Using CNN to recognize hotel images to combat human trafficking

Machine Learning

Group 32

Abesová Sara (saa275)
Hajková Karolína (kha520)
Šikulová Lucia (lsa890)
Ramadan Yozlem (yrn420)
Zdych Malgorzata (mzh800)

Abstract

This report aims to depict the extent to which the training on a large dataset of images with occlusions impacts the accuracy of a CNN algorithm, as compared to training without occlusions. The ultimate goal was to attempt to find a machine learning method that would recognize those hotel rooms that are likely to be targeted by human-traffickers. The dataset is consisted of approximately 50 000 images and was found on Kaggle, originating from a non-active competition *Hotel-ID to Combat Human Trafficking 2022*. The results were showed that there is no difference between using a partially occluded image set compared to an image set without occlusions.

1 Introduction

Human trafficking activities are affecting the lives of millions of people worldwide. Often for the vision of financial gain, many are forcefully separated from loved ones and exploited, either for sexual, forced labor, or other reasons involving involuntary activities. Vacation resorts are often targeted for these crimes. Having a system that would be capable to find the location of a hotel based on an input of a picture of its room could significantly simplify work of law enforcement companies and potentially save lives. For these reasons, the group of students from group 32 has decided to participate in competition on Kaggle.com, a popularized platform for data science and machine learning tasks. The challenge is titled "Hotel-ID to Combat Human Trafficking 2022 - FGVC9" and it aims to ultimately combat human trafficking by recognizing hotel locations based on pictures of hotel rooms. The team hopes to evaluate the performance of Convolutional Neural Network (CNN), a renowned deep learning technique, trained on a dataset of images with additional occlusions compared to not occluded images. Therefore, the research question can be stated:

How training on a labelled dataset of hotel room images, on an image dataset with occlusions versus without occlusions in the frame, influences the performance of a CNN?

This is measured by the accuracy of the CNN model trained on the two different datasets. The ultimate goal is to use these measurements to find the most suitable CNN model capable of recognizing those hotel rooms that are likely to be targeted by human traffickers and provide a tool that would prevent these crimes from happening in hotel complexes.

2 Data pre-processing

To ensure that working with a large dataset, such as this one containing around 50 000 images, goes smoothly, it is important to prepare the data in a way that makes it easier to implement it in ML algorithms.

Before the process of preparing the data can begin, the procedure of splitting data for testing purposes has to be determined. Although it is highly preferable to have a train, validation and test set, this was not possible for this case (Brownlee, 2020). Although the prior decision to train-validation split with an 80/20 ratio, as it is one of the most commonly used techniques, the validation set had later to be used as a test set, as it was impossible to access the test set of the competition (Team, 2023). More of this issue is discussed in Section 7.

2.1 Incorporation of masks on images

The first step in being able to provide an answer to the research question, the team had to incorporate occlusions, also called masks, on the images of the hotel rooms. The exact number of applied masks was 44711, as the remaining images were used for the test set. There are around 5000 train masks in the dataset of various dimensions and placements, introducing variability of the masked region, as well as the specific content that is obscured. The various perspectives and conditions help improve

the robustness of the model. The incorporation of masks on images considers selecting one randomly generated mask from the train masks folder and applying it to the picture of a hotel room, contributing to the uniqueness of the placement and size of occlusions guaranteeing a more diverse dataset.

2.2 Data augmentation

Data augmentation techniques can be employed to enhance the effective size of the dataset, reduce overfitting, and improve the model's robustness. Based on the team's research on the format of augmented data, there is nothing as the "most preferred" or "perfect" image size for CNN training images, which applies to all tasks and datasets. However, the optimal image size is determined by a number of factors, including the task at hand and available processing resources (Brownlee, 2019a).

In general, training CNN resolutions typically range between pixel sizes of 64x64 and 256x256 (Sabottke and Spieler, 2020). While it may appear counter-intuitive, a reduced number of input variables or features is typically beneficial in deep architecture applications, due to the fact that reducing the number of parameters needed to be tuned reduces the danger of model overfitting. Nowadays, images are often down-sampled to a fraction of the original resolution based on previous results to save processing time and resource needs. However, substantial image resolution reduction eventually can lead to the deletion and loss of key information in the image utilized for categorization, especially if the crucial information is disguised in minor details (Thambawita et al., 2021). It may also introduce errors or distortions that impact the model's performance. To overcome this, it is typically recommended to utilize interpolation algorithms that maintain as much of the image's high-frequency information as feasible. For these reasons, the team has searched various interpolation methods and in the end implemented Lanczos interpolation with pixel size 224x224 (Paige, 1980). This method produces sharper and more detailed results, compared to other less computationally demanding techniques, such as bilinear or bicubic interpolation.

However, despite the amount of research put into the aforementioned methods for resizing images, the team discovered that, for the image classification task using CNNs, it is not necessary to resize, crop, or interpolate images to a standard size. Instead, despite the fact that for CNNs, receiving images of the same size is a strict requirement (Manishgupta, 2020), the team has discovered that all the tasks can be done directly within the CNN architecture. Thus, it was enough to proceed with the original sizes, dimensions, and resolutions of the different hotel room pictures.

3 Structure, architecture and modelling decisions of the CNN

The architecture of the convolutional neural network (CNN) is inspired by the VGGNet architecture and is commonly used for image classification tasks. VGGNet is a good choice for image classification tasks that require high accuracy and robustness to variations in lighting, scale, and orientation (Simonyan and Zisserman, 2014). Therefore, it was an appropriate architecture for the task at hand to identify the Hotel ID using images of the various hotel rooms. The use of deep convolutional layers followed by fully connected layers in VGGNet enables the network to learn high-level features and make accurate predictions on the input images. However, due to a large number of parameters, such networks are computationally expensive to train and require large amounts of data to avoid overfitting (Sapijaszko and Mikhael, 2018). These issues will further be discussed in Section 7, Training the models.

3.1 Structure of the CNN

The developed CNN has 16 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. The input to the network is a 224x224 RGB image (3 channels), as they were scaled to this format in the data preprocessing stage. The reasons for selecting this format are discussed in Section 2, Data Preprocessing. The first convolutional layer has 64 filters, each with a 3x3 kernel size and using "same" padding. It is followed by another convolutional layer with the same number of filters and kernel size. Both layers use the Rectified Linear Unit (ReLU) activation function. The first max pooling layer has a pool size of 2x2 and a stride of 2x2, which reduces the spatial size of the output by a factor of 2. This is followed by two more pairs of convolutional and max pooling layers with 128 and 256 filters, respectively. The fourth pair of convolutional and max pooling layers has 512 filters, and the final pair has 512 filters as well. The final output is then passed through three fully connected

layers, with 4096 units each and the ReLU activation function, and a final output layer with 3116 units (corresponding to the number of classes in the dataset) and softmax activation function. Further specification of the CNN as well as the number of parameters corresponding to each layer, can be seen in Appendix A.1. [1]

3.2 Activation function modelling decisions

The ReLU activation function was selected due to its simplicity and effectiveness in overcoming the problem of vanishing gradients (Bloem, 2023a). The vanishing gradient problem occurs when the gradients of the weights in the network become very small during backpropagation, which can slow down or even prevent learning (Pascanu et al., 2013). One of the main advantages of the ReLU activation function is that it is computationally efficient to compute since it involves a simple thresholding operation that only keeps positive values and sets negative values to zero (Nwankpa et al., 2018). This simplicity allows the network to train faster and with fewer computational resources. Moreover, ReLU has been shown to perform well in deep neural networks with many layers, where other activation functions, such as sigmoid, may suffer from the vanishing gradient problem (Tan and Lim, 2019).

The softmax activation function is typically preferred in the output layer of neural networks for classification tasks, where the goal is to assign input data to one of several categories. The main advantage of softmax is that it produces a probability distribution over the output classes (Bloem, 2023c), which makes it easy to interpret the network's predictions and compare them with the truth labels. That's why the team has chosen to employ the softmax function in the last fully connected layer.

3.3 Optimization

Optimization algorithms are used to update the model's parameters during training in order to minimize the loss function (Bloem, 2023b). The Adam optimizer is a popular choice because it adapts the learning rate for each parameter based on its gradient and previous updates, making it efficient and effective in many scenarios (Kingma and Ba, 2014). In this case, the categorical cross-entropy loss function is used for multi-class classification problems, and the accuracy metric is used to evaluate the performance of the model during training and testing. Even though the actual learning rate used during training can vary based on the adaptive learning rate mechanism of the Adam optimizer, which has to be set to the initial learning rate. A learning rate of 0.001 has been selected as it is the most commonly used default value, and it has performed best when monitoring the validation accuracy of the training data.

3.4 Hyperparameter tuning

The number of epochs was selected based on the image classification task and the available resources for the project. Initially, gridsearch was the desired method to implement a systematic search for the best hyperparameters. However, this proved to be very challenging due to the long training times of the model. Hence, a more sparse search process was employed. The team started training with 50 epochs, but the results showed that the accuracy was poor. A larger number of epochs allows a model to see more data and potentially learn more complex features that can improve its accuracy. Although keeping in mind that preventing overfitting has to be ensured. In the end, 150 epochs were selected, and the validation accuracy was far from a model that is overfitted, but it was unfeasible to train for more epochs, given the task and the computational resources at hand.

The aim was to ensure the batch size is not too small so as not to cause overhead costs associated with loading data from disk to become significant, which can slow down the training process. Yet not too large, as it may not fit into memory and may cause the training process to crash (Brownlee, 2019b). As a rule of thumb, 32 is a good default value, with values above 10 taking advantage of the speed-up of matrix-matrix products over matrix-vector products (Bengio, 2012). Therefore, the lower value of this range was selected, and the value 32 was chosen. It might have been beneficial to tune this parameter more, but there was a lack of time considering the running time of the training process,

The theoretical value of a good hyperparameter value of steps per epoch was calculated based on the dataset. The number of steps per epoch should be set such that the model sees all the data in the

[1]The inspiration for the CNN structure came from https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

dataset at least once during each epoch. Therefore, the number of steps per epoch can be calculated by dividing the total number of samples in the dataset by the batch size (Duca, 2022). Since the dataset consisted of roughly 50,000 images and the chosen batch size number was 32, the plan was to aim for 1500 steps per epoch. However, that calculation was not a realistic estimate. Considering the available resources and the time frame of the project experimenting with such large numbers was unfeasible. Hence, the team tried several values lower than 100 for the training, yet the results were not satisfactory. Thus, the team continued with 100 steps per epoch despite the belief that the model would greatly benefit from a greater number of steps per epoch.

4 Training the models

The training process started with researching the method in which the CNN can be trained. Operating with a large set of around 50 000 image files that was provided by Kaggle competition requires a remote environment that would provide processing power. The problem was that all the images corresponding to a specific hotel were in the subfolders, which made accessibility to the dataset more difficult. Thus, the way to incorporate the ID of hotels in the image name needed to be considered, as the data was unlabeled. Therefore, a short Python code was developed to automate the process. It loops through the images in the subfolder, which has the hotel ID in its name, and renames files to the scheme hotelID + _ + imageName. Correspondingly, a new folder was created, where all the files from the subfolders with changed names were moved to, and it was used for training both model one and two.

For a medium-sized dataset, as is the one being used by the team, using GPU is preferred over CPU due to its power to perform faster computations, resulting in a reduced training time and more precise outcomes (BUBER and DIRI, 2018). Therefore, it was decided based on that the first notebook with CNN model will be run on Kaggle, as the platform offers access to GPU P100 (with a limit of 30 hours per week and 12 hours per one run). Therefore, it was estimated that a minimum of 150 epochs should be run, with 100 steps per epoch. Approximately one step was taking around 2-5 seconds, so the estimated running time of the training was predicted to be less than the upper 12 hours. Eventually, the first model was run for about 10 hours and later saved in two formats .json and .h5 with the use of Tensorflow Keras library.

According to the presented research question, the training on a labelled dataset of hotel room images with occlusions in the frame was performed and it will be referred to as a second model. Firstly, the occlusions - masks were incorporated on the images and the dataset was created in the Google Colab, which is extensively described in the Section 2.1. Since the dataset was created in the Google Colab, the reasonable solution was to train the data in the same place. However, the GPU - P100 available on the platform turned out to be much slower than the one on Kaggle, in comparison it was approximately 20 seconds per step, which would accumulate to more than 50 hours, which was simply not a realistic solution.

Therefore, the dataset with masks incorporated on the pictures needed to be transformed to the Kaggle platform, which took an extensively long time - around 6 hours to download it from Google Colab as a .zip folder and then 5 hours to upload it as the dataset in Kaggle. Eventually, training the second model was performed in the same environment and conditions as it was with the first model, with the use of GPU P100. Deep learning libraries - Keras and Tensorflow Keras were used in the training of both models. Additionally, to differentiate the pictures of the first and second dataset, the "_1" was incorporated to the names of the files in the second dataset, so that the scheme is hotelID + _ + imageName + _1.

5 Evaluation of models

The models first underwent an 80/20 split to create a data set on which their performance was evaluated. This split is done by Keras automatically, as is the selected ratio. Given the use of a CNN model, multiple metrics are applicable in evaluating and comparing the resulting model performance. Before evaluation can begin, it has to be established what can be learned and applied to the models produced. Both models had the same measurement methods applied and then compared, being plotted against each other to provide a visual representation of their differences.

5.1 Permutation Test

The permutation test, also known as the resampling test, is a useful tool in evaluating classification

models comparatively given that there is no other relevant statistic it can be compared to (Golland and Fischl, 2003). In this task, there is no high-quality and relevant measure the model could be compared to so the permutation method was chosen to establish a baseline. Establishing a baseline is crucial in this CNN task as there needs to be a certainty that the model performance is not the result of random chance. That is, the accuracy and other statistical metrics show that there is a significant difference between random chance and a model that went through training. Given the specific and sensitive nature of the task, it was agreed that there is no more reasonable baseline metric than a random guess.

Given a random classification assignment model, the accuracy approaches 0.0032%. This gives a baseline for the accuracy of both models and allows for a more sound evaluation. A full permutation test is not possible in this case, but we can establish a baseline based on it nonetheless, as described above.

5.2 Accuracy

The most common and comprehensive metric is accuracy, which measures the proportion of the correctly classified instances, measured in probability percentage. It is important to note that this metric has to be taken into account, as some models cannot reach a near 100% accuracy given the number of classes possible. In the case of the models, it is essential to note that a large number of classes will be reflected in the number, making it seem small. This is the case even given a highly accurate model.

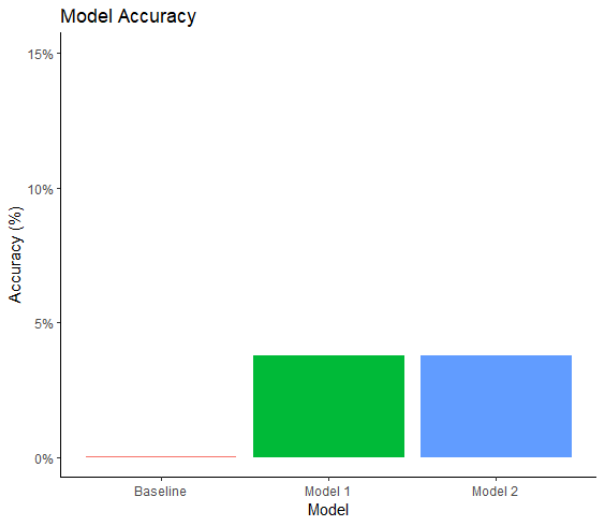

Figure 1: Model Accuracy plotted against Baseline

5.2.1 Full Picture Model

The first model, trained and evaluated on complete images returned an accuracy of 3.75%. While this is very low, it can still be seen that the model is learning something from the data. The baseline for a classification task of 3116 classes would be $1/3116$ which is 0.00032 or 0.032%. This means that the first model is 117 times better than the baseline statistic of an average guess. Thus, it can be concluded that the model is learning and applying the the knowledge currently during testing, but not yet accurate enough for real-world deployment, as the accuracy is still low overall.

5.2.2 Occlusion Model

The second model uses the same baseline, as random guess is not affected by occlusion, only by the number of instances and classes. The second model, using masks, had an accuracy of 3.75%. Again, this is much better than the baseline but brings up an issue that will be discussed in the comparison section.

5.3 Comparison

The similarity in the performance of the two models can be explained in several ways. The first, most obvious is that the feature selection that trains the model does not accommodate for the difference in the two data sets, that is, the mask does not obscure enough relevant information in the image to make a difference in the accuracy of the model either way. The model might have learned to ignore occluded parts of the pictures (Bolei Zhou, 2015). It is also possible that there is a bigger flaw with the approach to training overall, that we were unable to identify. Another less likely probability is that this accuracy is the convergence point of the base of the model and approach to pre-processing that is the same in both models (Charles and Papailiopoulos, 2018).

5.4 Precision, Recall & F1 score

Given that it has been established that the model performance is the same, the metrics will be reported for a single model, and apply to both unless specified otherwise.

$Precision$: 0.9638672156567022
$Recall$: 0.037542201215395005
$F1Score$: 0.0027168376774384514

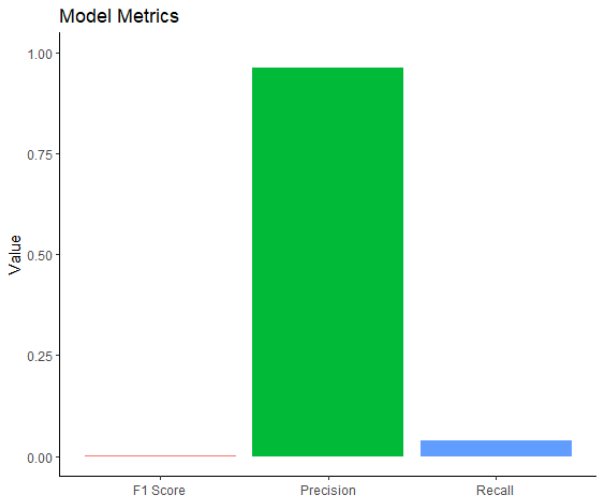

Figure 2: F1 Score, Precision and Recall

The high precision is very good in this case, as considering the subject, it is extremely important to consider class balance and thus the high likelihood of identifying the trafficking rooms is more important than wrongly identifying a normal hotel room. F1 score is more difficult to interpret given the highly complex nature of the task, though it is less important than precision given the cost imbalance inherent to the task topic. It is low, but it could be due to an unbalanced dataset, as well as the complexity.

6 Discussion of the results

As mentioned in the Comparison part, the difference in the models was negligible due to multiple possible reasons. It is important to note, however, that even given their identical performance, they may differ in other aspects such as their interpretability, computational efficiency, or ability to handle different types of images. For example, the occluded model may run faster, given the more easily machine-readable nature of the image, and be able to perform better on future lower-resolution images. Overall, in the future, there is a lot of opportunity for improvement, but the team is satisfied with being able to create a model that learns some patterns and applies this learning successfully. The biggest hurdle that prevents a deeper understanding of why the models perform as they do, is simply the incredible computational demand of a CNN model with 3116 classes. This level of complexity is the field of leading research universities, so the task was monumental for the team (Sapijaszko and Mikhael, 2018). Thus, interpreting the results had to be scaled to what the team had the computation power to be able to run.

7 Limitations and further research

The project's results should be considered in light of various restrictions that may impact the project's efficacy and sustainability. In particular, the number of epochs is an important factor which impacts the performance of the machine learning model. If the number of epochs is too low, this results in not giving the model enough opportunities to learn the patterns in the data hence it may underperform. However, if the number of epochs is too high, this may lead to overfitting the training data, therefore, performing poorly on the test data. The limitation has been encountered due to the restricted number of epochs. Despite the big dataset of around 50,000 images, it was not possible to run a large set of epochs due to time constraints. Consequently, a smaller number of epochs has been run, such as 150, for both datasets with and without masks.

The use of GPU technology in this project indicated a new limitation, which is related to a shortage of time. In particular, the first dataset without occlusion took more than 10 hours to run on Kaggle, causing delays in the project timeline. To overcome this problem, the team created a second dataset on Google Collab after purchasing Collab Pro to facilitate its execution. However, even this approach did not solve the problem of time exhaustion. Eventually, the team compressed the masked images dataset and upload it to Kaggle. It was successfully uploaded and executed on Kaggle after six hours of zipping the masked photos into a single folder.

There are several dataset properties that can be viewed as limitations for the task at hand. Firstly, the ratio of images per class can be a concern since each class represents one hotel. Therefore, the total number of hotels in the dataset is 3116. However, some classes only contain less than 10 images, which may not be sufficient for the model to differentiate between hotels. Additionally, the lack of distinguishing features of the room images, particularly in hotel chains or brands with the same interior designs, presents a further challenge. Although this issue could potentially impact the effectiveness of the model, the team has no control over the dataset, which just mimics the real world. Despite these restrictions, the dataset's large number of hotels is both a strength and a future research opportunity.

Techniques to address the class imbalance and improve feature extraction in hotel images could be investigated.

On top of that, exploring either CNNs other than the VGGNet or other neural networks could help find a different, more optimal way of classifying the images. Due to previous inexperience in the area, it was very difficult to estimate what the best machine-learning approach to the task at hand was. The aim was to use as much theoretical knowledge to use the best practices but, in reality, tacit knowledge on hyperparameter tuning and building a CNN with the different layers and sizes would have been greatly beneficial. It leaves a lot of options unexplored, which could be tested with greater computational power. An approach that is less computationally expensive would be a recommendation if the project had been done all over again.

8 Conclusion

In conclusion, this report set out to investigate the impact of training a CNN algorithm on a large dataset of images with occlusions compared to training without occlusions. The primary objective was to tune a CNN that would accurately recognize hotel rooms that are likely to be targeted by human-traffickers. The dataset used for the study consisted of about 50,000 images obtained from a non-active competition on Kaggle. The findings of the study revealed that there was no significant difference between using partially occluded images and using images without occlusions in terms of accuracy. Therefore, it can be concluded that occlusions may not have a significant impact on the accuracy of CNN algorithms in identifying hotel rooms that are likely to be targeted by human-traffickers. However, due to the low accuracy in both models it can be assumed that conclusion about the hypothesis cannot be drawn. There is not sufficient proof that for a correctly tuned CNN model with considerable accuracy there is no difference between using an occluded dataset of images compared to one where the images are untouched. The team was excited to tackle a project with real-world implications and that seemed challenging. The topic was interesting to the entire group and despite apprehension after considering what the average difficulty level compared to the other groups' projects was, they decided to go through with it. CNNs are understood to be difficult not only in terms of comprehension but also computational power and that turned out to be one of the biggest challenges during the project. It was incredibly difficult to iterate over the datasets, however, in the end, it was relatively successful. The performance of the models can be further improved. It is undoubtedly a sufficient result, considering the complexity of the project. There is abundant opportunity for improvement in terms of the model setup as well as accuracy. The project group is excited to consider future tuning of the hyperparameters and models and further their considerable learning by tackling this overwhelming yet eventually gratifying project.

References

Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478.

Peter Bloem. 2023a. Machine learning - lecture 11: Sequences. VU Amsterdam. Https://mlvu.github.io/lecture11/.

Peter Bloem. 2023b. Machine learning - lecture 2: Linear models and search. VU Amsterdam. Https://mlvu.github.io/lecture02/.

Peter Bloem. 2023c. Machine learning - lecture 6: Beyond linear models. VU Amsterdam. Https://mlvu.github.io/lecture06/.

Agata Lapedriza Aude Oliva Antonio Torralba Bolei Zhou, Aditya Khosla. 2015. Learning deep features for discriminative localization. *Computer Vision Foundation*.

Jason Brownlee. 2019a. Best practices for preparing and augmenting image data for cnns.

Jason Brownlee. 2019b. How to control the stability of training neural networks with the batch size.

Jason Brownlee. 2020. Train-test split for evaluating machine learning algorithms.

Ebubekir BUBER and Banu DIRI. 2018. Performance analysis and cpu vs gpu comparison for deep learning. In *2018 6th International Conference on Control Engineering Information Technology (CEIT)*, pages 1–6.

Zachary Charles and Dimitris Papailiopoulos. 2018. Stability and generalization of learning algorithms that converge to global optima. In *International Conference on Machine Learning*, pages 745–754. PMLR.

Angelica Lo Duca. 2022. What are steps, epochs, and batch size in deep learning.

Polina Golland and Bruce Fischl. 2003. Permutation tests for classification: Towards statistical significance in image-based studies. In *Information Processing in Medical Imaging*, pages 330–341, Berlin, Heidelberg. Springer Berlin Heidelberg.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Manishgupta. 2020. Yolo-you only look once.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

C.C. Paige. 1980. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear Algebra and its Applications*, 34:235–258.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr.

Carl F. Sabottke and Bradley M. Spieler. 2020. The effect of image resolution on deep learning in radiography. *Radiology: Artificial Intelligence*, 2(1).

Genevieve Sapijaszko and Wasfy B Mikhael. 2018. An overview of recent convolutional neural network algorithms for image recognition. In *2018 IEEE 61st International midwest symposium on circuits and systems (MWSCAS)*, pages 743–746. IEEE.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Hong Hui Tan and King Hann Lim. 2019. Vanishing gradient mitigation with deep learning neural network optimization. In *2019 7th international conference on smart computing & communications (ICSCC)*, pages 1–4. IEEE.

The Investopedia Team. 2023. What is the pareto principle-aka the pareto rule or 80/20 rule?

Vajira Thambawita, Inga Strümke, Steven A. Hicks, Pål Halvorsen, Sravanthi Parasa, and Michael A. Riegler. 2021. Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images. *Diagnostics*, 11(12):2183.

A Appendix

A.1 Appendix A

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	1792
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_7 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_8 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_10 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102764544
dense_1 (Dense)	(None, 4096)	16781312
dense_2 (Dense)	(None, 3116)	12766252

Total params: 147,026,796
Trainable params: 147,026,796
Non-trainable params: 0