Bachelor thesis
Artificial Intelligence

Radboud University

CNN Image Classification on Military Pictures

Author:
Jan Uwe Feldmann
s4465679

First supervisor/assessor:
dr., Pim Haselager
w.haselager@donders.ru.nl

Second supervisor/assessor:
dr., Franc Grootjen
f.grootjen@ai.ru.nl

June 18, 2018

Abstract

This project is part of a pipeline under the title "Adopt a bullet" that aims at gathering information of different weapon transports by using AI-techniques. One stage in this pipeline consists of identifying the information that is relevant for solving this problem. To approach this stage I implemented a convolutional neural network (CNN) and trained it on a large set of images. The research question was, if it would be able to distinguish between images depicting military armoury and those that are not reliably. In this case, images of tanks have been used for training. After an initial training over 10 epochs, an accuracy of 74.33% was achieved. A second, smaller CNN was trained in an attempt to prevent overfitting. This second CNN achieved a final accuracy of 82.05%. This is a good result, but overfitting still occurred. Further experimentation on its prevention as well as further field-testing of the CNN is recommended, for example by applying it to a web-crawler.

Contents

Chapter 1

Introduction

1.0.1 Motivation

This project arose from a pipeline of projects that share a common goal, which will be explained further in this chapter. First of all I am going to explain our motivation for this pipeline.
Aside from theoretical and ethical research, artificial intelligence has also been applied to a number of projects practically. Most prominently for the construction of military robots and drones, as the media reports more and more frequently in the last few years. We asked ourselves why AI has been used for military practice for some time now but a possible pacifistic movement does not seem to make use of it at all. Is it possible to show with our research that artificial intelligence could in some way support such a movement? To understand in what way AI could help, we investigated some organizations in this field a little bit further.
EMERGENCY is an Italian organization that strives to provide medical treatment to victims of wars for free. They are devoted to their principle that the right to care is a fundamental human right. Equality, quality and social responsibility are the foundations of their organization. Logically following are the objectives to spread peace and abolishing war (*Who We Are—EMERGENCY*, n.d.). Their hospitals represent one possible last destination of a bullet fired by a weapon. The whole route a weapon or bullet takes is not easy to follow, although interfering anywhere in this route requires this knowledge.
DutchArms is a project that could be described as part of the anti-war movement. It was initiated by Lighthouse Reports and produced by Bellingcat and a data-journalism team of KRO-NRCV (a Dutch public broadcaster). They intend to track transactions of Dutch armoury by linking official documents with images and videos on social media. More information can be found on their website (*Doel van het project*, n.d.).
The goal of our pipeline is to continue the tracking of weapons with the use of AI techniques and contribute to a broad movement. I will specifically fo-

cus on the sub-problem of detecting weapons and munition in conflict zones. Recognizing the fact that a lot of data is made public intentionally through governments or unintentionally by reporters and that data may very well be accessible on the internet, I concentrated my research on publicly available data. Since a lot of information either is conveyed in pictures that a journalist might have taken or is accompanied by a picture, finding those relevant pictures could speed up the process of detecting that information. This is why I decided to design an image classifier. One of the most popular image classifiers in the AI-field is a convolutional neural network. Training such a CNN on a set of military pictures should make it capable of classifying a broad range of images and selecting the ones that are relevant for our pipeline. The research question I formulated to evaluate the success of this project is the following: "Can a convolutional neural network reliably classify military images?". The reliability of the classifier will be evaluated on its accuracy on a test set, where an accuracy of 75% of higher denotes a reliable result. This threshold was chosen, because a lower accuracy would make the classification not informative enough, especially if it is used to filter relevant data from the web. The higher the accuracy will be, the more useful the CNN becomes as a tool for journalists or others. It might be specifically interesting how good a CNN-classifier can become with only few very specific classes instead of classifying a wide range of classes.

1.0.2 Pipeline

This section is written in cooperation with other students and will be appear in all our theses. The highlighted part describes the project covered in this thesis:
At the start of January 2018, a team of data-experts, journalists and open-source researchers started a quest, the Dutch Arms boot camp, that concerned the military arms export of the Netherlands. With the use of open-source information, the Dutch Arms team tried to find out as much as possible about the route Dutch military products follow on the way to their final destination that is stated in licences that the Dutch government releases. Whether these products reach this intended destination, and whether the weapons stay there or get distributed further remains questionable, which is why the boot camp was started. The team's editing crew made a call to everyone who is interested in helping out in the boot camp.
This call served as an inspiration for creating a pipeline of projects that will aid the search for information concerning military arms transport in the world.

The first project (see Figure 1.1) in the pipeline will consist of a convolutional neural network, that will serve as an image clas-

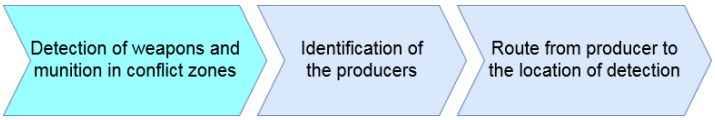


Figure 1.1: Pipeline: Step 1

sifier (Feldmann, 2018). It will be trained on images that either include tanks or something else and discriminate between those. Its strength will be evaluated by its accuracy with a test set of images.
The convolutional neural network will be used by a focussed web crawler that searches the internet for pictures (Roefs, 2018) (see Figure 1.1). The set of pictures the focussed web crawler will search for is determined by the output of the convolutional neural network. The focussed web crawler takes the output of the convolutional neural network as feedback when evaluating a web-page. The evaluation of web-pages guides the focussed web crawler over relevant parts of the internet, to maximize the amount of detected relevant images. Users can ask directed questions to the system in the sense that giving feedback to the algorithm about images will result in more of the positively marked images. There is no possibility to ask a more directed question. These first two projects (the web crawler and the neural network) can together be seen as one step in the pipeline.

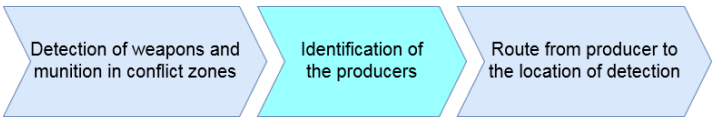


Figure 1.2: Pipeline: Step 2

The third part (see Figure 1.2) of the project deals with the identification of the producer given the information that is gathered by the preceding projects. (Bliek, 2018)
This project will be focusing on the identification of the producers of ammunition by implementing an Artificial Neural Network that can automatically recognize characters of a headstamp code on ammunition. Headstamp codes contain information about the producer and on military ammunition also about the year of production. This information can be used in a later step to obtain the country of production.

The final project (see Figure 1.3) in the pipeline deals with the handling of information the preceding projects have gathered (de Jonge, 2018). The purpose of this project is to identify military arms transport hubs in order

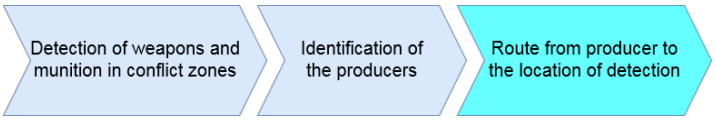


Figure 1.3: Pipeline: Step 3

to help to prevent arms reaching locations that are not stated in licenses governments release.
In short, an algorithm that can make a mapping of the route from production country to final location country of certain types of weapon will be implemented. With the information this algorithm provides, an indication is given of which country produces and ships the military ammunition to certain other countries/regions. So predictions on certain transport routes (e.g. which airports) can be made. This will result in the identification of the most influential military transport hubs. With this information transformed in human-readable output, teams like the Dutch Arms Bootcamp are one step closer to their goal of tracing military products.

Chapter 2

Background

2.0.1 Related Work

In the previous section I described my approach to answering the research question. Since I will use a convolutional neural network, it is important to know how it has been used before. Also, the construction of such a network can highly differ in regards of depth (how many layers it will have) and in regards of operations used (think of activation functions).
Cireşan, Meier, Masci, Gambardella and Schmidhuber have tried to implement a CNN that excels on flexible, high performance. They observed that computational speed is one, if not the most, limiting factor for CNN architecture. They desired to construct a fast CNN that would run on GPUs (graphical processing units). We know that a convolutional layer is parametrized by the size and number of maps, the kernel sizes and some other variables. So the specifics of my convolutional layers will probably differ from Cireşan et al.'s implementation. They used a maximum-pooling layer instead of a sub-sampling layer. From another research, we know that this can lead to a faster convergence and selecting superior invariant features (Scherer, Müller, & Behnke, 2010). It also improves generalization. It is logical that the "top layer is always fully connected, with one output unit per class label" (Cireşan et al., 2011, p.1238), and this will not be different in my implementation.
The results of this paper show that all CNN parameters are adaptable, depending on its particular application. The CNN types discussed in the paper "seem to be the best adaptive image recognizers, provided there is a labeled dataset of sufficient size" (Cireşan et al., 2011, p.1241). They also mention, that good results require big and deep but sparsely connected CNNs, which is computationally prohibitive on CPUs, but feasible on GPUs. Since I will focus on only a few classes, a less big and deep network will probably be sufficient and running it on a CPU will not be as punishing.
Krizhevsky, Sutskever and Hinton trained a large CNN to classify 1.2 million high-resolution images from the ImageNet LSRVRC-2010 contest. In total

there were 1000 different classes. This is relevant for this study, since I will also use high-resolution images from the same source. However it is necessary to keep in mind that with such a high amount of classes, the structure of the CNN will be inherently different from what I will use. Since in this dataset the images are from different sizes, but a CNN requires a constant input dimensionality, downscaling is applied to the images. This will also be necessary in this project. Krizhesky et al.'s output is fed to a 1000-way softmax activation function, which then produces the distribution over the class labels. Krizhevsky et al. mentioned a few ways to reduce overfitting. First of all they artificially enlarged the data set by using label-preserving transformations like generating image translations or altering intensities in the RGB-channels (red, green and blue channels, roughly following the colour receptors in the human eye). They also used droupout, which sets the output of each hidden neuron with a certain probability to zero. Consequently these neurons do not contribute to the forward pass and back-propagation. This way the network samples different architectures, which share weights, but the neurons cannot rely on the presence of other particular neurons. Krizhevsky et al. implemented dropout in the first two fully-connected layers. The results show, that "a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning" (Krizhevsky et al., 2012).
The construction of my network will be inspired by the previously mentioned papers, it will mostly differ in size and depth (since less classes will be used), but it will definitely implement maximum-pooling, a softmax activation function and dropout.

2.0.2 Libraries and Packages

I use some libraries and packages for the implementation of my network and in this section I will explain which and why.
Tensorflow is a simple dataflow-based programming abstraction. Its scripting interface allows to experiment with different model architectures without modifying the core system. It represents the individual mathematical operators as nodes in the dataflow graph, which makes it easier to compose novel layers. The mutable states and operations are treated the same, so it is easy to experiment with different update rules. Execution is done in two phases. First of all the network and its update rules have to be defined, then the optimized version has to be executed, which is optimized by using global information about the computation (Abadi et al., 2016).
Theano improves the execution and development time of machine learning. The repeated computation of complex mathematical expression can be simply written in terms of matrix or tensor operations. The compiler uses "tricks of the trade, heuristic, auxiliary libraries", which "makes it easier

to leverage the rapid development pattern" and "to profit from fast code" (Bergstra et al., 2011, p.2).

Chapter 3

Method

To test, whether a CNN can reliably classify images on their containment of military elements, further specifications have to be made. First of all I will solely focus on pictures depicting tanks (armoured combat vehicles) and control images that do not depict those tanks. I made this choice, since those images have often been taken in a relevant context, for example a conflict zone, or at least while the military object was in use. Photographs of alternative military objects, like bullets, are less feasible, since most of those have been taken in the production facility or in a hospital. Meanwhile images of tanks are available in a large amount and relate to the objective of this research, which is collecting information and tracking the path of military weapons. Focusing on tanks will not give us the opportunity to give a general answer to this research question, but it will at least hint to the right direction. Also some thresholds have to be agreed upon, as when to call a classifier 'reliable'. I would suggest the following: Since an accuracy of 50% would not be better than random guessing (assuming an equal distribution between both classes), I would call everything between 50-75% an okay result. Anything beyond that (75% or higher) I would call reliable. With a rate lower than that, too much information might be lost, and too many images might be falsely flagged as military pictures. This would highly restrict the usefulness of the CNN as an information-filter. Meanwhile a rate of 75% of higher should be informative enough. It is natural that the more reliable the network's classification is, the more valuable it is as a filter.
The images of tanks were taken from the image database ImageNet, using the tags 'tank', 'army tank' 'amored combat vehicle' and 'armoured combat vehicle'. The images not containing tanks were taken from the test collection of the ImageNet Object Detection Challenge, which were available on kaggle.com. A total of 2.976 images were used as training and evaluation data, of which exactly one half were depicting tanks, whereas the other half was composed of random images (not depicting tanks). The images were preprocessed, resized to be of the size 200x200 pixels and put in grayscale.

This was done to reduce the dimensionality of the input data. Consequently the CNN could solely focus on one colour-channel and would not correlate between the sizes of the images. After preprocessing, the images were randomly split into a training set and a test set, where 80% of the images were put in the former and the remaining 20% in the latter.
The next step was to actually implement the convolutional neural net, which was done in the programming environment "Jupyter Notebook", a tool for using python cells as well as markdown cells in one file. This way it was easy to keep an overview and the code was well documented. Also a bunch of libraries were used, which enabled me to set up a network quickly and implement small changes immediately. Tensorflow, Keras and Theano are the most prominent members of said libraries. To see an overview of all libraries used, look at Appendix A.1.
The design I chose for the network was the following: Two convolutional layers, followed by a maximum pooling layer, whereafter a dropout of 50% was implemented. This was followed by a dense layer with 128 output units, which was fed to a dropout layer of again 50%. The last layer was a dense layer with output units corresponding to the number of classes (in this case 2, the 'tank' class and the 'control' class). As an activation function I used softmax.
Since there are no rules or guidelines on high many convolutional layers to use, I used the design of the two networks, that were discussed in the background-section, as an inspiration. Since I had a lot of data for each class and only two classes, two convolutional layers combined with a maximum pooling layer seemed deep and complex enough.
The network was trained on the training set of images for 10 epochs, and evaluations on the test set after each epoch were saved. The final results will be displayed in the next chapter.
Afterwards a second version of this CNN was trained. The design was almost the same, but it was made less complex and deep by removing one convolutional layer. It was trained on the same data, with a different random split on training and evaluation sets, also for a duration of 10 epochs. Again, evaluations after each epoch were saved and will also be displayed in the next chapter.

Chapter 4

Results

4.0.1 First attempt

The CNN was trained with 2.380 samples per epoch, in batches of 30 samples. It was validated on 596 samples after each epoch. One epoch took about 1304 seconds, with an average of 547.9 ms per step. In total the CNN was trained over 10 epochs, it took 3 hours and 37 minutes. The training accuracy and loss, as well as the evaluational accuracy and loss after each epoch were recorded (see graphs 4.1(a) and (b)). The accuracy is the percentage of images that were correctly classified by the network respectively throughout the training phase or the evaluation. The loss is like a summation of the errors made for each image, the goal is to minimize this.
The final accuracy of the trained network on the evaluation set was 74.33%. This means that the classifier works, but it did not achieve an accuracy of 75% or higher, which I would have considered a reliable result.
An explanation for this is indicated very distinctly in graph 4.1(b), but also in graph 4.1(a): while the loss on the training set decreases over epochs, the validation loss actually increases significantly. This can be partly explained by overfitting. Overfitting occurs when you have limited training data, so that "many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution" (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). The point where overfitting becomes quite noticeable is after the second epoch, where both curves intersect. The accuracy on the training set increases in a logarithmic function, which is logical as more and more training should have less and less effect on the accuracy (comparable to the law of diminishing returns). It almost reaches 100% (99.41%), making it seem like it is learning the training set by heart, instead of detecting general features of tanks in the images. Meanwhile the evaluation accuracy fluctuates between a value of 70% and 85%.
Understanding where the CNN makes the most errors could help in preventing overfitting. So I created a confusion matrix with the results of the

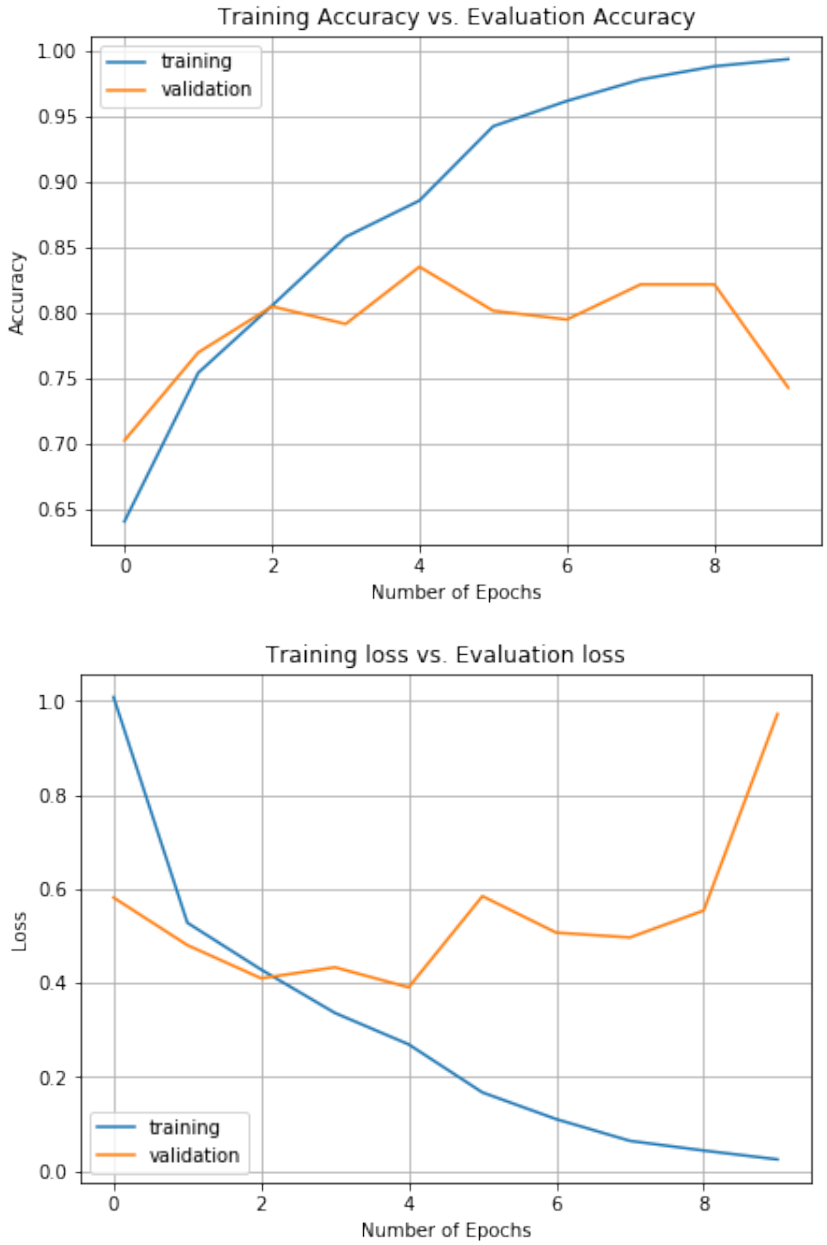


Figure 4.1: (a) Training vs Evaluation Accuracy of the first CNN (b) Training vs Evaluation Loss of the first CNN

evaluation (see figure 4.2).
What we can see in the confusion matrix is, that the biggest group are the true positives (thus, tanks that are actually classified as tanks), which might be promising for further research. False positives play the biggest role in the

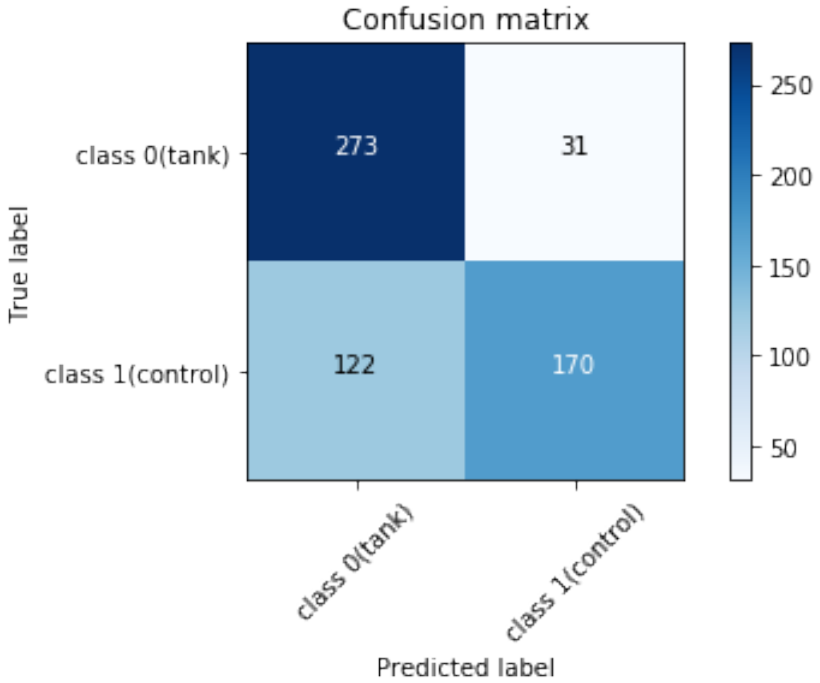
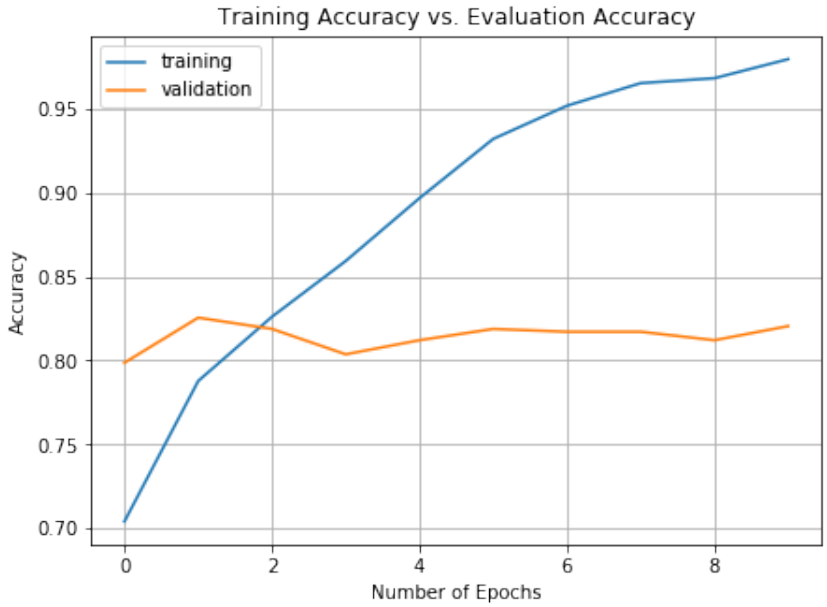


Figure 4.2: A confusion matrix over 596 samples classified by the first CNN

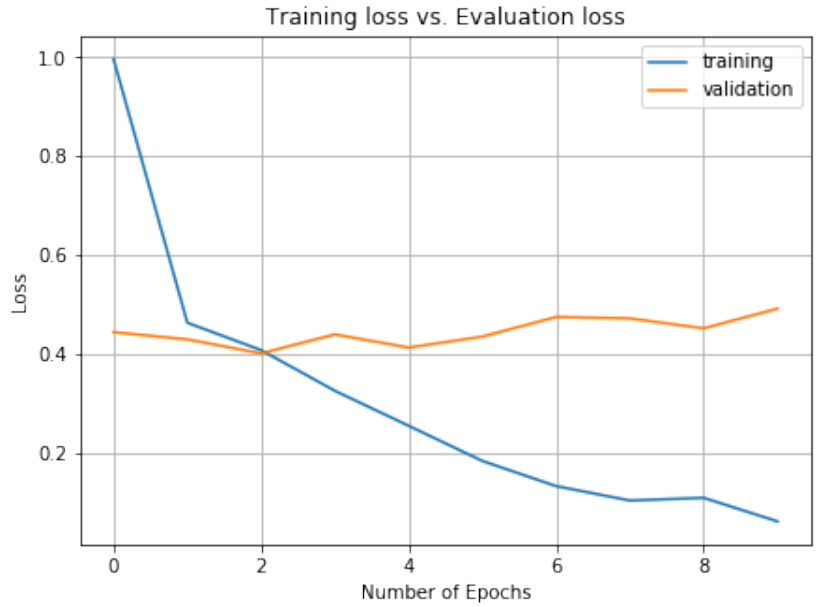
mistakes the classifier makes, so a lot of images are falsely flagged as tanks. Meanwhile only 31 tank images are mistakenly not classified as tanks, which is quite remarkable.

4.0.2 Second attempt

In an attempt to prevent overfitting and increase the training speed I trained a second CNN on the same data, with a different random split. The setup is almost the same except for one convolutional layer missing. A less complex neural net should still be able to learn this task reliably.
It turned out that the training speed did indeed increase significantly. It took about 555 seconds per epoch, with 233 ms per step in average. In total it trained over a time period of 1 hour and 32 minutes, which is less than half the time the first CNN took. Such a smaller network can be trained on more data easily without decreasing its training speed too much.
Again, accuracy and loss values were recorded and graphs were created in the same manner as before (see figure 4.3(a) and 4.3(b)).
We can see that once more the training accuracy rises quite steadily. It peaks after 10 epochs with an accuracy of 97.94%, so not much lower than the more complex network. The evaluational accuracy however seems to be



(a)



(b)

Figure 4.3: (a) Training vs Evaluation Accuracy of the second CNN (b) Training vs Evaluation Loss of the second CNN

more stable at around 81% at all times. Also the loss curve is very similar to the first network. What we can observe is, that the validation loss does

not increase significantly. So while this negative effect of overfitting might be reduced, it is not enough to justify a long training over 10 epochs. If we would have stopped after 1 or 2 epochs, the same result might have been achieved. With a final accuracy of 82.05% this network can somewhat reliably classify images on whether they depict a tank or not.
To showcase the workings of this CNN, I extracted the feature maps of the second layer with all 32 filters on an example image (see figure 4.4). You can clearly see that the shape of the tank is important for most of the filters. Some filters seem to focus more on the continuous track (see picture 1.0/1.0 or 0.8/0.8), but most filters are focusing on the outline of the tank and the shaft of its gun (e.g. 0.0/0.8). One filter in especial (located at 0.4/0.4) is very sensitive to the background or sky.
I also created another confusion matrix with the CNN's results on the evaluation set (see figure 4.5). The confusion matrix is very unremarkable in a positive way. There are not many more false positives than there are false negatives and the biggest group is again the true positives. This speaks for a reliable classification.

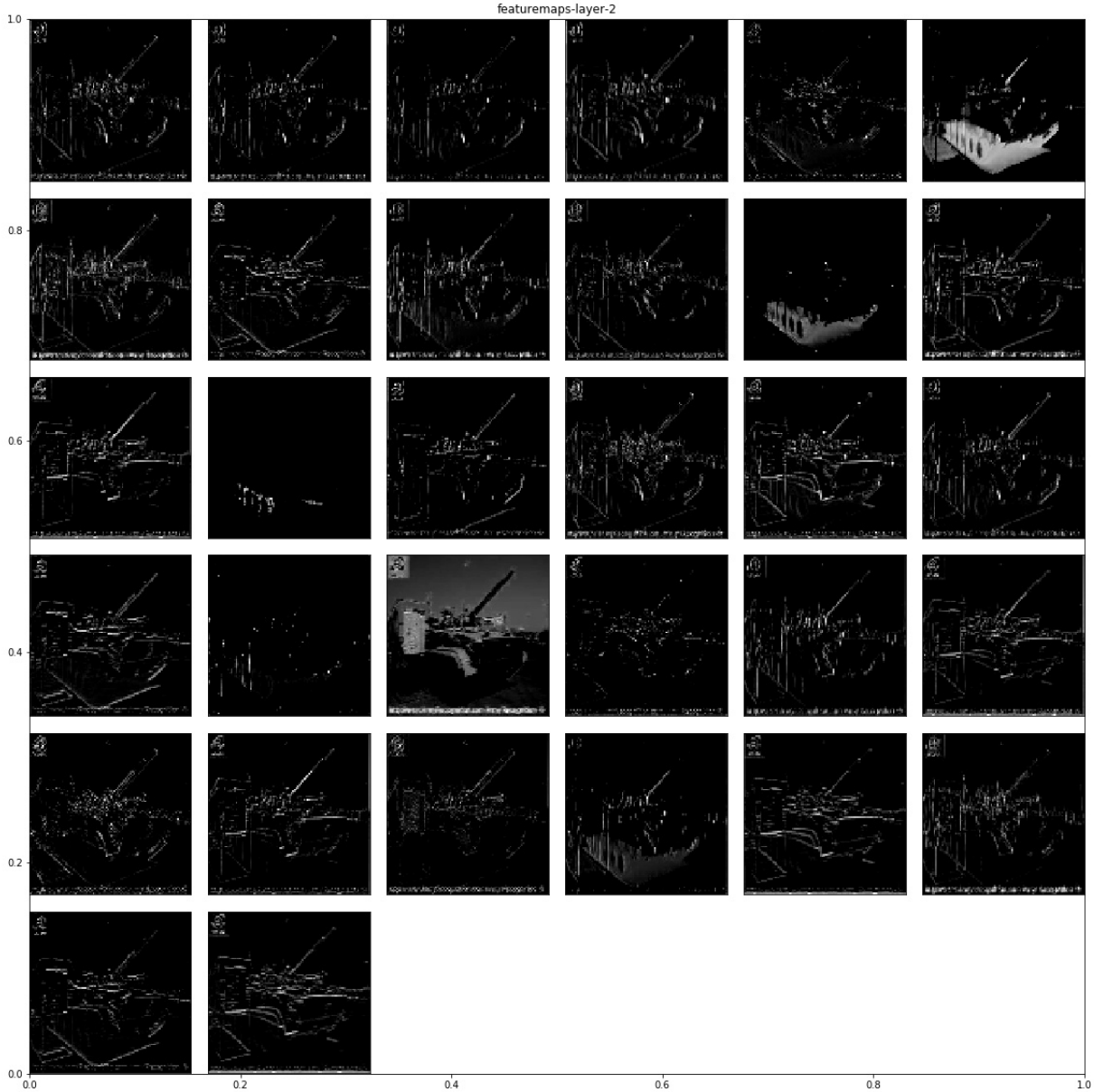


Figure 4.4: Feature maps of the 32 filters after the first layer

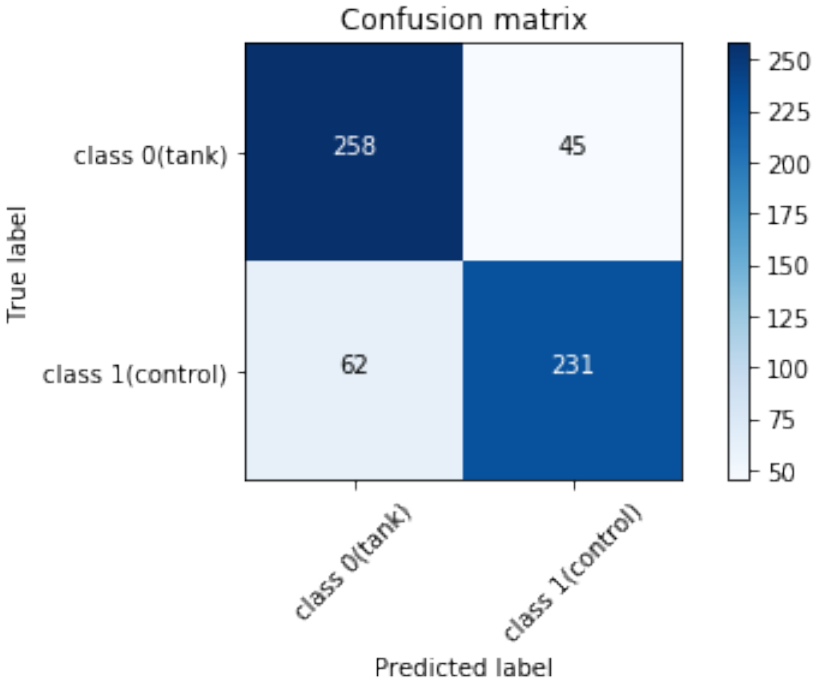


Figure 4.5: A confusion matrix over 596 samples classified by the second CNN

Chapter 5

Conclusions

In conclusion, the research question, whether a convolutional neural network can reliably classify military weapons in an image, can not yet be fully answered. Since only images of tanks have been used, this will only give us a hint towards the answer of this question. However the second neural network, that has been constructed, can indeed reliably classify tanks in an image with an accuracy of 82.05 %. This is more than the 75 % threshold I set for this research question. The biggest challenge I encountered, was the network overfitting on the training data. While it has been tackled in some way, due to the less complex nature of the second CNN, more strategies exist, that could prevent overfitting overall. Then maybe an even higher accuracy could be achieved and the CNN would probably be more applicable to different datasets. Data augmentation would have been a good approach, as more data allows more training and overfitting on the test-set would take longer. Also the dropout rates for both CNNs were steady with a rate of 0.5, but higher or lower dropouts rates could be experimented with. Also I have some assumptions, which could have been tested in this project. I trained both networks over ten epochs, which was not really needed as the evaluational accuracy did not benefit from epochs 5 and higher. So I would recommend either not using the whole test set in each epoch, or using less epochs overall. Also the distribution of classes in my sets were not really realistic, as 50% of all images on websites do not contain tanks. However I wanted to have a high frequency of tank-images, so that the classifier would really learn their features and the resulting accuracy can be critically analysed, since an accuracy of 50% would mean random guessing and everything above would mean, that the classify is more reliable than that. In my research I used a CPU to train the network on. Using a GPU would result in faster training times and can only be recommended. Reducing the complexity of the convolutional neural network was a success and had the results I hoped for. The negative effects of overfitting were reduced, training speed increased and the network had a higher final accuracy. Also the data pre-

processing was sufficient. One channel is more than enough for the network to pick out features with its filters (see Figure 4.4).
In how far is this result meaningful to the motivation behind this study? First of all, it should be possible to train the CNN on different military images as well. A CNN like this can be applied to a web-crawler (Roefs, 2018) and help in identifying sources that connect to weapon transports. The rate of false positives is not too high (see Figure 4.5), which means that only few websites will be downloaded, that got falsely flagged by the CNN. The rate of false negatives is also not too high, which means that the web-crawler would only miss few websites that would have contained meaningful information. In future, the CNN should be trained with more data, and more measures preventing overfitting should be applied to the data and the network. Hopefully an even higher accuracy can then be achieved. A CNN like that can then be applied to a web-crawler and be tested on real web data.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... Brain, G. (2016). TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 265–284. Retrieved from `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi` doi: 10.1038/nn.3331

Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., ... Bengio, Y. (2011). Theano: Deep Learning on GPUs with Python. *Journal of Machine Learning Research*, *1*, 1–48.

Bliek, A. (2018). Recognizing Headstamp Codes using a Convolutional Neural Network. *Bachelor thesis proposal, department of Artificial Intelligence, Radboud University*.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. *IJCAI International Joint Conference on Artificial Intelligence*, 1237–1242. doi: 10.5591/978-1-57735-516-8/IJCAI11-210

de Jonge, M. (2018). Adopt a Bullet: Identifying Hubs of Military Arms Export by Using Supervised Learning Mechanisms on Open-Source Information. *Bachelor thesis proposal, department of Artificial Intelligence, Radboud University*.

Doel van het project. (n.d.). Retrieved 12.06.2018, from `https://dutcharms.nl/about`

Feldmann, J. U. (2018). Identifying Arms in Images with a Convolutional Neural Network. *Bachelor thesis proposal, department of Artificial Intelligence, Radboud University*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 1–9. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007

Roefs, D. N. G. (2018). Focussed image crawling. *Bachelor thesis proposal, department of Artificial Intelligence, Radboud University*.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling op-

erations in convolutional architectures for object recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6354 LNCS*(PART 3), 92–101. doi: 10.1007/978-3-642-15825-4_10

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958. doi: 10.1214/12-AOS1000

Who we are—emergency. (n.d.). Retrieved 12.06.2018, from `https://en.emergency.it/who-we-are/`

Appendix A

Appendix

A.1 Code: Imports and Libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import image
import keras as ks
from keras import backend as K
K.set_image_dim_ordering('th') #this defines the dimension ordering for keras, we use Theano
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.optimizers import SGD, RMSprop, adam
import os
from PIL import Image
from sklearn.cross_validation import train_test_split
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from keras.models import model_from_json
from keras.models import load_model
```

A.2 Code: CNN Construction 1

```
#Choose a sequential model
model =Sequential()
#add a convolutional layer with relu activation
model.add(Convolution2D(32,(3,3),padding='same',input_shape=input_shape))
model.add(Activation('relu'))
#add another convolutional layer
```

```
model.add(Convolution2D(32,(3,3)))
model.add(Activation('relu'))
#add a maximum pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
#we use a 0.5 dropout rate
model.add(Dropout(0.5))
#a layer that flattens the input
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
#another 0.5 dropout
model.add(Dropout(0.5))
#a dense layer with the number of classes and softmax
                                activation for the output
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

A.3 Code: CNN Construction 2

```
#Choose a sequential model
model2 =Sequential()
#add a convolutional layer with relu activation
model2.add(Convolution2D(32,(3,3),padding='same',input_shape=
                                input_shape))
model2.add(Activation('relu'))
#add a maximum pooling layer
model2.add(MaxPooling2D(pool_size=(2,2)))
#we use a 0.5 dropout rate
model2.add(Dropout(0.5))
#a layer that flattens the input
model2.add(Flatten())
model2.add(Dense(128))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
#a dense layer with the number of classes and softmax
                                activation for the output
model2.add(Dense(num_classes))
model2.add(Activation('softmax'))
```