Image classification on picked up litter: a comparison

Luka de Vrij

Bachelor Thesis - Kunstmatige Intelligentie - Universiteit Utrecht

Supervisor	2025.01.26	Second Examiner
Thijs van Ommen	7795912	Kristina Gogoladze

Abstract
This paper examines three types of models for image classification of a picked up litter dataset. A VGG, ResNet and Vision Transformer are used to compare accuracy and training time on a dataset of roughly 100.000 images. The three pretrained model types were fine-tuned on the dataset. For the VGG and ResNet, this is accomplished by transfer learning, where the model is frozen, and the classifier is trained on the newly defined classes. For the Vision Transformer, a similar process, but one where the inner weights of the Transformer are updated, is used. After training and comparing these models, the newer Vision Transformer attained an accuracy of 84%, while the older ResNet and VGG were limited to around 55% accuracy. This difference can be explained by the different architecture, as well as the efficacy of the fine-tuning process within the Vision Transformer. This suggests the attention mechanism within the Transformer is responsible for the accuracy gain.

Contents

Introduction

Image classification is a rapidly developing field within computer vision. Repetitive tasks that traditionally require humans to correctly classify images are needed for data collection, sorting, autonomous driving and much more. One such example is that of data collection of litter.

There are people who pick up litter all around the country, and record all this data to further help local governments in effectively cleaning up communities. Recording this data is often done by hand: A photo is made of the item, and properties such as ‘Type’, ‘Raw material’ and ‘Brand’ are classified. Additionally, the location is attached to the image. This data can then be sold or used to gain insight on what areas are most polluted, and what can be done to minimize this. Brands can be pressured into changing their packaging material, or legislative action can be taken against them. One such example is Antaflu, a throat lozenge manufacturer who changed their plastic wraps to paper ones after the data showed their wrappers were overrepresented in the data.

Usually, these people are out and about, picking up litter all day, after which they do the classification process at home. Adding the data to, for example, an Excel sheet. This process can be tedious. If this process could in some way be automated, these people could spend longer doing the thing they actually want to do, which is cleaning up neighborhoods. The classification process can be streamlined using automatic image classification. *This paper explores three different image classification models and evaluates which is best suited for classifying picked up litter.*

Traditionally, image classification is done with a convolutional neural network (CNN), a model architecture which learns features through one or more convolutional layers. Many variants exist of the classic CNN, varying the amount of layers or changing more radical aspects. One more radical improvement is the ResNet, which was introduced around 2015. ResNet uses residual connections to allow for deeper networks [2]. This model allows for a more direct comparison to be made between it and a more standard CNN. Additionally, a more recently developed model will be applied, namely the Vision Transformer. Transformers have made great strides in natural language processing, and are seemingly very useful for image classification as well. These Transformer-based models often outperform other state-of-the-art image classification models [1]. The accuracy of these three models will be compared, along with considerations such as training time.

It is expected that the Vision Transformer will perform best in terms of accuracy. This is a newer architecture, and has been proven state-of-the-art on the ImageNet test set and various other benchmarks [1]. The difference between the ResNet and CNN will become visible when using deep models, where the ResNet will outperform the standard CNN in terms of accuracy [2].

First, some background is given on these models. How these models are trained and used will be explained in Methods, along with what dataset will be used and where it was obtained. The results will follow, along with some analysis on these results, after which the paper will conclude.

Background

Convolutional neural networks

Convolutional neural networks find their basis in the mathematical concept of a convolution. In a convolutional network for 2D images, the operation can be defined as follows:[1]

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

For every position (i,j) in the output S, the convolution operation calculates the weighted sum of the surrounding values from the input image I, using the filter K - consisting of weights. Simply put, it is element-wise multiplication, which is then summed.

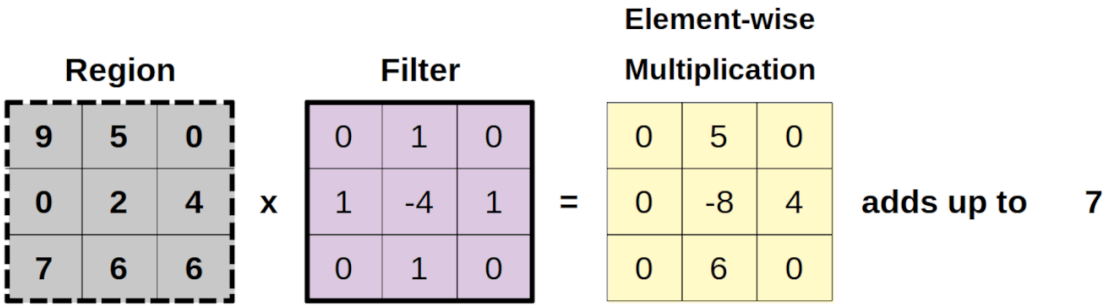

Figure 1: Visual example of a 2D convolution, taken from [10]

By sliding a window (receptive field) over the image, and applying the convolution operation with a given filter (often called the kernel), it is possible to reveal features in the output, called the feature map. This is a crucial part of what a convolutional layer inside a CNN does.

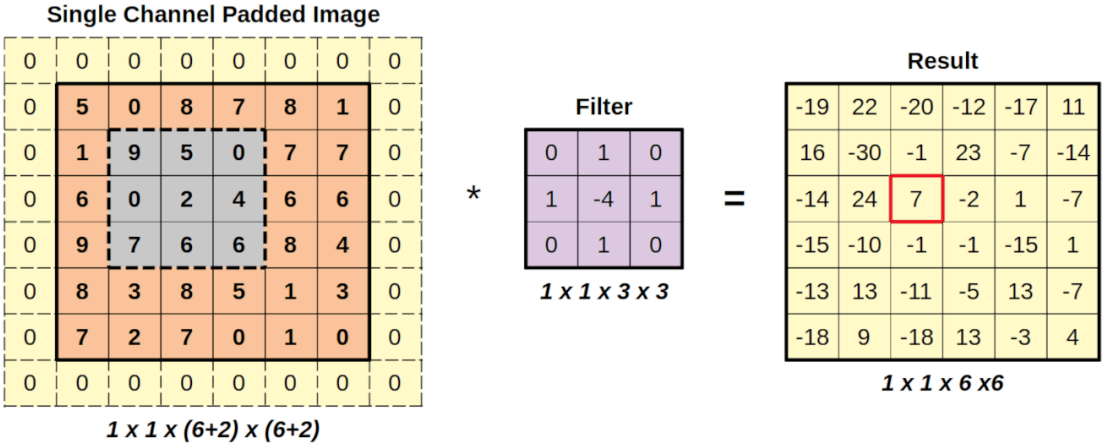

Figure 2: Visual example of the workings of a convolutional layer, applying convolutions on local patches of the input image, taken from [10].

[1] Mathematically, this would be defined as cross-correlation, which is what a convolution within machine learning usually stands for. The difference between true mathematical convolution and machine learning convolution has to do with whether the kernel is flipped. In this paper, a convolution is defined regardless of kernel flipping, since it is not relevant. Learning a kernel can be done regardless: it will simply learn the flipped kernel [9].

These examples are all applied on single channel images, so they would be grayscale; on RGB images this is done thrice over, on each of the channels, and then summed together for one feature map. Additionally, zero-padding is used to keep values balanced around the edges [12].

Generating this feature map can produce various effects, such as blurring and edge detection - with some kernels revealing features that come in handy for classification of objects. Basic features such as edges, corners and textures in an image are recognised by early convolutional layers in a network, with deeper layers attempting to recognise more complex patterns, shapes, objects or parts of objects [12]. However, how do we know which kernels to pick? Luckily, this is where machine learning comes in to help. The most useful kernels are learned by training the CNN.

A convolutional layer inside a CNN consists of more than just applying convolutions to generate a feature map. A nonlinear activation function is applied to the feature map, but more importantly, a subsequent pooling layer follows. [13] states: “Max-pooling creates slight position invariance over larger local regions and down-samples the input image [...]”. This means that we can still recognize a certain feature, even if it is translated slightly. As Goodfellow et al. state: “A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the max pooling (...) operation reports the maximum output within a rectangular neighborhood.” [9] Consider the following examples using max pooling:

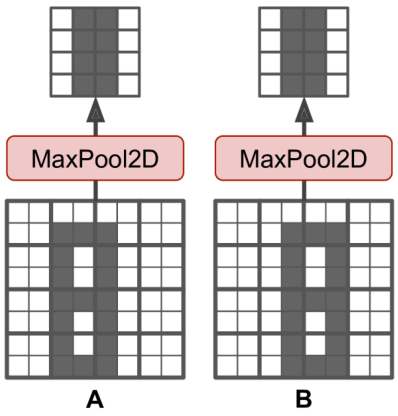

Figure 3: Taken from [12], max pooling is automatically invariant to small translation; a translation of one horizontal pixel has no effect on the output. Additionally, after pooling, 64 outputs are reduced to 16.

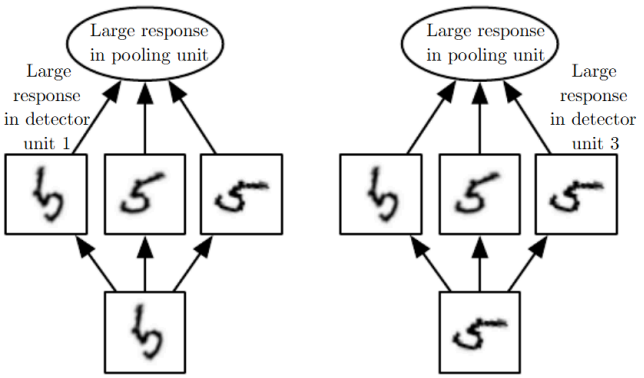

Figure 4: Taken from [9], rotational translation has to be captured by multi-channel pooling. The amount of computation is reduced, since fewer pooling units than outputs (detector units) are required.

The convolution stage, activation function, plus the pooling layer make up a convolutional layer [9]. Most CNN architectures stack a multitude of convolutional layers, followed up by one or more fully connected layers, and a softmax layer that reduces the outputs to a distribution of probabilities on the amount of possible output classes.

Several good CNN architecture variants have gone on to perform well in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), driving more and more innovation. Examples include AlexNet, GoogLeNet, VGG and the ResNet architecture.

ResNet

The ResNet architecture is a response to the question: “Can CNN performance be improved by simply adding more layers?”. The answer is no, as a couple problems become apparent.

Firstly, exploding/vanishing gradient: With traditional CNN’s, the update rule of backpropagation attempts to nudge the weight in the right direction. With neural networks that are quite shallow, this works well. However, as soon as very deep networks are used, a problem arises: the gradient used to update the weights becomes smaller and smaller as it propagates further down the network. As a result, the starting layers will not be updated significantly enough, resulting in next to no new knowledge being learned by the network. Or conversely, when a large gradient keeps growing as it is passed along, resulting in an exploding gradient problem [3, 4].

Secondly, deep networks seem to suffer from a degradation problem. Adding more layers does not make the performance better, or even comparable: it worsens it. This is counterintuitive, since adding more layers should in no way make the performance worse: the added layers could simply do identity mapping, having no effect. He et al. write: “The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers.” [2]

The degradation problem is addressed with shortcut/skip connections between layers. These connections explicitly map identity, whereafter it is summed up element-wise with the output of the layers [2]. While the exploding/vanishing gradient is mostly solved by normalized initialization and intermediate normalization layers, the skip connections also help with backpropagating gradients. Since there are more paths this gradient will flow down, and these paths are shorter, the gradient is more meaningful when fully backpropagated through the network [7]. This allows for deeper networks, built out of several residual blocks, as displayed in Figure 5.

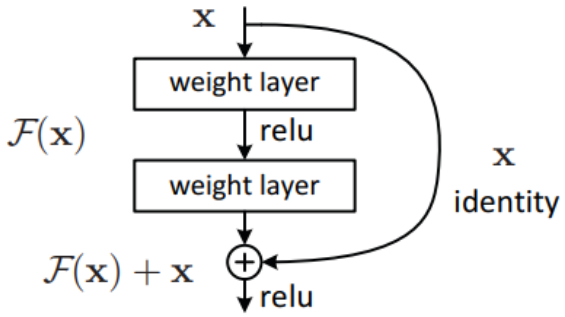

Figure 5: A single residual block; taken from [2]

These residual blocks are combined to form a Residual Network (ResNet), as proposed in Deep Residual Learning for Image Recognition by He et al. in 2015 [2]. They propose various sizes of ResNets, with different amounts of layers: ResNet-18, up to ResNet-152. All variants start with a 7x7 convolutional layer, and a pooling layer. Various amounts of residual blocks are then used, but all end with an average pool, a fully connected layer for classification, and a softmax to normalize them into probabilities per class.

In Figure 6, the ResNet-18 is displayed. Solid lines indicate skip connections where no dimension change is necessary, dotted lines indicate a dimension change; a linear projection is made to match the dimensions [2].

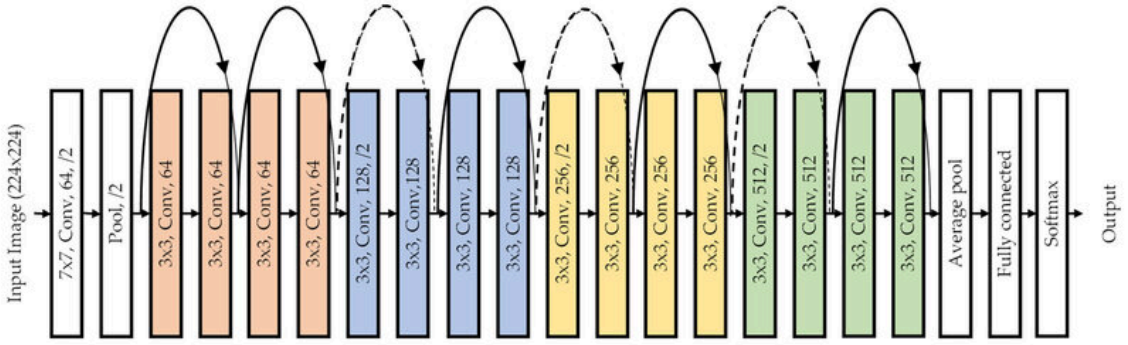

Figure 6: ResNet-18 overview, from [11].

To apply a ResNet on a new problem, one can fully train it on that dataset, but another possibility is to do fine-tuning via transfer learning. A pretrained ResNet-18 is already trained on the ImageNet dataset, with 1000 classes. To retain the general performance of the ResNet, and classify on a new dataset, the fully connected classification layer has to be retrained. The amount of outputs is set to the amount of classes to classify, instead of the pretrained 1000, and the weights are reinitialized at random. All other layers are frozen, and the data is passed through to train the fully connected layer [13].

Vision Transformer

The Vision Transformer (or ViT) was introduced in the paper 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale' by Google Research in 2021. Transformers have been used extensively in the field of natural language processing, to which comparison can be made. The Vision Transformer uses an attention based transformer for fine tuning on image classification tasks.

A crucial aspect to the Transformer model is that of attention. Attention is responsible for allowing the model to attend to other tokens, essentially modelling how important or relevant a current token is to one before. In natural language, these are usually words that connect to relevant other words. The process that decides how important certain tokens are is supported by three pre-trained matrices [8].

When applying this architecture to images, some changes have to be made. Firstly, we only use the encoder. In traditional Transformers, the encoder's job is to process input tokens, while the decoder uses these and its own outputs for generating probabilities on the next token in the sequence [8]. Since we merely require encoding our images, the decoder is discarded. The transformer encoder takes flattened patches of images as the input, instead of words. A 2D image is divided up into patches of a defined resolution, left to right, top to bottom. Each of these patches are then linearly projected into a patch embedding, along with a position embedding. This is the only place where positional information is fed into the model [1].

Similarly to BERT, a learnable embedding token, [class], is prepended to the input. This token will represent the entire image; attention from all subsequent patches will influence this learnable token until it is a representation of the image as a whole. A single multi-layer perceptron (MLP) with one hidden layer is used at the end of the process to classify the learned class embedding to any of the known classes [1].

The encoder contains an amount of layers, dependent on the type of model. In this layer, the multi-head attention is performed, on either 12 or 16 inputs simultaneously. As visualized in Figure 8b, these layers contain two sublayers, each having a residual connection around them, similarly to the ResNet.

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Figure 7: Taken from [1]. Different model variants and their details.

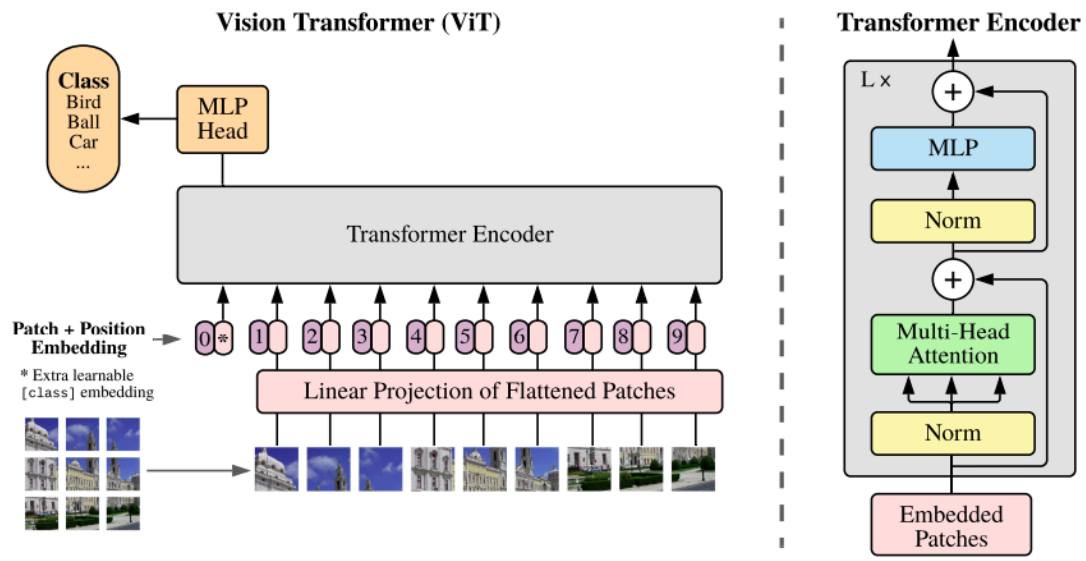

Figure 8: Vision Transformer architecture overview; directly from [1]. The encoder content is seen as two sublayers. This layer is repeated L times depending on the model type.

Fine tuning this model is done by getting rid of the pre-trained MLP head that is responsible for the final classification, and replacing it with a $D \times K$ feed forward layer, with K being the amount of classes, and D the dimension of the patch embeddings. The weights of this layer are zero-initialized [1]. Fine-tuning is done at a higher resolution, since this can improve accuracy [5]. This introduces a problem: “When feeding images of higher resolution, we keep the patch size the same, which results in a larger effective sequence length. [...] the pre-trained position embeddings may no longer be meaningful.” [1] The ViT solves this issue by interpolating the pre-trained position embeddings, so they line up with the new resolution.

Methods

All models were trained on the same dataset. This dataset was provided by the Netherlands' most famous 'litter collector' - Dirk Groot, dubbed Zwerfinator. The dataset consists of 112522 images (1080x1080), each classified on four properties: 'Category', 'Material', 'Type' and 'Brand'. The first two variables are always defined, 'Type' and 'Brand' are not. An example of a data point is as follows:

File	Category	Material	Type	Brand
Fotos/B/BagPlasticChipsLays/BagPlasticChipsLays-00197.jpg	Bag	Plastic	Chips	Lays

Figure 9: Example of a best case scenario image file[2], before processing

Since only 'Category' and 'Material' are always defined, these were used for training. All combinations of these two variables resulted in 1068 categories. Of these, those that had less than 250 training examples, were omitted. This process reduced the total amount of images in the dataset from 112522 to 108207, meaning the omitted classes accounted for 3.8% of the dataset. After this process, 96 classes remained.
These 96 classes include examples such as *Bag_Plastic* and *Bag_Paper*. In order for the models to understand this, the labels are translated to a number, and later to a tensor.

[2] None of the sample images used in this paper are from the original dataset, as the distribution of the original dataset is restricted.

File	ClassLabel	Class
Fotos/B/BagPlasticChipsLays/BagPlasticChipsLays-00197.jpg	0	Bag_Plastic

Models used are:

1. A ResNet model, pre-trained on ImageNet1k. The ResNet model serves as an upgraded CNN, seeing the effect of the architectural change: residual connections.
2. A CNN-like architecture, namely a pretrained VGG-11, to establish a baseline to compare the ResNet to.
3. A Vision Transformer, pre-trained on ImageNet21k. This serves as a newer, different architecture. It is also quite relevant to the current AI boom which popularized Transformers.

Each of these models once were state-of-the-art, and were used in various papers like [1], [2] and [5] for comparison. Additionally, the VGG architecture secured second place in the ImageNet Challenge 2014 on classification, and ResNet won this task in 2015.

The VGG model used is its most simple one, a model with 11 convolutional layers. ResNet has many variants, of which a multitude were tested and evaluated. All of these ResNet variants are pre-trained on ImageNet with 1000 classes, as is the VGG-11. For the Vision Transformer, the base model, vit-base-patch16-224-in21k, was used. This model is pre-trained on ImageNet with 21000 classes.

ResNet

To train the ResNet model, the pretrained weights were used and transfer learning was applied to the 96 classes. The dataset was randomly split into two sets, a training set (80%) and a test set (20%). Each class was evenly split, meaning each class had the same amount of examples when comparing to other classes in that dataset. Images were transformed exactly like that particular ResNet model was trained, i.e. resized to 224x224, and normalized according to the mean and standard deviation of the ImageNet dataset. This resulted in a tensor of images, influenced by the mini-batch size. All layers of the pretrained model were frozen to then override the fully connected classifier head with an unfrozen one, with an output size of 96 (the amount of output classes). Cross entropy loss and an Adam optimizer with a learning rate of 0.001 was used. To keep the training time within limits, an increased resolution (as with the ViT) was not employed. Accuracy is used to evaluate the model on the test set.

Two data preparation pipelines were used: in one, called ‘alldata-96’, uses all the data of each class that has 250 or more examples (before the test-train split). The other (‘max250-96’) uses a maximum of 250 examples per class, making each class balanced. This also decreased training time substantially, resulting in quicker iteration of various configurations. The ResNet model used varied, from ResNet-18, the lowest, to ResNet-101. ResNet-18 has 18 convolutional layers, while ResNet-101 has 101.

VGG-11

VGG-11 was trained similarly to the ResNet model. Since ResNet uses a single fully connected layer as its classifier, overwriting it was straightforward. However, the VGG-11's classifier consists of three fully connected linear layers. Of these, only the last was adapted to the amount of classes.

ViT

To successfully fine-tune the ViT, the total dataset was split up into 2 parts: a training dataset (60%), a validation set (20%) and a test set (20%). Before feeding the training data to the transformer, it was resized from 1080x1080 to either double the native resolution of the model, 448x448, or the original resolution the model was trained on, 224x224. Literature shows fine-tuning on higher resolution can have beneficial effects [5]. When collating images for a batch, positional encodings were interpolated to allow for this higher resolution. After resizing, the images were transformed into tensors, and normalized using the image mean and standard deviation of 0.5, since these are the same values that the model was pre-trained with. The model was fine-tuned with 3 training epochs, with evaluation on the validation set in between epochs and a learning rate of 0.00005. Afterwards, the test set accuracy was evaluated.

Results

All training was done on a GPU using CUDA: NVIDIA GeForce RTX 3070. Further specifications can be found in Appendix B. To attain an accuracy score that was useful, various models and variations were trained and tested.

ResNet

Model	Mini-batch	Epochs	Data preparation	Final train accuracy	Final test accuracy	Training time (seconds)
ResNet-18	16	5	max250-96	0.4138	0.3588	1726
ResNet-18	32	5	max250-96	0.4412	0.3565	1570
ResNet-34	32	5	max250-96	0.4564	0.3623	1620
ResNet-34	32	10	max250-96	0.5287	0.3731	3083
ResNet-18	32	1	alldata-96	0.3903	0.4439	2179
ResNet-34	32	5	alldata-96	0.4883	0.4790	10455
ResNet-50	32	1	alldata-96	0.4781	0.5478	2227
ResNet-50	32	5	alldata-96	0.6261	0.5577	9896
ResNet-101	32	1	alldata-96	0.4514	0.4992	2274
ResNet-101	32	1	alldata-96	0.5506	0.5116	2233
ResNet-101	32	5	alldata-96	0.5976	0.5253	8964

It is clear that ResNet is unable to capture the data fully, at least in these configurations. Adding more convolutional layers is beneficial, as ResNet-50 performs notably better than the lower model variations. ResNet-18 is pre-trained on 1000 ImageNet classes, but is unable to capture just 96 classes in the litter dataset.

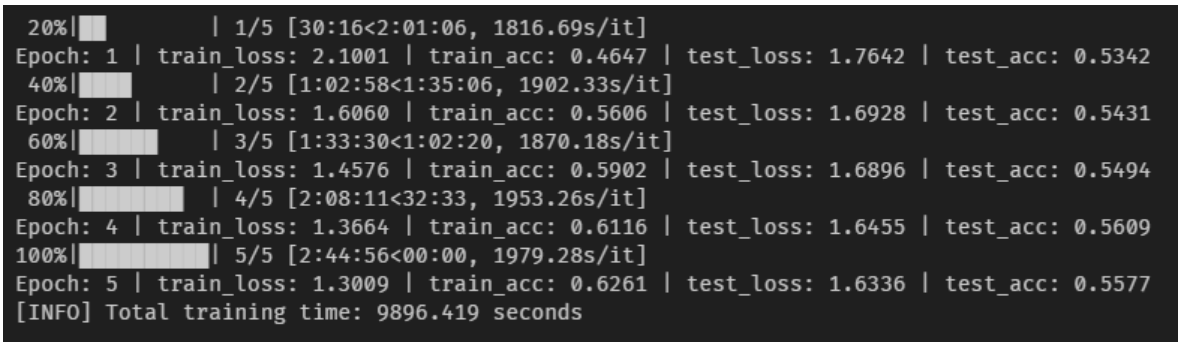

Figure 10: Training output: ResNet-34, 5 epochs. All ResNet training outputs can be found in Appendix A.

Do note that train and test accuracy increase over time, until the 4th epoch. This would imply additional epochs would not further increase accuracy, at least not in this configuration. Larger ResNets, such as the ResNet-101 could benefit from additional training, as the accuracy does not dwindle within 5 epochs (see Appendix).

VGG-11

Model	Mini-batch	Epochs	Data preparation	Final train accuracy	Final test accuracy	Training time (seconds)
VGG-11	32	5	alldata-96	0.3871	0.4667	8953
VGG-11	32	1	alldata-96	0.3493	0.4465	2089

The VGG shows similar performance to the ResNet-18 architecture, with no substantial change in either accuracy or training time.

ViT

Model	Fine-tune resolution	Final eval accuracy	Final test accuracy	Training time (seconds)
vit-base-patch16-224-in21k	224	0.7838	0.7842	6248
vit-base-patch16-224-in21k	448	0.8427	0.8439	10033

Increasing the fine-tuning resolution had a positive effect, bringing the accuracy from 79% to around 84%. Do note training took substantially longer, almost three hours.

Analysis

The Vision Transformer vastly outperforms the other models in terms of accuracy. To begin to understand why this is the case, attention was visualized with Attention Rollout, as proposed in [14]. This allows for the final classification token to be rolled out over the patches that it has most attended to, visualizing which patches were most important for the final classification [1].

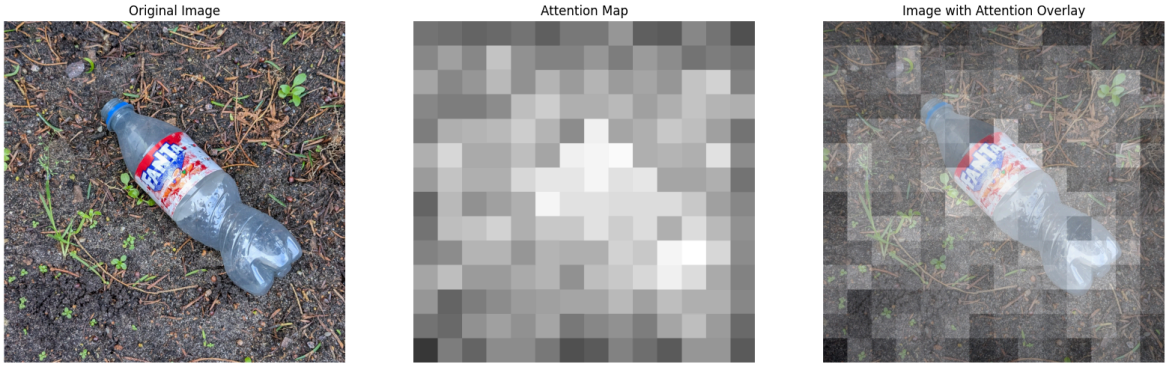

Figure 11: Attention rollout over a sample image: lighter patches indicate higher importance for the final classification.

The CNN architecture tries to accomplish similar behaviour using early convolutional layers, which are supposed to pick up on simple features such as edges, shapes and corners. Subsequent layers are then to ‘combine’ these into more complex features. Both the CNN and ViT may thus attempt to filter out anything unimportant. However, it is possible both the ResNet and VGG are fooled more easily by other objects in the picture, as the dataset contains a myriad of different photos with a lot of variance: photos taken at night, close-up and far away, multiple objects in a single photo, and so forth. The Vision Transformer is better at this filtering process, because the attention mechanism can immediately see all image patches, and isn’t limited to the receptive field of a kernel. The CNN requires more convolutional layers, i.e. a deeper model, to get a more global understanding.

Additionally, the two architectures employ a different fine-tuning process. When fine-tuning the ViT, the classification head is replaced with a D×K feed forward layer, which has to be trained using the new class labels. Training affects the attention matrices within the Transformer that are responsible for self-attention, which allows them to better learn what to attend to. Conversely, the transfer learning process does not allow the pre-trained layers to change according to the new data. Merely the last layer, which is responsible for making a classification, is re-trained. The deeper layers are fully frozen.

The difference between the VGG-11 and ResNet architecture is hard to quantify, since the amount of layers differ between the VGG-11 and the ResNet-18. In these experiments, the performance of the VGG-11 and ResNet-18 was comparable, and training time was similar. The difference between the VGG and ResNet will only become apparent when attempting to train the VGG with a lot of layers, an amount it might not be able to handle. In that case, the ResNet could be used, as it was specifically designed to allow for very deep models.

Conclusion

This paper explored three image classification models and evaluated which was best for classifying picked up litter. Three models were picked, mainly from existing literature. The VGG-11, with a traditional CNN architecture, an upgraded version with residual connections in its convolutional layers, ResNet, and a more recently developed architecture, the Vision Transformer. Each of these were trained on a dataset provided by Dirk Groot, consisting of roughly 100.000 images with labels. Classes were created out of the Category and Material label, and classes with less than 250 images were excluded. 96 classes remained to be fine-tuned with. For the ResNet and VGG-11 model, this dataset was temporarily scaled down, to allow for quicker iteration, until a good set of parameters were found. Fine-tuning the CNN architectures constituted transfer learning, where all layers but the classifier are frozen, and the classifier is retrained on the new data. Similarly, the Vision Transformer was fine-tuned by replacing the MLP head with a zero-initialized feed forward layer, without freezing anything else.

Out of the three models tested, the Vision Transformer vastly outperformed the VGG-11 and ResNet models. Its accuracy was easily pushed to 80 percent, while the VGG and ResNet models struggled to attain anything higher than 55 percent. In terms of training time, both the Vision Transformer and CNN models were trained for similar time periods, reinforcing once more that the Vision Transformer is the optimal model for this problem.

The difference in accuracy between the two model architectures can be explained in two ways: the feature extraction process and the fine-tuning process. The ResNet and VGG are unable to get a global understanding of the image, due to how the convolutional layers operate on a local subspace of the image. The ViT, on the other hand, is able to make global connections using self-attention. The fine-tuning process of the ViT allows this attention mechanism to become even better than it is pre-trained, while the ResNet and VGG have to operate with a mostly frozen network.

Additional improvements could be made by letting the ViT run for longer. The ViT requires a substantial amount of data [1], and could benefit from additional epochs. The ResNet-50, fine-tuned with 5 epochs, would not, as the test accuracy started decreasing after further training. Additionally, for both the CNN architectures and the ViT, bringing down the amount of classes to classify could help: separating the Type and Category labels into individual models would require double the training, but could help in classifying labels which in this approach would have more data.

In short, after evaluating the VGG, ResNet and Vision Transformer, it is clear the Vision Transformer performs best on the litter dataset. Both the ResNet and VGG were not able to match the ViT's performance, even with sizes such as the ResNet-50 or ResNet-101. This is in line with the hypothesis, since the ViT is the newest architecture, and enough data was supplied to successfully train it. Its supremacy over the older architectures comes down to feature extraction: the ViT uses attention to successfully isolate the objects required for classification. In addition, it is further fine-tuned on the dataset, something the CNN architectures are unable to do due to their transfer learning process. To further improve the accuracy, the ViT is the way forward; once more proving that attention is all you need.

Bibliography

1. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, & Neil Houlsby. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. https://arxiv.org/abs/2010.11929
2. Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2015). Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385
3. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (pp. 249–256). PMLR. https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
4. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, 5, 157-66. https://ieeexplore.ieee.org/document/279181
5. Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, & Neil Houlsby. (2020). Big Transfer (BiT): General Visual Representation Learning. https://arxiv.org/abs/1912.11370
6. Brown, J., Gharineiat, Z., & Raj, N. (2023). CNN Based Image Classification of Malicious UAVs. Applied Sciences, 13(1), 240. https://doi.org/10.3390/app13010240
7. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, & Andrew Rabinovich. (2014). Going Deeper with Convolutions. https://arxiv.org/abs/1409.4842
8. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, & Illia Polosukhin. (2023). Attention Is All You Need. https://arxiv.org/abs/1706.03762
9. Ian Goodfellow, Yoshua Bengio, & Aaron Courville (2016). Deep Learning. MIT Press. https://www.deeplearningbook.org/contents/convnets.html
10. By Daniel Voigt Godoy - https://github.com/dvgodoy/dl-visuals/ , under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/deed.en), from https://commons.wikimedia.org/w/index.php?curid=150823503, no changes made
11. Brown, Jason & Gharineiat, Zahra & Raj, Nawin. (2022). CNN Based Image Classification of Malicious UAVs. Applied Sciences. 13. 240. 10.3390/app13010240. https://www.mdpi.com/2076-3417/13/1/240
12. Géron, Aurélien (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol, CA: O'Reilly Media. ISBN 978-1-492-03264-9., pp. 436
13. Cireşan, D., Meier, U., & Schmidhuber, J. (2012). Transfer learning for Latin and Chinese characters with Deep Neural Networks. In The 2012 International Joint Conference on Neural Networks (IJCNN) https://doi.org/10.1109/IJCNN.2012.6252544
14. Samira Abnar, & Willem Zuidema. (2020). Quantifying Attention Flow in Transformers. https://arxiv.org/abs/2005.00928

Appendix

Training outputs

ResNet

(resnet18 - minibatch16 - epoch5 - max250on96)

```
 20%|██        | 1/5 [07:52<31:28, 472.16s/it]
Epoch: 1 | train_loss: 3.5330 | train_acc: 0.1930 | test_loss: 3.0288 | test_acc: 0.2765
 40%|████      | 2/5 [13:11<19:05, 382.00s/it]
Epoch: 2 | train_loss: 2.7321 | train_acc: 0.3331 | test_loss: 2.8734 | test_acc: 0.3137
 60%|██████    | 3/5 [18:30<11:47, 353.60s/it]
Epoch: 3 | train_loss: 2.5060 | train_acc: 0.3738 | test_loss: 2.7898 | test_acc: 0.3423
 80%|████████  | 4/5 [23:43<05:37, 337.54s/it]
Epoch: 4 | train_loss: 2.3910 | train_acc: 0.3951 | test_loss: 2.7414 | test_acc: 0.3554
100%|██████████| 5/5 [28:46<00:00, 345.35s/it]
Epoch: 5 | train_loss: 2.3116 | train_acc: 0.4138 | test_loss: 2.7705 | test_acc: 0.3588
[INFO] Total training time: 1726.757 seconds
```

(resnet18 - minibatch32 - epoch5 - max250on96)

```
 20%|██        | 1/5 [05:11<20:46, 311.71s/it]
Epoch: 1 | train_loss: 3.5746 | train_acc: 0.1926 | test_loss: 3.0589 | test_acc: 0.2808
 40%|████      | 2/5 [10:27<15:42, 314.05s/it]
Epoch: 2 | train_loss: 2.7100 | train_acc: 0.3443 | test_loss: 2.8154 | test_acc: 0.3204
 60%|██████    | 3/5 [15:42<10:28, 314.35s/it]
Epoch: 3 | train_loss: 2.4528 | train_acc: 0.3915 | test_loss: 2.7444 | test_acc: 0.3377
 80%|████████  | 4/5 [21:03<05:17, 317.28s/it]
Epoch: 4 | train_loss: 2.3137 | train_acc: 0.4176 | test_loss: 2.6627 | test_acc: 0.3552
100%|██████████| 5/5 [26:10<00:00, 314.00s/it]
Epoch: 5 | train_loss: 2.2019 | train_acc: 0.4412 | test_loss: 2.6878 | test_acc: 0.3565
[INFO] Total training time: 1570.025 seconds
```

(resnet34 - minibatch32 - epoch5 - max250on96)

```
 20%|██        | 1/5 [05:19<21:17, 319.38s/it]
Epoch: 1 | train_loss: 3.5725 | train_acc: 0.1955 | test_loss: 3.0124 | test_acc: 0.2869
 40%|████      | 2/5 [10:47<16:12, 324.24s/it]
Epoch: 2 | train_loss: 2.6669 | train_acc: 0.3553 | test_loss: 2.7492 | test_acc: 0.3373
 60%|██████    | 3/5 [16:12<10:49, 324.59s/it]
Epoch: 3 | train_loss: 2.4065 | train_acc: 0.4040 | test_loss: 2.6984 | test_acc: 0.3385
 80%|████████  | 4/5 [21:36<05:24, 324.53s/it]
Epoch: 4 | train_loss: 2.2654 | train_acc: 0.4282 | test_loss: 2.5898 | test_acc: 0.3598
100%|██████████| 5/5 [27:00<00:00, 324.08s/it]
Epoch: 5 | train_loss: 2.1486 | train_acc: 0.4564 | test_loss: 2.6177 | test_acc: 0.3623
[INFO] Total training time: 1620.379 seconds
```

(resnet34 - minibatch32 - epoch10 - max250on96)

```
  10%|          | 1/10 [05:13<47:05, 313.97s/it]
Epoch: 1 | train_loss: 2.0657 | train_acc: 0.4729 | test_loss: 2.6136 | test_acc: 0.3648
  20%|          | 2/10 [10:29<42:00, 315.05s/it]
Epoch: 2 | train_loss: 2.0060 | train_acc: 0.4835 | test_loss: 2.6121 | test_acc: 0.3742
  30%|          | 3/10 [15:44<36:45, 315.02s/it]
Epoch: 3 | train_loss: 1.9650 | train_acc: 0.4886 | test_loss: 2.6595 | test_acc: 0.3623
  40%|          | 4/10 [20:35<30:33, 305.56s/it]
Epoch: 4 | train_loss: 1.9301 | train_acc: 0.4948 | test_loss: 2.6005 | test_acc: 0.3756
  50%|          | 5/10 [25:25<24:58, 299.76s/it]
Epoch: 5 | train_loss: 1.8778 | train_acc: 0.5102 | test_loss: 2.6430 | test_acc: 0.3733
  60%|          | 6/10 [30:17<19:48, 297.17s/it]
Epoch: 6 | train_loss: 1.8683 | train_acc: 0.5085 | test_loss: 2.6393 | test_acc: 0.3775
  70%|          | 7/10 [35:08<14:45, 295.31s/it]
Epoch: 7 | train_loss: 1.8379 | train_acc: 0.5142 | test_loss: 2.6404 | test_acc: 0.3746
  80%|          | 8/10 [40:00<09:47, 293.98s/it]
Epoch: 8 | train_loss: 1.8174 | train_acc: 0.5235 | test_loss: 2.6620 | test_acc: 0.3790
  90%|          | 9/10 [45:32<05:06, 306.07s/it]
Epoch: 9 | train_loss: 1.7865 | train_acc: 0.5266 | test_loss: 2.6889 | test_acc: 0.3756
 100%|          | 10/10 [51:23<00:00, 308.31s/it]
Epoch: 10 | train_loss: 1.7699 | train_acc: 0.5287 | test_loss: 2.6895 | test_acc: 0.3731
[INFO] Total training time: 3083.110 seconds
```

(resnet18 - minibatch32 - epoch1 - alldataon96)

```
100%|          | 1/1 [36:18<00:00, 2178.74s/it]
Epoch: 1 | train_loss: 2.3779 | train_acc: 0.3903 | test_loss: 2.1000 | test_acc: 0.4439
[INFO] Total training time: 2178.744 seconds
```

(resnet34 - minibatch32 - epoch5 - alldataon96)

```
  20%|          | 1/5 [36:49<2:27:16, 2209.09s/it]
Epoch: 1 | train_loss: 2.3310 | train_acc: 0.4073 | test_loss: 2.0563 | test_acc: 0.4513
  40%|          | 2/5 [1:11:43<1:47:04, 2141.55s/it]
Epoch: 2 | train_loss: 1.9978 | train_acc: 0.4647 | test_loss: 1.9883 | test_acc: 0.4652
  60%|          | 3/5 [1:45:14<1:09:24, 2082.21s/it]
Epoch: 3 | train_loss: 1.9262 | train_acc: 0.4766 | test_loss: 2.0010 | test_acc: 0.4637
  80%|          | 4/5 [2:18:50<34:15, 2055.93s/it]
Epoch: 4 | train_loss: 1.8918 | train_acc: 0.4819 | test_loss: 1.9818 | test_acc: 0.4706
 100%|          | 5/5 [2:54:15<00:00, 2091.04s/it]
Epoch: 5 | train_loss: 1.8606 | train_acc: 0.4883 | test_loss: 1.9849 | test_acc: 0.4790
[INFO] Total training time: 10455.192 seconds
```

(resnet50 - minibatch32 - epoch1 - alldataon96)

```
 100%|          | 1/1 [37:07<00:00, 2227.99s/it]
 Epoch: 1 | train_loss: 2.0448 | train_acc: 0.4781 | test_loss: 1.7001 | test_acc: 0.5478
 [INFO] Total training time: 2227.996 seconds
```

(resnet50 - minibatch32 - epoch5 - alldataon96)

```
 20%|██        | 1/5 [30:16<2:01:06, 1816.69s/it]
Epoch: 1 | train_loss: 2.1001 | train_acc: 0.4647 | test_loss: 1.7642 | test_acc: 0.5342
 40%|████      | 2/5 [1:02:58<1:35:06, 1902.33s/it]
Epoch: 2 | train_loss: 1.6060 | train_acc: 0.5606 | test_loss: 1.6928 | test_acc: 0.5431
 60%|██████    | 3/5 [1:33:30<1:02:20, 1870.18s/it]
Epoch: 3 | train_loss: 1.4576 | train_acc: 0.5902 | test_loss: 1.6896 | test_acc: 0.5494
 80%|████████  | 4/5 [2:08:11<32:33, 1953.26s/it]
Epoch: 4 | train_loss: 1.3664 | train_acc: 0.6116 | test_loss: 1.6455 | test_acc: 0.5609
100%|██████████| 5/5 [2:44:56<00:00, 1979.28s/it]
Epoch: 5 | train_loss: 1.3009 | train_acc: 0.6261 | test_loss: 1.6336 | test_acc: 0.5577
[INFO] Total training time: 9896.419 seconds
```

(resnet101 - minibatch32 - epoch1 - alldataon96)

```
100%|██████████| 1/1 [37:54<00:00, 2274.56s/it]
Epoch: 1 | train_loss: 2.1616 | train_acc: 0.4514 | test_loss: 1.9124 | test_acc: 0.4992
[INFO] Total training time: 2274.565 seconds
```

```
100%|██████████| 1/1 [37:12<00:00, 2232.82s/it]
Epoch: 1 | train_loss: 1.6376 | train_acc: 0.5506 | test_loss: 1.8635 | test_acc: 0.5116
[INFO] Total training time: 2232.828 seconds
```

(resnet101 - minibatch32 - epoch5 - alldataon96)

```
 20%|██        | 1/5 [32:37<2:10:31, 1957.80s/it]
Epoch: 1 | train_loss: 2.1616 | train_acc: 0.4514 | test_loss: 1.9124 | test_acc: 0.4992
 40%|████      | 2/5 [1:01:52<1:31:55, 1838.39s/it]
Epoch: 2 | train_loss: 1.6966 | train_acc: 0.5366 | test_loss: 1.8678 | test_acc: 0.5089
 60%|██████    | 3/5 [1:32:46<1:01:30, 1845.43s/it]
Epoch: 3 | train_loss: 1.5460 | train_acc: 0.5666 | test_loss: 1.8640 | test_acc: 0.5188
 80%|████████  | 4/5 [2:01:16<29:52, 1792.09s/it]
Epoch: 4 | train_loss: 1.4614 | train_acc: 0.5868 | test_loss: 1.8838 | test_acc: 0.5191
100%|██████████| 5/5 [2:29:23<00:00, 1792.77s/it]
Epoch: 5 | train_loss: 1.3958 | train_acc: 0.5976 | test_loss: 1.8202 | test_acc: 0.5253
[INFO] Total training time: 8963.845 seconds
```

Vision Transformer

vit-base-patch16-224-in21k, 3 epochs, finetuned at 224x224
"epoch": 3.0, "eval_accuracy": 0.7837538120321597, "eval_loss": 0.8692321181297302, "eval_runtime": 369.2227, "eval_samples_per_second": 58.615, "eval_steps_per_second": 3.664, "test_accuracy": 0.784169670085944, "test_loss": 0.8736757040023804, "test_runtime": 524.9169, "test_samples_per_second": 41.229, "test_steps_per_second": 2.578, "total_flos": 1.5105753092484956e+19, "train_loss": 0.6318401496790695, "train_runtime": 6248.1316, "train_samples_per_second": 31.172, "train_steps_per_second": 1.948
vit-base-patch16-224-in21k, 3 epochs, finetuned at 448x448
"epoch": 3.0, "eval_accuracy": 0.84266703631827, "eval_loss": 0.6445400714874268, "eval_runtime": 477.5118, "eval_samples_per_second": 45.322, "eval_steps_per_second": 2.833, "test_accuracy": 0.8439146104796229, "test_loss": 0.6500927805900574, "test_runtime": 601.8901, "test_samples_per_second": 35.957, "test_steps_per_second": 2.248, "total_flos": 6.0423012369939825e+19, "train_loss": 0.8124008000102638, "train_runtime": 10033.0271, "train_samples_per_second": 19.413, "train_steps_per_second": 1.213

Training hardware

AMD Ryzen 5 5600X
NVIDIA GeForce RTX 3070
32 GB of 3200 MHz DDR4 RAM

Images were read from a SATA SSD, with reading speeds of 515MB/s.
Original images were 1080x1080 in JPEG format, averaging between 100 and 500 KB per image.

Code

Code can be found in this repository:
https://github.com/LukaDeVrij/KI_BSc_Thesis