Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Mathias Bynke

Multi-label image classification with language-image models

An approach for a fine-grained domain-specific dataset

Master's thesis in Computer Science
Supervisor: Kerstin Bach
Co-supervisor: Bernt Ivar Utstøl Nødland
June 2022

NTNU
Norwegian University of
Science and Technology

FFI Forsvarets forskningsinstitutt

Mathias Bynke

Multi-label image classification with language-image models

An approach for a fine-grained domain-specific dataset

Master’s thesis in Computer Science
Supervisor: Kerstin Bach
Co-supervisor: Bernt Ivar Utstøl Nødland
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sammendrag

Nylige framskritt innen selvveiledede fortrente bildemodeller (self-supervised pre-trained image models) har gjort det mulig å bygge gode modeller for oppgavespesifikk (task-specific) bildeklassifisering lettere og med mindre treningsdata enn før. Dette har senket terskelen for å lage modeller for nye oppgaver. Utviklingen innen språk-bilde-modeller og utgivelsen av modeller som OpenAIs CLIP [Radford et al., 2021] tillater sågar eksempelløse (zero-shot) bildeklassifikatorer – klassifikatorer konstruert uten noen oppgavespesifikke eksempelbilder – uten annet enn naturlig språk til å definere klassene. For finkornet (fine-grained) klassifisering har eksempelløs bruk av CLIP vist seg å egne seg i svært varierende grad. I noen tilfeller fungerer det godt, i andre tilfeller ikke. Vi undersøker hvor godt denne modellen fungerer som basis for en finkornet flerannotasjonsklassifikator (fine-grained multi-label classifier) på et datasett med spesialiserte militærrelaterte bilder med inkonsekvente tekstannotasjoner. Dette gjør vi ved å bruke språk-bilde-egenskapene til CLIP til å konstruere og evaluere en eksempelløs klassifikator, samt ved å bruke CLIPs innlærte bilderepresentasjoner direkte til å utvikle rent bildebaserte modeller inspirert av k-nærmeste naboer og hurtigminnemodeller (cache models) som drar nytte av bilder med kjente annotasjoner uten å måtte trenes. Under utviklingen av modellene studerer vi effekten av ulike designvalg, blant annet om det lønner seg å trene deler av modellen videre. Siden datasettet har norske tekstannotasjoner, undersøker vi også hvilken effekt valget av språk har på den eksempelløse klassifikatoren.

Vi finner at det for våre data, som tilhører et spesialisert domene, gir mye bedre resultater å lage en klassifikator som bruker CLIPs bilderepresentasjoner direkte, og som sammenligner med bilder med kjente annotasjoner, enn å bruke den til å lage en eksempelløs språk-bilde-klassifikator. Språk-bilde-klassifikatoren gjør det betydelig dårligere enn referansen, en bildebasert 1-nærmeste nabo-modell, mens våre bildebaserte modeller gjør det bedre enn referansen. Det viser seg også at språket i tekstannotasjonene er viktig for hvor godt språk-bilde-klassifikatoren yter. Engelske annotasjoner gjør det bedre enn norske, og manuelle engelske oversettelser av høy kvalitet gjør det bedre enn automatiske oversettelser. Vi peker på forhold vi tror hindrer språk-bilde-klassifikatoren i å nærme seg ytelsen til de bildebaserte klassifikatorene. Den best egnede modellen vi finner, er et søk som finner de bildene blant en mengde referansebilder som ligner mest på søkebildet, og vekter annotasjonene deres ut fra hvor mye de ligner.

Abstract

Recent development in self-supervised pre-trained image models has made it possible to build good models for task-specific image classification more easily and with less training data than before. This has lowered the barrier to creating models for new tasks. The development of language-image pre-training and the release of models like OpenAI's CLIP [Radford et al., 2021] even allow for zero-shot image classifiers – classifiers built without a single task-specific image sample – using natural language to specify the classes. For fine-grained image classification tasks, zero-shot usage of CLIP is shown to have unpredictable performance, doing well with some tasks but not with others. We explore the effectiveness of this model as a basis for a multi-label fine-grained classifier on a dataset of specialized domain military-related images with inconsistent text labels. We do this by using the language-image properties of CLIP to create and evaluate a zero-shot classifier as well as using its learned image representations directly to develop purely image-based models inspired by k-nearest neighbors and cache models which utilize available images with known labels while not requiring any training. As part of the development of the models, we study the effects of various model design choices, one of which is whether to fine-tune parts of the model. As the dataset has Norwegian text labels, we also investigate the effects of the language of the labels on the zero-shot classifier.

We find that, for our specialized domain data, making a classifier that uses CLIP's image representations directly and utilizes images with known labels is drastically more effective than using it to make a zero-shot language-image classifier. The results of the language-image classifier are considerably worse than an image-based 1-nearest neighbor baseline, whereas our image-based models' results are better than this baseline. We also find that the language of the labels is important for the language-image classifier's performance, with English labels performing better than Norwegian labels, and manual, high-quality translation into English performing better than automatic translation. We point to issues that we believe keep the language-image classifier from coming close to the performance of the image-based classifiers. The best performing model we find is a search procedure that identifies the most similar images from a set of reference images and weights their labels according to their similarity to the query image.

Preface

This thesis is the final part of my Master of Science (MSc) degree with a specialization in artificial intelligence at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). The work was carried out in collaboration with the Norwegian Defence Research Establishment (FFI). It was supervised by Kerstin Bach, professor at the Department of Computer Science, and co-supervised by Bernt Ivar Utstøl Nødland, researcher at FFI.

I would like to thank Kerstin and Bernt Ivar for their excellent guidance and support throughout the work on this thesis. I would also like to thank FFI for providing access to the dataset used in this work as well as computational resources and for allowing me to work alongside their researchers. I also appreciate the help of Ole Arne Øverland at FFI, who provided domain expertise for keyword translation and other issues.

Mathias Bynke
Kjeller, June 20, 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since machine learning methods came to dominate the field of image processing in general and image classification in particular, building state-of-the-art classifiers for a new task has required gathering a significant amount of training data specific to the task. For some tasks, getting such data is costly or difficult. There have been efforts to better train machine learning models with fewer training samples. One such effort is the development of general self-supervised pre-trained image models, which can be specialized for specific tasks with significantly less data than models trained for the task from scratch. This is achieved by fine-tuning or otherwise adapting the general model for the specific task. With pre-trained language-image models, one can even build specialized classifiers and other image models with no new training and no other data than a short text description of the desired classes. The opportunities provided by such general models for solving ever-new tasks are still being explored. In this thesis, we contribute to this effort by investigating the effectiveness of specializing a general language-image model for the fine-grained multi-label image classification task of predicting keywords of mostly military-related images.

1.1 Background and Motivation

Artificial intelligence as a field of research has come a long way since the term was first used in 1956 [Russel and Norvig, 2010, p. 17]. Originally aspiring to define any intelligent process so exactly that it can be simulated, it has gone through periods of optimism and bloom as well as pessimism and stagnation. Along the way, artificial intelligence methods have been able to perform ever new tasks previously reserved for humans. In the last decade, a combination of advances in artificial neural networks, increased computing power, and large amounts of available data have accelerated the development, use, and publicity of these methods enormously. Artificial neural networks are a *machine learning* method, meaning that the behavior of the system is not explicitly coded – perhaps not even understood – by humans, but is rather "learned" from some form of experience. Problems that were recently seen by some as unsolvable in the near future, have been solved using neural networks, such as reaching and surpassing human-level performance in the games Go [Silver et al., 2016] and Starcraft 2 [Vinyals et al., 2019], predicting protein folding [Jumper et al., 2021] or generating and manipulating complex, functioning computer code from natural-language instructions [Chen et al., 2021; Li et al., 2022b].

As the effectiveness of neural networks became apparent, actors from an increasing number of fields and with a variety of problems adopted them for their own use. Image processing is such a field. It has gone through a revolution where classic methods carefully crafted by humans have been supplemented with or even replaced by machine learning methods. This was in particular due to convolutional neural networks, which are designed specifically for grid-like data such as images. This change has helped solve a range of image-related problems that are often straightforward for humans to solve, but where a solution is very difficult to formulate explicitly. Examples of this are image classification and semantic segmentation.

The shift towards machine learning methods in image processing has created a need for data to train the models. However, gathering data for a new task can be time-consuming and labor-intensive. For some tasks or domains, it can be infeasible in practice. In other cases, data is available but is of low quality, as it may not be optimized for machine learning. A way to reduce this problem is *transfer learning*, which is the process of adapting a model that was trained for one task on a certain kind of data to another task, possibly using a different kind of data. This can be done by training the model again on new data. Training for the first task is then called *pre-training*, and training for the new task is called *fine-tuning*. For this to be helpful, the old and new tasks need to have something in common, so that the pre-trained model is in a better position to learn the new task than an untrained model would be. For example, a neural network classifying images may have learned to recognize features such as edges or color patterns in photographs of real-world objects, and this knowledge may be relevant across a wide range of tasks involving photographs.

If a task is quite general and requires a model to learn a broad range of concepts, it may be a suitable task to pre-train models for before fine-tuning for any of a wide variety of other tasks. Instead of doing pre-training again for each new specialized problem one wants to solve, a single model pre-trained on such a general task may form the basis for solving several different specialized tasks, saving time, computation, and work. For example, it is common to fine-tune image classification models on the Imagenet dataset [Deng et al., 2009] in order to later perform more specialized image classification. Imagenet has the advantages of being publicly available, large, and quite varied, whereas task-specific data may be sparse. In fact, if the first task is general enough, one may even use the model without fine-tuning at all, either as it is or adapting it somehow. For example, a model trained on Imagenet may be reused without fine-tuning by using the values from one of its final layers, treating them as an image representation vector that is assumed to capture important features of the image.

Imagenet is a labeled dataset, and (pre-)training a model to predict these labels is a supervised learning problem. Even though pre-training allows specialized tasks to be learned with less specialized data during fine-tuning, labeling the large pre-training dataset is still a labor-intensive manual process. This limits the size of such datasets and thus the capabilities of the models pre-trained on them. Recently, there has been great progress in *self-supervised* pre-training, making use of datasets without labels. For instance, contrastive learning has allowed models to learn useful image representations from large sets of images without labels. This removes the need for manual labeling, unlocking a vast amount of unlabeled data for training.

Contrastive language-image pre-training (CLIP) is a neural network model by Radford et al. [2021] aiming to leverage transfer learning to perform new tasks with little or no task-specific data, known as *few-shot* and *zero-shot* learning, respectively. After its release, it has gained attention due to its unprecedented performance in these types of learning and in image representation learning. It achieved this through self-supervised training, outperforming existing models with supervised as well as self-supervised pre-training. Rather than using only images, it is pre-trained on a 400-million-sample dataset of various images and text captions found together on the Internet. Instead of classification, the task was to identify whether or not a certain image-caption pair belongs together. This type of training data is readily available online and does not need to be manually labeled, saving labor. The idea is also that this data format allows the model to learn a broad range of concepts since image captions on the Internet are more varied and rich in information than one-hot class labels, and matching them to the right image requires knowledge of a variety of concepts. The authors do indeed find that CLIP can perform several image classification tasks with good performance even without task-specific data. With the way it uses text during pre-training, it has the benefit that one can tap into its knowledge using text. This makes it possible to use it zero-shot without any fine-tuning, lowering the bar for using it to solve new tasks. Indeed, this property has been used with success by others since the model's release, and new work using it is still being done.

The Norwegian Defence Research Establishment (FFI) has a dataset of military-related images, some of which are labeled with text keywords in Norwegian. The keywords specify different aspects of the images' contents, such as the specific make of a truck or a broad category of equipment. The dataset is fine-grained and of a specialized domain, and there is considerable noise in the keywords. FFI is interested in exploring how machine learning can be used to add value to the dataset, for instance by inferring keywords for the unlabeled images from the labeled ones. CLIP is a promising model to apply in this case. This is partly because it handles both images and text and is specifically designed to look for relationships between them, which is at the core of this problem. In addition, CLIP can be used as is – without any training other than the pre-training which has already been performed by Radford et al. [2021]. Even disregarding the language abilities of the model, its high-quality learned image representations are promising as the basis for new image models. Investigating the application of CLIP to this task is part of the collective, ongoing effort to investigate how well pre-trained task-agnostic models perform on ever new tasks, in this case fine-grained multi-label classification on military-related images with noisy labels.

FFI's dataset was collected for different purposes over time, impacting its quality in a machine learning setting. Curated and high-quality datasets are useful and popular for training and evaluating machine learning models. However, such datasets do not realistically represent the available data in all cases. In real situations, the available data may be less suitable for machine learning; it may have been gathered at different times, by different people, and for different purposes. This may cause inconsistencies across the dataset, and notably in the labels, as the labeling policy may not have been the same throughout the data gathering. Knowing how to make use of such data despite its shortcomings may be valuable in situations where improving the quality of the data is impractical.

1.2 Goals and Research Questions

Goal Determining how well a general pre-trained image model performs on fine-grained multi-label classification on a specialized image dataset with inconsistent labels.

In this work, we aim to explore ways to utilize the pre-training of a general pre-trained model for our task, which is to predict keywords for the images in FFI's dataset. This task is a fine-grained, multi-label classification problem with images from the military domain and with noisy text labels. While other models could be used as well, there is value in investigating the capabilities of a self-supervised, pre-trained, and task-agnostic model by studying its ability to solve this type of task. In addition to adding to the knowledge about such pre-trained models, this can be seen as part of a broader goal of finding good models for this type of task.

Research question 1 How have previous works approached tasks similar to ours?

For informing the choice of approach for our problem and putting it into context, we want to know what existing approaches work best for problems with similar characteristics, as these may be promising for our problem as well.

Research question 2 How can the knowledge from CLIP's pre-training best be harnessed for our task?

We choose to focus our attention on the CLIP model due to its language understanding, broad semantic knowledge, and unprecedented performance in zero- and few-shot learning and representation learning. This makes it a promising pre-trained model to make use of for our problem. However, what way to best utilize it is not clear. This is a broad question, and we define two subquestions specifying more specific paths of investigation.

Subquestion 2 a How well suited is the language-image zero-shot classification method of Radford et al. [2021] for our task?

Subquestion 2 b How well suited is CLIP's pre-trained image encoder for our task without using text?

Since CLIP handles text, it is possible to use the text labels of the dataset to create a classifier like Radford et al. [2021] do. However, the text is in Norwegian rather than English. This gives us a chance to investigate the impact of this difference on the classifier's performance. The vocabulary may be too specialized for CLIP's text encoder to perform well. Still, its image encoder may have learned concepts that can be useful even when processing images from foreign domains. It is thus not immediately clear if the best way to use CLIP is to use a language-image classifier or some other architecture making use of the image representations learned during CLIP's pre-training.

1.3 Research Method

We apply an experimental methodology to examine the research questions. We build several models and evaluate them on the same dataset. This allows us to compare them to each other directly. We evaluate the zero-shot language-image approach of Radford et al. [2021], only adjusting it to handle the multi-label case. We also introduce our own methods based only on the image representation vectors generated by CLIP, comparing them to a simple 1-nearest neighbor model baseline, as used by Conde and Turgutlu [2021]. These methods are not zero-shot. We do not fine-tune CLIP itself, but inspired by Zhang et al. [2021] we experiment with fine-tuning an add-on to the CLIP-model's image encoder.

1.4 Contributions

In this thesis, we present evaluations of a set of multi-label image classifiers based on the learned representations of the generally pre-trained language-image model CLIP on a specialized domain, fine-grained task: classifying military-related images with text labels. We adapt a zero-shot language-image classifier for our task and evaluate it, finding that its performance is relatively poor. We also find that this classifier is sensitive to the language of the text labels, with even a poor translation from Norwegian to English resulting in an improvement, and a manual, high-quality translation improving it even more. We design and evaluate several models that use only image representations along with reference images with known labels. We show the effects of various model design decisions. Of all the models, a training-free search procedure inspired by k-nearest neighbors gives the best results on our task.

1.5 Thesis Structure

In chapter 2 we present theory and research that we use directly or let inform our work in this thesis. In chapter 3 we describe FFI's dataset and the processing we perform on it. We also present the architectures of the models we build and evaluate. In chapter 4 we describe what experiments we perform and how we perform them, and we present their results. In chapter 5 we discuss the results of the experiments and what we can learn from them. Finally, in chapter 6 we sum up the findings and suggest paths for further research.

Chapter 2

Background

Our work builds on that of others. This includes both established techniques we use for creating and evaluating models and the results of other research informing our approach. In this chapter, we present theory and research underlying our work. Parts of this chapter are adapted from Bynke [2021].

2.1 Background Theory

This section presents theory that we use in this thesis. We describe the development of image processing leading up to our task. We then go into the technical details underlying the CLIP model as well as methods we use for building and evaluating our own models.

2.1.1 Image processing

Image processing means transforming or generating images in some way, or extracting knowledge about the content of images. This field, which has traditionally made extensive use of human-designed algorithms and transformations, has undergone great development since the rise of artificial neural networks. Convolutional neural networks (CNNs) exploit the grid structure of images by learning transformations that are invariant to translation and thus do not need to learn each visual concept again for each position in the image. They revolutionized image processing with their superior ability to classify images when compared to earlier approaches. The annual ImageNet competition, where models competed to classify images from the large ImageNet dataset [Deng et al., 2009], was won by a great margin in 2012 by the CNN AlexNet [Krizhevsky et al., 2012]. This brought increasing attention to this type of model, and the CNNs continued to be improved in later iterations of the ImageNet competition and elsewhere. More recently, a newer network architecture originally designed for text, the transformer, is being used for images in place of CNNs with great success [Dosovitskiy et al., 2020].

Large and high-quality datasets like ImageNet, which contains over a million manually labeled images of certain objects, have been important for the development of image processing and other machine learning methods. In real-world applications, however, the data available might not be this plentiful or well labeled. For such cases, evaluating models on these clean datasets gives unrealistic results. Actors may have collected data for other purposes, and seeing the success of modern artificial intelligence methods, they may want to make use of them on their own data. Since the data was not collected with machine learning in mind, and perhaps not even meant for consumption by a computer, one may face certain problems, such as labels missing or being inconsistent or the data not covering all cases that a machine learning model would face when used. In particular, neural networks can require a large amount of training data to perform well, which poses a problem when the task is complex and the data is insufficient.

Despite ImageNet's large size, language models have benefited from datasets that are much larger still. This is made possible by models that do not require labeled data but can be trained unsupervised on vast amounts of text, which is an abundant resource online. Notable examples of large language models trained this way are BERT [Devlin et al., 2018] and the GPT models, most recently GPT-3 [Brown et al., 2020]. There have been efforts to replicate this success of unsupervised training for image models. One way to do this is to train a model to recognize whether or not two images are altered versions of the same image. This does not require any labeling of the images, unlocking vast amounts of data. This method uses *contrastive training*, which presents true and false matches to the model and uses a special contrastive loss to make the model associate only the true matches with each other. As a consequence, the model learns to create useful representations of images that can then be used for another task directly or as a starting point for fine-tuning a model for a specific task. Unsupervised training of large, general models has proved effective, and the field has seen great success in the last half-decade.

Marking a milestone in this area, an alternative approach is to use contrastive training on pairs of *images and text captions.* The model is then trained to recognize whether the pairing is a "correct" one or randomly chosen. We refer to this use of both images and text as *language-image* training. This can be achieved with an image model and a text model working in tandem. The text captions are assumed to contain useful information about the images, requiring the model to acquire knowledge about the contents of the images in order to learn the task successfully. Like the pure image contrastive training, this learns useful image representations. In fact, CLIP, which is such a model, pushed the frontier on learning such representations, surpassing existing methods [Radford et al., 2021]. However, this approach comes with the added advantage that the model also has a text component. This unlocks new ways of interacting with the model, such as designing text captions that correspond to classes and using the model to assign images to the most relevant class, effectively classifying the images. This means that a classifier can be built without training on or even seeing a single task-specific sample image.

2.1.2 Contrastive language-image pre-training

In this section, we describe theory that underlies the Contrastive language-image pre-training (CLIP) model before describing the model itself, focusing on the technical details.

Softmax

The softmax operation is defined like this: [Russel and Norvig, 2010, p. 848]

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

It has two useful properties: The components of the resulting vector sum to one, and large components of the input vectors are given exponentially more weight than small entries. It is commonly used to normalize a vector in order to be able to interpret it as a probability distribution. The function of softmax can be interpreted in multiple ways. It can be seen as a continuous and differentiable version of the one-hot argmax function, which identifies which component of the input vector is the highest, giving a value of one at the corresponding position in the output vector and zero everywhere else. It is also a generalization of the logistic function $(1 + e^{-z})^{-1}$ from one to several dimensions.

Cosine similarity

Cosine similarity is one of several ways to measure how similar two vectors are to each other. It is defined like this, the last formulation being the one used by the CLIP model:

$$\text{Cosine similarity}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\vec{x}}{\|\vec{x}\|} \cdot \frac{\vec{y}}{\|\vec{y}\|} \tag{2.1}$$

The name “cosine similarity” refers to its geometric interpretation as the cosine of the angle between the two vectors. This follows from a formulation of the dot product: $\vec{x} \cdot \vec{y} = \|\vec{x}\| \|\vec{y}\| \cos(\vec{x}, \vec{y})$. If the two vectors have the same direction, the similarity is 1, and if they have opposite directions, it is -1. If the two vectors are perpendicular to each other, the similarity is 0. Note that since the vectors are normalized by dividing by their magnitude, their magnitudes are ignored when computing the cosine similarity.

Attention and transformers

When processing sequences such as text with neural networks, a common approach has been to use so-called recurrent neural networks to convert the sequence to a single representation vector for further processing. This type of model consumes the sequence one element at a time, gradually updating the representation vector. Two consequences of this limit the training of high-performance models. First, all the relevant information in the sequence needs to be encoded in the representation vector. This requires this vector to be large enough. Even if it is, it is hard for a model to learn to capture the entire sequence equally well. When moving from one end of the sequence to the other, it is likely that the later elements will be better represented than the earlier elements because they were more recently seen by the model. Second, the sequential nature of the model makes it impossible to process all the elements in parallel. Parallelization is desirable and often used with other types of neural networks because it can greatly speed up the training process.

Bahdanau et al. [2014] address the problem of the limiting representation vector in text translation. They introduce a mechanism called *attention*, which supplements the recurrent neural network by learning to recognize which parts of the original sentence are relevant at each position in the translated sentence. Information from these relevant parts is then used directly instead of having to be encoded implicitly in the representation vector. For each position in the output sentence, the attention mechanism considers all positions in the input sentence and estimates how relevant the information there is for the output position in question. It then takes the softmax of these estimates. This result is used as weights for computing the weighted average of the information at all input positions. This weighted average is then one of the inputs when deciding what to put in the output. The attention is trained along with the rest of the model.

The attention mechanism has since been developed further. The sequence is processed and divided into different parts at each position of the input and output sequences: *queries*, *keys* and *values*. The queries can be thought of as representing what information is required at each output position considering what has been put in the output sequence so far. The keys represent what information is provided at each input position. The values provide the information of each input position. When the queries are drawn from the output sequence and the keys and values from the input sequence, this is called cross-attention. In contrast, self-attention, a modification to the attention mechanism, has also been introduced. It takes queries, keys, and values all from the input sequence. This makes the attention mechanism function more like a layer in a neural network, allowing several self-attention layers to be stacked for increased learning capacity.

Vaswani et al. [2017] address the problem of parallelization. They do this by doing away with the recurrent neural networks, instead introducing a new architecture called a *transformer*. It is based on attention alone, heavily utilizing self-attention. This allows the models to be trained significantly more quickly. This has become the state-of-the-art architecture for various forms of natural language processing.

Even though they were invented for natural language, transformers have been shown to perform very well on images as well, challenging the dominance of convolutional neural networks. Dosovitskiy et al. [2020] apply a transformer to images with minimal changes to its architecture. To obtain a sequence in the format expected by the transformer, they split the image into patches of equal size and transform them into embeddings as one would with words when processing text. The resulting models achieve results matching the state of the art while requiring fewer resources to train.

Contrastive learning

When training a model for classification, one might make it predict the probabilities of a sample belonging to each possible class. The loss function is chosen so that minimizing it makes the probability of the correct class be close to 1, while all other probabilities are close to 0. In contrastive learning, however, the model is trained to transform samples into representation vectors that are close to each other if the samples are similar, and far apart if the samples are dissimilar [Hadsell et al., 2006].

A contrastive loss is used that considers pairs of representations and a label signifying whether they are “similar” according to some definition of this term. When minimized, this loss pushes the dissimilar samples apart in the representation space and pulls the similar samples together. The representations can then be used for classification or other tasks.

Prior knowledge is used to decide which samples are similar and thus should be pulled together. For instance, an image can be manipulated in several ways that are assumed to not remove relevant information, such as rotating it. In that case, two manipulated images can be defined as similar if and only if they are drawn from the same original image. This way, the model can be trained unsupervised, making use of unlabeled samples. Approaches like this have been used with success to create representations with unsupervised training [Oord et al., 2018; Chen et al., 2020a]. If the samples are labeled with classes, they can instead be defined as similar if and only if they belong to the same class.

The CLIP model

CLIP (Contrastive language-image pre-training) [Radford et al., 2021] is a model that compares images to text captions. Given a set of images and a set of text strings as input, it outputs an estimated probability of each text-image pair belonging together.

CLIP consists of an image encoder and a text encoder, which produce vector representations (or feature vectors) of the same length. These feature vectors are compared using cosine similarity. The model is trained on a large dataset of images and corresponding image captions retrieved from public sources on the Internet. The training is contrastive, making the model output a high similarity if the text-image pair is a real pair occuring in the dataset, and a low similarity if it is a random pairing. This self-supervised training allows it to utilize large amounts of available unlabeled data for learning good representations instead of requiring manually labeled image datasets. This is similar to how language models have been able to learn from vast unlabeled text datasets. The image captions provide richer information about the images than a one-hot class label, and CLIP’s creators show that it does indeed learn concepts relevant for several tasks, as long as they are not too niche or complex. The trained model’s parameters have been publicly released.

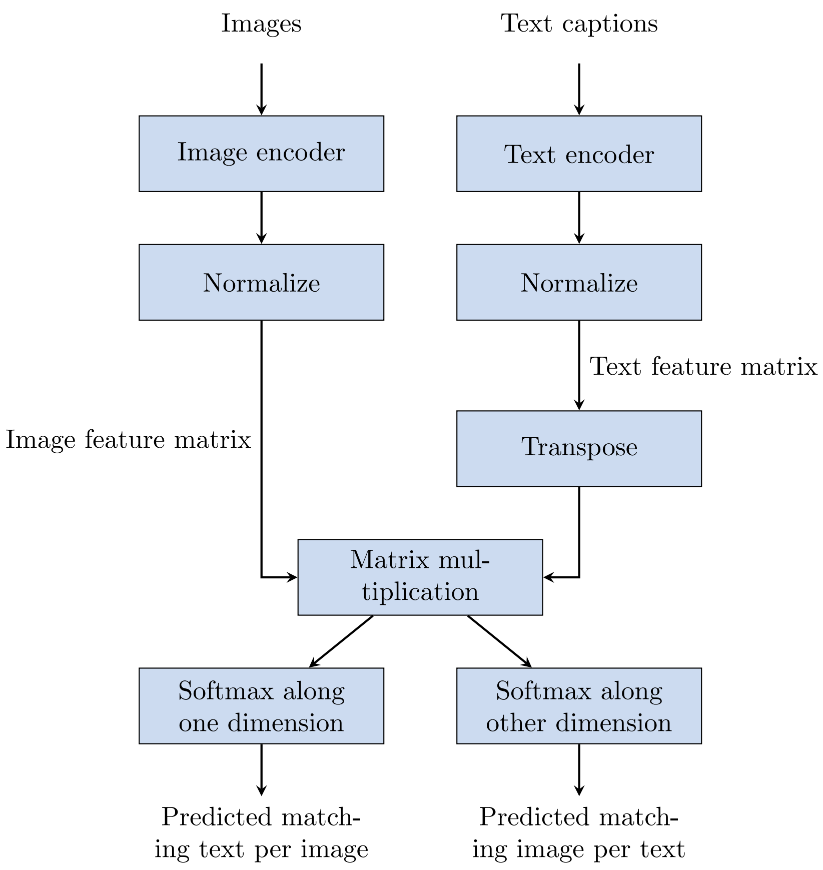

Figure 2.1: The architecture of CLIP

As shown in Figure 2.1, when given a set of images and captions, the encoders produce the respective representations vectors, normalize them by dividing by their vector lengths, and stack them in two matrices, one image representation matrix and one text representation matrix. These matrices are multiplied together with one of them transposed. This is equivalent to taking the dot product of each image-caption pair of representations. Since the representation vectors have length 1, this dot product is equal to the cosine similarity for each pair. The result is a matrix of similarities with each row representing an image and each column a text caption, or vice versa. Depending on the use case, the softmax can be taken of either each row or each column. The result is, for each row or column, a probability distribution over all captions (or images) estimating the probability that each of them belongs to the image (or caption) in question.

CLIP has been trained and released with two different image encoder architectures: a convolutional network and a vision transformer. Radford et al. [2021] found the vision transformer to be the best performing, and this is the one we will consider. This is backed by related work finding vision transformers to be suitable, as mentioned in subsection 2.2.1. The text encoder also uses a transformer architecture. The encoders produce representation vectors of size 512.

The similarities calculated by CLIP can be used as is for tasks such as searching for images semantically given a text prompt, searching for relevant text captions given an image, or helping a generative adversarial model make an image based on text. Alternatively, the raw representations calculated by CLIP's encoders could be used directly, for instance for training an image or text classifier or searching for images with similar content. Even when processing images or text only, ignoring the other modality, the model is still indirectly benefiting from the other due to the model being pre-trained on it. When using the representations, one should keep in mind how the model was trained and how that affects what information is included in the representations – in other words, what information would be relevant for matching it with a text caption. Since the model is pre-trained, it can be applied to other tasks zero-shot, that is without any additional training. Depending on the task and the available data, however, it may be beneficial to fine-tune the model for the task at hand. Radford et al. [2021] use the model for zero-shot image classification by crafting text captions such as "a picture of a [class name]" and using CLIP's similarity measure to identify which caption fits each image best. This is described in more detail in 2.2.2.

2.1.3 k-nearest neighbors

k-nearest neighbors (k-NN) is a machine learning model for classification or regression [Russel and Norvig, 2010]. Instead of training, when asked to predict the value of a new sample, the model considers the known samples directly. It finds the k known samples closest to the new sample (its k nearest neighbors) and uses their values to decide on a prediction. This can for instance be the mean or the most common value, depending on whether it is solving a regression or classification problem. The idea is that similar samples will also have similar values. The hyperparameter k can be set according to the availability of known samples. The higher this parameter is, the more protected the model is against overfitting because it bases every decision on more samples and is less likely to be dominated by random noise. However, it risks underfitting and thus not capturing the true pattern of the data if the available samples are few and far between. On the other hand, a lower value of k puts more weight on each available sample. As a special case, if $k = 1$, only the single nearest neighbor is used, and its value is returned unmodified.

2.1.4 Cache model

A cache model is a classification or regression model. It is similar to a k-NN model in that it keeps a set of known samples and compares new samples to them. Unlike k-NN, however, it compares a new sample to *all* known samples, weighting their labels by their similarity. In the form of the cache model we refer to, the similarity between two samples is calculated as the dot product of the samples' vector representations. The model then applies the softmax operation to these similarities, after multiplying by a temperature parameter to adjust the sharpness of the weight distribution.

The name "cache model" goes back to Kuhn [1988]; Kuhn and De Mori [1990], who – inspired by computer memory caching – introduced a speech recognition model that took recently used words into account to avoid ambiguities. Inspired by this, a continuous cache model has been shown to work well with neural networks for language [Grave et al., 2016]. It stores hidden activations from the network and uses them to search for similar instances using the dot product operation. This approach is reminiscent of the attention mechanism, using the activations for a new instance as a query and the activations for the previously seen instances as keys and using the dot products of the query and keys as a measure of relevance. However, unlike attention, the cache model does not require training. The continuous cache model has been adapted to image recognition as an add-on to pre-trained networks [Orhan, 2018]. Here, the representations learned by the network are used as keys and queries. This allows making an image classifier without training using the pre-trained network and labeled images as reference.

2.1.5 k-means clustering

k-means clustering is an unsupervised machine learning algorithm that groups vectors into k groups ("clusters"), such that similar vectors are assigned to the same cluster [Wu et al., 2008]. More specifically, it attempts to minimize the distance from each vector to its cluster's mean vector ("centroid"). Various distance measures can be used, but euclidean distance is common. The process is initialized by creating k centroids according to some rule, such as sampling random vectors from the input. The algorithm then iterates between two steps: First, allocate each vector to its closest centroid, forming clusters. Second, recalculate each cluster's centroid, moving it to the mean of the vectors belonging to the cluster. This loop is terminated when the process has converged, that is when each iteration no longer makes any changes to the clusters. When using euclidean distance, the process is guaranteed to converge to a minimum, albeit not necessarily to the global one. To improve the chances of finding a good minimum, the algorithm can be repeated with another choice of initial centroids.

The choice of the parameter k affects the possible solutions. In general, a higher k allows a lower distance from the vectors to their closest centroids, at the cost of higher model complexity. In the extreme case, when k is equal to the number of input vectors, there can be a centroid at each input vector, resulting in a "perfect" solution with zero distance from input vector to centroid. The best choice of k depends on the data and the desired properties of the clustering. In our case, we mostly have a pre-determined k, so we do not go into this.

Mini-batch k-means clustering [Sculley, 2010] is a stochastic version of k-means clustering. In each iteration, it considers only a randomly sampled mini-batch of the input vectors instead of all of them. Instead of setting the centroid to the cluster's mean in each iteration, it updates it using one input vector at a time, decreasing the learning rate gradually as more input vectors have been considered. This algorithm reduces the running time by orders of magnitude compared to the original k-means clustering, at the cost of a somewhat worse solution after convergence.

2.1.6 Performance metrics

For comparing different approaches to solving our task of multi-label classification, we measure their performance with various metrics. The purpose of the metrics is to take the raw results from a classification model – consisting of a confidence score given to each class for each image – compare them to the true labels, and condense them to a single number reflecting how "good" the results are. There is no single metric that perfectly captures all relevant aspects of a result, and for each metric there is a tradeoff between advantages and drawbacks. We therefore use several metrics to measure the performance of various approaches on our task, but we focus more on some than others. Some metrics take into account the confidence given by models to each class, but others require a pure binary prediction. In the latter case, we set a confidence threshold specific to each model such that confidences above or equal to the threshold are considered a positive prediction, and confidences below the threshold are considered a negative prediction.

Some of the metrics are designed for binary classification. Multi-label classification is equivalent to several binary classifications being performed at once, one for each class. The binary classification metrics can be extended to the multi-label case in several ways: Assume that there are M samples and C classes. The metric can be calculated for each class independently, resulting in C different values. The average of these class-specific results can be used as the final metric. This is called the *macro average*. This average could be weighted by the number of actual samples from each class, known as the class' support. This is called the *weighted average*. Alternatively, the classes can be ignored, and all $M \times C$ predictions are handled together as a single binary classification result. For the accuracy and mean squared error metrics, the macro and micro averages are equal due to their linearity.

In the definitions below, we define P and N as the sets of true positive and true negative samples, respectively, and P_P and N_P as the sets of predicted positive and predicted negative samples, respectively.

Accuracy and subset accuracy

Accuracy is a widely used metric for classification problems. It is defined as the proportion of the predictions that are correct. It is in the range $[0, 1]$, where 0 is worst and 1 is best. This can be written as:

$$\text{accuracy} = \frac{|P \cap P_P| + |N \cap N_P|}{|P| + |N|}$$

This metric is intuitive and easy to understand. However, it can be misleading when the classes are imbalanced. For instance, if 99 % of the samples are negative, a naive classifier that always gives a negative prediction will give the correct prediction in 99 % of the cases, a seemingly good score that does not capture the classifier's failure to recognize any positive samples.

For multi-label classification, there is also the subset accuracy metric. This is the proportion of samples for which all labels are predicted correctly. This is a quite strict metric compared to for instance micro averaged accuracy, especially when the number of classes is large. For instance, if there are 159 classes (like in our dataset after data preparation) and the label for each class is predicted correctly with a probability of 99 % independently of each other, the probability of getting all classes right is approximately 20 % (0.99^{159}).

Mean squared error

Mean squared error (MSE) is a common measure of the difference between two vectors or sets of numbers. It is defined like this:

$$\text{MSE} = \overline{(y_i - \hat{y}_i)^2} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n is the number of samples, y_i is 1 or 0 if sample i is positive or negative, respectively, and $\hat{y}_i$ is the model's confidence of sample i being positive. In our case, where y_i and $\hat{y}_i$ are in $[0, 1]$ for all i, the MSE will be in the same range, where 0 is best and 1 is worst. Minimizing MSE is equivalent to minimizing the euclidean distance between the vectors y and $\hat{y}$. This is the only performance metric we use where lower values are better.

Recall and precision

In order to avoid the problem of accuracy being a misleading metric for unbalanced datasets, two more specific aspects of the classifier's behavior can be measured instead: The ability to correctly identify *positive* samples, and the ability to correctly identify *negative* samples. By giving the classifier a lower threshold for giving a positive result, the former will get worse and the latter will get better, and vice versa. Balancing these two thus becomes a design question when building the classifier, and will depend on the use case and the importance of the two abilities.

Recall, also known as sensitivity, is a measure of the classifier's ability to correctly identify positive samples. It is the proportion of the true positive samples that are correctly identified as positive:

$$\text{recall} = \frac{|P \cap P_P|}{|P|}$$

Precision is a measure of the classifier's ability to correctly identify negative samples. It is the proportion of the samples predicted to be positive that really are positive:

$$\text{precision} = \frac{|P \cap P_P|}{|P_P|}$$

It can be interpreted as the probability that a sample predicted to be positive really is positive: $\mathbb{P}(s \in P \mid s \in P_P)$, where s is a sample. A weakness of this metric is that it depends on the prevalence $\mathbb{P}(s \in P)$. Intuitively, "guessing" that a sample is positive, is more likely to be correct if a large proportion of the samples are in fact positive. This means that precision measures not only the quality of the classifier but the quality of the classifier's result on a specific data distribution. This can be both a strength and a weakness. It provides a useful statistic in practice for a given classifier and dataset, directly addressing the question: "The classifier said this sample is positive. How much can I trust that result?" However, one should be wary of using it to compare classifiers if they have been given data with different distributions.

F_β score

To compare classifiers quantitatively, one may want a single measure that sums up the two abilities measured by recall and precision. The F_1 score, or balanced F measure [Schütze et al., 2008, p. 156], is an attempt at this. It is the harmonic mean of recall and precision. If both precision and recall have a value of 1, the F_1 score is 1 as well. If one or both of them are 0, the F_1 score is also 0.

The F_1 score gives equal weight to recall and precision. This might not reflect their true relative importance in every use case. The F_β score, or F measure, generalizes the metric by assigning any weight β to recall relative to precision, with F_1 being the special case with equal weight. This generalized weighted harmonic mean can be written like this, where p is precision and r is recall:

$$F_\beta = (1 + \beta^2)\frac{pr}{\beta^2 p + r}$$

For instance, the F_2 score, which we will use in this work, gives recall a weight two times as high as precision. This can be suitable in a use case where the model's predictions are to be shown to a human who can identify false positives. In such a case the model should be incentivized to show the edge cases to the human rather than hide them. In a case such as ours where the labels are noisy, giving recall more weight than precision has the advantage of being less punishing towards false positives [Conde and Turgutlu, 2021].

Since the F_β score is derived from precision, it inherits precision's property of depending on $\mathbb{P}(s \in P)$ as discussed in the section about recall and precision. This means it should not be used to compare methods if they were evaluated on data with different distributions.

Specificity

Specificity is an alternative to precision for measuring a classifier's ability to correctly label negative samples. Unlike precision, it is independent of the prevalence. Instead of asking how much we can trust that a sample is positive if it was classified as such ($\mathbb{P}(s \in P \mid s \in P_P)$), it turns the question around, asking how much we can trust the classifier to classify a sample as negative if it is truly negative ($\mathbb{P}(s \notin P_P \mid s \notin P) = \mathbb{P}(s \in N_P \mid s \in N)$). It can be calculated as:

$$\text{specificity} = \frac{|N \cap N_P|}{|N|}$$

By being independent of the prevalence, specificity is useful for measuring the quality of classifiers even if the prevalence is not the same between measurements. However, it does not directly answer how much a positive prediction can be trusted for a given dataset, as the precision does.

Area under ROC curve (ROC AUC)

As mentioned, one can adjust a classifier's threshold for giving a positive result to balance recall against precision or specificity. Sometimes one wants to study how these measures respond to different thresholds. This can be done by studying an ROC curve, which is a plot of the true positive rate against the false positive rate, in other words, sensitivity/recall against (1 - specificity), as the threshold varies [Schütze et al., 2008, p. 162]. An example of an ROC curve is shown in Figure 2.2. The name ROC (receiver operating characteristics) stems from its use with radar during World War II [Streiner and Cairney, 2007]. Studying this curve can help in choosing the right threshold. It also gives an indication of the quality of the classifier. In general, one wants points on this curve to be close to the top left corner, corresponding to a high true positive rate and a low false positive rate. In contrast, a useless, randomly guessing classifier would have an ROC curve along the diagonal from the bottom left to the top right corner. This means that the degree to which the curve is close to the top left corner is a measure of the quality of the model across all thresholds. This is commonly measured as the area under the ROC curve, or ROC AUC (with AUC standing for "area under curve"). This number is 1 for a perfect classifier, (approximately) 0.5 for a randomly guessing classifier, and 0 for a perfectly wrong classifier.

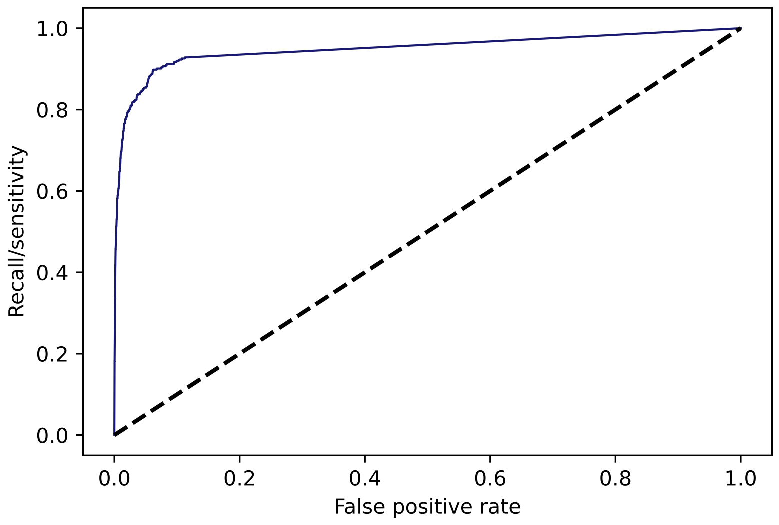

Figure 2.2: An example ROC curve. The diagonal which is characteristic of a randomly guessing classifier is shown as a dashed line.

2.1.7 Hyperparameter optimization

In addition to the normal parameters of a machine learning model, whose values are decided through some kind of training, models typically have a set of *hyperparameters*. These constitute design decisions that must be made about the model before it can be trained. Examples of hyperparameters are the number or size of layers in an artificial neural network, the number of trees in a random forest, or k in k-nearest neighbors. Finding good hyperparameters for a given problem is often non-trivial, and hyperparameter optimization is a field of research in its own right. We will describe a few approaches for hyperparameter search: random search, grid search, and Bayesian optimization.

Random search and grid search

Random search is a simple method that samples a value randomly for each hyperparameter between upper and lower bounds specified by the user. The resulting combination of values is used to train and evaluate the model according to a specified metric. This is repeated for any desired number of iterations, and the hyperparameter combination with the best performance is returned.

With grid search, the user specifies a set of values to evaluate for each hyperparameter, typically evenly spaced between some upper and lower bounds. Every possible combination of these values is then tested, and the best-performing one is returned. These combinations can be visualized as lying on a grid in the multi-dimensional search space, hence the name "grid search". In contrast to random search, the number of iterations in grid search is implicitly defined as the number of combinations of hyperparameter values. This number is $\Pi_{i=1}^{n} k_i$, where n is the number of hyperparameters and k_i is the number of values to test for each parameter. If all k_i are the same value k, this simplifies to k^n, growing exponentially with the number of parameters. This means that the number of iterations and thereby the running time of grid search quickly grows out of hand unless the number of parameters and values to test are kept quite small. However, grid search is deterministic and can, given enough time, cover the search space well compared to random search, which may by chance miss good parameter combinations.

As a compromise between the efficiency of random search and the thoroughness of grid search, it is common to combine them. A relatively short random search is performed, and the best combination found by random search is used to inform the search space of a grid search. It is now assumed that the actual optimal solution is nearby, in effect narrowing down the possibly vast search space significantly in only a constant number of iterations. A grid search is now performed in this new neighborhood, possibly improving the solution found by random search. It is also possible to run two grid searches, one with a relatively coarse grid and another one with a smaller grid to allow for an increasingly focused search.

Bayesian optimization

Another compromise between randomness and thoroughness is Bayesian optimization. This method estimates the performance of a model as a function of the hyperparameters, as well as the uncertainty, through a Gaussian process. Using this estimate, it suggests new points in the search space to evaluate and uses them to update the estimate iteratively. Like random search, it runs for a specified number of iterations. The points are chosen because it seems to be a promising point (exploiting the current information) or because it is likely to improve the existing estimate (exploring the search space). The estimate can be initialized by evaluating one or more points at random, like random search. Through this process, Bayesian optimization starts out as a random search, but it gradually becomes more focused, prioritizing promising areas in the search space. An example of this process can be seen in Figure 2.3.

2.2 Related work

2.2.1 Image classification

While we are specifically interested in applying general pre-trained image models for our task, other methods may inform the development of our models. We describe some of the best performing concurrent image classification models applied to multi-label, fine-grained, or few-shot classification or image representation learning, highlighting their suitability for our task and relationship to language-image pre-training models.

Learning task-agnostic image representations

Chen et al. [2020a] introduce the framework SimCLR, which learns image representations from self-supervised contrastive pre-training. Instead of comparing images to text as CLIP does, it creates training data by modifying unlabeled images and learns to identify image pairs that stem from the same image. They evaluate the learned image representations by attaching a linear classifier and training it on ImageNet. When the model was released, its results were state of the art for self-supervised image representation learning. Chen et al. [2020b] go on to improve SimCLR further, resulting in the framework SimCLRv2. This version performs semi-supervised learning in several steps: First, it pre-trains a larger network than SimCLR on unlabeled images with contrastive training. Second, it fine-tunes the network on labeled images. Finally and optionally, it “distills” the network by training a new, much smaller network to mimic the behavior of the large one on labeled as well as unlabeled images. The first step is task agnostic, and the others are task specific. The authors reason that learning task-agnostic features requires

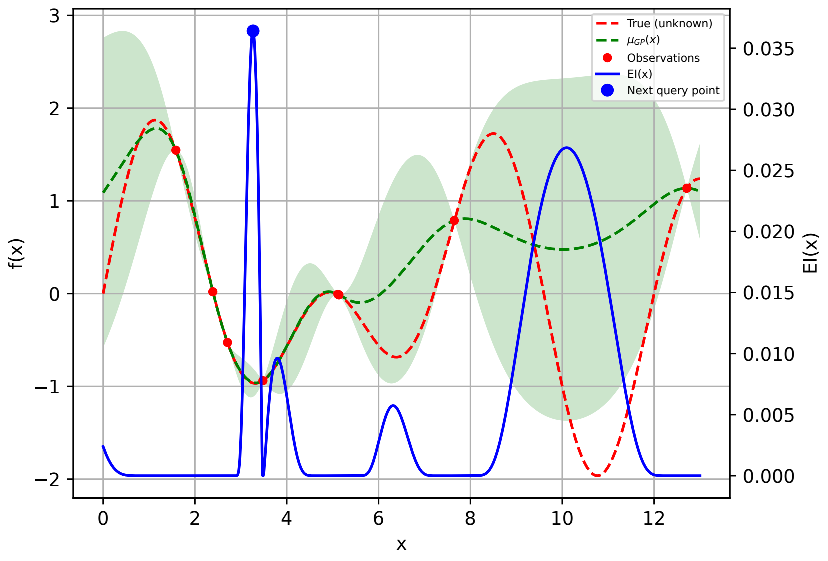

Figure 2.3: Illustration of a one-dimensional Bayesian optimization process. The red dashed line shows $f(x) = \sin(x) + \sin(1.618x)$, which is the function to minimize, and the red points are the values observed so far. The green dashed line and surrounding area show the current estimate of $f(x)$ and its uncertainty. The blue line shows the expected improvement ($\mathrm{EI}(x)$) that would result from evaluating f at the given value of x, with the blue dot marking the most promising value to evaluate next. At the moment, the process has found a sub-optimal local minimum around $x = 3$, but is aware that there may be a better minimum around $x = 10$.

a model with greater capacity than when the task is known, and that this is the reason why the increased model size during pre-training improves the performance.

The SimCLR models are no longer state of the art. Notably, with the CLIP model, Radford et al. [2021] demonstrate that task-agnostic language-image pre-training learns more effective image representations for linear probe classification on ImageNet than other publicly released models at the time like SimCLRv2 and BiT-M [Kolesnikov et al., 2020], shortly before the start of our work. Our task differs from ImageNet classification, limiting the usefulness of this result, but while SimCLRv2 and BiT have the advantage of being trained on classification with dataset specifically, CLIP is trained on a separate, task-agnostic dataset. CLIP's good results on ImageNet despite this are likely to be more representative of its representations' effectiveness when transferred to new tasks. We therefore believe that CLIP's representations are the most suitable for our work.

Multi-label image classification

Query2Label, introduced by Liu et al. [2021], is the state-of-the-art approach for multi-label image classification for the datasets PASCAL VOC 2007 and 2012 [Everingham et al., a,b] and NUS-WIDE [Chua et al., 2009] and on par with the state of the art for MS-COCO [Lin et al., 2014]. The architecture can be divided into two stages. The first stage uses an image classification backbone to extract spatial image features. The second uses transformer decoders to search for the classes in these features. The model learns to predict each class's presence in an image by learning label embeddings for each class. The label embeddings are given as queries to a transformer decoder, and the values and keys are derived from the image, in effect performing cross attention looking for relevant parts of the image given the label. The authors argue that this approach helps the model attend to different parts of the image for different classes, which is useful when the image contains several objects of interest in different regions. In our dataset, however, different classes do not typically correspond to multiple objects spread around in the same image, but rather different attributes of the same objects, so this may not be as useful for our task.

Fine-grained image classification

He et al. [2021] introduce the architecture TransFG, which is close to state of the art for the fine-grained image classification datasets Caltech-UCSD Birds-200-2011 [Wah et al., 2011] and Stanford dogs [Khosla et al., 2011].[1] These datasets consist of images of variants of birds and dogs, respectively. This means they are more uniform than our dataset. They argue that vision transformers are a better choice of model for this type of task than convolutional neural networks due to their innate ability to identify important regions, and thus take into account global and local discriminative features at once. They do indeed find that standard vision transformers (see section 2.1.2) are well suited for fine-grained classification. In addition, they make several adjustments further improving performance, resulting in the TransFG architecture. These adjustments include a "Part selection module" to help the model find and focus on the discriminative regions and a contrastive loss to force the representations of similar images from different classes further apart.

The CLIP model is released in several versions, one of the differences being whether the image encoder is a visual transformer or a convolutional net. As our dataset is characterized by several fine-grained classes, the work on TransFG gives reason to think that the visual transformer is the

[1]It was state of the art when work on this thesis began, but recently it has been surpassed in accuracy by about 1 percentage point on both CUB-200-2011 and Stanford dogs.

most promising choice of image encoder for our task. This is also somewhat supported by the work on Query2Label mentioned above, which also highlights the effectiveness of a transformer, albeit used differently, on multi-label classification.

2.2.2 Applying language-image pre-training

It is not obvious how to best use a pre-trained language-image model to create an image classifier. Several works have been published that use the language-image model CLIP to perform image classification with or without using its text encoder. This includes the original work presenting CLIP [Radford et al., 2021], where the authors describe a classifier architecture using both the text and image encoders of CLIP. We consider some of these works to inform our approach.

Language-image pre-training is an active area of research. Since CLIP's original release and since we began work on this thesis, several other models similar to it have been released, and the state of the art in pre-trained models has improved. These models include ALIGN [Jia et al., 2021], Florence [Yuan et al., 2021], LiT [Zhai et al., 2021], BASIC [Pham et al., 2021], FLAVA [Singh et al., 2021], BLIP [Li et al., 2022a], and CoCa [Yu et al., 2022]. We use CLIP as our language-image model in this thesis, but our work could be repeated replacing CLIP with one of these models, possibly improving results.

Language-image classifier

Radford et al. [2021] invented the original CLIP model. The way the model is implemented (see 2.1.2 for details), it can easily be repurposed from the contrastive pre-training to language-image classification. The authors do this by giving the names of the classes to CLIP as text captions and presenting it with an image. The model then gives each class name a score indicating how well it matches the image. Finally, they reinterpret these scores as confidences in the image belonging to each class. The result is a classifier that performs a task it was not trained for. In other words, it performs the task zero-shot. The authors test this model on various datasets and find that in many cases, it performs very well. Notably, it achieves an accuracy on ImageNet as high as that of ResNet-50 [He et al., 2016], a commonly used image model, specifically trained on ImageNet.

They note some interesting patterns in which datasets CLIP does and does not work well with, compared to linear probing on ResNet50 pre-trained on ImageNet. The performance on the datasets of general images is decent. For fine-grained classification datasets, however, it varies a lot, doing best on car model recognition and worst on flower species recognition. This leaves great uncertainty as to how well zero-shot CLIP classification can be expected to perform on our fine-grained dataset. The model performs poorly on complex tasks including counting and judging distance, but this is not likely to affect our use case.

The authors find that using class names as text captions for this kind of classifier can be problematic when they were not originally written with this use in mind. They note that the class names in image datasets are often written in such a way that they may confuse CLIP. For instance, ambiguous label names are common, and they exist in our dataset as well.

It is also a problem that class labels are typically single words that do not resemble most captions in the dataset CLIP was pre-trained on, which are more likely to be sentences than individual words. The authors find that it helps to add additional text – a context – around the class name, for instance "A photo of a [class].", and use that as the text prompt. Contexts that are tailor-made for the dataset, such as "A photo of a [class name], a type of pet." for a dataset of pet pictures, help more than a context designed to be used with all datasets. Improving model performance by looking for good prompts in this way is referred to as *prompt engineering*.

Ensembling over many prompts per class gives an even greater improvement to the model. They do this by creating several different prompts and ensembling over the representations of these prompts before comparing them to the images.

Prompt engineering is likely to be relevant for our dataset as well, as the keywords, even if translated from Norwegian to English, are often short and provide very little context on their own. It is also reasonable to suspect that performing prompt engineering on each class individually will enable the model to perform even better, even though this is not mentioned by the authors. This might however be labor-intensive to do for our dataset due to the large number of classes, some of which require expert knowledge.

Prompt engineering can be a tedious process, as the exact wording in the prompt can have large and counter-intuitive effects on the classifier. Zhou et al. [2021] remove the need for manual prompt engineering by optimizing the context automatically, a procedure they call CoOp. They do not actually use a text string as context. Instead, they optimize a list of continuous tokens to use in place of the language tokens that are usually read by CLIP's text encoder. This is combined with the class names to create the final prompts. By training the classifier while keeping CLIP's weights frozen, the context is optimized for the task at hand. This means that their approach requires training and uses training samples, and the classifier is no longer zero-shot.[2]

Gao et al. [2021] propose a different approach, fine-tuning a two-layer addition to CLIP's text and image encoders with task-specific data. They call their method CLIP-adapter and find it to perform better than CoOp. Zhang et al. [2021] continue this work by finding a way to set the weights of the adapter without training, calling this new approach Tip-adapter. They set the weights such that the adapter implements a query-key cache model making use of a set of labeled reference images. This exhibits similar performance to CLIP-adapter despite being trained, and it improves further with only minimal fine-tuning.

Image-based classifier

Conde and Turgutlu [2021] use CLIP to perform classification on images of art. Their art classification task has several features in common with our task, making their work relevant: It is multi-label and fine-grained image classification, the images are annotated with keyword-like text, and the domain is quite specific. The art images differ from the images in our dataset (which is described in subsection 3.1.1) in that they are cleaner and mostly depict single objects with plain backgrounds and little irrelevant information. This could make it easier for an image encoder to focus on the relevant task, especially for CLIP, which is not trained for the specific task. In our case, this could mean that the image representations waste space encoding irrelevant information like weather and orientation of objects instead of the interesting properties of the objects.

They evaluate CLIP's image encoder zero-shot by getting its representation of images in a reference set and a test set. They then assign labels to each test set image by finding its nearest neighbor in the reference set and copying the labels of that reference image. This is a 1-NN model as described in subsection 2.1.3. They achieve an F_2 score of 51.61 %, which is similar to the score of state-of-the-art convolutional neural nets trained on 10 % of the dataset. This model has the advantages of being simple and not requiring any training. However, using only the nearest neighbor seems restrictive. It seems reasonable that performance might benefit from changing the model to a k-nearest neighbors model or in another way allowing it to take more reference images than one into account.

The authors find two ways of improving the model through fine-tuning. First, they fine-tune CLIP itself using contrastive training in the same way it was originally pre-trained, but with

[2]Arguably, manual prompt engineering also requires training samples.

task-specific data. This improves the F_2 score of the 1-NN model by 3.46 percentage points. Second, they use the image encoder from this fine-tuned version of CLIP as the basis for a classifier network, which they train fully supervised. This network improves the F_2 score by an additional 4.9 points. With both of these improvements, the performance is close to a ResNet [He et al., 2016] trained fully supervised. These results suggest that a form of fine-tuning may help with our dataset as well.

Chapter 3

Method

In this chapter, we present FFI's image dataset, which we use to evaluate models, and provide details on its composition and properties. We then introduce the pipeline we use to prepare this data for use with the models. Finally, we present the architectures of our suggested models and the reasoning behind them. The software we have used and corresponding version numbers are listed in Appendix A.

3.1 Dataset

3.1.1 Dataset description

Our dataset is a collection of approximately 290 000 military-related images collected from various sources and for various purposes over time. The dataset is provided by FFI.[1] Most of the images are photographs of types of vehicles or equipment, but the dataset is not limited to this, and there are also images of other objects such as buildings or soldiers as well as drawings. Most images have a single object clearly in the foreground, but some contain several objects, some of which may be in the background and/or partially occluded. Figure 3.1 shows a photo typical of the dataset.

Approximately 130 000 of the images are labeled with one or more keywords describing the content of the images, and these are the ones we use in this work, as explained in subsection 3.1.2. For example, the photo in Figure 3.1[2] could be labeled with the keywords "tankvogn" (fuel truck), "lastevogn" (truck), "fireakslet" (four axle), and "Kamaz" (a truck manufacturer).[3] Some keywords do not refer to physical objects, but other concepts, such as "logo". For our work, we treat the keywords as labels and attempt to predict them given an image. Since each image can have any number of labels, this is a multi-label classification problem. Most of the images are also labeled with information other than the keywords, such as location or the relevant military branch, but we do not consider this.

Approximately 160 000 of the images have no keywords. It can either be assumed that in fact, no keywords were relevant to these images, or that they are just not labeled and cannot be used for supervised machine learning without manual labeling effort by domain experts. Certain

[1]We do not have the license to share or show these images, so any images shown here are drawn from sources where the license allows sharing, and not from the dataset.

[2]The example photo is by Yuriy Lapitskiy and is licensed under CC BY-SA 2.0. It is fetched from `https://www.flickr.com/photos/74292825@N00/1295897644`.

[3]Where useful, we use English translations of the keywords in this thesis for readability.

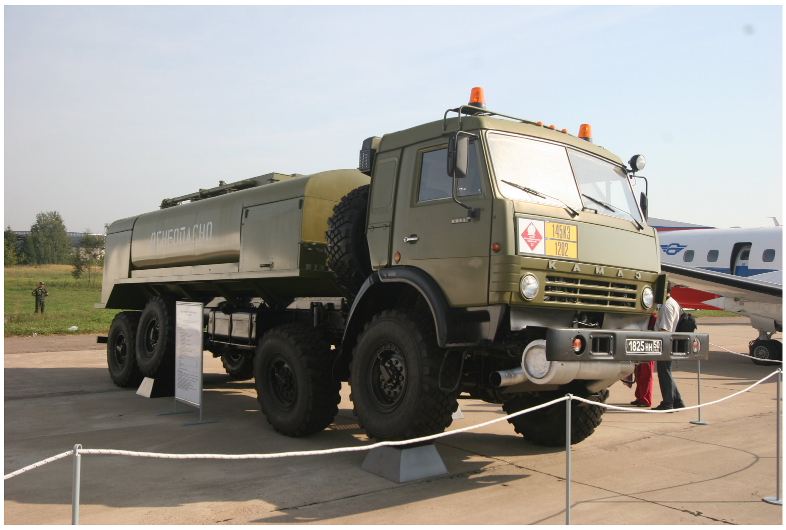

Figure 3.1: Example image of a four axle tank truck. This image is not taken from the dataset, but depicts the same vehicle as an image in the dataset and is representative of it.

types of images seem to be more likely to have keywords than others. For instance, images of trucks are likely to have them, whereas images of airplanes are not.

Certain keywords denote highly domain-specific objects that would require specialized knowledge to recognize. Examples of this are the keywords "Troposcatter" or "Reconnaissance vehicle". The separation between keywords is also fine-grained, meaning that the classes can be very similar to each other and hard to tell apart. An example of this is the set of keywords "two -", "three -", "four -", "seven -" and "eight axle vehicle". To tell the difference between these, a model would have to know specifically what part of the vehicle to consider, as the vehicles may otherwise be very similar.

Some of the images in the dataset are duplicates of each other. Some are completely identical, others are cropped slightly differently or have other insignificant differences. This is likely because the same image was gathered from different places without the person gathering noticing the situation.

The images are inconsistently labeled, owing to the way the dataset was built. Some keywords are present in only some of the relevant images. A good example of this is the keyword "truck", which is only present in about 3 000 images. Compare this to the two most common keywords, "three axle" and "Kamaz", which have approximately 16 000 and 12 000 occurrences respectively, even though a large proportion of these occurrences are in reality images of trucks. For example, an image in the dataset of the exact same vehicle as in Figure 3.1 is labeled with "fuel truck", "Kamaz", and "four axle", but not "truck" even though it would be relevant to this image. This issue is at its most apparent in duplicates, where some identical images are labeled differently. Instead of interpreting the presence or absence of a keyword as a "yes" or "no", it may be better to interpret them as "yes" or "maybe". Although we do still interpret absence as "no", we compensate for this by optimizing a metric (the F2 score) that puts more emphasis on sensitivity than precision, as explained in section 2.1.6.

There are 570 distinct keywords in the dataset, but they are not evenly distributed, with a few being very common and many being quite rare. Each keyword occurs between 1 and 15964 times, on average 426. The 25., 50., and 75. percentiles are 1, 4, and 139, respectively.

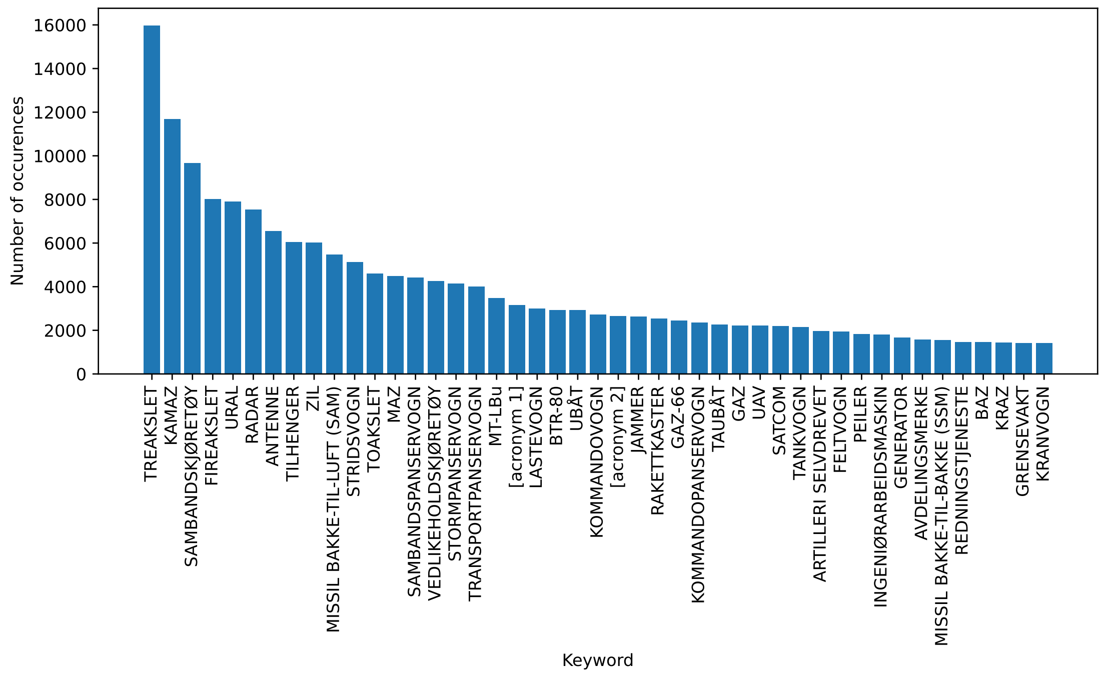

Figure 3.2: The 45 most common keywords and their number of occurrences.

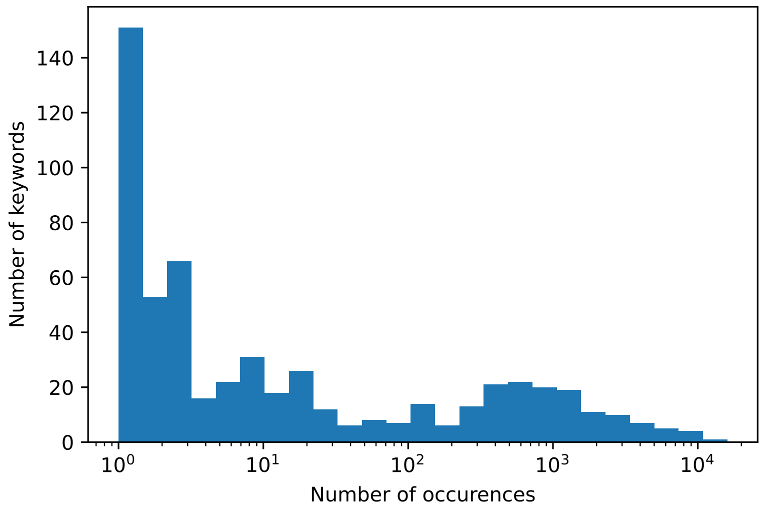

Figure 3.3: Distribution of the number of occurrences per keyword.

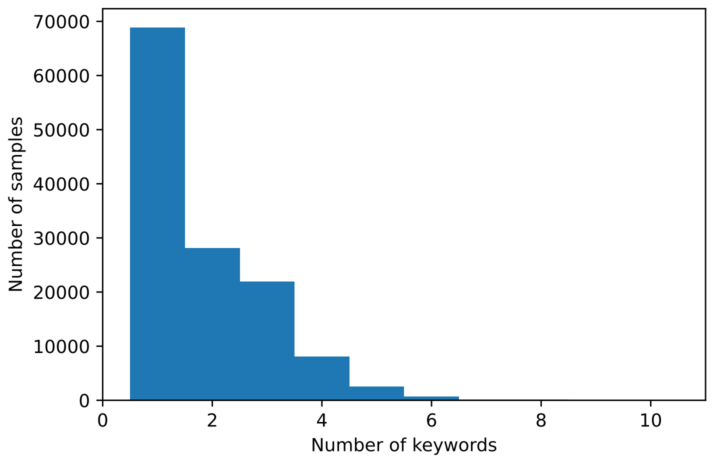

Figure 3.4: The distribution of the number of keywords per image, limited to the range $[1, 10]$.

151 keywords occur exactly once. Figure 3.2 shows the number of occurrences of the 45 most common keywords. The distribution of the number of occurrences per keyword can be seen in figure 3.3.

Of the approximately 131,000 images that have associated keywords, each one has between 1 and 50 keywords, on average 1.86. The 25., 50., and 75. percentiles are 1, 1, and 3, respectively. Figure 3.4 shows the distribution of the number of keywords per image, excluding those without any keywords. Very few images have more than a handful of keywords.

3.1.2 Data preparation

Before training, optimizing, and evaluating the models, we prepare the dataset. The data preparation is based on Bynke [2021]. Figure 3.5 shows the data preparation pipeline. The first step is removing any images that have no keywords. This follows the assumption described in subsection 3.1.1 that these images have not been considered for labeling and thus should not be used for training or evaluation.

We then give the images as input to CLIP's image encoder, obtaining a representation vector for each image. All further processing will make use of this vector instead of the raw image data. We normalize the vectors by dividing them by their norms. This has the effect that taking the dot product between any such vectors later will be the same as finding the cosine similarity between them, as demonstrated in Equation 2.1. In Figure 3.5 the normalization is not shown but assumed to be part of the encoding step.

Using the representation vectors we calculate the cosine similarities between all images. These similarities are then used to remove duplicates. If two images have a similarity above a certain threshold, they are considered suspected duplicates, and one of them is removed. If more than two images are duplicates of each other, all except one are removed. If the duplicates have a similarity above an even higher threshold, they are considered certain duplicates, and any keywords in the removed images are copied to their duplicate. The purpose of this is to not waste label information when the labels of the duplicate images differ. To decide on the values of the two thresholds we manually inspect randomly sampled images along with the image most

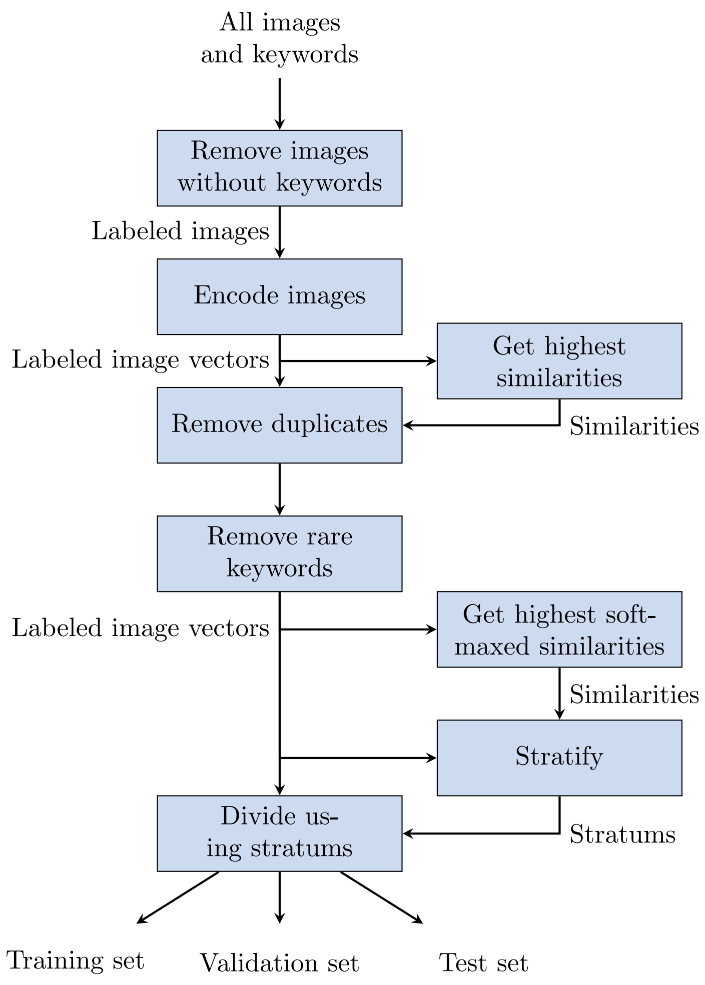

Figure 3.5: The data preparation pipeline. Each box represents a process in the pipeline. The figure is adapted from Bynke [2021].

similar to them and label them as duplicates or not duplicates. We then use logistic regression to model the probability of the two images being duplicates as a function of the cosine similarity between them. We calculate the two similarity values that, when given as input to the regression model, return probabilities of 1 % and 99 % respectively. These two are the values we set as our two thresholds. We continue to sample and label image pairs until these thresholds converge.

As shown in Figure 3.3, most of the keywords occur very few times. We choose to exclude keywords occurring less than 50 times in the training set. This is because we suspect that some of these rare keywords disturb the results. Some seem to be sporadically applied only to one or a few images even though they are in fact relevant for many others. Assuming that a model correctly suggests such a label for those other images, these suggestions would be counted as false positives and impact the result unfairly. Some of the rare keywords come from image keywords that do not fit in with the rest or are completely irrelevant, for instance dates, corrupt strings, or strings of only question marks. These stem from the non-homogeneous sources and use cases of the images gathered for this dataset. After removing rare keywords, 159 distinct keywords remain.

For building and evaluating models we split the data into a training set, a validation set, and a test set. We want each of these sets to preserve the variation and the distribution of all properties of the full dataset as well as possible. Sampling images randomly for creating these sets runs the risk of producing sets that are not representative of the full dataset. This happens the most easily for properties that are represented by only a few samples, such as a very small class, as the chance of sampling a disproportionate amount of a group is higher when the group is small. The large number of samples helps reduce this problem, but to reduce it further, we use stratification when distributing the samples. Stratification means to divide the data into relatively homogeneous groups (stratums) before dividing each of these groups proportionally. This has the effect of keeping the distribution of a property of interest.

For single-label classification problems, stratification can be done by sampling the specified proportion of images from each class, thus keeping the distribution of the labels. This simple approach does not work with multi-label samples because each sample may belong to several classes and may thus be picked several times. We instead stratify using a surrogate measure: highest "softmaxed" similarity. For each image, the cosine similarity with every other image is calculated, and the softmax operation is applied to these similarities.[4] The highest of the resulting similarities is then used as a stratification value for that image. The idea is that it serves as a measure of the number of relevant examples available: many relevant examples give a low maximum "softmaxed" similarity and vice versa. Since this is a continuous variable, the samples are divided into ten equally sized groups using quantiles. These are the groups used in the stratification. We split the data into training, validation, and test sets according to the proportions 60 %, 20 %, and 20 %. When the data preparation is done, the training set has 75 992 samples while the validation and test sets have 25 331 samples each. This means that 126 654 images remain in the final sets, a reduction from 290 070 images in the original set.[5]

3.2 Model architectures

We evaluate three models: text similarity, image similarity search, and an image similarity network based on a cache model. They all take CLIP image representation vectors as input and output a confidence score for each class for each input image vector. Another image encoder

[4]Note that the cosine similarity between every pair of images is used twice in the data preparation pipeline – here and when finding duplicates. By reusing the results from earlier instead of calculating them again, this process can be sped up.

[5]Counting 2 images that were removed manually as they were corrupt, the original dataset had 290 072 images.

than CLIP's could be used instead. Figure 3.6 shows how each model is then used to create label suggestions for images. The class confidence scores are finally compared against a threshold. If an image's score for a class is at or above this threshold, the class is returned as a suggested label for the image. The threshold is decided by optimizing the model's performance on the validation set, as described in section 4.2. There is no limit for the number of classes that could be suggested as labels for the same image, making this a multi-label classifier.

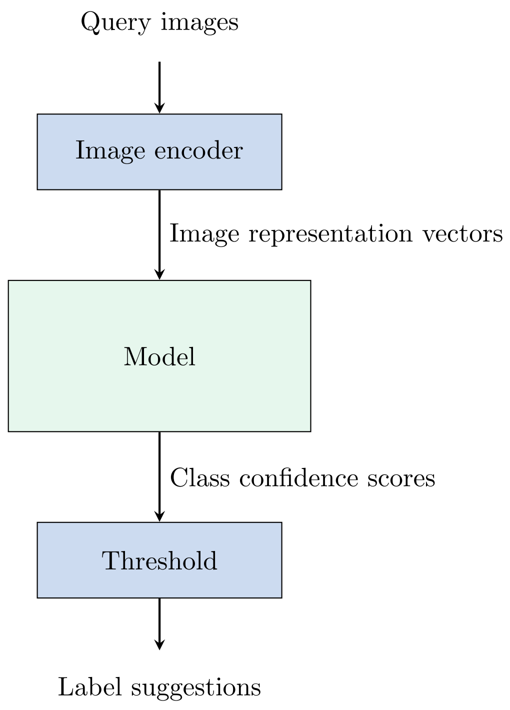

Figure 3.6: The architecture common to all models

3.2.1 Language-image model

This method is adapted from the original classification approach of Radford et al. [2021]. For the purposes of this thesis, we refer to it as the *language-image* model. Each class name is converted into an artificial caption by placing it in a *context* such as "a picture of a [class name]". Each caption is then fed through CLIP's text encoder, converting it to a text representation vector. The input image is fed through CLIP's image encoder, and the resulting image representation vector is compared to each caption's representation vector with cosine similarity. To adapt the method to the multi-label case, we do not perform a softmax operation on the cosine similarity scores of the image, but instead rescale them from the range $[-1, 1]$ to $[0, 1]$ and interpret them as confidence scores. In practice, the scores do not typically use the entire range, but tend to lie around 0.5–0.7. We could adjust for this and use the entire range in a more meaningful way, but this is not necessary as we are only interested in whether each score is above or below a certain threshold.

Like Radford et al. [2021] we allow the model to aggregate several contexts. We provided various contexts such as "a photo of a [class]", "a drawing of a ", and "a picture from the military of a [class]". An artificial caption is created for each class name and each context before all captions are converted to text representation vectors. For each class, the model takes a weighted average of all the corresponding text representation vectors. This weighted average is then used when the model compares images to captions. The context weights are tuned as hyperparameters.

Label translation

Since the labels are mostly in Norwegian, we translate them to learn how this affects the performance. We translate them both automatically and manually, resulting in three different sets of labels to evaluate the model with: untranslated, automatically translated, and manually translated. We use Google Translate to generate the automatic translations. The manual translation was done in collaboration with a domain expert at FFI. Table 3.1 shows examples of the translations. The automatic translations have obvious flaws, and it is easy to find examples of incorrect translations. Only 3 of the 10 labels in Table 3.1 are correctly labeled. For instance, "stridsvogn" (main battle tank) is translated to "trailer", "tankvogn" (tank truck) to "tank", and "sambandspanservogn" (armoured communication vehicle) to "community travel". Translations like these are likely to mislead and confuse the model. Nevertheless, on manual inspection, it seems reasonable that these labels might be more helpful than the original Norwegian labels for the text encoder of CLIP, which was trained on English captions.

Original label	Automatically translated	Manual translation
treakslet	treakslet	three axle vehicle
sambandskjøretøy	**communication vehicle**	communication vehicle
fireakslet	fireakslet	four axle vehicle
antenne	**antenna**	antenna
tilhenger	supporter	trailer
missil bakke-til-luft (SAM)	missil ground-to-air (SAM)	surface-to-air missile (SAM)
stridsvogn	trailer	main battle tank
toakslet	toakslet	two axle vehicle
sambandspanservogn	community travel	armoured communication vehicle
vedlikeholdskjøretøy	**maintenance vehicle**	maintenance vehicle

Table 3.1: The 10 most common labels, excluding those that would be the same in English as in Norwegian, and their automatically and manually generated translations. The correct automatically generated translations are highlighted in bold.

3.2.2 Neighbor search model

We have implemented a model using CLIP's image encoder to predict image labels based on already labeled reference images. We refer to it as the *neighbor search* model.

Figure 3.7 shows the model's architecture. It takes as input an image to analyze (the query image) as well as a set of reference images, and it gives suggested labels as output in the form of a set of classes with associated scores. The label scores reflect to what degree each label is present on reference images that are similar in content to the query image. This is used as a proxy for the probability of the query image belonging to each class.

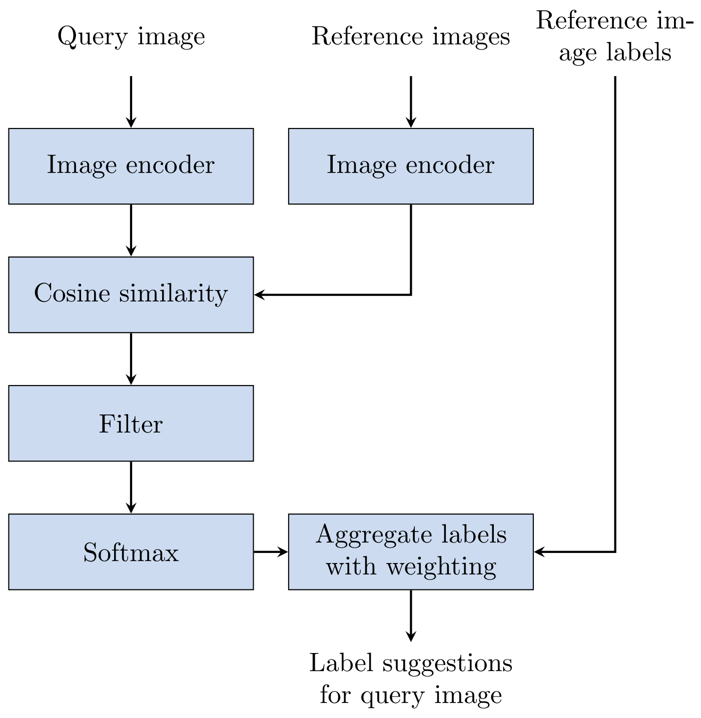

Figure 3.7: The architecture of the neighbor search model

The core of the model is an image similarity search utilizing image representations provided by an image encoder. We use the image encoder from CLIP, but this could in principle be any model converting images into representation vectors. We compute the representations for the query image as well as for all of the reference images. With many reference images, computing all the representations is quite costly, taking approximately 5 hours for our labeled dataset of 130 000 images on a GPU. When several searches are to be performed with the same reference images, we therefore cache the image representations and reuse them in subsequent searches, greatly reducing the running time. The computation could likely be optimized, but with caching it was unnecessary for our use.

We then calculate the cosine similarity between the query representation and each reference representation. Like CLIP, we multiply these similarities by a certain factor before using them. CLIP uses a factor of 100, resulting in similarity scores in the interval $[-100, 100]$.[6] For our model, we optimize this factor as a hyperparameter, which we call the *softmax temperature* due to its effect on the softmax operation that comes next.

The resulting scores are sorted and filtered, keeping only the best matches, defined as images having a similarity score above a certain threshold, restricted to at most a certain number of images. If the query image is itself also present in the reference set, this also needs to be removed in the filter, or else it will be a perfect match and completely dominate the search result. We set the similarity score threshold for the filter as the score of the best match found minus a *maximum similarity difference*, optimized as a hyperparameter. For instance, if this maximum difference is set to 7, images with a score lower than the threshold would have a score after softmax of at

[6] In practice, the number 100.125 is observed, probably due to float precision errors when calculating the cosine similarity.

most 0.0009, which would be negligible. Setting a low value reduces the necessary computation in the later steps. The *maximum number of neighbors* is also optimized as a hyperparameter. For performance reasons, we do not let the maximum similarity difference or maximum number of neighbors go above 8 or 200 respectively during hyperparameter search.

After the filter, we take the softmax of the remaining similarity scores and collect the labels of the corresponding reference images. The similarity scores after softmax are now interpreted as weights representing how relevant each reference image is for suggesting labels for the query image. By extension, we interpret them as weights for the labels of the reference images. For each label present in the reference images, we take the sum of all the weights given to it. We now have a set of labels, each with an assigned score between zero and one, which we call the label's confidence score. Ideally, this score should be close to one for correct labels, and close to zero for incorrect labels. For performance reasons, we add the weights of the most similar reference images first and stop when the cumulative weight of all the processed images reaches 0.99, as any further processing will have a minimal impact on the result.

The model can also present the reference images that were important for the result, to the user, making it possible to verify that they are in fact relevant. This can help the user understand what informed the result, and thus how much they can trust it. Alternatively, the user can ignore the labels suggested by the model, only looking at the best matches and their labels.

Other approaches for converting search results into labels are possible. One could choose not to weight the reference images, but let each one count equally to the result. This is equivalent to setting the softmax temperature to 0. In that case, the filter should probably be made more restrictive, allowing fewer reference images to pass through. This can be accomplished by lowering the maximum similarity difference or maximum number of neighbors. The filter could also be changed. It could select a certain number of the most similar images each time and otherwise ignore the similarity scores. This would make the process more similar to a standard k-nearest neighbors model. It could also select every reference image above a certain constant similarity threshold. This might especially make sense if it is necessary to reduce the dependence on the user inspecting the matching reference images and verifying that they are reasonable matches.

3.2.3 Neighbor net models

During work on this thesis, Zhang et al. [2021] introduced the model Tip-adapter, mentioned in section 2.2.2. Inspired by this, we adopt the idea of implementing an image cache model like the one presented by Orhan [2018] as a two-layer neural network, to be used as is or with fine-tuning. Such a cache model works similarly to our image similarity search but does not filter the matching reference images. Like the other models, it takes as input the features of the query image as calculated by an image encoder and gives as output a confidence score between 0 and 1 for each class. We present two versions of this model: one largely unmodified version of a cache model and one with more substantial modifications. We refer to these as the *simple neighbor net* model and the *class-wise neighbor net* model respectively.

Simple neighbor net model

Without training, the calculations of the simple neighbor net are equivalent to the neighbor search described in subsection 3.2.2 and shown in Figure 3.7 except that it does not have a filter to restrict the number of neighbors to consider. Implementing the model as a neural network makes it possible to fine-tune it.[7]

[7]It also has the added benefit of making the model much faster than the neighbor search, as the entire model can be efficiently run on a GPU, as opposed to only a few of the operations with the neighbor search.

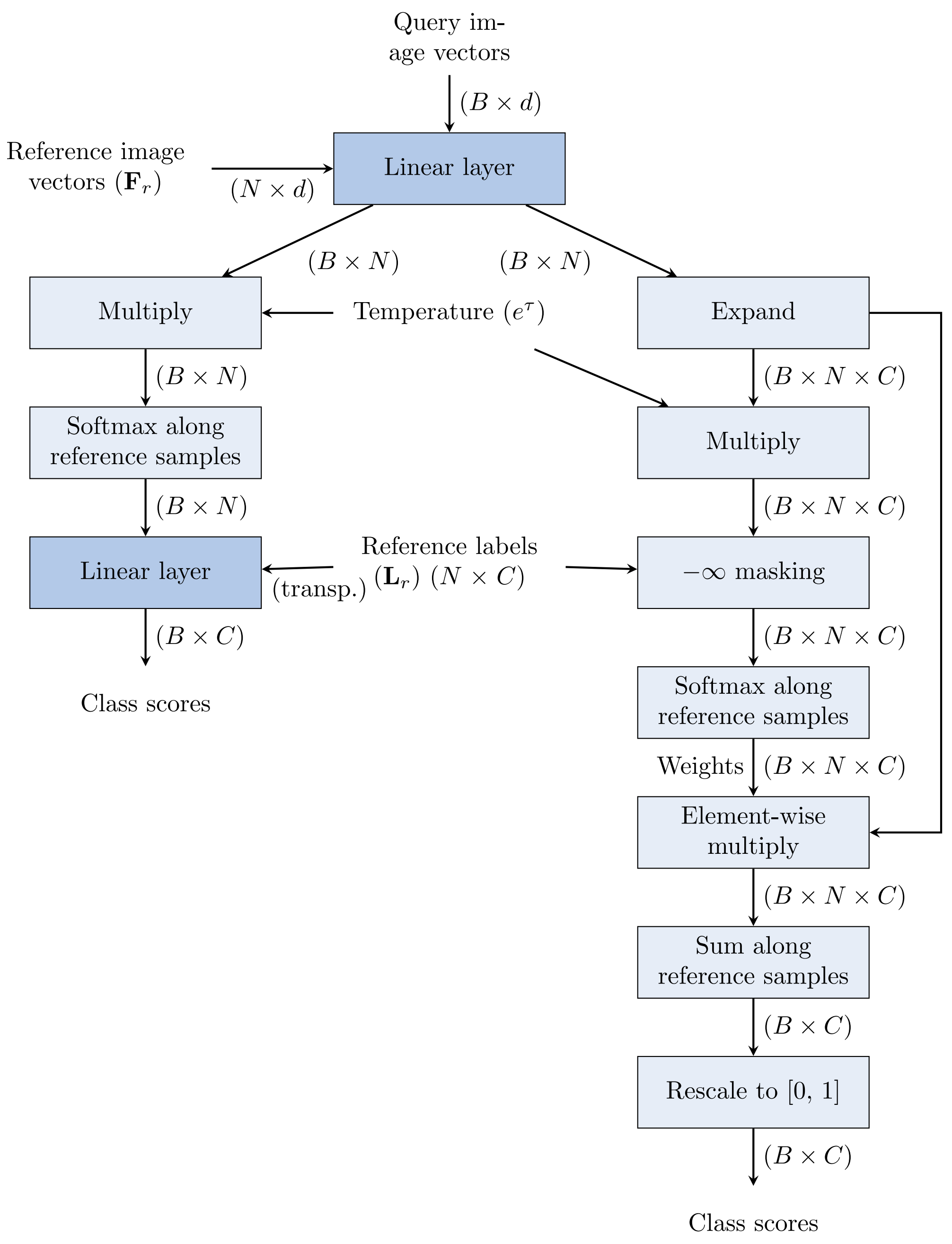

Figure 3.8: The architecture of the simple neighbor net (left branch) and the class-wise neighbor net (right branch). The middle part of the figure is common to both models. B is the number of query images in a batch. Tensor dimensions are shown in parentheses.

The leftmost branch of Figure 3.8 shows the architecture of this model. The first linear layer of the network calculates the cosine similarity between the query image and each reference image. Adapting the notation from Zhang et al. [2021], this is achieved by setting its weights to $\mathbf{F}_r$, a $N \times d$ matrix whose rows are the feature vectors of all the reference samples. Here, N is the number of reference samples, and d is the size of the feature vectors. Taking the feature vector f_q of the query image as input, the layer computes the matrix-vector product $\mathbf{F}_r f_q$, which is a vector of size N containing the cosine similarity between the query image and each of the reference images.

To obtain normalized weights, we multiply this vector by a softmax temperature factor e^τ and then perform the softmax operation σ. Thus, the first layer along with the softmax can be summarized as $S = \sigma(e^\tau \mathbf{F}_r f_q)$.

The second layer weights the labels of the reference images by their similarities by computing $\hat{y} = \mathbf{L}_r^T S$. Here, $\mathbf{L}_r$ is a $N \times C$ matrix, where each of the N rows contains the labels of a reference image as C numbers, either 0 or 1, one for each class. $\hat{y}$ now contains confidence scores for each class and is the final output of the network.

Although the modifications made to the neighbor search model to create the simple neighbor net model were inspired by Tip-adapter [Zhang et al., 2021], some differences remain between our model and Tip-adapter. In what the authors call a residual connection, Tip-adapter combines the class scores from the cache model with those of zero-shot CLIP, utilizing both reference images and text. Tip-adapter's cache model also uses a different function to determine the weights of the reference images, whereas our simple neighbor net model uses the same function as Orhan [2018], which is also equivalent to our neighbor search model's weights. Our model is also adapted to handle multi-label classification, which Tip-adapter does not. We achieve this mainly by normalizing the weights of the reference images such that the final linear combination of reference labels contains numbers between 0 and 1. A difference worth mentioning outside of the architecture itself is that Zhang et al. [2021] test Tip-adapter on the task of classifying ImageNet images and not a specialized domain dataset.

Class-wise neighbor net model

The simple neighbor net model applies the softmax operation across all reference image similarities without regard to their labels. This normalizes the similarities making them sum to one. Especially when aggregating the reference images and using a one-hot encoding for the labels (see the explanation of aggregation below) this can make the classes compete for the same confidence score points if reference images from several classes get a high similarity score compared to the query image. This is not desirable in a multi-label setting, where several classes should be able to get a high confidence score at the same time.

To prevent this, the class-wise neighbor net model looks at reference images of each class separately, explicitly considering how similar the reference images of each class are to the query image. It then summarizes these similarities into one number for each class, which we interpret as the class confidence score. This raises the question of how to summarize the similarities. Natural ways to do this are to take their average or their maximum. However, both of these approaches may introduce issues. Using the maximum similarity within each class can be thought of as answering the question "How similar is the query image to the most similar reference image of each class?". An issue with this is that it does not at all take into account any other reference images than the most similar one and might be too easily fooled by a single noisy or unrepresentative reference image. Using the average similarity for each class instead would be less susceptible to noise. However, we suspect that it would hinder the model's ability to recognize classes that have great variation in image appearance. For example, the class "Search and rescue (SAR)" in our dataset contains images of helicopters, of people, and of other pieces of equipment. Averaging similarities across these very different images could lose important information.

We try to avoid these issues by, instead of the (equally weighted) average or maximum, choosing to use a weighted average of the similarities, using the similarities after a softmax operation as weights. This is a generalization of the two approaches which allows the model optimization to find a good middle ground between them. By multiplying with a softmax temperature first, softmax can generate weights focusing completely on the most similar reference image (using an infinitely high temperature) or weight each reference image equally (using a temperature of 0). Since this model variant considers the similarities of the reference images from each class separately, we call this the class-wise neighbor net.

The rightmost branch of Figure 3.8 shows the architecture of this model. It implements the class-wise similarity softmax described above. To avoid loops and allow for running efficiently on a GPU, it is implemented as a series of tensor operations that processes all classes at once. The "Expand" operation adds a new dimension for the classes by repeating the input along that dimension. The similarities are then multiplied by the softmax temperature. $-\infty$ is inserted at all positions where the reference label is 0, leaving the original input only in positions (b, n, c) such that reference image n belongs to class c. This is followed by a softmax operation along the reference sample dimension. Inserting $-\infty$ before the softmax has the effect of applying a mask for the softmax, making it place a 0 in all positions that are not relevant for the class in question, and thus letting the values within each class sum to 1. The output of the softmax operation is now the weights that are to be used for the weighted average of the similarities. This weighted average is produced by multiplying these weights element-wise by the original similarities after the expansion from earlier and then summing along the reference sample dimension. The output is now a weighted average for each class of the similarity with the reference images belonging to that class, giving a higher weight to the most similar reference images. Finally, the results are rescaled linearly from [-1, 1] to [0, 1].

Fine-tuning

Due to the design of the network and the way its weights are initialized, it can be used zero-shot, or it can be fine-tuned. The only parameters that we train during fine-tuning, are the weights of the first linear layer, which are initialized to $\mathbf{F}_r$, and the softmax temperature e^τ, which is trained as its logarithm τ as in Radford et al. [2021].[8] As opposed to Radford et al. [2021], we do not clamp the temperature to a maximum of 100. The weights of the second linear layer of the simple neighbor net are frozen. The linear layers do not have any biases.

Aggregating reference samples

Using all reference samples in the first layer of the network has the disadvantage of resulting in a very large weight matrix for large training sets. This can impact performance and hinder fine-tuning due to overfitting. In our case, with approximately 80.000 reference images and feature vectors of size 512, this matrix contains around 40 million weights. This is fine performance-wise, but we do see signs of overfitting.

Zhang et al. [2021] explore averaging the representation vectors of groups of samples to create fewer "prototype" samples. This makes it possible to reduce the number of samples, and thus parameters, to a constant number per class regardless of the original number of samples. They

[8]To circumvent numerical issues during training, the softmax temperature for the class-wise neighbor net is optimized as a hyperparameter instead of trained as a normal parameter.

use 16 as their number of prototypes per class. They find that these 16 prototype samples can capture information from the added additional available samples, and that fine-tuning helps to avoid diminishing returns as the number of samples grows. We adopt this method but adjust it to account for our problem being multi-label instead of multi-class classification. Zhang et al. [2021] also limit this exploration to the relatively few-shot case, testing a maximum of 128 samples per class, whereas our most common classes have several thousand samples. It is thus not known how well the approach works in our case.

Since Zhang et al. [2021] are concerned with single-label classification, they can exploit some simplifying properties of this problem that simplify the aggregation, but that are not valid for the multi-label case. One such property is that each reference sample contributes to exactly one class's prototype sample. With multi-label samples, however, a choice must be made: Should a sample with several labels be included in prototype samples of all relevant classes, or only one? We choose to include it in all relevant classes, avoiding the need to choose between the classes, and utilizing the information of every label. This means that the more classes a sample belongs to, the more influence it will have in the model's decisions. This may not be a bad thing, as a sample belonging to many classes does indeed mean that it is relevant in many cases.

Another simplifying property is that creating *labels* for the prototype samples is trivial with single-label samples, as they are all identical within each group – they belong to the same class. where labels within each class are identical, they do not address how to aggregate the *labels* of the reference images during aggregation. In multi-label classification, reference images belonging to the same class can have different labels, as they can also belong to other classes apart from the one in question. This means that not just the samples' representation vectors must be aggregated, but also their labels.

We implement two ways to aggregate the labels: simple averaging and one-hot encoding. Averaging allows a prototype sample that was created for one class, to contribute to other classes as well. For example, if a prototype sample for class 1 is made from 10 samples, all of which belong to class 1, 5 of which to class 2 and none to class 3, the label of the prototype sample will be [1, 0.5, 0], and any similarity between the query image and this prototype sample will contribute half as much to class 2 as it will to class 1. With one-hot encoding, we ignore the labels of the reference samples that make up the prototype sample, and replace them with a one-hot encoding of the class that the prototype sample was made for. In the example above, this would result in the label for the prototype sample being [1, 0, 0] even though some of the samples belonged to class 2. Since the samples belonging to class 2 are included in the prototype samples of that class anyway, the samples can still convey this information through them.

Zhang et al. [2021] aggregate samples into prototype samples by creating groups randomly before averaging their representation vectors. As the number of samples to aggregate, and with it the number of samples per group to average, grows, we expect these averages to become less and less meaningful. Due to the law of large numbers, we can expect the averages from the different aggregation groups for a class to tend to the same vector, losing information about any variation within the class. This could be solved by increasing the number of prototype samples per class, but this would partially defeat their purpose, letting the number of parameters grow along with the number of available samples. To avoid this we evaluate grouping the samples by similarity instead of randomly before averaging representation vectors. We use clustering to put similar reference samples in the same aggregation group, letting each cluster serve as an aggregation group. The motivation is that this will allow the prototype samples to each capture their own variation of the class and better represent the entire class. We use mini batch k-means clustering for creating the clusters to speed up calculation with large numbers of samples compared to normal k-means clustering. This is explained in subsection 2.1.5. An artifact of the clustering is that some of the clusters may be empty, especially if the number of samples is low compared to the number of clusters. We detect this and set the aggregated representation vector of empty clusters to the zero vector, causing the dot product with any query image representation vector to be 0 and thus in practice not being considered further.

Chapter 4

Experiments and Results

Research question 2, along with its subquestions, asks how to best harness the knowledge from CLIP's language-image pre-training. In this chapter, we present the experiments that we perform to shed light on this. We describe the experiments and how they are carried out, present the specific hyperparameters used for each variant of the models, and present the results of the experiments.

4.1 Experimental Plan

We perform experiments to answer research question 2 from section 1.2, which concerns how to best utilize CLIP for our task. We evaluate the language-image model, the neighbor search model, different variations of the neighbor net model, and a 1-NN as a baseline. These models are described in section 3.2. Some of the models have more room for variation than others. We experiment with different variants and identify each model's best-performing variant. This variant is what we finally use to compare the models to each other.

4.1.1 Data and model pipeline

We implement the models with similar interfaces such that they can be used as part of the same pipeline. Figure 4.1 shows our pipeline for training, optimizing, and evaluating the models on our dataset. The pipeline is automated only to a certain extent, with certain parts being performed manually.

The dataset is first prepared by the process outlined in subsection 3.1.2. This process produces three datasets: a training set, a validation set, and a test set. When working with the image-similarity-based models, we also refer to the training set as the reference set, as these are the images used as reference images by these models. Depending on the model architecture, we either use the training/reference set for constructing the model and possibly fine-tuning it, or we do not use it at all. We perform hyperparameter optimization using the validation set. This includes setting the class confidence threshold. Finally, we evaluate the optimized models using the test set.

Following Conde and Turgutlu [2021], we use the F_2 score (described in 2.1.6) as a metric for optimizing and comparing methods. This takes both precision and recall into account, but gives recall twice as much weight as precision. All the models output class confidence scores between 0 and 1 for each image-class pair, but the F_2 score does not consider the exact value of the confidence scores, only whether or not it is above the model's threshold. Other metrics

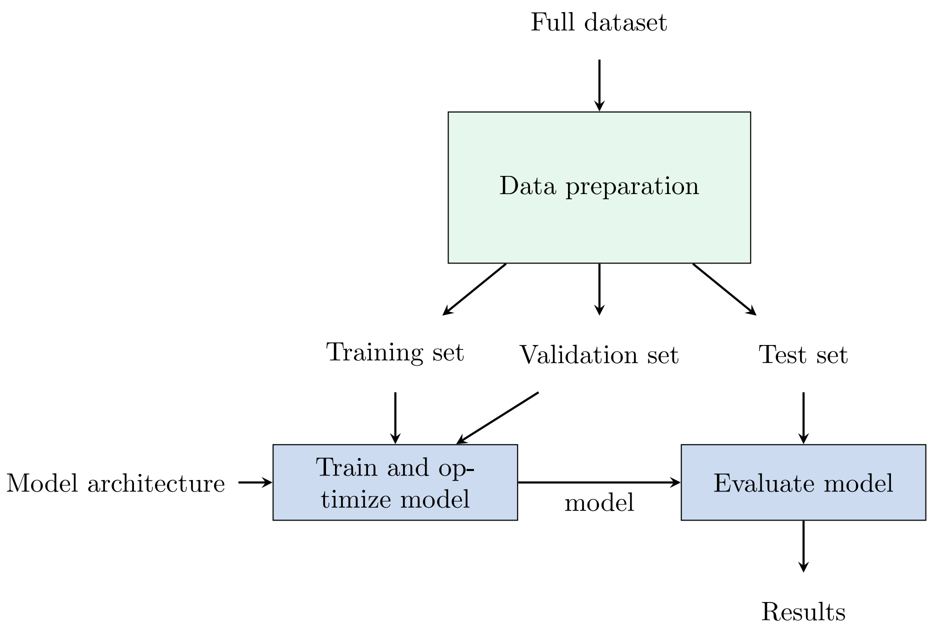

Figure 4.1: The model pipeline

such as mean squared error or area under the ROC curve do consider the confidence scores[1] and may give a better indication of the model's ability to differentiate high and low confidence answers. However, we believe that the F_2 score captures the most essential aspects of the models' performance well, focusing on the task of either suggesting a label for an image or not suggesting it. For fine-tuning the neighbor nets we use mean squared error as loss, as this requires a differentiable measure.

4.1.2 Language-image model

Our language-image model is based on the zero-shot classifier by Radford et al. [2021]. In that work, the model is evaluated on multi-class classification using various datasets. We aim to answer subquestion 2 a from section 1.2 asking how well the approach works on our type of data and for multi-label classification. The model is described in subsection 3.2.1.

For this model, we experiment with three versions of the class names: the original Norwegian labels, automatically translated labels, and manually translated labels. We store the translations as mappings from original name to translated name and use these mappings when building the model. We evaluate all of these versions with context ensembling. To investigate the impact of context ensembling, we also evaluate the best of these versions with only one context.

When using context ensembling we have created a list of suggested contexts and optimize the weight of each context as hyperparameters. The weights are chosen from the range [0, 100] and then normalized such that their sum to 1. If the weight of a context is set to 0, it is effectively unused. When not using ensembling, the model is optimized by evaluating one context at a

[1]The area under the ROC curve depends on the confidence scores indirectly by considering many threshold values.

time and choosing the best one. When using the original Norwegian labels, we supply suggested Norwegian contexts as well as the same English contexts that we use for the English labels. The Norwegian contexts are merely translated versions of the English contexts. It is then up to the hyperparameter optimization to decide whether to use English or Norwegian contexts or both.

4.1.3 Image-based models

To answer subquestion 2 b from section 1.2, which asks how well suited the image features learned by CLIP's image encoder are for our task, we perform several experiments using only these image features. These experiments evaluate the neighbor search model and the neighbor net model variants, which are described in section 3.2, as well as the baseline 1-NN model.

1-NN model

As a baseline, we evaluate a simple 1-nearest neighbor (1-NN) model like the one used by Conde and Turgutlu [2021] for fine-grained multi-label classification. k-NN models are described in subsection 2.1.3. In practice, we use our implementation of the neighbor search described in section 3.2 to implement the 1-NN baseline model. This is possible because 1-NN is a special case of the neighbor search. We do this by setting the hyperparameter for the maximum number of neighbors of the neighbor search to one. Doing this effectively puts the other two hyperparameters out of effect, as they make no difference when the number of neighbors is only one.

Neighbor search model

The neighbor search model has no variants to test other than varying hyperparameters, which is handled by hyperparameter optimization. We therefore only present one experiment using this model.

Neighbor net models

The neighbor net model has several variants. Variants include the simple and class-wise architectures, and nets with and without prototyping, clustering, and fine-tuning and with different ways to create the labels of the prototype samples. These properties and the motivation for introducing them are described in subsection 3.2.3. There are five properties with two options each that can be varied and combined, resulting in close to $2^5 = 32$ possible model variants.[2] We have not tested all of these, but rather let the evaluation of each change to the models inform further experimentation. In this process, we use evaluations on the validation set to make model design decisions during experimentation. However, the results presented in subsection 4.3.3 are from evaluations on the test set.

During experiments, it turned out that introducing prototype samples reduced performance when *not* fine-tuning. However, a motivation for introducing prototype samples was specifically to help with fine-tuning by reducing overfitting. Since hyperparameter optimization is considerable faster without fine-tuning, we experiment further with prototype samples without fine-tuning. Then, after identifying which properties give the best result, we reintroduce fine-tuning.

When using prototype samples, we mostly use 16 prototypes like CLIP-art [Conde and Turgutlu, 2021], but we also make an attempt to optimize this number as a hyperparameter.

[2]In reality, not all combinations are possible – for instance – clustering requires using prototypes, so the real number is a bit lower.

4.2 Experimental Setup

This section contains information about the optimization results necessary for reproducing the experiments.

We optimize each method that uses hyperparameters using the validation set and Bayesian optimization, which is explained in section 2.1.7. We start the process with 10 random initialization iterations. The optimization stops when the 10 best F_2 scores are all within a window of 0.1 percentage points, or after a maximum of 1000 iterations. The best set of hyperparameters found are used for evaluation on the test set, and we report the result of this evaluation.

The models output a confidence score for each class, but calculating certain metrics – F_2 score, for instance – requires binary predictions. Depending on the use case, binary predictions may also be required when putting the model to use, such as when deciding whether to suggest the class to the user. We set a threshold for the confidence score above which the class is included as a suggested label. This threshold is set at the value that maximizes the F_2 score on the validation set for the model in question.

4.2.1 Language-image model

The hyperparameters of this model are the relative weights of the different contexts. The suggested contexts are written manually based on the experiences reported by Radford et al. [2021]. Table 4.1 shows the hyperparameters for the language-image model with context ensembling as well as the full set of suggested candidates. The model seems to have a certain preference for the endpoints 0 and 100, meaning no weight and equal weight. The reason for this is unknown; there might be an artifact of the Bayesian optimization that makes the minimum or maximum allowed values more likely to be chosen. For the experiment without context ensembling, the context chosen by optimization was "a photo from the military of a {}".

For the automatically translated labels, the hyperparameter search returned a weight of 38.13 for the context "a picture of a{}" and 100 or 0 for all other contexts. We discovered that setting this to zero as well, giving equal weight to the remaining 6 contexts, gave equal or slightly improved results, and we use this manual modification in our experiment for simplicity. The same was not true for the single non-endpoint weight of the model with manually translated labels, which was 12.57.

Table 4.2 shows the optimal class confidence threshold for each model.

4.2.2 Image-based models

1-NN model

Since k is fixed to 1, this model has no meaningful hyperparameters and thus requires no optimization. The class confidence threshold is also unimportant, as this model can only output class confidences of exactly 0 or 1.

Neighbor search model

This model has three hyperparameters that are optimized through Bayesian optimization: the softmax temperature – a factor to multiply the similarity score of each reference image by before performing softmax, the maximum similarity difference, and the maximum number of neighbors (matching images) to consider, as explained in subsection 3.2.2. As with the other models, we also find the optimal class confidence threshold. Table 4.3 shows the hyperparameters for the

	Label translation		
Context	None	Automatic	Manual
a photo of {}	0	0	0
a photo of a {}	0	0	0
a picture of a {}	0	0	0
a drawing of a {}	100	100	100
an illustration of a {}	100	100	0
a photo of a big {}	100	100	0
a photo of a small {}	55.78	0	0
a military photo of a {}	100	0	12.57
a photo from the military of a {}	100	0	100
a picture from the military of a {}	46.01	100	100
this photo from the military contains a {}	23.21	100	0
this photo from military contains a {}	100	100	0
a photo of a {}, a type of military equipment	0	0	0
a photo of a {}, a type of vehicle	0	0	0
et fotografi av {}	0	-	-
et fotografi av en {}	0	-	-
et bilde av en {}	72.07	-	-
en tegning av en {}	83.99	-	-
en illustrasjon av {}	0	-	-
et fotografi av en stor {}	0	-	-
et fotografi av en liten {}	0	-	-
et militært fotografi av en {}	0	-	-
et fotografi fra militæret av en {}	0	-	-
et bilde fra militæret av en {}	22.97	-	-
dette fotografiet fra militæret inneholder en {}	0	-	-
dette fotografiet fra militær inneholder en {}	0	-	-
et fotografi av {}, en type militært utstyr	0	-	-
et fotografi av {}, en type kjøretøy	0	-	-

Table 4.1: The hyperparameters (relative context weights) used for the language-image model. "{}" denotes the position of the class label.

Context ensembling	Label translation	Threshold
Context ensemble	Original labels	63.30 %
	Automatic	63.70 %
	Manual	64.06 %
Single context	Manual	64.90 %

Table 4.2: The class confidence threshold used for each variant of the language-image model.

image similarity search model that were found during optimization on the validation set and used for evaluation on the test set.

Hyperparameter	Value
Softmax temperature	68.566
Maximum similarity difference	5.2568
Maximum number of neighbors	36
Class confidence threshold	14.34 %

Table 4.3: The hyperparameters used for the neighbor search model

Neighbor net model

Model	Prototypes	Clustering	Prototype labels	Fine-tuned	Softmax temperature	Epochs	Mini-batch size	Learning rate	Momentum
Simple	*No*	-	-	*No*	105.60	-	-	-	-
	No	-	-	*Yes*	*105.60*	6	319	0.000016878	0.10277
	16	*No*	*One-hot*	*No*	10.557	-	-	-	-
	16	*Yes*	*One-hot*	*No*	57.704	-	-	-	-
	16	*Yes*	*Average*	*No*	43.653	-	-	-	-
	16	*Yes*	*Average*	*Yes*	*43.653*	12	33	9.8052	0.36597
	4241	*Yes*	*Average*	*No*	102.13	-	-	-	-
Class-wise	*16*	*Yes*	-	*No*	14.193	-	-	-	-
	16	*Yes*	-	*Yes*	4.9755	63	64	100	0.90390
	16	*No*	-	*Yes*	1	100	64	52.2653	0.99

Table 4.4: The hyperparameters used for each variant of the neighbor net model. The values in italics are not optimized. For the simple neighbor net with fine-tuning, the softmax temperature is trained, and the value shown is its initial value before training.

Table 4.4 shows the hyperparameters used for the different variants of the neighbor net model. When not fine-tuning, the softmax temperature is the only optimized hyperparameter. For fine-tuning we use stochastic gradient descent (SGD) with a cosine annealing learning rate scheduler like Tip-adapter [Zhang et al., 2021]. We do not use warm restarts for the scheduler, as it does not seem to improve performance. We found that the mini-batch size and learning rate used by Tip-adapter[3] did not work well for training our model. Instead, we optimize them as hyperparameters. We also optimize the number of epochs as well as a momentum factor for SGD. As loss, we use mean squared error instead of the binary cross entropy used by Tip-adapter. This

[3]Tip-adapter uses a mini-batch size of 256 and a learning rate of 0.001.

is to keep the loss from exploding when confidence scores outputted by the model are exactly 0 or 1, which they can be as there is no activation function at the end of the network.

When fine-tuning the simple neighbor net model, like Radford et al. [2021], we train the softmax temperature as a normal parameter during the fine-tuning to avoid having to optimize it as a hyperparameter. We then initialize the temperature to the value used by the corresponding model without fine-tuning, which was found through hyperparameter optimization. This way the models are identical before fine-tuning starts, and the training can start improving it immediately instead of finding a good softmax temperature separately. For the class-wise neighbor net model, the softmax temperature is not trainable, so we optimize it as a hyperparameter.

4.3 Experimental Results

We now present the results of the experiments grouped by type of model. We then compare the best variant of each model along with the 1-nearest neighbor baseline. We present the F_2 score of all model variants and a broader selection of metrics for the final comparison. We also show how well the models recognized individual classes. Finally, we show each model's output for an example image.

As explained in section 2.1.6, the precision metric and, through it, the F_2 score depend on the prevalence of the dataset and can not be used to meaningfully compare classifiers that have been evaluated on datasets with different prevalences. The prevalence of the test set used to generate the results presented in this section is 1.16 %. In Table 4.8 we also present metrics that do not depend on prevalence.

4.3.1 Language-image model

Context ensembling	Label translation	F_2 score
Context ensemble	Original labels	11.6 %
	Automatic	15.8 %
	Manual	19.1 %
Single context	Manual	18.6 %

Table 4.5: The results of different variants of the language-image model

Table 4.5 shows the results from experiments with the language-image model. The effect of translating the labels is clear, with automatic translation performing better than the original, Norwegian labels, and the manually translated labels performing better still. We also see that using context ensembling has only a minimal impact on the performance.

4.3.2 Neighbor search model

Table 4.6 shows the performance of the neighbor search model, both restricted to 1 neighbor and without such a restriction. We also show the result of evaluating the baseline 1-nearest neighbor model, which we implement as a special case of the neighbor search model. Comparing the neighbor search to the 1-NN model, we see that allowing the use of several neighbors has a significant impact on the F_2 score, increasing it by 10.9 percentage points.

Model	F_2 score
1-NN	45.8 %
Neighbor search	56.8 %

Table 4.6: Results of the neighbor search model and the 1-NN model, where the latter is implemented as a special case of the former.

4.3.3 Neighbor net model

				Fine-tuned	
Model	Prototypes	Clustering	Prototype labels	No	Yes
Simple neighbor net	No	-	-	55.8 %	55.8 %
	16	No	One-hot	22.5 %	
	16	Yes	One-hot	27.6 %	
	16	Yes	Average	38.6 %	54.8 %
	Optimized	Yes	Average	55.1 %	
Class-wise neighbor net	16	Yes	-	17.8 %	55.3 %
	16	No	-		55.9 %

Table 4.7: F_2 scores of different variants of the neighbor net model.

Table 4.7 shows a comparison of the different variants of the neighbor net models that we have tested.

Introducing prototype samples without fine-tuning worsens the F_2 score considerably from 55.8 % to 22.5 %, as well as all other metrics (not included in the table). When introducing fine-tuning in addition to the prototype samples, the F_2 score increases to 54.8 %, 1.0 percentage points short of the performance without prototype samples.

When optimizing the number of prototypes, the hyperparameter search found 4241 prototype samples to be the optimal number, resulting in an F_2 score without fine-tuning of 55.1 %, which is still slightly lower (0.7 percentage points) than the score without prototype samples. However, this is without fine-tuning. Doing the same optimization with fine-tuning would be interesting in future work.

We see that using clustering improved the F_2 score of the simple neighbor net without fine-tuning from 22.5 % to 27.6 %, an improvement of 5.1 percentage points. The class-wise neighbor net with fine-tuning performed slightly better without clustering, with an F_2 score of 55.9 % as compared to 55.3 %. It is not clear that this small difference is more than random variation, and it is certainly not large.

Using average labels for the prototype samples instead of one-hot encoded labels improved the F_2 score from 27.6 % to 38.6 %, an improvement of 11.0 percentage points.

Without fine-tuning, the class-wise neighbor net performs much worse than the simple neighbor net. Compared to the best performing simple neighbor net without fine-tuning, the F_2 score was reduced from 38.6 % to 17.8 %, a reduction of 20.8 percentage points. However, with fine-tuning, the performance of the class-wise neighbor net is 1.1 percentage points better than the simple neighbor net. The class-wise neighbor net is not made to use the average prototype labels, and this column is thus unused in the table.

The best performing variation is the class-wise neighbor net with fine-tuning and without clustering, which achieved an F_2 score of 55.9 %. This is less than 0.1 percentage points better than the simple neighbor net without prototypes, which achieved an F_2 score of 55.8 % both with and without fine-tuning. This difference is less than the stopping criteria for the hyperparameter optimization, which means that it is not significant. This means that the best variation was able to match, but not surpass the performance of the simple neighbor net without fine-tuning or prototype samples.

4.3.4 Model comparison

Model	F_2 score	MSE	Recall	Precision	Accuracy	Subset acc.	ROC AUC
Language-image	19.1 %	0.3760	38.9 %	6.3 %	92.6 %	2.5 %	0.805
1-NN	45.8 %	0.0126	45.9 %	**45.6 %**	**98.7 %**	**35.8 %**	0.726
Neighbor search	**56.8 %**	**0.0075**	**66.5 %**	35.8 %	98.2 %	18.2 %	0.945
Simple neighbor net	55.8 %	0.0077	65.8 %	34.7 %	98.2 %	19.7 %	**0.969**
Class-wise neighbor net	55.9 %	0.0078	65.0 %	35.8 %	98.2 %	11.3 %	0.942

Table 4.8: Comparison of different methods, using micro averaging where applicable.

Table 4.8 compares the best variants of each model, also showing other metrics than the F_2 score. We see that all image-based models greatly outperform the zero-shot language-image model. The F_2 score of the language-image model is 26.7 percentage points below the 1-NN baseline.

The neighbor search model is the best performing model and improves the F_2 score with 11.0 percentage points compared to the 1-NN baseline. The neighbor net models come close to this result, but do not improve upon it. The simple and class-wise neighbor nets improve the F_2 score by 10.0 and 10.1 percentage points respectively.

To give a more complete view of the differences between the various models, we include several metrics in addition to the F_2 score in Table 4.8. These metrics are described in subsection 2.1.6. The F_2 score is what we optimize during hyperparameter search and model design decisions such as choosing the best variant of the neighbor net models.

Table 4.9 shows the ROC AUC for each of the 10 most common classes for the different methods. We choose to show this metric instead of the F_2 score because we believe it better shows the model's ability to discriminate positive and negative samples across thresholds, not being restricted by a common threshold for all classes. The neighbor search model has the best performance in 6/10 cases, and the simple neighbor net is best in the remaining 4/10 cases. There is very little difference between the two models' results, owing to the models being nearly equivalent.

We look more closely at the class-specific performance of the language-image model and the best image-based model. The language-image model performed at its best on the classes "Patch", "Transport helicopter", "Passenger aircraft", "Aircraft carrier", and "Attack helicopter". It performed the worst on the classes "Customs" and "Garrison" as well as several acronym names. On several classes, it even performed worse than random chance. The neighbor search model performed the best on the classes "Passenger aircraft", "Attack helicopter", "Fishing vessel", "Emblem", and "Logo". It performed the worst on the classes "Minelayer", "Harbor", "Self-propelled mortar", and "Armored medical evacuation vehicle".

Class	Neighbor search	1-NN	Language-image	Simple neighbor net	Class-wise neighbor net
Three axle vehicle	**0.910**	0.726	0.744	0.906	0.895
Kamaz	**0.936**	0.761	0.769	0.933	0.930
Communication vehicle	**0.926**	0.711	0.815	0.924	0.923
Four axle vehicle	**0.926**	0.744	0.700	0.925	0.909
Ural Automotive Plant (UralAz)	**0.910**	0.682	0.688	0.907	0.899
Radar	0.926	0.776	0.812	**0.926**	0.898
Antenna	0.929	0.771	0.812	**0.935**	0.920
Zavod imeni Likhachyova (ZiL)	**0.917**	0.673	0.612	0.914	0.902
Trailer	0.909	0.690	0.724	**0.913**	0.891
Surface-to-air missile (SAM)	0.905	0.724	0.769	**0.911**	0.878

Table 4.9: Comparison of the class-specific ROC AUC of different methods on the 10 most common classes.

4.3.5 Example

To demonstrate the model, we perform inference on the image in Figure 3.1, which is not taken from our defense dataset. The image depicts a four-axle tank truck from the manufacturer Kamaz. Table 4.10 shows the final label suggestions for the query image based on these and other reference images. Demonstrating that the neighbor search model can be used to identify relevant reference images and present them to the user for inspection, Table 4.11 shows the keywords and normalized similarity scores of the most similar reference images as judged by the model. These are the top five images contributing to the final label suggestions for this model in Table 4.10.

Label	Neighbor search	1-NN	Language-image	Simple neighbor net	Class-wise neighbor net
Four axle vehicle	**33.8 %**		**65.4 %**	**29.0 %**	**19.0 %**
Kamaz	**79.5 %**	**100.0 %**	**66.3 %**	**71.4 %**	**53.3 %**
Tank truck (POL)	**19.8 %**		**65.8 %**	**14.0 %**	**59.2 %**
Truck	**19.8 %**		**64.1 %**	**14.6 %**	**19.4 %**
Armoured bridgelayer			**64.2 %**		
Armoured command vehicle			**65.9 %**		
Armoured communication vehicle			**65.5 %**		
Armoured engineer reconnaissance vehicle			63.9 %		
Armoured medical evacuation vehicle			**65.3 %**		
Armoured NBC reconnaissance vehicle			64.0 %		
Armoured recovery vehicle (ARV)			**65.2 %**		
Armoured transport vehicle			**65.3 %**		
Artillery command vehicle			**65.3 %**		
Bridgelayer	7.8 %				
Command vehicle			**65.0 %**	6.3 %	11.9 %
Communication vehicle	**18.4 %**		**65.0 %**	**16.1 %**	
Container				5.3 %	
Crane truck			**65.0 %**		
Decontamination vehicle			**65.2 %**		
Direction finder	4.9 %				
Eight axle vehicle			**65.4 %**		
Engineering vehicle			**65.2 %**		
GAZ-66			**64.3 %**		
Generator truck			**64.4 %**		
Heavy equipment transporter (HET)			**64.5 %**		
Kraz			**65.5 %**		
Light utility vehicle (LUV)			**64.2 %**		
Maintenance vehicle	10.8 %		**65.2 %**	10.2 %	10.6 %
Medical evacuation vehicle			**64.5 %**		
Missile (SSC)			**64.1 %**		
Multiple rocket launcher (MLR)			63.9 %		
NBC decontamination vehicle			**64.8 %**		
Reconnaissance vehicle			**64.1 %**		
Recovery vehicle			**65.1 %**		
Self-propelled			**64.5 %**		
Seven axle vehicle			**65.5 %**		
Surface-to-air missile (SAM)			**64.1 %**		
Test vehicle			**65.1 %**		
Three axle vehicle	**49.6 %**	**100.0 %**	**65.5 %**	**51.5 %**	**83.1 %**
Two axle vehicle	**16.6 %**		**65.5 %**	**13.1 %**	10.4 %
Ural Automotive Plant (UralAz)				**15.1 %**	**24.0 %**

Table 4.10: Label suggestions with corresponding confidence scores of different models using the image of Figure 3.1 as query image. All confidence scores above its model's threshold are included (shown in bold), as well as the three runners-up. The ground-truth labels are written in bold.

	Match 1	Match 2	Match 3	Match 4	Match 5
Weight	4.7 %	4.6 %	4.0 %	3.5 %	3.5 %
Labels	**Kamaz** Three-axle	**Truck** **Kamaz** Two-axle	**Kamaz** Communic. vehicle Satcom Three-axle Antenna	**Truck** Baz **Four-axle** Heavy equip. transporter	**Tank truck (POL)** **Kamaz** **Four-axle**

Table 4.11: The weights and labels of the five top matching reference images found by the neighbor search model when using the image of Figure 3.1 as query image. The ground-truth labels of the query image are shown in bold. Some labels are shortened. Match 5 is an image of the same exact vehicle.

Chapter 5

Discussion

The results presented in section 4.3 show significant differences in performance across different models. They also indicate whether the different modifications made to the models improve them. In this chapter, we evaluate these results and discuss what we can learn from them. First, following research subquestion 2 a, we consider the zero-shot language-image model, noting its apparent poor results and discussing possible reasons for it. We then move to research subquestion 2 b by discussing the neighbor search and net models and their variations. Finally, following research question 2, we compare all models.

5.1 Language-image model

The fact that the language-image model's performance increases with each translation – first automatic and then manual – is not completely surprising. CLIP's text encoder can be assumed to have received minimal exposure to Norwegian text and can be expected to perform better with English text than with Norwegian text. However, as exemplified in section 3.2.1, the automatically translated labels contain mistakes that sometimes completely lose their meaning. As such, it was not clear that the automatic translation would increase the F_2 score by as much as 4.2 percentage points, a large increase relative to the score with the original labels of 11.6 %. The manual translation increases the score further by 3.3 percentage points. This means that the overall improvement achieved by translating the labels from Norwegian to English is 7.5 percentage points, a substantial improvement.

Context ensembling makes considerably less of a difference, only increasing the F_2 score by 0.5 percentage points compared to using only the best performing context. Note that it should be impossible for the context ensembling to have a negative impact, as the hyperparameter search space for the context ensembling variant is a superset of that of the single context variant of the model.

As seen in Table 4.10, the language-image models can return a very long list of suggested labels. This indicates a poor ability to discriminate between classes, leading to the model "guessing" when in doubt. Contributing to this is the fact that we choose the class confidence threshold that optimizes the F_2 score, which punishes false negatives more than false positives. This is also reflected in Table 4.8, which shows that the model has a very low precision of 6.3 %. This is even worse than the particular example used in Table 4.10, where 11 % (4/38) of the suggestions were correct. However, in Table 4.8 we also see that one ground-truth label barely had a high enough confidence (64.1 %) to be suggested, demonstrating the model's need for the low threshold.

A clear conclusion from the results is that the language-image model performs quite poorly compared to the purely image-based models. There are several differences between these models that can be the reason for this difference. The language-image model is zero-shot, while the image-based models make use of the tens of thousands of reference images in our dataset. The language-image model's only way to distinguish the classes is to consider the captions, where the class names are all that differ between them. This means that the language-image model depends on the class names to convey the concept of the class. If the class name is too general or even misleading, or if the concept is not contained in the learned representation space of the text and image encoders, the model will be unable to accurately recognize the class. These problems are especially pronounced in our case due to the fine-grained nature and quite specialized domain of our dataset. For instance, the dataset has classes for different numbers of axles on a vehicle. It would not be surprising if the text encoder does not capture this distinction accurately due to limited exposure to it in its pre-training data. While the language-image model has to represent each class only through a single representation vector, the image-based models directly consult similar reference images from the same distribution and specialized domain, helping it recognize obscure or varied classes by example.

The language-image model does not have any knowledge of what the dataset looks like, whereas the neighbor search and neighbor net models use the training set. This is a disadvantage for the language-image model in cases where the class names are obscure, not self-explanatory, or ambiguous and where the classes are more easily understood through examples. If a keyword has poor coverage in the dataset in the sense that it is relevant to many more images than are actually labeled with it, this can hurt the performance of the language-image model because it could give what appears to be false positives when an image lacks a keyword that it should have had. We believe this is not necessarily a weakness of the model, but an artifact of the inconsistent labels in the test set, as even a perfect model could suffer from this problem. Compared to this, if the labeling errors are consistent, the neighbor search and neighbor net may profit from being able to copy the errors of the training set, getting an artificially high score.

On the other hand, the image models' ability to adapt to the specific distribution might become a problem if they are used with a different distribution. The language-image model would likely not be affected by a distribution shift in the same way, but we have not tested this. Combining the language-image model and an image-based model could get the best of both worlds, utilizing both reference images and class names. The ablation study for Tip-adapter [Zhang et al., 2021] indicates that such a combination performs better than either of the models alone. However, the language-image model performs reasonably well on their task of classifying ImageNet images, while it does not on our task.

The language-image model may be a good starting point for fine-tuning similar to what we do with the neighbor net models. The large improvement we see when fine-tuning the neighbor net models despite having reduced the number of weights by creating 16 prototype samples, may be possible to replicate at least partially with the language-image model. This would give a similar weight matrix as using 1 prototype sample per class for the neighbor net, but this number could be increased, for example by duplicating rows in the matrix while adding some noise to help gradient descent update the rows differently.

5.2 Image-based models

The neighbor search comfortably outperforms the 1-NN baseline model. The fact that it at least does not do worse than it, is to be expected. This is because the 1-NN model is a special case of the neighbor search. Thus, if 1-NN had been optimal, we could expect the hyperparameter optimization of the neighbor search to find it. In other words – the neighbor search performing worse than 1-NN would indicate a flaw in the hyperparameter search.

The simple neighbor net model with no sample aggregation and no training is almost, but not quite as good as the neighbor search model. The two are nearly equivalent. The only difference is that the neighbor search model filters the matching reference images before using them. This shows that this filter improves the model compared to considering the labels of all reference images every time. This is further demonstrated by the hyperparameter optimization of the neighbor search model, which made the filter significantly less inclusive than required by the search space.

When optimizing the number of prototype samples, the resulting number is higher than the number of reference samples in almost every class in our dataset. For these classes, the prototype samples map to the reference samples one-to-one[1]. This means that the prototype samples have little effect, and the model is largely the same as a model without prototype samples. The fact that the hyperparameter search identified this high number as optimal, indicates that reducing the number of samples at all is not optimal, at least without fine-tuning.

Fine-tuning made no meaningful difference on the simple neighbor net without prototype samples. All metrics we gathered are extremely close for these two models. The hyperparameter search also chose a low number of training epochs and a low learning rate, suggesting that training the model further did not improve it. When using prototype samples, however, fine-tuning improved performance drastically compared to not fine-tuning. This suggests that the aggregation of samples into prototypes removed information beyond what was made necessary by the reduced number of model parameters. It also demonstrates that the fine-tuning was able to reintroduce this information. However, the performance increased only to around the level of the model without prototypes, indicating that the prototypes did not help the fine-tuning improve the original model.

A strength of using CLIP to calculate image similarities is the model's generality, which arises from the varied and non-specialized data it has been pre-trained on. However, this can also be a weakness. Since CLIP's image encoder is task agnostic – or rather is trained to match general images with text captions – it might not pay attention only to aspects of the images that are relevant to our task. For example, for a picture of a vehicle, it might assign high importance to elements in the environment such as the weather, the type of landscape, or whether there is snow in the image. When all we are interested in, is the specifics of the vehicle, these elements may confuse the model and make it pair the image with images of similar landscapes rather than similar vehicles. If the image encoder had been more specialized for the task, it would likely be better at knowing what aspects of an image matter. As our task requires fine-grained classification, the lack of task-specific data during pre-training is likely to hinder its performance.

When given the example image in Figure 3.1 of a tank truck, an image of the exact same vehicle is only considered to be the fifth most similar reference image (Table 4.11). The four more similar images contain other green military-looking trucks. This may be caused by confusion by irrelevant elements in the image as mentioned above. For example, the lighting is so different between the query image and the real matching reference image that it might not be immediately obvious even to a human that the color of the truck is the same in the two images. However, the calculated similarities are nearly the same for all these images, suggesting that the most important issue is that the image encoder simply struggles to separate the fine-grained classes. As can be seen from the labels of the identified matches, the model does not seem to give very high importance to the number of axles. In contrast, it seems to be better at recognizing the truck's manufacturer (Kamaz), as confirmed by the very high confidence (79.5 %) in Table 4.10. Manual inspection indicates that this may be due to the distinct shape of the cabins of the

[1]In practice, the clustering algorithm may still combine some reference samples into one prototype sample.

trucks in these images, which seems to be unique to this manufacturer in our dataset and is easily recognizable by a human.

5.3 Model comparison

It is interesting that while the various models achieve very different F_2 scores, there is a striking number of models in a narrow range between 54.8 % and 56.8 %. This includes the neighbor search model, the simple neighbor net model without prototype samples and with an optimized number of prototype samples, and all fine-tuned neighbor net models. This raises the question of whether there is a limit to how well a model can perform on this dataset, perhaps due to the inconsistent labels, or if the limitation lies in a part of the model design that is common to the different models. If the limit lies in the dataset, steps could be taken to make it more consistent, as suggested in section 6.3.

Mean squared error (MSE) is the metric we optimize during fine-tuning, as it is differentiable, as opposed to the F_2 score. Note that of the models in Table 4.8, the class-wise neighbor net is the only model that has been fine-tuned. Even though it has been trained directly for minimizing this metric, it does not have the lowest MSE. The much higher MSE of the language-image model is a consequence of its confidence scores not using the entire [0, 1] range, as mentioned in subsection 3.2.1.

The 1-NN model has the highest precision of all models, 9.8 percentage points higher than that of the neighbor search model. Note however that the recall is 20.6 percentage points lower, and we have weighted this twice as high as precision through the F_2 score. Unlike the neighbor search model, the 1-NN model does not have a threshold to adjust to make the optimal tradeoff between recall and precision. It is thus unable to adapt to the difference in importance between these two metrics. It is likely that if the recall and precision were weighted the same when choosing the threshold, the neighbor search model would be able to outperform the 1-NN model in both metrics.

The accuracy of all image-based methods is above 98 %, demonstrating that the dataset is very imbalanced, as explained in section 2.1.6. As mentioned in section 4.3, the prevalence is 1.16 %, meaning that 98.84 % of the labels are negative. A model could then label all samples as negative and still get an accuracy of more than 98 %. The differences in subset accuracy are more pronounced. Notably, the subset accuracy of the 1-NN model is quite high, and considerably higher than that of any other model. This suggests that there are connections between the classes that this model is particularly good at recognizing. It may be that by looking only at the most similar reference image, this model has a higher chance of getting lucky and finding a very similar image with all the same keywords, and that this information is lost in the noise when factoring in more reference images.

Also when looking at the ROC AUC, the image-based models perform considerably better than the language-image model. It is interesting that the simple neighbor net model has a higher ROC AUC than the neighbor search model despite having a lower F_2 score. This indicates that the simple neighbor net is less sensitive to the choice of threshold than the neighbor search. The ROC AUC for the 1-NN model is less interesting because it is impossible to adjust the sensitivity of it other than setting the threshold to 0 (and predicting all samples as positive) or above 1 (and predicting all samples as negative).

Looking at the class-specific ROC AUC scores of Table 4.9 and what classes are handled best by the language-image and neighbor search models, we see some similarities and some differences. The language-image model seems to be at its weakest when given acronym class names while handling more common object names like “Attack helicopter” and “Passenger aircraft” better.

The neighbor search model also performs best with the same kind of common objects, but does the worst with less common objects such as “Self-propelled mortar” and “Armoured medical evacuation vehicle”.

The difference in the models’ weak classes is likely due to the language-image model’s reliance on the class names. While the neighbor search model uses the image encoder of CLIP, the language-image model requires both the text encoder and image encoder to be familiar with the concept to recognize. When the class name is obscure, either due to uncommon words or unknown concepts, it is unable to recognize the class well. The similarity in the models’ strengths is interesting. It may indicate a correlation between concepts that are commonly depicted in images online, and concepts that we can easily describe with common words, and thus have simple, yet accurate class names. This would be natural, considering that natural language has evolved to be efficient in day-to-day use. It can also indicate the simple fact that common concepts are more prevalent in CLIP’s pre-training dataset, and thus both its text and image encoders were rewarded during training for being able to recognize those. However, as our dataset comes from a quite specialized domain, the lacking performance with uncommon concepts impacts the overall performance negatively.

Although the metrics show a great divide in performance, it is interesting to note that, unlike the 1-NN model, all our own models returned all four ground-truth labels as suggestions for our example, as can be seen in Table 4.10. Still, the image-based models do this with drastically fewer suggestions. This is only one example, however, and we must take care to not draw grand conclusions from this alone. Looking at the models’ recall in Table 4.8 we see that the perfect recall in our example is indeed above average, with the best methods at around 66 %.

The neighbor search model has the advantage of being explainable. The reference images that it uses to create its output can be reported and displayed to the user along with their labels. This allows the user to consider these images and potentially decide that they are in fact irrelevant or even incorrectly labeled and thus judge whether the model’s label suggestions can be trusted. In fact, the user can choose to ignore the suggestions altogether and use the model only to look for relevant reference images. In that case, it may not be problematic if there are irrelevant reference images at the top of the list, as long as actually relevant images are there as well for the user to find. An example of such a case is the matches found using Figure 3.1 as query image, shown in Table 4.11, as discussed in section 5.2.

A disadvantage of the neighbor net models with fine-tuning is that we lose this explainability. Without fine-tuning, it is possible to make the neighbor net models still report exactly what reference images contributed to the result and by how much. With fine-tuning, the connection to the original reference images is lost. The weights that were initialized to the reference images’ representation vectors are no longer L2 normalized, and taking the dot product with a query image’s representation vector can no longer be interpreted as a cosine similarity. Instead of using the reference images directly, the model has now learned implicit knowledge that it is unable to explain to the user, and it has become a black box. In this regard, it becomes similar to neural networks trained from scratch.

With some modification[2], the language-image model has the advantage that it can be used for semantic search – using text descriptions to search for images. As with the neighbor search model as described above, the user can choose to ignore the model’s label suggestions and use the model only to look for relevant images and then inspect them manually.

[2]The modification would involve comparing one text string to many reference images instead of comparing many text strings to one query image.

Chapter 6

Conclusion and future work

6.1 Conclusion

We conclude by reviewing the research questions and seeing what our results tell us about them.

Research question 1 How have previous works approached tasks similar to ours?

In section 2.2 we have presented previous work on image classification problems with properties similar to our task. Work on fine-grained and on multi-label image classification highlights the usefulness of the transformer architecture for these tasks. Self-supervised methods exist that learn image representations well suited for classification. Of these, CLIP is quite suitable for our task. We have made use of an existing language-image classifier using CLIP's text and image encoders, adapting it for our task. We have also adopted an existing 1-NN approach for fine-grained multi-label classification using CLIP's image representations as a baseline model. A cache model has been shown to be effective in combination with CLIP's image representations, especially by implementing it as a neural network and fine-tuning it. We are inspired by this when developing our models.

Subquestion 2 a How well suited is the language-image zero-shot classification method of Radford et al. [2021] for our task?

A language-image zero-shot classifier adapted for multi-label classification did not perform well on our dataset. Using the original Norwegian labels, it performed very poorly. Translating the labels to English helped substantially, but not enough to bring the performance close to the image-based baseline 1-NN model from Conde and Turgutlu [2021].

Subquestion 2 b How well suited is CLIP's pre-trained image encoder for our task without using text?

The best method we have identified for utilizing CLIP's pre-trained image encoder to do multi-label classification on our dataset is a nearest-neighbor-inspired model, called the neighbor search model in this thesis. This performed better than the baseline 1-NN model.

Research question 2 How can the knowledge from CLIP's pre-training best be harnessed for our task?

The best method we have found to utilize CLIP's pre-training knowledge to perform multi-label classification on our dataset is our neighbor search model. This is a purely image-based model and does not use CLIP's text encoder. This performed better than zero-shot language-image classification and other image-based models we have tested.

6.2 Contributions

We have explored ways to use a self-supervised language-image pre-trained model, CLIP, as a basis for an image classifier. Specifically, we have built and evaluated such classifiers on a set of military-related images.

The task we have evaluated the classifiers on has a set of interesting properties. The images in the dataset are labeled in the form of text keywords, inviting the use of CLIP's cross-modal knowledge. The text labels are in Norwegian, allowing us to investigate the performance of CLIP's text encoder on Norwegian text compared to an English translation. The images belong to a quite specialized domain and are unlikely to have been well represented in CLIP's generic pre-training dataset. The task requires fine-grained image classification, which was found by Radford et al. [2021] to work well with CLIP zero-shot only in some cases. The task is a multi-label classification task, which none of the tasks tested by them were. The labels in the dataset are inconsistent, which can confuse models and disturb the evaluation of the models, but is a realistic scenario in use-cases where high-quality curated datasets for machine learning are not available.

We have built and evaluated several models based on CLIP's text and image encoders to perform this task. One model is a zero-shot classifier making use of text labels similar to the one created by Radford et al. [2021], adapted for multi-label classification. With this model, we investigate the effects of the language of the labels. We find that the language of the labels is important for the language-image classifier's performance, with English labels performing better than Norwegian labels, and manual, high-quality translation into English performing better than automatic translation. However, even with the manual translations, the performance of the language-image classifier is considerably worse than an image-based 1-nearest neighbor baseline. We have pointed to issues keeping the language-image classifier from coming close to the performance of the image-based classifiers. It especially struggles with names of companies and fine-grained distinctions such as three- versus four-axle vehicles, but even for common objects such as passenger planes, where it is at its best, it lags behind our other models.

We have also built and evaluated a variety of purely image-based models using CLIP's learned image representations directly. These models are inspired by k-nearest neighbors and cache models. Unlike the zero-shot classifier, they utilize reference images with known labels while still not requiring any training. We have studied the effects of various model design choices to find a well performing model. One of these design choices is whether to fine-tune parts of the model. We find that these image-based classifiers outperform the 1-NN baseline and perform drastically better than the the zero-shot language-image classifier. Of all models we have evaluated, the best performing one is a k-NN inspired search procedure that identifies the most similar images from a set of reference images and uses their labels weighted according to their similarity to the query image. Our attempts to improve this model by implementing it as a cache model neural network and fine-tuning it did not help the performance.

6.3 Future Work

In this work we have compared a zero-shot language-image model to image-based models with access to tens of thousands of training samples, concluding that the latter perform the best. In a situation where no training data was available, however, our image-based models would not work, and the language-image model would be better. An interesting question is at what amount of data the language-image model is outperformed by the best image-based model. This would give new insight into which situations would benefit the most from the different approaches.

As we have acknowledged, our dataset suffers from problems such as inconsistent labels. While we made some improvements to the dataset in the data preparation pipeline (subsection 3.1.2), further improvements can be done by manually correcting individual sample labels. Improving the dataset may help answer whether its inconsistencies are the reason for what seems like a limit in performance, as discussed in section 5.3. Inconsistencies can be identified by using our models to rank the samples by loss. In this way, one can focus the manual inspection on the samples with a high loss, working on the assumption that these are more likely to deviate from the majority of the training set. Similarly, the language-image model can be used to identify which class names are the most unhelpful to it, as indicated by a low F_2 score or ROC AUC for the individual class. These class names can then be manually changed to something that is believed to be more helpful, the evaluation can be run again and the process repeated for as long as desired. Our manual class name translation was a way to improve the quality of the data for use by that model, and we have identified some poorly performing class names, but we have not done systematic improvement of specific class names based on their performance.

We suspect that a straightforward, but effective way to improve all of our models is to implement class-wise thresholds. Our models all have a single threshold across all classes that decides how high a class confidence score must be before the corresponding class is included as a suggested label. By optimizing a threshold for each class instead, we believe that the models will be better able to handle the large differences between the classes, for example in how specific they are. This may especially help the language-image model, which has very limited information about each class. For the fine-tuned neighbor net models, a similar effect could likely be achieved by instead adding a trainable bias to the final layer.

We have tried optimizing the number of prototype samples in the simple neighbor net model without fine-tuning. Doing this with fine-tuning as well would be a natural continuation of this work. It could identify whether 16 prototype samples are too few or too many to avoid overfitting while keeping enough model parameters to successfully learn the task.

We have found that the simple neighbor net model without prototype samples and without fine-tuning performs slightly worse than the neighbor search model, despite the two being nearly equivalent. Since the neighbor search model is the best performing model, it would be interesting to see if it could be improved by fine-tuning. This could be done by adding a filter to the simple neighbor net such that it is completely equivalent to the neighbor search model, before fine-tuning.

Although the language-image model is clearly outperformed by the image-based models, it has access to information that the image-based models do not have: class names. A single model that makes use of this textual information in addition to the reference images could get the best of both worlds and perform better than a purely image-based model. This could be achieved by ensembling the existing models or designing a new model that uses both the class names and the reference images.

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Bynke, M. (2021). Preparing noisy multi-labeled image datasets for evaluating language-image models. Specialization project. Norwegian University of Science and Technology (NTNU).

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020b). Big self-supervised models are strong semi-supervised learners.

Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., and Zheng, Y.-T. (July 8-10, 2009). Nus-wide: A real-world web image database from national university of singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece.

Conde, M. V. and Turgutlu, K. (2021). Clip-art: Contrastive pre-training for fine-grained art classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3956–3960.

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Gao, P., Geng, S., Zhang, R., Ma, T., Fang, R., Zhang, Y., Li, H., and Qiao, Y. (2021). Clip-adapter: Better vision-language models with feature adapters. *arXiv preprint arXiv:2110.04544*.

Grave, E., Joulin, A., and Usunier, N. (2016). Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

He, J., Chen, J.-N., Liu, S., Kortylewski, A., Yang, C., Bai, Y., and Wang, C. (2021). Transfg: A transformer architecture for fine-grained recognition.

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

Head, T., Kumar, M., Nahrstaedt, H., Louppe, G., and Shcherbatyi, I. (2021). scikit-optimize.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Jia, C., Yang, Y., Xia, Y., Chen, Y.-T., Parekh, Z., Pham, H., Le, Q., Sung, Y.-H., Li, Z., and Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916. PMLR.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Applying and improving alphafold at casp14. *Proteins: Structure, Function, and Bioinformatics*, 89(12):1711–1721.

Khosla, A., Jayadevaprakash, N., Yao, B., and Fei-Fei, L. (2011). Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and development team, J. (2016). Jupyter notebooks - a publishing format for reproducible computational workflows. In Loizides, F. and Scmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, Netherlands. IOS Press.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

Kuhn, R. (1988). Speech recognition and the frequency of recently used words: A modified markov model for natural language. In *Coling Budapest 1988 Volume 1: International Conference on Computational Linguistics*.

Kuhn, R. and De Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 12(6):570–583.

Li, J., Li, D., Xiong, C., and Hoi, S. (2022a). Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. *arXiv preprint arXiv:2201.12086*.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., et al. (2022b). Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Liu, S., Zhang, L., Yang, X., Su, H., and Zhu, J. (2021). Query2label: A simple transformer way to multi-label classification.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Orhan, E. (2018). A simple cache model for image recognition. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pham, H., Dai, Z., Ghiasi, G., Kawaguchi, K., Liu, H., Yu, A. W., Yu, J., Chen, Y.-T., Luong, M.-T., Wu, Y., Tan, M., and Le, Q. V. (2021). Combined scaling for open-vocabulary image classification. *arXiv preprint arXiv:2111.10050*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

Russel, S. J. and Norvig, P. (2010). *Artificial intelligence: a modern approach.* Pearson Education, Inc., Upper Saddle River, New Jersey.

Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*. Cambridge University Press.

Sculley, D. (2010). Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Singh, A., Hu, R., Goswami, V., Couairon, G., Galuba, W., Rohrbach, M., and Kiela, D. (2021). Flava: A foundational language and vision alignment model. *arXiv preprint arXiv:2112.04482*.

Streiner, D. L. and Cairney, J. (2007). What's under the roc? an introduction to receiver operating characteristics curves. *The Canadian Journal of Psychiatry*, 52(2):121–128.

The pandas development team (2021). pandas-dev/pandas: Pandas 1.3.3.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The caltech-ucsd birds-200-2011 dataset.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37.

Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., and Wu, Y. (2022). Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917*.

Yuan, L., Chen, D., Chen, Y.-L., Codella, N., Dai, X., Gao, J., Hu, H., Huang, X., Li, B., Li, C., et al. (2021). Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*.

Zhai, X., Wang, X., Mustafa, B., Steiner, A., Keysers, D., Kolesnikov, A., and Beyer, L. (2021). Lit: Zero-shot transfer with locked-image text tuning. *arXiv preprint arXiv:2111.07991*.

Zhang, R., Fang, R., Gao, P., Zhang, W., Li, K., Dai, J., Qiao, Y., and Li, H. (2021). Tip-adapter: Training-free clip-adapter for better vision-language modeling. *arXiv preprint arXiv:2111.03930*.

Zhou, K., Yang, J., Loy, C. C., and Liu, Z. (2021). Learning to prompt for vision-language models. *arXiv preprint arXiv:2109.01134*.

Appendix A

Software

This is a list of software we use and its version numbers.

Python 3.8.10.

PyTorch 1.9.1 [Paszke et al., 2019]. For implementing all models.

CLIP 1.0 [Radford et al., 2021]. For loading the CLIP model and using its image and text encoders.

Scikit-learn 1.0 [Pedregosa et al., 2011]. Calculating metrics, clustering samples and dividing the dataset into train, validation and test sets.

Skopt/Scikit-optimize 0.9.0 [Head et al., 2021]. Bayesian hyperparameter optimization.

Deep-translator 1.6.1. Automatically translating class names from Norwegian to English.

NumPy 1.21.2 [Harris et al., 2020]. Matrix and vector manipulation for implementing models and calculating metrics.

Pandas 1.3.3 [The pandas development team, 2021]. Storing and handling the dataset and its samples.

Matplotlib 3.4.3 [Hunter, 2007]. For plotting figures.

h5py 3.4.0. Storing and retrieving pre-calculated image features.

Jupyter [Kluyver et al., 2016]. Running Python code in notebooks. It consists of the following packages: IPython (7.27.0); ipykernel (6.4.1); ipywidgets (7.6.5); jupyter_client (7.0.3); jupyter_core (4.8.1); nbclient (0.5.4); nbconvert (6.1.0); nbformat (5.1.3); notebook (6.4.4); qtconsole (5.1.1); traitlets (5.1.0).