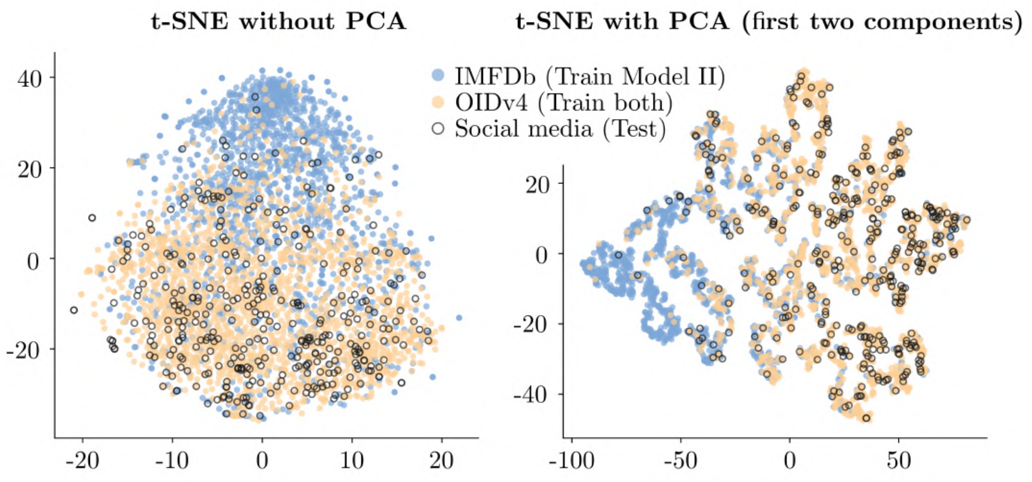


Figure 5.4: t-SNE & PCA dimensionality reduction on subsamples for training and testing data (left: t-SNE for full image variance, right: t-SNE only for first two principal components, which account for the most variability).

IMFDb training images do not share many characteristics with the test set that can be identified by t-SNE, but the OIDv4 dataset does. This finding can be a reason for the insufficient classification performance of Model II as evaluated in section 4.2, and supports the satisfying findings from binary classification of Model I (just based on OIDv4) vice versa.

In other research [2, 40, 50, 63], t-SNE is also produced for single classes to determine their similarity. If applied here for the OIDv4 training images for the three classes of Model II, a more scattered pattern is visible in figure 5.5. While the class *vehicle* shows some clustering, the class *soldier* is completely dispersed. This finding adds to the classification difficulty of Model II, as dimensionality reduction performed on the whole image cannot produce clear clusters. Most of the other studies mentioned – which use more distinct classes and bounding boxes – show less dispersed t-SNE patterns. Consequently, object detection based on bounding-boxes might be more valuable than image-level classification. This way, much background image noise can be omitted.

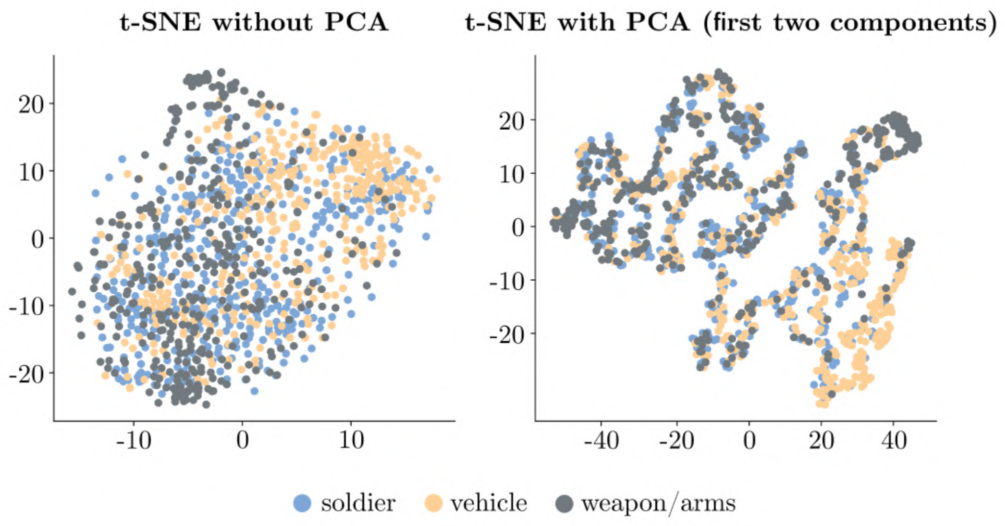


Figure 5.5: t-SNE & PCA dimensionality reduction on subsamples for OIDv4 training classes for Model II (left: t-SNE for full image variance, right: t-SNE only for first two principal components, which account for the most variability).

5.2.2 Data preprocessing issues

Cartoon/Animation detection. The approach to sort out animated images was based on general distinctive characteristics [27] in contrast to real photographs. Thresholds had to be chosen (unique color pixel count and gradient strength after bilateral filtering) for separation, and obviously there are some edge cases that were hard to filter. For example in practice, animated images with high color variety of $n > 100k$ pixels could not automatically be differentiated from photographs with this method. Analog to this, real photographs with $n < 100k$ unique color pixels were identified as animation and therefore sorted out in preprocessing (see figure 5.6). This is rather a limitation in preprocessing, which is also in line with other research (e.g., classifier with $acc = 0.92$ in [27]) and does not pose a significant error to the training in this work. The fact that some animated images are still left in the data could even contribute to dataset variety. Other papers utilizing IMFDb [6, 32, 44, 74] and OIDv4 images [4, 48, 58] did not mention the cartoon issue previously.

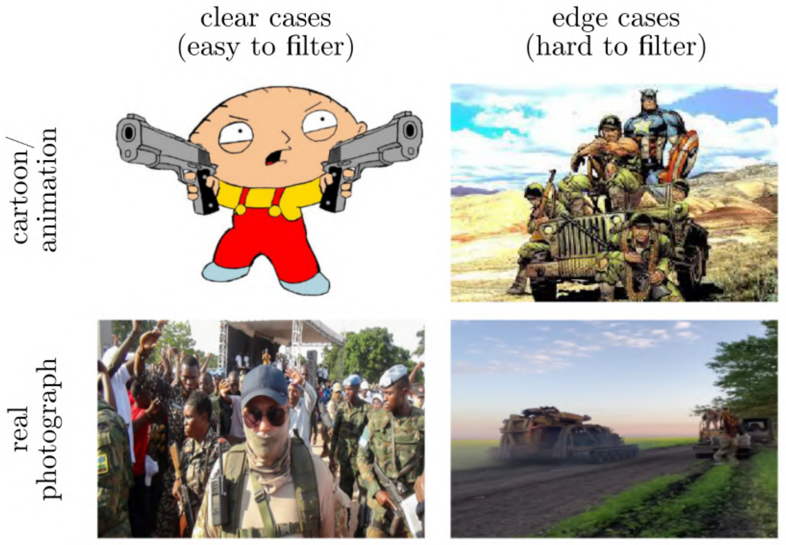


Figure 5.6: Examples for clear (left) and edge cases (right) in cartoon/animation (top) filtering from real photographs (bottom).

IMFDb annotation. Annotation of $n = 150k$ unlabeled IMFDb training images was not possible within the time frame of this work. Obviously, with more (and especially larger variation) of training images, more robust models can be trained. As soon as the full IMFDb dataset is annotated, new trainings can be performed to test improvement. In the domain of arms goods, there are not many labeled open-source databases, increasing manual efforts for any application. Also, a structured class label taxonomy as for other domains [4] would enhance annotation quality, possibly with more accurate labels allowing for more precise classifications.

5.2.3 Technical issues

Two different pretrained CNN architectures – YOLOv8m and ResNet50 – have been used for the models. The choice was mainly based on the classification tasks: Currently, the ULTRALYTICS library developing the latest YOLO-model does only allow for multi-class (including binary) and not multilabel classification (see section 2.2, but is described as the state-of-the-art CNN [51] performing best on popular benchmarks [29]). This is why the older but common ResNet architecture was used for Model II, as it can be finetuned as a multilabel classifier. Both model trainings are based on multiple 10k images, with Model I combining OIDv4 and negative social media instances and Model II combining IMFDb and OIDv4. This is more than in many similar studies [21, 32, 33, 49, 71], most of which are performing object detection based on bounding boxes. For classification on image-level, there is research with way more training images [40] – so limitations due to dataset size should be considered.

5.3 Ethical considerations: social media ans bias

After already indicating that this work aims at advocating for the oppressed by targeting violence and helping to control arms goods (see section 1.3), ethical aspects still need to be considered in the context of critical AI [18]. For example, the possible introduction of a background bias by region as mentioned in section 4.4 is a problem that needs to be monitored in further training. As findings indicate high performance differences of Model I for different regions, this bias could already be introduced with the training images from OIDv4 – but even if not, it will most likely influence further training based on the classified social media images. This can be concluded from looking at the intensity histograms averaged for each conflict region (see figure 5.7). After applying the GrabCut algorithm [57] to segment fore- and background using the full image as ROI rectangle (after resizing images to 500×500 pixel), it is possible to calculate the mean pixel frequency for all test image backgrounds of each region against intensity for the color channels. Frequency was capped at $f = 2000$ to enable

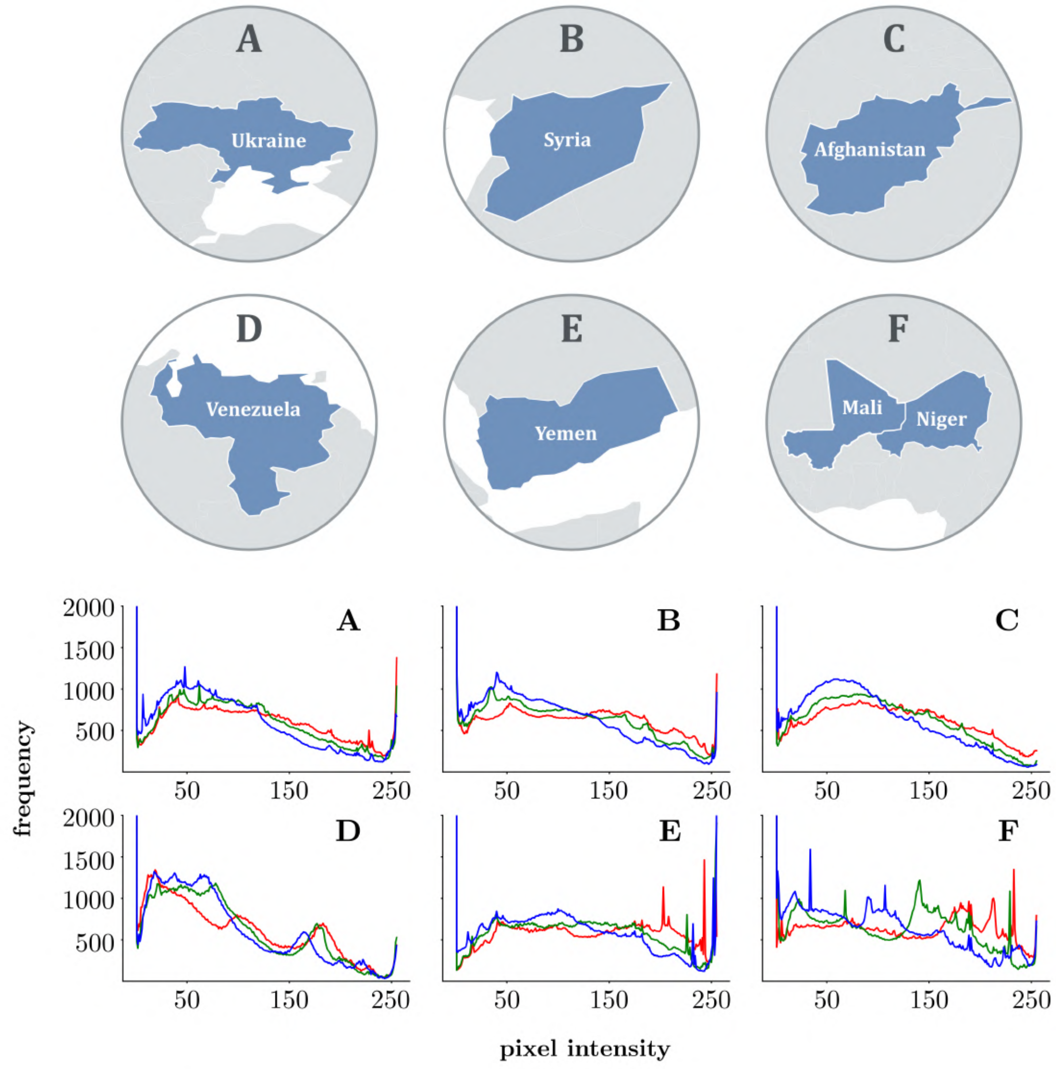


Figure 5.7: Pixel intensity histogram by color channel (for backgrounds segmented with GrabCut algorithm, frequency is averaged per conflict region).

a detailed view on RGB color channel differences. Whereas overall histogram distribution looks similar for most regions, some differences are apparent. More precisely, backgrounds in the Sahel and Yemen images show higher average pixel frequencies for intensity $i > 100$ compared to other regions. Images from Venezuela show a peak in frequency in between $150 \leq i \leq 200$. It is not clear from this finding how strong the effect in further CNN training would be, but a bias resulting from the variation in color channels is possible, especially when classifying the full images instead of introducing bounding boxes, which would rule out background effects to some degree.

Other biases are likely, for instance raised through non-representativeness in social media [82] or other causes [14]. A common call in research is to demand for a human-in-the-loop [18] to be aware and take counter-mechanisms against those biases. This can also be defined as a post-classification process to identify errors [68]. Additionally, there are privacy concerns in any application dealing with social media, especially as images are a sensitive form of media that might contain disclosure of private information [42]. Then again, all images used in this work are publicly available – and their use in an OSINT-context is not prohibited. In an application, training images are not publicly visible anyway, but the models would classify a continuous stream of social media images. A possible strategy to minimize privacy harm on individuals is omitting any link of a TP prediction to its source, optimally not even collecting metadata such as the image URL or user ID [14]. Another issue is the misuse of such an application in a harmful manner as also pointed out in other research [18, 82], for example forwarding images to other actors such as local law enforcement. This is why it is necessary to exclusively use the proposed models for research purposes.

6. Summary

This chapter will summarize the most important findings of the thesis in order to formulate a conclusion for developing an application for mentioned use-cases. It will also point out future directions inferred from that. Especially, further CNN training improvement such as switching to an object detection task using bounding-boxes as a basis for a useful application will be addressed.

6.1 Conclusion

The evaluation of trained networks Model I (*is_interesting*) and Model II (*is_object_class*) in this thesis has proven that CV for arms control applied on social media images can produce valuable outcome for an application with some limitations.

Model I. The binary classification of Model I showed slight overfitting behaviour, while still predicting on social media test data with average metrics of $\overline{acc} = 0.93$, $\overline{rec} = 0.75$, $\overline{prec} = 0.84$ and $\overline{spec} = 0.97$ across the four bias classes. There is a TP/TN trade-off present that can be influenced by the training bias. This bias should be based on the application requirements I, II and III as defined in section 4.1, while also considering for interpretation that ground truth data contains way more negative than positive samples. Moreover, two other findings were pointed out and can be concluded: Recall is way higher when testing on a different dataset (IMFDb), suggesting it is likely that the binary model can also produce increased TP on social media images, for instance when being trained on more positive social media samples. Secondly, very high prediction confidence scores towards $p = 0$ and $p = 1$ were visible in the classification on test images, leaving minimum space for uncertainty.

Conflict zones. Also regional differences could be detected after training. A very high span in recall values is most prominent with $rec_{(max)} = 0.88$ for Afghanistan and $rec_{(min)} = 0.58$ for Yemen, whereas TN and precision are rather stable across regions. Taking into account the high amount of negative samples, this results in an $acc \geq 0.9$ for all conflict zones. For interpretation, it is important to consider the varying share of $n_{(interesting)}/n_{(not\_interesting)}$ images in the social media test images across regions – underlining the need for a larger and more solid test set acquired over months of scraping. Lastly, in the context of responsible AI there should be high awareness of possible biases, especially biases introduced through different background characteristics of conflict regions.

Model II. The multilabel classifier of Model II showed high overfitting behaviour in training even after many finetuning efforts, and produced unsatisfying prediction quality on social media images as clarified in section 4.2. Major differences across the three classes were visible, with $rec_{(max)} = 0.81$, $prec_{(max)} = 0.93$ and $AUC - ROC_{(max)} = 0.75$ for the class *soldier*, but way lower metrics for the other two classes. The presence of soldiers in most images, background noise and insufficient training samples can be a reason. Lowering prediction confidence thresholds in an application to $p < 0.5$ can raise TP and adjust practical use to some degree, and layer-wise activation maps the issue rather than pointing to a solution. A possible strategy is to increase class samples with more annotations, and more importantly switch to bounding-boxes instead of image-level classification.

Two-model pipeline. As a final conclusion, the two model approach can be reviewed very useful as it splits up the problem and allows for individual evaluation of model performance. Because both models cannot be directly compared in the current setup as indicated in section 4.3, Model I functions as a filter for Model II instead. Based on that, further work could include integrating both classification tasks into each other to raise a single network solution. Finally, in application-context it became clear that regardless of the exact network implementation, CV-based machine learning produces promising results (with multiple hundreds of true positive test samples from conflict regions) – based on the definition of *interesting* (see section 1.2) in this work. Content-related assessment of those images is left to expert researchers in arms control and conflict studies.

6.2 Future directions based on this work

Multiple measures can be taken to improve the prediction results of this work. While for Model I it makes sense to classify the whole images, the downstream Model II could be implemented with an object detection task instead of image-level classification as indicated above. This would require much more manual annotation work – which was out of scope during this project. Labeling the training images would then not be based on quick annotation by keystrokes, but hand-drawn *bounding boxes* framing the classes arms, vehicle and soldier. As for the use-cases it is not necessary to locate the objects within images, the main benefit would be to ease the learning process for the model by reducing the background error in the images – only feeding the relevant object features into the network. This can also help to tackle a possible bias through different regional background characteristics (see section 4.4). Although not necessary for recognition, object localization using bounding boxes are also a beneficial visual feature in an application. In this case, many resources need to be planned for the annotation process. Using bounding boxes instead of image-level classes for arms detection has resulted in promising results in other research [18, 28, 32, 33, 71].

Another way to improve the prediction is using TP and FN images resulting from the test set to train a new CNN. As presented in chapter 4, the generalization from the two training sets to the test set was likely improvable, which could be due to different image properties in the two sets. Enriching the training data with actual social media images that were correctly identified as interesting will most likely improve the results when predicting on new social media images, because new data is gathered. This is a common procedure in deep learning applications (p.424 [23]). In this context, the current work can also be seen as a fundamental tool to generate such useful training data, used to develop networks that achieve sufficient accuracy to be integrated in an application for expert inspection. With the recent advances in image-generation abilities of ViT, another future possibility is the creation of artificial training imagery based on the classes defined in this work – which would generate fully annotated training data.

Additionally, in future work automation and optimization approaches should be used to enhance the full development process. For example, to find the optimal CNN architecture and size, neural architectural search (NAS) (p.305f [66], [83]) approaches should be used. For optimal finetuning (which was a trial-and-error process to some extent), hyperparameter optimization techniques [81] will save time compared to manually adjusting parameters such as the batch size, learning rate, optimizer or the number of epochs. Additionally, performance visualization systems such as presented in previous literature [50] or integrated into large cloud computing platforms can be helpful for the model development.

Lastly, the scraping so far only filtered images from the social media platform, ignoring all video material. Manual inspection has shown that also many videos hold possibly useful information in the arms control context. Future work could include this video material – both to enrich training data and test set – with obvious computational limitations regarding runtime.

Bibliography

[1] Abien F. Agarap. Deep Learning using Rectified Linear Units (ReLU). *CoRR*, abs/1803.08375, 2018.

[2] Samet Akcay, Mikolaj E. Kundegorski, Chris G. Willcocks, and Toby P. Breckon. Using Deep Convolutional Neural Network Architectures for Object Classification and Detection within X-ray Baggage Security Imagery. *IEEE Transactions on Information Forensics and Security*, 13:2203 – 2215, September 2018.

[3] J. Arterburn, E.D. Dumbacher, and P.O. Stoutland. *Signals in the Noise: Preventing Nuclear Proliferation with Machine Learning & Publicly Available Information.* C4ADS, 2021.

[4] Amna Asif, Shaheen Khatoon, Hasan Maruf, Majed A. Alshamari, Sherif Abdou, Khaled Mostafa Elsayed, and Mohsen Rashwan. Automatic Analysis of Social Media Images to Identify Disaster Type and Infer Appropriate Emergency Response. *Journal of Big Data*, 8, December 2021.

[5] Bellingcat. MH17 - The Open Source Evidence: A Bellingcat Investigation. October 2015.

[6] Muhammad Tahir Bhatti, Muhammad Gufran Khan, Masood Aslam, and Muhammad Junaid Fiaz. Weapon Detection in Real-Time CCTV Videos Using Deep Learning. *IEEE Access*, 9:34366 – 34382, 2021.

[7] BICC, HSFK, IFSH, and INEF. *Noch lange kein Frieden - Friedensgutachten 2023.* transcript Verlag, 2023.

[8] Markus Borg, Ronald Jabangwe, Simon Åberg, Arvid Ekblom, Ludwig Hedlund, and August Lidfeldt. Test Automation with Grad-CAM Heatmaps - A Future Pipe Segment in MLOps for Vision AI? *IEEE 14th International Conference on Software Testing, Verification and Validation Workshops*, 2021.

[9] Kendrick Boyd, Kevin H. Eng, and C. David Page. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 451 – 466, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[10] Andrew P. Bradley. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[11] Gregoire Burel, Hassan Saif, Miriam Fernandez, and Harith Alani. On Semantics and Deep Learning for Event Detection in Crisis Situations. In *Workshop on Semantic Deep Learning (SemDeep) at ESWC*, May 2017.

[12] Saúl Calderón Ramírez, Raghvendra Giri, Armaghan Moemeni, Mario Umaña, David Elizondo, Jordina Torrents-Barrena, and Miguel A. Molina-Cabello. Dealing with Scarce Labelled Data: Semi-supervised Deep Learning with Mix Match for Covid-19 Detection Using Chest X-ray Images. In *25th International Conference on Pattern Recognition (ICPR)*, October 2020.

[13] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. *CoRR*, 2020.

[14] Yan Chen, Kate Sherren, Michael Smit, and Kyung Y. Lee. Using Social Media Images as Data in Social Science Research. *New Media & Society*, 25(4):849 – 871, 2023.

[15] Alex Clark. Pillow (PIL Fork) Documentation. *GitHub*, 2015.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*, abs/2010.11929, 2020.

[17] Frédo Durand and Julie Dorsey. Fast Bilateral Filtering for the Display of High-Dynamic-Range Images. *ACM Trans. Graph.*, 21(3):257 – 266, July 2002.

[18] Alexander Egiazarov, Vasileios Mavroeidis, Fabio Massimo Zennaro, and Kamer Vishi. Firearm Detection and Segmentation Using an Ensemble of Semantic Neural Networks. In *European Intelligence and Security Informatics Conference (EISIC)*, pages 70 – 77, 2019.

[19] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust Physical-World Attacks on Deep Learning Visual Classification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625 – 1634, 2018.

[20] Joseph Farha. Small Arms and Light Weapons Guide (SALW). *Bonn International Center for Conflict Studies*, 2024.

[21] Chiara Francalanci, Paolo Guglielmino, Matteo Montalcini, Gabriele Scalia, and Barbara Pernici. IMEXT: A Method and System to Extract Geolocated Images from Tweets — Analysis of a Case Study. In *11th International Conference on Research Challenges in Information Science (RCIS)*, pages 382 – 390, 2017.

[22] Zoe N. Gastelum. *Societal Verification for Nuclear Nonproliferation and Arms Control*, pages 169 – 183. Springer International Publishing, Cham, 2020.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015.

[25] Mohammadreza Heydarian, Thomas E. Doyle, and Reza Samavi. MLCM: Multi-Label Confusion Matrix. *IEEE Access*, 10:19083 – 19095, 2022.

[26] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *CoRR*, abs/1608.06993, 2016.

[27] Tzveta Ianeva, Arjen de Vries, and Hein Rohrig. Detecting Cartoons: A Case Study in Automatic Video-Genre Classification. *IEEE International Conference on Multimedia and Expo*, 1, August 2003.

[28] Oussama Jarrousse, Thomas Fritz, Jens Elsner, Stefan Taing, Laura Henke, and Mathias Uhlenbrock. Automatic Weapon Detection in Social Media Image Data using a Two-Pass Convolutional Neural Network. *European Law Enforcement Research Bulletin*, October 2018.

[29] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLOv8. *GitHub*, 2023.

[30] Jungseock Joo and Zachary C. Steinert-Threlkeld. Image as Data: Automated Content Analysis for Visual Presentations of Political Actors and Events. In *Computational Communication Research*, volume 4, 2022.

[31] JustAnotherArchivist. Snscrape Python Library. *GitHub*, 2021.

[32] Rodrigo Kanehisa and Areolino Neto. Firearm Detection using Convolutional Neural Networks. In *11th International Conference on Agents and Artificial Intelligence*, pages 707 – 714, January 2019.

[33] Volkan Kaya, Servet Tuncer, and Ahmet Baran. Detection and Classification of Different Weapon Types using Deep Learning. *Applied Sciences*, 11(16), 2021.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*, 2017.

[35] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *CoRR*, abs/1811.00982, 2018.

[36] Linnea Laestadius and Alice Witt. *Instagram Revisited*, chapter 40, pages 581 – 597. SAGE Publications Ltd, 2022.

[37] Mehdi B. Lazreg, Morten Goodwin, and Ole-Christoffer Granmo. Deep Learning for Social Media Analysis in Crises Situations. In *29th Annual Workshop of the Swedish Artificial Intelligence Society*, June 2016.

[38] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436 – 444, May 2015.

[39] Lea Lehner. *Social Media – Vox populi oder Forum der Agitatoren? Elementare Aufgaben-, Wirkungs- und Problemfelder sozialer Netzwerke im syrischen Bürgerkrieg*, page 135 – 154. September 2020.

[40] Wen-Hung Liao, Yen-Ting Huang, Tsu-Hsuan Yang, and Yi-Chieh Wu. Analyzing Social Network Data Using Deep Neural Networks: A Case Study Using Twitter Posts. In *IEEE International Symposium on Multimedia (ISM)*, pages 237 – 2371, 2019.

[41] Nico Lück. Machine Learning-Powered Artificial Intelligence in Arms Control. Technical report, Peace Research Institute Frankfurt, Leibniz-Institut Hessische Stiftung Friedens- und Konfliktforschung, 2019.

[42] Xiaoyue Ma and Xu Fan. A Review of the Studies on Social Media Images from the Perspective of Information Interaction. *Data and Information Management*, 6(1), 2022.

[43] Pablo Martí, Leticia Serrano-Estrada, and Almudena Nolasco-Cirugeda. Social Media Data: Challenges, Opportunities and Limitations in Urban Studies. *Computers, Environment and Urban Systems*, 74:161 – 174, 2019.

[44] Parth Mehta, Atulya Kumar, and Shivani Bhattacharjee. Fire and Gun Violence based Anomaly Detection System Using Deep Neural Networks. In *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 199 – 204, 2020.

[45] Diganta Misra. Mish: A Self Regularized Non-Monotonic Neural Activation Function. *CoRR*, abs/1908.08681, 2019.

[46] Sanam Narejo, Bishwajeet Pandey, Doris Esenarro Vargas, Ciro Rodriguez, and Rizwan M. Anjum. Weapon Detection Using YOLO V3 for Smart Surveillance System. *Mathematical Problems in Engineering*, 2021.

[47] Dat T. Nguyen, Shafiq R. Joty, Muhammad Imran, Hassan Sajjad, and Prasenjit Mitra. Applications of Online Deep Learning for Crisis Response Using Social Media Information. *CoRR*, abs/1610.01030, 2016.

[48] Yusuke Niitani, Takuya Akiba, Tommi Kerola, Toru Ogawa, Shotaro Sano, and Shuji Suzuki. Sampling Techniques for Large-Scale Object Detection From Sparsely Annotated Objects. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6503 – 6511, 2018.

[49] Roberto Olmos, Siham Tabik, and Francisco Herrera. Automatic Handgun Detection Alarm in Videos using Deep Learning. *Neurocomputing*, 275:66 – 72, 2018.

[50] Chanhee Park, Hyojin Kim, and Kyungwon Lee. A Visualization System for Performance Analysis of Image Classification Models. *Electronic Imaging*, November 2019.

[51] Muralidhar Pullakandam, Keshav Loya, Pranav Salota, Rama Muni Reddy Yanamala, and Pavan Kumar Javvaji. Weapon Object Detection Using Quantized YOLOv8. In *5th International Conference on Energy, Power and Environment: Towards Flexible Green Energy Technologies (ICEPE)*, pages 1 – 5, 2023.

[52] Thomas Reinhold. *Arms Control for Artificial Intelligence*, pages 211 – 226. Springer International Publishing, Cham, 2022.

[53] Leonard Richardson. Beautiful Soup Documentation. *GitHub*, April 2007.

[54] Richard Rogers. Visual Media Analysis for Instagram and Other Online Platforms. *Big Data & Society*, 8(1), 2021.

[55] Candace Rondeaux. *Decoding the Wagner Group.* Oxford Academic, 2023.

[56] Giulio Rossolini, Alessandro Biondi, and Giorgio Buttazzo. Increasing the Confidence of Deep Neural Networks by Coverage Analysis. *IEEE Transactions on Software Engineering*, 49(2):802 – 815, 2023.

[57] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut": Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[58] Abhishek Sarda, Shubhra Dixit, and Anupama Bhan. Object Detection for Autonomous Driving using YOLO (You Only Look Once) algorithm. In *Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pages 1370 – 1374, 2021.

[59] Tobias Schreck and Daniel Keim. Visual Analysis of Social Media Data. *IEEE Computer*, 46(5):68 – 75, 2013.

[60] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization. *CoRR*, abs/1610.02391, 2016.

[61] Saleh Shahinfar, Paul Meek, and Greg Falzon. How many Images do I Need? Understanding how Sample Size per Class Affects Deep Learning Model Performance Metrics for Balanced Designs in Autonomous Wildlife Monitoring. *Ecological Informatics*, 57, May 2020.

[62] Andrew Shin, Masato Ishii, and Takuya Narihira. Perspectives and Prospects on Transformer Architecture for Cross-Modal Tasks with Language and Vision. *CoRR*, abs/2103.04037, 2021.

[63] Connor Shorten and Taghi M. Khoshgoftaar. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(60), 2019.

[64] Avantika Singh and Shaifu Gupta. Learning to Hash: A Comprehensive Survey of Deep Learning-Based Hashing Methods. *Knowledge and Information Systems*, 64, August 2022.

[65] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929 – 1958, 2014.

[66] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[67] K. K. Thyagharajan and G. Kalaiarasi. A Review on Near-Duplicate Detection of Images using Computer Vision Techniques. *Archives of Computational Methods in Engineering*, 28(3):897 – 916, January 2020.

[68] Michelle Torres and Francisco Cantú. Learning to See: Convolutional Neural Networks for the Analysis of Social Science Data. *Political Analysis*, 30:1 – 19, April 2021.

[69] Robert Trevethan. Sensitivity, Specificity, and Predictive Values: Foundations, Pliabilities, and Pitfalls in Research and Practice. *Frontiers in Public Health*, 5, November 2017.

[70] Joshua Tucker, Andrew Guess, Pablo Barbera, Cristian Vaccari, Alexandra Siegel, Sergey Sanovich, Denis Stukal, and Brendan Nyhan. Social Media, Political Polarization, and Political Disinformation: A Review of the Scientific Literature. *SSRN Electronic Journal*, January 2018.

[71] Erik Valldor, KG Stenborg, and David Gustavsson. Firearm Detection in Social Media Images. In *Swedish Symposium on Deep Learning*, September 2018.

[72] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579 – 2605, 2008.

[73] Bruno R. Vasconcellos, Marcelo Rudek, and Marcelo de Souza. A Machine Learning Method for Vehicle Classification by Inductive Waveform Analysis. *IFAC-PapersOnLine*, 53(2):13928 – 13932, 2020. 21st IFAC World Congress.

[74] Gyanendra K. Verma and Anamika Dhillon. A Handheld Gun Detection Using Faster R-CNN Deep Learning. In *Proceedings of the 7th International Conference on Computer and Communication Technology*, page 84 – 88, New York, NY, USA, 2017. Association for Computing Machinery.

[75] Angelo Vittorio. Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset. *GitHub*, 2018.

[76] Pieter D. Wezeman, Justine Gadon, and Siemon T. Wezeman. Trends in International Arms Transfers, 2022. *Stockholm International Peace Research Institute (SIPRI)*, 2023.

[77] Simone Wisotzki. Die grenzenlose Verbreitung von Klein- und Leichtwaffen: Argumente für eine restriktive deutsche Rüstungsexportpolitik. *Zeitschrift für Friedens- und Konfliktforschung*, 3:305 – 321, December 2014.

[78] Jamie Withorne. Machine Learning Applications in Nonproliferation: Assessing Algorithmic Tools for Strengthening Strategic Trade Controls. *James Martin Center for Nonproliferation Studies, Middlebury Institute for International Studies at Monterey*, August 2020.

[79] Mingfang Wu, Hans Brandhorst, Maria-Cristina Marinescu, Joaquim More Lopez, Margorie Hlava, and Joseph Busch. Automated Metadata Annotation: What is and is not Possible with Machine Learning. *Data Intelligence*, 5(1):122 – 138, March 2023.

[80] Pavinder Yadav, Nidhi Gupta, and Pawan K. Sharma. A Comprehensive Study towards High-Level Approaches for Weapon Detection using Classical Machine Learning and Deep Learning Methods. *Expert Systems with Applications*, 212:118698, 2023.

[81] Tong Yu and Hong Zhu. Hyper-Parameter Optimization: A Review of Algorithms and Applications. *CoRR*, abs/2003.05689, 2020.

[82] Han Zhang and Jennifer Pan. CASM: A Deep-Learning Approach for Identifying Collective Action Events with Text and Image Data from Social Media. *Sociological Methodology*, 49(1):1 – 57, 2019.

[83] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *CoRR*, abs/1611.01578, 2016.

Appendix: Algorithms

The appendix for this work contains the main training and evaluation modules for Model I and Model II developed in Python. Other modules regarding preprocessing, specific evaluation features and visualizations are not included.

Model I – Binary classification

The following modules *training.py* and *prediction.py* are the main components of the training and evaluation for the binary classification model *is_interesting*. Additional components – which are not included in the appendix – were created for evaluation, mainly for visualizing results. Model I is finetuned on a pretrained YOLOv8 architecture.

Training module *training.py*

```python
import os, time
from ultralytics import YOLO
import torch
import torch.nn as nn

# Set working directory.
current_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(current_dir)

# Define my weighted loss class.
class WeightedLoss(nn.Module):
    def __init__(self, class_weights):
        super(WeightedLoss, self).__init__()
        self.class_weights = torch.FloatTensor(class_weights)
        self.bce = nn.BCEWithLogitsLoss(reduction='none',
         ↪  pos_weight=self.class_weights)
    def forward(self, inputs, targets):
        return self.bce(inputs, targets)

# Load and print pretrained YOLO model.
model = YOLO('yolov8m-cls.pt')
print(model)

# Override the forward method with the custom loss. (Different
↪  settings here!)
model.forward = WeightedLoss([1.0, 1.0]).forward # No weights at the
↪  moment (Bias in training data).

# Save starting time.
start_time = time.time()

images_dir = "" # Local images dir removed after all trainings.

# Train the model.
epochs = 100
model.train(data=images_dir, save = True, epochs=epochs, pretrained =
↪  True, patience=epochs, batch = config["batch_size"], freeze =
↪  config["unfreezing_layers"], imgsz = config["img_size"], dropout =
↪  config["dropout"], plots = False, seed = 7) # Check seed for new
↪  runs!

# Print training duration.
print("Training runtime:", int(time.time() - start_time), "seconds")
```

Evaluation module *prediction.py*

```
from ultralytics import YOLO
from PIL import Image
import torch
import os, csv

# Import trained model.
model = YOLO("local\\path") # Removed local path after implementation!

# Run detection on directory.
image_directory = "local\\path" # Test images path, removed after
↪  implementation.
file_list = os.listdir(image_directory)
total_images = len(file_list)
```

```python
max_images = 1672 # Change to actual max in folder!
counter = 0

# Define the CSV file path (adjust for each bias).
csv_file_path = "conf_results_all_strong_positive.csv"

# Open the CSV file in write mode.
with open(csv_file_path, mode='a', newline='') as file:
    writer = csv.writer(file)

    # Write the header row.
    writer.writerow(["conf_interesting", "true"])

    # Iterate over all images in directory.
    for image in file_list[:max_images]:
        results = model(os.path.join(image_directory, image))

        # Initialize variables.
        conf_interesting = None
        conf_not = None

        # Extract confidence values.
        for r in results:
            if r.probs.top5[0] == 0:
                conf_interesting = r.probs.data[0].item()
                print(conf_interesting)
                conf_not = r.probs.data[1].item()
                print(conf_not)
            else:
                conf_not = r.probs.data[1].item()
                conf_interesting = r.probs.data[0].item()

        # Write the row to the CSV file.
        writer.writerow([conf_interesting, 0])

        # Increment counter.
        counter += 1

        # Stop iterating after reaching the specified max_images.
        if counter >= max_images:
            break
```

Model II – Multilabel classification

The following modules *main.py*, *data.py*, *model.py* and *multiclass_prediction* are the main components of the training and evaluation for the multilabel classification model *is_object_class*. Additional components – that are not included in the appendix – were created for evaluation, mainly for visualizing results. Model II is finetuned on a pre-trained ResNet50 architecture.

Training module *main.py*

```
import os, csv, time, json
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
from sklearn.metrics import multilabel_confusion_matrix,
↪  precision_score, recall_score, accuracy_score, confusion_matrix
from varname import nameof
import numpy as np
import socket

# Import classes from my other files.
import data
import model
import visualization

# Set working directory and load trainingConfig JSON file that stores
↪  parameters.
current_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(current_dir)
config_filepath = os.path.join(current_dir, "trainingConfig.json")
with open(config_filepath, "r") as jsonfile:
    config = json.load(jsonfile)

# Set dataset path. As I am working on different PCs, set path
↪  device-dependent.
pc_name = socket.gethostname()
if pc_name == "R184W10-CELSIUS":
    dataset_path = config["train_path_BICC"]
elif pc_name == "LAPTOP_UBUH5BJN":
    dataset_path = config["train_path_laptop"]
```

```python
else:
    print("Different device!")

model_output_name = config["model_output_name"]

# Set transforms: data augmentation.
transform = transforms.Compose([
    transforms.Resize(list(config["transform_resize"])),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=config["rotation_degrees"]),
    transforms.ToTensor()
])

# Set training and validation sets (0.8/0.2 split in folder paths).
train_path = dataset_path + "\\train"
val_path = dataset_path + "\\val"
train_dataset = datasets.ImageFolder(root=train_path,
↪   transform=transform)
val_dataset = datasets.ImageFolder(root=val_path, transform=transform)

# Define the custom collate function.
def custom_collate(batch):
    images, labels = zip(*batch)
    return torch.stack(images), torch.stack(labels)

# Function to save training results to a csv file.
def results_to_csv(file_path, *lists):
    # Transpose the lists to create columns.
    columns = list(map(list, zip(*lists)))

    with open(file_path, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write header row.
        csv_writer.writerow(["epoch", "train/loss", "train/f1",
        ↪   "val/loss", "val/f1", "val/acc", "val/prec", "val/rec"])

        # Write data rows.
        for epoch, values in enumerate(columns, start=1):
            rounded_values = [round(value, 5) for value in values]
            csv_writer.writerow([epoch] + rounded_values)

# Save starting time.
start_time = time.time()
```

```
# Workaround, as there were runtime issues.
if __name__ == "__main__":
    # Create data loaders with the custom collate function
    train_loader = DataLoader(train_dataset,
    ↪   batch_size=config["batch_size"], shuffle=True, num_workers=4,
    ↪   collate_fn=custom_collate)
    val_loader = DataLoader(val_dataset,
    ↪   batch_size=config["batch_size"], shuffle=False, num_workers=4,
    ↪   collate_fn=custom_collate)

    # Instantiate the model
    num_classes = len(train_dataset.classes)
    model = model.CNN(num_classes)

    # Define loss function and optimizer: BCE loss function and Adam
    ↪   optimizer is good for multilabel classification problem.
    criterion = nn.BCEWithLogitsLoss(pos_weight =
    ↪   torch.FloatTensor(config["class_weights"]))
    #criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr =
    ↪   config["learning_rate"])

    # Training loop
    num_epochs = config["epochs"] # For testing

    # Metrics for evaluation and visualisation.
    train_losses = []
    val_losses = []
    train_f1_scores = []
    val_f1_scores = []
    val_accuracy_scores = []
    val_precision_scores = []
    val_recall_scores = []
    val_specificity_scores = []
    best_val_loss = float("inf")

    for epoch in range(num_epochs):

        all_labels = []  # Store all labels from validation set
        best_predictions = []  # Store best predictions from
        ↪   validation set

        model.train()
        epoch_train_losses = []
        epoch_train_predictions = []
```

```
epoch_train_labels = []

for inputs, labels in train_loader:
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels.float())
    loss.backward()
    optimizer.step()

    epoch_train_losses.append(loss.item())

    predictions =
     ↪  torch.sigmoid(outputs).round().detach().cpu().numpy()
    epoch_train_predictions.extend(predictions)
    epoch_train_labels.extend(labels.cpu().numpy())

# Calculate and log average training loss for the epoch.
average_loss = sum(epoch_train_losses) /
 ↪  len(epoch_train_losses)
print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss:
 ↪  {average_loss:.4f}")
train_losses.append(average_loss)

# Calculate training F1 score.
train_precision = precision_score(epoch_train_labels,
 ↪  epoch_train_predictions, average='micro')
train_recall = recall_score(epoch_train_labels,
 ↪  epoch_train_predictions, average='micro')
train_f1 = 2 * (train_precision * train_recall) /
 ↪  (train_precision + train_recall)
train_f1_scores.append(train_f1)

# Validation loop within epoch.
model.eval()
epoch_val_losses = []
epoch_val_predictions = []
epoch_val_labels = []
current_best_predictions = []

with torch.no_grad():
    for inputs, labels in val_loader:
        outputs = model(inputs)
        val_loss = criterion(outputs, labels.float())
        epoch_val_losses.append(val_loss.item())
        predictions =
         ↪  torch.sigmoid(outputs).round().detach().cpu().numpy()
        epoch_val_predictions.extend(predictions)
        epoch_val_labels.extend(labels.cpu().numpy())

        # Store labels and predictions for the confusion
         ↪  matrix
        all_labels.extend(labels.numpy())
        current_best_predictions.extend(torch.sigmoid(outputs)
        .round().numpy())

average_val_loss = sum(epoch_val_losses) / len(val_loader)
print(f"Epoch [{epoch+1}/{num_epochs}], Validation Loss:
 ↪  {average_val_loss:.4f}")
val_losses.append(average_val_loss)

# Update best predictions based on the highest predicted
 ↪  probability for each class.
if not best_predictions or val_loss < min(val_losses):
    best_predictions = current_best_predictions

# Update best predictions based on the lowest val/loss and
 ↪  save best model.
if average_val_loss < best_val_loss:
    best_val_loss = average_val_loss
    torch.save(model.state_dict(),
     ↪  f"{model_output_name}_best.pth")
    if epoch > 0:
        print("Model improved in epoch", epoch + 1)

# Calculate validation evaluation metrics.
val_accuracy = accuracy_score(epoch_val_labels,
 ↪  epoch_val_predictions)
val_precision = precision_score(epoch_val_labels,
 ↪  epoch_val_predictions, average='micro')
val_recall = recall_score(epoch_val_labels,
 ↪  epoch_val_predictions, average='micro')
val_f1 = 2 * (val_precision * val_recall) / (val_precision +
 ↪  val_recall)
val_f1_scores.append(val_f1)
val_accuracy_scores.append(val_accuracy)
val_precision_scores.append(val_precision)
val_recall_scores.append(val_recall)
print(val_accuracy, val_precision, val_recall, val_f1)
```

```
    conf_matrices = multilabel_confusion_matrix(epoch_val_labels,
     ↪  epoch_val_predictions)
    specificities = []
    for conf_matrix in conf_matrices:
        tn, fp, fn, tp = conf_matrix.ravel()
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0.0
        specificities.append(specificity)

    val_specificity_scores.append(specificities)

    if epoch == 0:
        # Print training duration for first epoch to calculate
         ↪  total time beforehand.
        print("Epoch 1 duration: ", int(time.time() - start_time),
         ↪  "seconds")

# Print training duration and save trained model.
print("Training runtime:", int(time.time() - start_time),
 ↪  "seconds")
torch.save(model.state_dict(), f"{model_output_name}_last.pth")

# Save results to csv file.
results_to_csv('results.csv', train_losses, train_f1_scores,
 ↪  val_losses, val_f1_scores, val_accuracy_scores,
 ↪  val_precision_scores, val_recall_scores)

# Visualize training and validation loss and f1 over epochs.
visualization.visualize_losses(train_losses, "train_loss", "svg")
visualization.visualize_losses(val_losses, "val_loss", "svg")
visualization.visualize_losses(train_f1_scores, "train_f1", "svg")
visualization.visualize_losses(val_f1_scores, "val_f1", "svg")

# Create a confusion matrix based on the best predictions.
conf_matrix = multilabel_confusion_matrix(np.array(all_labels),
 ↪  np.array(best_predictions), labels=[0, 1, 2])
class_names = train_dataset.classes

# Normalize the confusion matrices
normalized_conf_matrix = [conf_matrix[i] /
 ↪  conf_matrix[i].sum(axis=1, keepdims=True) for i in
 ↪  range(len(class_names))]

# Create the multilabel confusion matrix using seaborn heatmap.
for i in range(len(class_names)):
    visualization.visualize_conf_matrix(conf_matrix[i],
     ↪  class_names[i], "svg", "d", "abs")
    visualization.visualize_conf_matrix(normalized_conf_matrix[i],
     ↪  class_names[i], "svg", ".2f", "norm")
```

Dataset module *data.py*

```
import os
import torch
from PIL import Image
from torch.utils.data import Dataset

# Function that creates my dataset (local structure)
class MultiLabelImageDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = sorted(os.listdir(root_dir))
        self.class_to_idx = {cls: i for i, cls in
         ↪  enumerate(self.classes)}
        self.imgs = self._make_dataset()

    def _make_dataset(self):
        images = []
        for class_name in self.classes:
            class_path = os.path.join(self.root_dir, class_name)
            if os.path.isdir(class_path):
                for img_name in os.listdir(class_path):
                    img_path = os.path.join(class_path, img_name)
                    item = (img_path, [self.class_to_idx[class_name]])
                    images.append(item)
        return images

    def __len__(self):
        return len(self.imgs)

    def __getitem__(self, idx):
        img_path, labels = self.imgs[idx]

        # Read the image.
        image = Image.open(img_path).convert('RGB')

        # Apply transformations.
        if self.transform:
            image = self.transform(image)

        # Convert labels to a binary vector.
        binary_label = torch.zeros(len(self.classes))
        binary_label[labels] = 1.0

        return image, binary_label
```

Finetuning module *model.py*

```
from torchvision.models import resnet50
import torch.nn as nn
import os, json

# Set working directory and load trainingConfig JSON file that stores
↪  parameters.
current_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(current_dir)
config_filepath = os.path.join(current_dir, "trainingConfig.json")
with open(config_filepath, "r") as jsonfile:
    config = json.load(jsonfile)

# Determine if model will be finetuned or not (currently used for
↪  testing).
finetune = config["finetuning"]

# Define the CNN model. Here, I can modify the network.
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.base_model = resnet50(pretrained=True)
        print(self.base_model)

        # Use pretrained model only.
        if not finetune:
            in_features = self.base_model.fc.in_features
            self.base_model.fc = nn.Linear(in_features, num_classes)
        # Use my finetuning settings (as in config).
        else:
            # Freeze pretrained model layers.
            for param in self.base_model.parameters():
                param.requires_grad = False
            # e.g., ["layer2", "layer3", "layer4", "avgpool", "fc"]
            # Unfreeze the specified (last n) layers to enable
            ↪  learning new features.
            unfreeze_layer_names = config["unfreeze_layers"] # Specific
            ↪  to ResNet!
            for name, param in self.base_model.named_parameters():
                if any(unfreeze_layer_name in name for
                ↪  unfreeze_layer_name in unfreeze_layer_names):
                    param.requires_grad = True

            in_features = self.base_model.fc.in_features

            # Remove the existing fully connected layer.
            self.base_model.fc = nn.Identity()

            # Add a new fully connected layer on top, where I can
            ↪  adjust size of dense layer and dropout.
            self.fc = nn.Sequential(
                nn.Linear(in_features, config["dense_layer_nodes"]),
                nn.ReLU(),
                nn.Dropout(config["dropout"]),
                nn.Linear(config["dense_layer_nodes"], num_classes)
            )

    def forward(self, x):
        if not finetune:
            return self.base_model(x)
        else:
            x = self.base_model(x)
            x = self.fc(x)
            return x
```

Test prediction module *multiclass_prediction.py*

```
from PIL import Image
import os, json, socket
import torch
import numpy as np
from torch import nn
from torchvision import transforms
import matplotlib.pyplot as plt
from skimage.transform import resize
```

```python
from torchvision.models import resnet50

# Import classes from my other files.
from model import CNN
from multiclass_model import model_output_name

# Set working directory and load trainingConfig JSON file that stores
↪  parameters.
current_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(current_dir)
config_filepath = os.path.join(current_dir, "trainingConfig.json")
with open(config_filepath, "r") as jsonfile:
    config = json.load(jsonfile)

# Set dataset path. As I am working on different PCs, set path
↪  device-dependent.
pc_name = socket.gethostname()
if pc_name == "R184W10-CELSIUS":
    dataset_path = config["data_path_BICC"]
elif pc_name == "LAPTOP_UBUH5BJN":
    dataset_path = config["data_path_laptop"]
else:
    print("Wrong dataset path, check again.")

# Count number of classes and load the model.
num_classes = len(os.listdir(dataset_path))
model = CNN(num_classes)
model.load_state_dict(torch.load(f"")) # Removed local path after
↪  prediction.
model.eval()

# Folder path containing images for prediction.
images_folder_path = "" # Removed local path after prediction.

# Initialize lists to store class-wise confidence scores and
↪  counters.
class_confidences = [[] for _ in range(num_classes)]
class_counters = [0] * num_classes

# Set threshold values for each class.
class_thresholds = config["class_thresholds"]  # For testing of
↪  different confidence thresholds for thesis.

# Iterate over images in the folder.
for filename in os.listdir(images_folder_path):
    if filename.endswith(('.jpg', '.jpeg', '.png', '.gif')):
        image_path = os.path.join(images_folder_path, filename)

        # Load image and ensure to have three channels.
        image = Image.open(image_path)
        if image.mode != 'RGB':
            image = image.convert('RGB')

        # Preprocess image.
        transform = transforms.Compose([
            transforms.Resize(list(config["transform_resize"])),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
            ↪   std=[0.229, 0.224, 0.225]),
        ])

        input_image = transform(image).unsqueeze(0)

        # Predict image classes.
        with torch.no_grad():
            output = model(input_image)
            predictions = torch.sigmoid(output)

        # Print the predicted classes and compare with threshold
        ↪   values.
        print(f"\nImage: {filename}")
        for class_idx, confidence in enumerate(predictions.flatten()):
            print(f"Class {class_idx}: Confidence = {confidence:.4f}")
            class_confidences[class_idx].append(confidence)

            # Compare with threshold and increase counter if above
            ↪   threshold.
            if confidence > class_thresholds[class_idx]:
                class_counters[class_idx] += 1

# Calculate and print the average confidence and counter for each
↪   class.
print("\nClass Statistics:")
for class_idx, (confidences, counter) in
↪   enumerate(zip(class_confidences, class_counters)):
    average_confidence = sum(confidences) / len(confidences) if
    ↪   confidences else 0
    print(f"Class {class_idx}: Average Confidence =
    ↪   {average_confidence:.4f}, Count Above Threshold = {counter}")
```