UNIVERSITY
OF OSLO

Master's thesis

Achieving Data-efficient Neural Networks with Hybrid Concept-based Models

Tobias Aanderaa Opsahl

Data Science, Specialisation Statistics and Machine Learning
60 ECTS study points

Department of Mathematics
Faculty of Mathematics and Natural Sciences

Autumn 2023

Tobias Aanderaa Opsahl

Achieving Data-efficient Neural Networks with Hybrid Concept-based Models

Supervisors:
Vegard Antun
Riccardo de Bin

Abstract

Most datasets used for machine learning consist of a single label per data point, which is used to optimise the model. However, in cases where more information than just the class label is available, would it be possible to train models more efficiently? We introduce two novel neural network architectures that train with both class labels and additional information in the dataset, referred to as *concepts*. We call these models *hybrid concept-based models*, since they use both concept predictions and information not interfering with the concepts to predict the class. In order to thoroughly explore their performance, we introduce ConceptShapes, an open and flexible class of datasets with concept labels. Through various experiments, we show that the hybrid concept-based models outperform standard computer vision models and previously proposed concept-based models with respect to performance, especially in sparse data settings. We also introduce an algorithm for performing *adversarial concept attacks*, where an image is perturbed in a way that does not change a concept-based model's concept predictions, but changes the class prediction. We argue that this puts the interpretable qualities promised from previously proposed concept-based models into question.

Contents

Chapter 1

Introduction

Understanding model behaviour is a crucial challenge in deep learning (DL) and artificial intelligence (AI) [1]–[4]. DL models are inherently chaotic, and give little to no insight into why a prediction was made. In order to use AI in high risk settings [5] and to get insight into the shortcomings of current models [6]–[9], it has been argued that understanding model behaviour is critical. In computer vision, early attempts for explaining a model's prediction consisted of assigning pixel-wise feature importance [10]–[13]. Even though these methods gained popularity and were visually appealing, they have been shown to perform a poor job at actually explaining model behaviour [2], [3], [14]–[18]. A more recent approach is to globally inspect if a model has learned human understandable concepts related to the predictions [19]–[21]. Concepts can also be used to attempt to construct models that are inherently interpretable [22]–[27]. However, despite many proposals for interpretable DL models, various experiments show that their interpretable qualities may not hold [28]–[30].

We define *concepts* in the context of machine learning (ML) as human meaningful features in the data that is different from the main target we try to predict. For a computer vision model that classifies bird species, a concept may be if the bird has a small size or if it has orange eyes [23], [31] (see Figure 1.1). For a chess computer that predicts the next best move given a board position, a concept may be whether or not the player to move is in check, or if they have a material advantage [21].

Datasets labelled with concepts have been used to attempt to make interpretable models [23]–[27]. The main feature of these models is that they are first used to predict the concepts, and then the concept predictions are used to predict the label (see Figure 1.2). The target prediction is made solely from the concept predictions. This way, one can interpret why a target prediction was made by inspecting the concept predictions.

Although such concept-based models intuitively seem interpretable, experiments have shown that this may not be the case in practice [28]–[30]. The main limitation for interpretability is that the concept predictions encode more information than just the concepts, referred to as *concept leakage*. In tasks where the concepts are uncorrelated and useless to predict the target, using the concept predictions may achieve substantially better performance than an oracle using the true concept labels [28], [29]. It has also been shown that concept-based models are susceptible to *adversarial attacks* [30]. One can find images that look identical, have the same target prediction, but different concept predictions. As one of the contributions in this thesis, we will add evidence to their lack of interpretability. We propose an algorithm that perturb an image so that it looks identical to the original image, the concept predictions are unchanged, but makes the

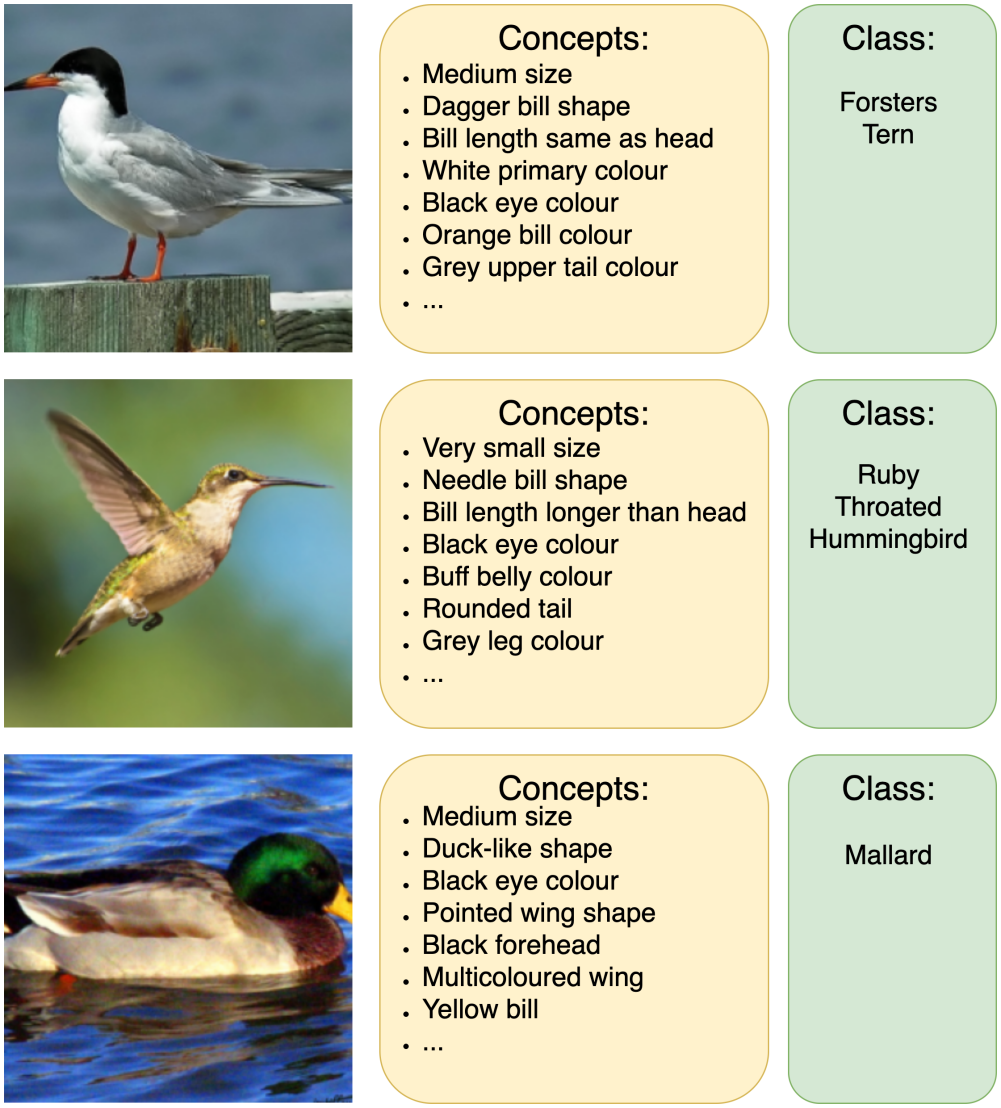


Figure 1.1: **Example of concepts**. Images are labelled both with concepts and classes. Images are taken from the Caltech-USCD Birds-200-2011 (CUB) dataset [31].

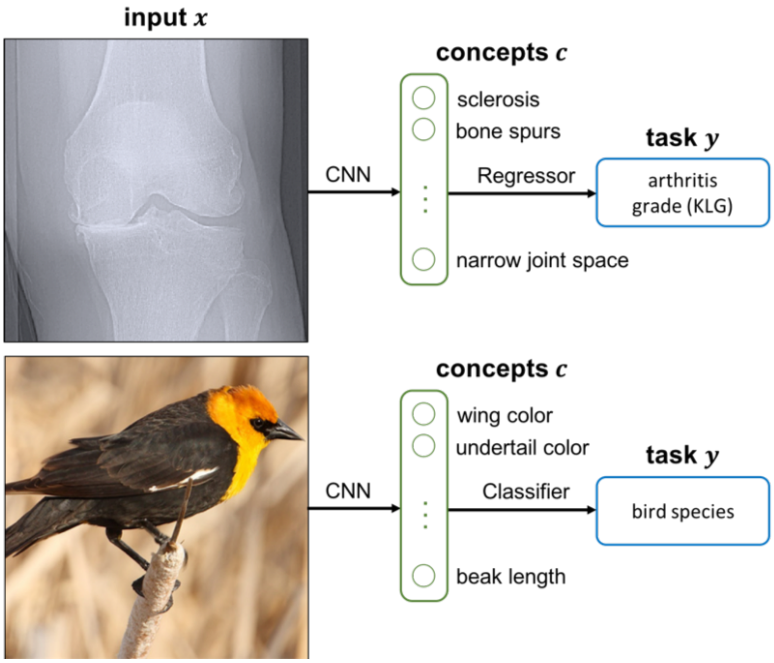


Figure 1.2: **Overview of a concept model**. The model first predicts the concepts in the dataset, and then uses those predictions to predict the target. This is figure 1 from [23].

target prediction change (see Figure 1.3).

1.1 The Issues with Data Hungry Algorithms

We will deviate from the goal of interpretability and use the framework of concept-based models to create models that tackle another crucial challenge in AI, namely large dataset requirements. ML models require vast dataset sizes in order to reach good performance. This results in large computational requirements [32], high energy usage [33], [34], and challenges in gathering the data [35], [36].

The need for a large dataset also demonstrates that modern AI learns differently than humans and animals. For instance, large language models (LLMs) fail on fundamental reasoning and planning [37]. This is expressed in a recent quote from Turing award laureate Yann LeCun [9]:

> '*Animals and humans get very smart very quickly with vastly smaller amounts of training data than current AI systems. Current LLMs are trained on text data that would take 20,000 years for a human to read. And still, they haven't learned that if A is the same as B, then B is the same as A. ... My money is on new architectures that would learn as efficiently as animals and humans.*'

Furthermore, gathering the datasets is a crucial bottleneck for many applications. In tasks like medical image processing, gathering data may require specialised equipment, domain expert knowledge, many human patients and carefully designed experiments. As it is put in *Deep Learning for Medical Image Processing: Overview, Challenges and the Future* [36]:

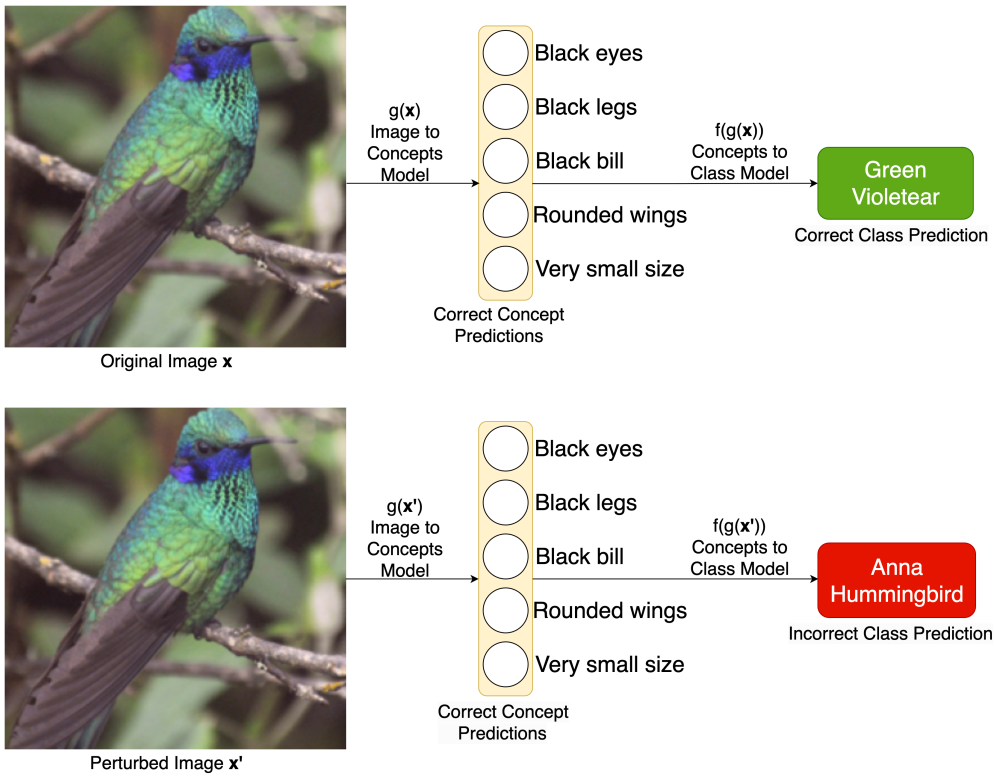


Figure 1.3: **Adversarial Concept Attack.** Images are perturbed in a way that does not change a concept bottleneck model's (CBM) [23] concept predictions, but change the class prediction. We suggest that this questions the interpretable qualities of CBMs.

> '*Deep learning requires immense training datasets to establish the accuracy of deep learning classifiers; the lack [of] suitable datasets is one of the biggest barriers to the success of deep learning in medical imaging.* '

and in *Advances in Deep Learning-Based Medical Image Analysis* [38]:

> '*Although deep learning models have achieved great success in medical image analysis, small-scale medical datasets are still the main bottleneck in this field.*'

It is easy to imagine problems with dataset sizes in other areas as well, such as satellite and aerial imaging, wildlife monitoring, microscopic imaging, underwater imaging and industrial inspection. Even in applications where gathering training data does not require immediate expenses, datasets may limit model performance. Consider for example the task of predicting housing prices. Datasets may consist of previous house sales in the region. Although collecting the existing data might not be expensive, the amount of existing house sales in a region might not be large enough to train accurate ML models. Additionally, researchers might in some cases be limited to predefined datasets, without being able to collect more data.

Motivated by the arguments presented above, we propose the following question to investigate in this thesis:

> *How can we make high performing ML algorithms that require less data?*

1.2 Our Contributions

We tackle the problem of large dataset requirements in computer vision by proposing new models that train with both concept labels and target labels. The models do not receive the concept labels as inputs, but uses them to train concept predictions in hidden layers, which is then used to predict the class. In order to evaluate their performance, we have also developed a new class of concept datasets (datasets with concept labels), as one of the contributions in this thesis.

Our models differ from existing concept-based models in that they are motivated by performance, not interpretability. Therefore, we construct the models without the limitation of making the final target prediction solely from the concept predictions. Our models use both concept predictions and information that has not interfered with the concepts in order to make the target prediction. An overview of one of the architectures can be seen in Figure 1.4.

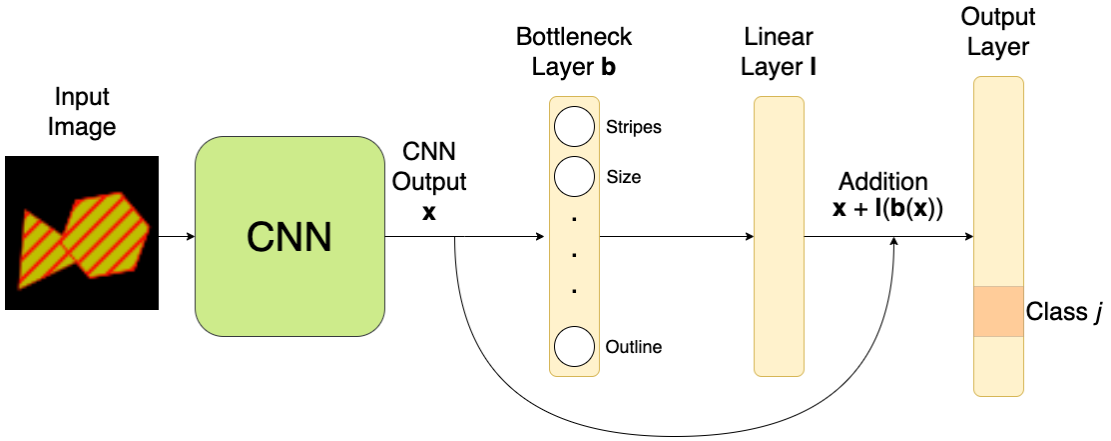


Figure 1.4: Architecture of a **Concept Bottleneck Model with Residual Connection (CBM-Res)**, one of the novel hybrid concept-based models proposed. The intermediary bottleneck layer predicts the concepts in the dataset. The final class prediction is made both with the concept predictions and with information just going through a normal convolutional neural network (CNN). This differs from earlier proposed concept-based models, which only use the concept predictions to predict the target class. The difference is rooted in that our models are motivated by performance, not interpretability.

By using the concept labels to add additional information to the dataset, one may perform better on a given dataset, or save expenses by requiring less data. For instance, assume a group of researchers wish to further develop models on a previously proposed medical dataset, but the amount of data was not sufficient to train accurate ML models. The researchers may not be able to gather more data, since that would require a new comprehensive study. However, they might have sufficient domain expertise to add additional labels to the images. This could give the model sufficient extra information to become satisfactorily accurate. If adding concept labels is cheaper than creating more training data, one might save time and money by using concept-based models.

In some cases, concept labels are already available, or can be easily inferred. In the context of developing a chess computer, it is trivial to calculate concepts such as material balance and opponents threats for a given game position. Reinforcement learning algorithms that operate in physical space may use orientation, velocity and coordinates as concepts, and use the concept predictions to predict the next best action.

We propose models that are able to efficiently train with the information given by the concept labels.

Our models are compatible with existing methods for training with small datasets. Instead of training an ML model from scratch, one can start with a large pre-trained model. The last layer of the pre-trained model can be changed and trained on the prediction task of interest. This is referred to as *transfer learning*. One can also augment the training data in ways one might expect data outside the given dataset to look like, called *data augmentation*. In computer vision, this may consist of horizontally flipping an image, slightly altering the colours or cropping it. Both transfer learning and data augmentation are popular methods in DL, and in the experiments we will use both of them combined with our proposed models.

In addition to proposing our novel concept-based model architectures and concept datasets, we explore limitations of interpretability in current concept-based models. We develop an algorithm that, given an image and a concept-based model, produces an identically looking image with the exact same concept predictions, but with different class predictions (see Figure 1.3). We refer to this as *adversarial concept attacks*. This is a different approach than in [30], which changes the concept predictions, but keeps the class predictions the same. Since our method demonstrates that identical concept predictions on almost similar images can produce completely different output, we argue that this further questions if concept-based models can be considered interpretable.

Our contributions are as follows:

- **Novel model architectures:** We propose novel DL architectures that use concept labels in addition to the class labels to train. Unlike previously proposed concept models that were created with the purpose of being interpretable, ours are motivated by achieving better performance when the amount of data is limited. We refer to the models as *hybrid concept-based models*, since they use concept predictions in addition to information not related to concepts to predict the class. To the best of our knowledge, these are the first concept-based models that are motivated by performance.

- **New flexible concept datasets:** We argue that datasets that are currently being used to benchmark concept-based models have limitations or are inaccessible, and therefore present a class of openly available synthetic datasets with concept labels called *ConceptShapes*. The framework for creating the datasets is generic, so the user can control the difficulty by the amount of classes, and decide the correlation between the classes and the concepts.

- **Experimental results:** We show that the novel models can outperform both previously proposed concept-models and standard convolutional neural networks (CNN) with respect to test-set accuracy, especially when the amount of training data is low. This is done on both a wide variety of the ConceptShapes datasets, in addition to the popular CUB dataset [31].

- **Adversarial Concept Attacks:** We propose an algorithm for generating adversarial examples for concept models. The adversarial examples look identical to the original images, produce the exact same concept predictions, but change the class prediction. We argue that this makes the interpretable qualities in existing concept-based models even more questionable.

1.3 Thesis Structure

The structure of the thesis is as follows: First, we cover the background of relevant topics in AI in Chapter 2, 3 and 4. We start by giving a brief overview of the DL framework in Chapter 2. We describe adversarial attacks, in order to present our own adversarial algorithm later. Because previous concept-based models have been motivated by understanding model behaviour, we explore why this is an important issue and discuss explainable artificial intelligence (XAI) and interpretability. We then give an overview of an existing class of popular methods for explaining model behaviour by making saliency maps in Chapter 3, along with comprehensive evidence for their lack of explainable power. Chapter 4 covers concept-based explanations and some interpretable models based on them, along with their shortcomings.

After the background, we continue with our contributions. Chapter 5 provides an overview of the issues in current concept datasets, and a comprehensive overview of the synthetic concept datasets *ConceptShapes* that we propose. Novel concept-based model architectures are covered in Chapter 6. We present our algorithm for performing *adversarial concept attacks*, along with corresponding experimental results and discussion in Chapter 7. We describe the experimental setup for evaluating the hybrid concept-based models' performance in Chapter 8, with results in Chapter 9. Chapter 10 summarises the results and discusses possible directions for future work.

Chapter 2

Deep Learning

2.1 The Deep Learning Framework

DL methods started the AI revolution when *AlexNet* won the Imagenet competition in 2012 [39]. Soon after, DL began to dominate areas such as speech recognition [40], [41], game intelligence [42], [43] and natural language processing [37], [41]. Deep learning consists of automatically learning multiple representations of input data, usually by utilising neural networks with many hidden layers.

In this introduction we consider the framework of supervised machine learning on a classification task. We are given N data points, $\{\mathbf{x}_i\}_{i=1,...,N}$, along with corresponding class labels $\{y_i\}_{i=1,...N}$. The data is d dimensional, and we have p classes, meaning each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, ..., p\}$ for every i. Given $y_i = j$, we write the *one-hot-encoding* as $\mathbf{y}_i = [0, 0, \ldots, 1, \ldots, 0] \in \mathbb{R}^p$, a p dimensional vector with zeros at every element, except input j, which is 1.

In *computer vision*, where the input data is images, d is the number of pixels. Our goal is to find a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^p$ that will accurately predict which class an image belongs to. Here θ is the vector of parameters for the model f_θ, which we will sometimes omit for simplicity and write f instead. In practice, f_θ will be a deep neural network.

The output of the model is denoted by $\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$. It is a p dimensional probability vector, where all entries lie in the interval [0,1], and the sum of them is 1. In order to achieve the probability vector, one can use the *softmax* function, defined as $s(\mathbf{z})_j = e^{z_j}/(\sum_{r=1}^{p} e^{z_r})$ for each $j \in [1, ..., p]$. The values before they are passed through the softmax function are called *logit* values.

We want f_θ to generalise beyond the dataset which is used to train the model and find θ. In order to do that, it is customary to partition our dataset in a *training set*, *validation set* and *test set*. We let $I_{\text{train}}, I_{\text{val}}, I_{\text{test}} \subset \{1, 2, \ldots, N\}$ be the sets of indices to the data points that belongs to the training set, validation set and test set, respectively. We use the training set to train models and find the parameters θ, and chose the best model according to how well they perform on the validation set. Finally, we report the performance on the test-set, which has not been used to train or select the models. The performance can be measured by the *accuracy*, $\frac{1}{|I_{\text{test}}|} \sum_{i \in I_{\text{test}}} \mathbb{I}(\text{argmax}(\hat{\mathbf{y}}_i) = y_i)$, where where $\text{argmax}(\hat{\mathbf{y}}_i)$ returns the index of the element in $\hat{\mathbf{y}}_i$ with the largest scalar value, and $\mathbb{I}$ is the indicator function, defined as

$$\mathbb{I}(x = y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

In order to find suitable parameters for our model, we chose a *loss function* $L : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ that we want to minimise. Its inputs are the predicted value $\hat{\mathbf{y}}_i = f_\theta(\mathbf{x}_i)$ and the one-hot-encoded true label $\mathbf{y}_i$. The loss function is low when the model assigns a high probability to the true class, and high otherwise. When using the softmax function to achieve $\hat{\mathbf{y}}_i$, it is computationally efficient to also use the *categorical cross entropy* loss function, defined as:

$$L(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\sum_{j=1}^{p} y_{i,j} \log(\hat{y}_{i,j}) \tag{2.2}$$

where $y_{i,j}$ and $\hat{y}_{i,j}$ are scalar values corresponding to the j'th elements of the vectors $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$. Note that exactly one element in the sum will be non-zero, since $\mathbf{y}_i$ is the one-hot-encoding of y_i.

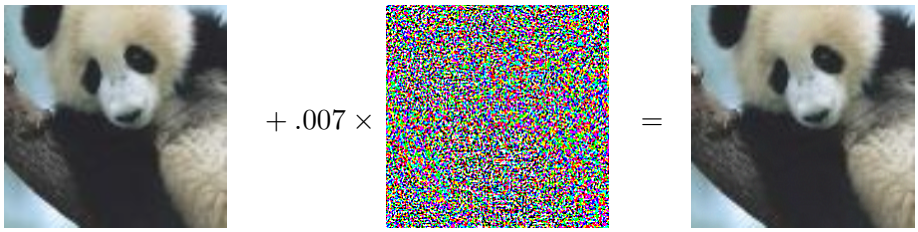
We use *stochastic gradient descent* (SGD) to iteratively update the parameters θ. Let $\{D_1, D_2, \ldots, D_M\}$ be a partition of the training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in I_{\text{train}}}$. We refer to each D_m as a *minibatch*, and let the minibatches be of approximately equal sizes. For each minibatch D_m, we calculate the average gradient of the loss function with respect to the parameters. The gradient is calculated with *backropagation* [44]. Then, the parameters are updated in the direction of the negative gradient, multiplied with a *learning rate* η. For a minibatch D_m at step t, the equation for calculating θ_{t+1}, the parameters at the next step, then becomes:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{|D_m|} \nabla_{\theta_t} \sum_{(\mathbf{x},\mathbf{y}) \in D_m} L(f_{\theta_t}(\mathbf{x}), \mathbf{y})$$

In practice, we often use a variation of SGD. Instead of directly going in the direction of the current gradient, one can use *momentum* to average over previous gradients [45]. The method *RMS-prop* [46] divides the learning rates for parameters elementwise by an exponential running average of the squared gradient for that parameter. Combining momentum and RMS-prop, in addition to bias correcting, yields the popular optimizer *Adam* [47].

In order for deep learning models to work well, various other methods are also used. We use the rest of this section to describe some of them. The layers in the network are passed through an *activation function*, which achieves a non-linear relationship between the input and output. The most popular activation function is *ReLU*, defined as $ReLU(x) = \max(x, 0)$. The *sigmoid* function $\sigma(x) = \frac{1}{1+e^{-x}}$ may also be used. The activation functions are applied elementwise.

In computer vision, the images are often processed and augmented before being passed through the network. The raw pixel values consist of a three dimensional vector, where every element is in $\{0, 1, ..., 255\}$. The three values represent the red, green and blue channels of the image, and the number represents the colour intensity. The values are often scaled down to $[0, 1]$, and then normalised with respect to each channel's average pixel value in the dataset. Data augmentation, like random cropping or flipping, may be applied to the training images. This artificially emulates more training data. Data augmentation can reduce *overfitting*, which is when the validation-loss starts to increase during training, despite the training-loss further decreasing. Another popular regularisation technique is *dropout* [48], which randomly sets elements from hidden layer outputs to 0 during training.



$\mathbf{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\mathbf{x}} L(f_\theta(\mathbf{x}), \mathbf{y}))$
"nematode"
8.2% confidence

$\mathbf{x} + \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} L(f_\theta(\mathbf{x}), \mathbf{y}))$
"gibbon"
99.3 % confidence

Figure 2.1: Demonstration of adversarial attack with fast gradient sign method (FGSM), performed on GoogLeNet [56] trained on imagenet [57]. **Left:** The original test image of a panda. **Middle:** The signed loss of the gradient with respect to the image. **Right:** The perturbed image, with $\alpha = 0.007$. The neural network predicts *gibbon* for the perturbed image, even though it still resembles a panda. This is Figure 1 from [54].

2.2 Adversarial Attacks

Soon after the DL revolution started, it was discovered that it was possible to easily fool the models by producing slightly perturbed inputs that resulted in vastly different outputs. This was coined as *adversarial attacks* [6]. For computer vision models, one can produce images that are only slightly changed, and look identical to humans. However, they could still produce completely different predictions from a DL model. This is illustrated in Figure 2.1.

The existence of adversarial examples limits AI's trustworthiness [49], and the presence of such examples have been documented in many domains. This includes image reconstruction [8], image classification [50], voice recognition [51], natural language processing [52] and tools for automating diagnosis in medicine [53].

Many methods for obtaining adversarial examples have been proposed. The *fast gradient sign method* (FGSM) was introduced as a fast and reliable algorithm to produce adversarial examples [54]. It differed from the original method that was computationally expensive [6]. Given a neural network f_θ, image $\mathbf{x}$, true label y and loss function L, the method alters $\mathbf{x}$ by adding $\alpha \cdot \text{sign}(\nabla_{\mathbf{x}} L(f_\theta(\mathbf{x}), \mathbf{y}))$, where α is an adjustable step size. This means that it takes a single step in the direction where the loss for the true class increases the most.

The FGSM proved that DL models were highly susceptible to adversarial attacks that were easy to generate, but did not always produce images that looked similar to the original. This is because it takes one big step in the direction of the gradient of the loss, which is usually a highly non-linear function. It was later generalised to the *basic iterative method* [55], which takes many small steps with updated gradient calculations. At step t, perturbed image number $t+1$ is generated by $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha \cdot \text{sign}(\nabla_{\mathbf{x}_t} L(f_\theta(\mathbf{x}_t), \mathbf{y}))$, with $\mathbf{x}_0 = \mathbf{x}$. Additionally, the images are clamped to be within the range of possible pixel values. This method more consistently generates adversarial examples that only change the original image slightly.

Further generalisations consist of targeting the attacks to a specific class, and projecting the perturbed images close to the original images. Instead of simply misclassifying, one can decrease the loss of a target class $y_k \neq y$, with $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \cdot \text{sign}(\nabla_{\mathbf{x}_t} L(f_\theta(\mathbf{x}_t), \mathbf{y}_k))$. It is also possible to increase the raw class logit output for

y_k, instead of minimising the signed loss. An additional improvement was with *projected gradient descent* (PGD) [58]. In addition to the iterative method, every image $\mathbf{x}_t$ is projected down on an epsilon ball around $\mathbf{x}$, such that the perturbed images do not move away from the original input image with more than epsilon. One can do this with respect to any distance metric, but one often uses the ℓ_2 or ℓ_∞ norm. Another popular method for generating adversarial examples is *DeepFool* [59], which iteratively updates images to move them closer to the model's decision boundary.

The reason for DL methods being susceptible to adversarial attacks, referred to as being *unstable*, partly stems from the high dimensionality of the data. Computer vision inputs often have hundreds of thousands of dimensions, which makes it easy to find a direction that increases the loss of the true class rapidly [54]. A small change in many of the input features can result in a big change overall, since the number of input features is high. It also means that there is no way to obtain a dataset that densely covers a grid of possible input images. It is argued that restricted models are easier to fool than complex ones, due to them not being able to make sufficiently flexible decision boundaries [54], [58]. Paradoxically, there exist theoretical results showing that stable and accurate neural networks do exist for some problems, but training algorithms are not able to find them [60].

The existence of adversarial examples prove that DL models do not learn what they are intended to do. If a computer vision model truly understood the contents a human sees in an image, it would not be fooled by very similar looking images. It is conjectured that the models learn *false structures* instead of the intended task [7], [61]. Since the DL models work arithmetically with the raw pixel values, the false structures may be any structure in the pixels that correlates well with the intended task, thus allowing for high accuracy. However, the false structure is not stable with respect to small perturbations in the pixel values, yielding the unstable results.

2.3 Explainable Artificial Intelligence

DL models give no valuable insight into *why* a prediction was made, and are therefore referred to as *black boxes*. One can give the models input and observe the outputs, but not decipher the process of what happened in between. This has led to a great demand from policy makers and users of the models to develop methods that give insight and transparency into the decision making processes.

For instance, consider medical imaging. If a model only predicts the presence of a disease, without giving any reason for why, it may not convince medical experts, especially if they do not agree with the conclusion [35], [36]. Regarding law, transparency in the model behaviour is necessary to make sure it does not act biassed and unfairly with respect to gender and ethnicity [3], [62].

Additionally, understanding how models work, and why they may perform poorly, is a useful step towards improving them. Although DL models often give impressive accuracy, this comes at the cost of *non-human* behaviour, which often gives unsatisfactory performance. As discussed in the previous section, they are sensitive to small perturbations in the input that humans do not notice. This suggests that they learn different structures than intended. They also require vast amounts of training data and computational power [32], [63]. If one had a better understanding of how the models work, then it would be easier to work with their limitations.

Furthermore, in high-risk applications, the use of DL will be restricted. The European Union newly developed a legal framework to regulate AI based on risk-level

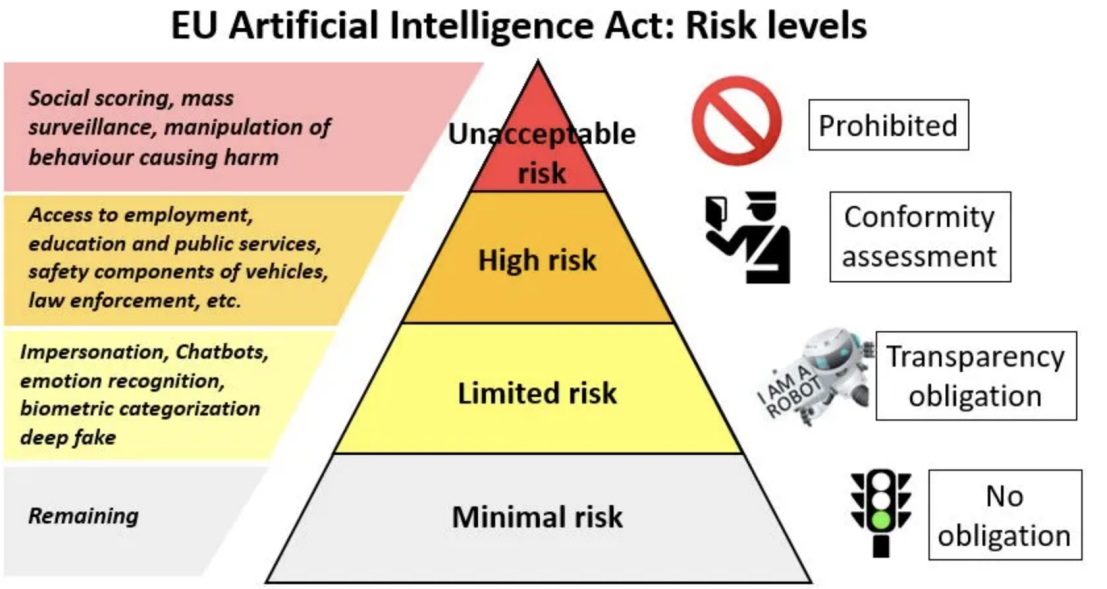


Figure 2.2: High-level overview of EU's Artificial Intelligence Act. The regulations increase with the risk of the applications. The image is taken from [64].

[5], [65]. If the application is deemed to be of higher risk, like in medicine and law, more restrictions will apply (see Figure 2.2). Model transparency is one of the requirements, which will not be possible without being able to explain the models' behaviour.

The approaches to understanding model behaviour are mainly divided into *explainability* and *interpretability*. Explainable Artificial Intelligence (XAI) aims to do post-hoc analyses of pre-trained models in order to understand their behaviour. A common strategy consists of ranking the importance of individual input features. Feature importances can be quantified with *Shapley values* [66], while *saliency maps* [10] may determine the importance of pixels for computer vision models. Another approach is through concept-based explanations, which globally inspect the model's behaviour through vectors aligned with human meaningful concepts [19]. On the other hand, *interpretability* is often used to describe models that are understandable by design. A classical example is linear regression, where the computed coefficients themselves can be interpreted as feature importances. DL models have mostly been constructed without this objective, but recent contributions have been aimed towards the goal of interpretability [3], [22], [23].

Chapter 3

The Rise and Fall of Saliency Maps

Figure 3.1: The explanation produced by a saliency map on an image of a dog and a slightly altered image of a dog. The model prediction is the same for both images. The upper right image has been altered so that the explanation will significantly change, while looking similar to the original image. This is figure 1 from [15].

3.1 Saliency Maps

In an attempt to understand the behaviour of DL computer vision models, various methods for producing pixel-wise explanations have been proposed. The produced output is called *saliency maps*, which attempts to answer how important each pixel of an image was for the model's prediction.

Early versions consisted of differentiating the class probability prediction with respect to the pixels [10]. If a pixel received a high absolute value of this derivative, it means that the model's behaviour was sensitive to changes in that pixel. This is then assumed to partly explain how the model has behaved. In order to also consider the actual input for the pixels, the method *gradient times input* multiplies the gradient with the pixel input values [13].

A more recent version of a saliency map is *integrated gradients* [11], which sums the average gradient over a series of images. It starts with a baseline image, often a completely black image, and then uses images that are gradually more similar to the

Figure 3.2: We have no idea why this image is labelled as either a dog or a musical instrument when considering only saliency. The explanations look essentially the same for both classes. Credit: Chaofen Chen, Duke University. This is Figure 2 from [3].

original image. For each of these images, one computes the gradient of the output with respect to the input pixels. The final saliency map is created by averaging over the gradients from the images. Another influential visualisation method is *Grad-CAM* [12]. Instead of solely using input gradients, Grad-CAM examines the feature maps of chosen convolutional layers. For a given class prediction, it calculates the gradient of this prediction relative to these feature maps, generating a heatmap. This heatmap is upscaled to the image size, highlighting key regions influencing the model's decision for that class.

3.2 The Lack of Explanations from Saliency Maps

While these kinds of saliency maps have been widely used in practice, it is unlikely that they describe the underlying reasons of how the models predict [2], [14], [18]. Although the images seem appealing as they often highlight the object that is being classified, there is no unified agreed way to quantify this. This issue is rooted in the difficulty of actually defining what an explanation is. For an explanation to be useful, it has to sufficiently convince a human why the model made a decision. The medical community has shown considerable demand for understanding model decisions [35], but it is reported that current explainability methods fail to do this [18]. It is written that:

> '*Explainability methods cannot yet provide reassurance that an individual decision is correct, increase trust among users, nor justify the acceptance of AI recommendations in clinical practice.* '

— From *The false hope of current approaches to explainable artificial intelligence in healthcare* [18]

One of the biggest limitations of current explainability methods is that there is no method of quantifying how good an explanation is. A main driver for the justification of the saliency maps have been that they produce images that are visually appealing. However, why should this mean that they actually capture the reasons why a model predicted in a certain way? Making a heatmap for why a model correctly predicted a Siberian husky seems to capture the important parts of the image, but the heatmap looks similar when asked why the image might depict a musical instrument instead (see Figure 3.2). Moreover, there are an abundance of different saliency maps to choose from. With no quantitative metric to determine which one that performs the best, how do you choose which one to represent the model behaviour (see Figure 3.3)?.

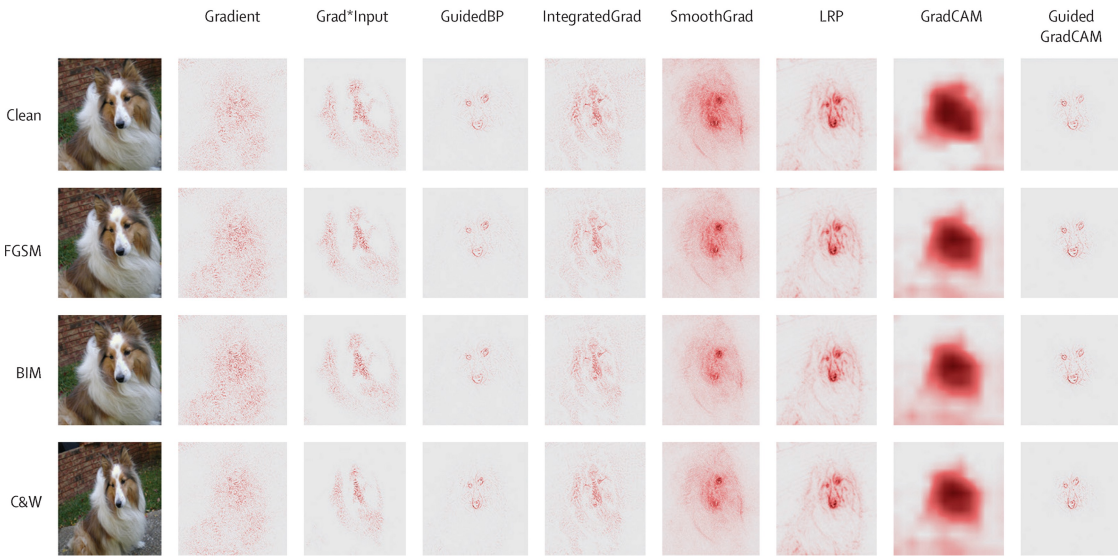


Figure 3.3: **Which explanation is correct?** Saliency maps produced by various popular explanation methods. Do one simply choose the heatmap that looks the best? The rows show different types of adversarial perturbations. Even though the perturbed images make the model misclassify the images, the explanations look similar. This Figure 2 from [18].

There is also evidence that saliency methods are not robust with respect to what one would expect from an explanation. Since there is no quantified way of explaining, the methods propose different metrics to represent an explanation. Most revolve around what happens to the class prediction if some small perturbation to the input pixels or feature maps take place. Although the sensitivity of a model's behaviour for inputs seems like an intuitive measure for how it behaves, does it really capture the rationale behind the decisions? From a proper explanation, it would be reasonable that:

1. **Dependence on models and data:** The explanation method should explain the model. Therefore, it should be highly dependent on the model's weights. Subsequently, since the data that were used to train the model is crucial for how the model ended up, the explanation method should be sensitive to changes in the data.

2. **Robustness when model behaviour does not change:** If some change occurs that does not change the models behaviour, they should not change the explanation.

In the next paragraphs we refer to experiments showing that neither of these hold for saliency-based explanations.

Experiments randomising data model-weights show that some saliency maps are not particularly sensitive to the models and data [14]. In Figure 3.4, we see the results from various popular saliency maps given a randomised effect on the weights in a pretrained network. From left to right, the layers of the network are gradually cumulatively randomised. Despite large changes in the network's weights, some of the explanations seem rather similar. A similar experiment was also conducted where the labels of training images were completely randomised. The networks were trained to a 95% accuracy on the training set, but performed equally to random guessing at the test set. Since this network had completely memorised the training instances and did

no generalisation at all, one would expect that the saliency maps look close to random. However, as one can see in Figure 3.5, the heat-maps do not appear to be random at all. If the explanation methods are partly invariant to the model and data it is trying to explain, does it perform any job explaining at all?

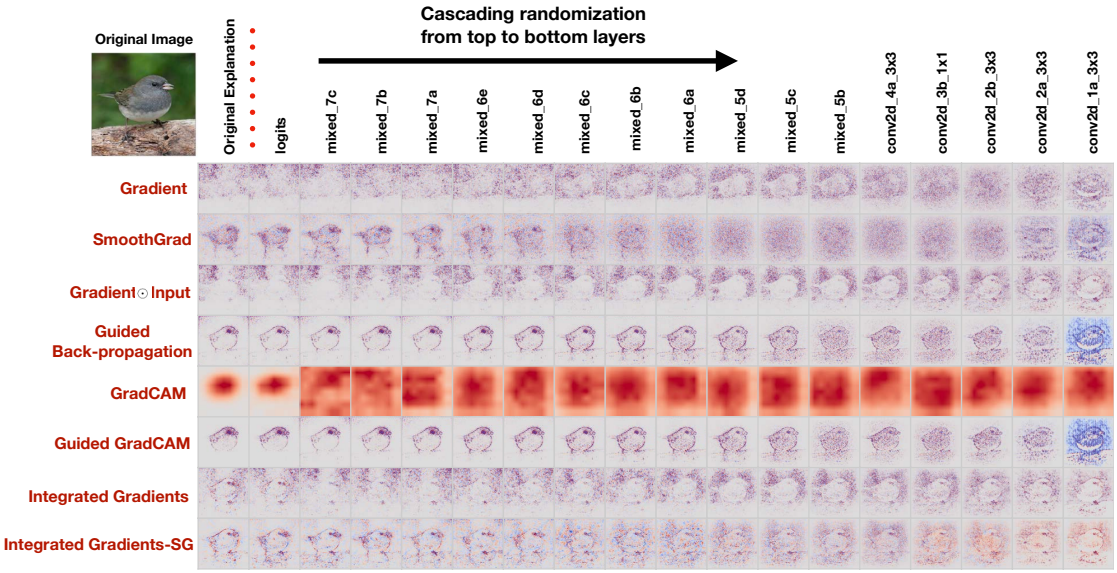


Figure 3.4: The following columns show the explanations when layers in the model have been randomised one by one. The column to the right corresponds to a completely randomised network. Despite this randomisation, some of the explanations seem rather unaffected. How can the explanation be useful if it is not truly dependent on the models' learned parameters? This is Figure 2 from [14].

The saliency maps may also change explanation even though the model does not change its behaviour. In [17], experiments were conducted on two networks that differed only by a bias term. The data used was only changed by a constant vector shift. The changed bias term was designed so that the constant vector added to the images was cancelled out by the bias term, and the network predicted exactly the same. However, the explanation methods did not. In Figure 3.6, one can see the results of a hand drawn cat added upon images from MNIST [67], and how the heatmaps differ for two networks with the exact same predictions.

Finally, we refer to experiments showing that saliency maps can be arbitrarily altered to any target explanation of choice. Just as DL models are unstable and show chaotic changes in behaviour from small perturbation of input images [54], [59], the pixel-wise explanations show similar signs of instability [15]. By adding small perturbations to input images that are unnoticeable to humans, one can use a weighted loss function to find images that the model predicts similarly, but the explanation can be arbitrarily chosen. This can be seen in Figure 3.1.

We argue that these limitations and shortcomings of saliency based explanation methods make them unsuitable for actually explaining how the model works. Although one might argue that there are methods that pass certain tests, like gradients actually being dependent on data in Figure 3.5, those methods fail in other tasks. Given the abundance of different saliency maps, the task of thoroughly exploring all of them are outside the scope of this thesis. Furthermore, all of the methods are based on pixel by pixel feature importance for a single image. Humans do not think pixel-wise, and this fact alone limits the explanatory powers of the saliency maps. Additionally, since every

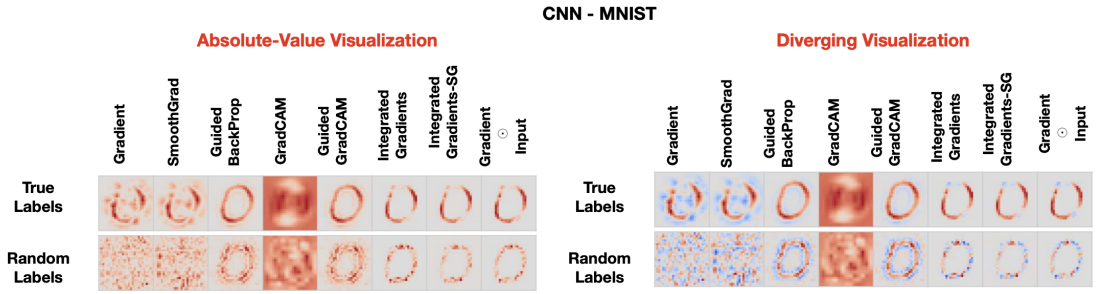


Figure 3.5: **Explanation for a model trained with the correct labels and a model trained with random labels**. Many explanations look similar for the model trained with correct labels and the model trained on random labels, even though they behave completely differently. **Left:** Absolute value visualisation of masks for digit 0 from the MNIST test set for a CNN. **Right:** Saliency masks for digit 0 from the MNIST test set for a CNN shown in diverging color. This is figure 6 from [14].

explanation is done image by image, it does not capture the global model behaviour. Next, we will look into concept-based ways of seeking understanding of model behaviour, along with their limitations.

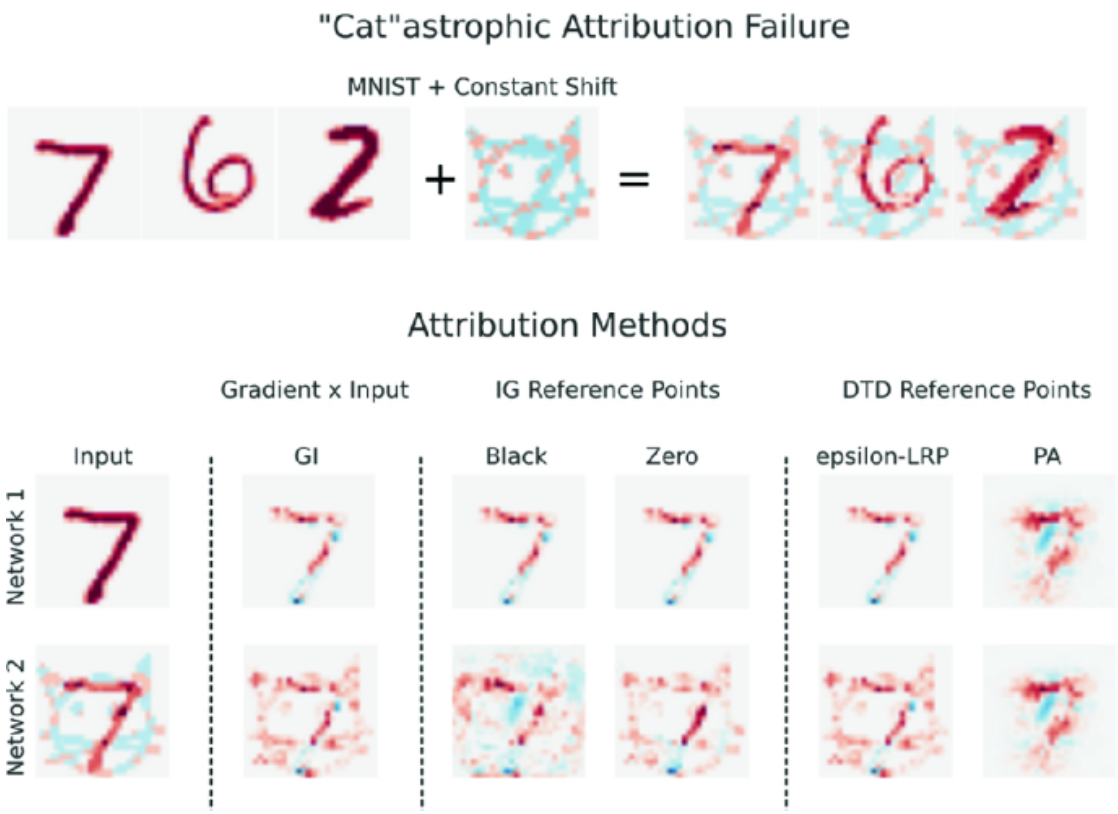


Figure 3.6: **Evaluation of attribution method sensitivity using MNIST.** Gradient times input, integrated gradient with both a black and zero reference point and two other saliency map methods do not display invariance and produce different attributions for each of the networks. This is figure 14.2 from [17].

Chapter 4

Concept-based Explanations and Models

4.1 Post-hoc Concept-based Explanations

Another method of understanding model behaviour is with global *concept-based* explanations. A concept is a human understandable and meaningful feature, different from the class label. For a computer vision model, a concept may be whether or not there are stripes in an image. For a chess computer that predicts the next best move, a concept can be whether or not the player is in check. In order to use this to explain the model's behaviour, one can first examine if the model shows signs of knowing the concept. One can then further explore how the presence or absence of a concept influences the model's predictions. Unlike saliency maps, concept-based explanations are global, meaning they examine the model's behaviour on sets of images, as opposed to an image at a time. Furthermore, they are rooted in human understandable concepts, as opposed to pixel-wise explanations.

We quantify how a concept is learned by how well a simple linear classifier is able to distinguish between the latent representation of inputs having the concept and inputs that do not have it. For each concept C, we acquire a set of images that contains the concept (for example images with stripes), and a set with images that does not contain the concept (for example images without stripes). We call these the *positive set* P_C and *negative set* N_C, respectively. Given a model f to explain, let $q_l : \mathbb{R}^d \to \mathbb{R}^k$ be the function that transforms images to the latent activation in hidden-layer l in f. We pass all images through q_l and achieve our dataset $\{q_l(\mathbf{x}) \,|\, \mathbf{x} \in (P_C \cup N_C)\}$, with labels $y_i = 1$ if $\mathbf{x}_i \in P_C$, and $y_i = 0$ if $\mathbf{x}_i \in N_C$. We then train a simple linear classifier on this dataset. The classifier creates a hyperplane in $\mathbb{R}^k$ that tries to distinguish between the activations of images from P_C and images from N_C. We let $\mathbf{v}_C^l$ be the vector perpendicular on this hyperplane. This is defined as the *concept activation vector* (CAV) [19] for concept C in layer l. If the classifier is able to achieve a high validation accuracy, we assume that the model has learned about the concept. This is outlined in Figure 4.1.

We can further check the model's prediction sensitivity to the CAVs, introduced by a method called *testing with concept activation vectors* (TCAV) [19]. For example, we might want to explore if the concept *stripes* was important for the class prediction *zebra*. This is done by moving the latent space of zebra-images slightly in the direction of the concept-vector. One can then calculate the proportion of images that increased the zebra-logit output. This score can then be compared to scores calculated with different classes or concepts, to get a sense of which concepts the model uses.

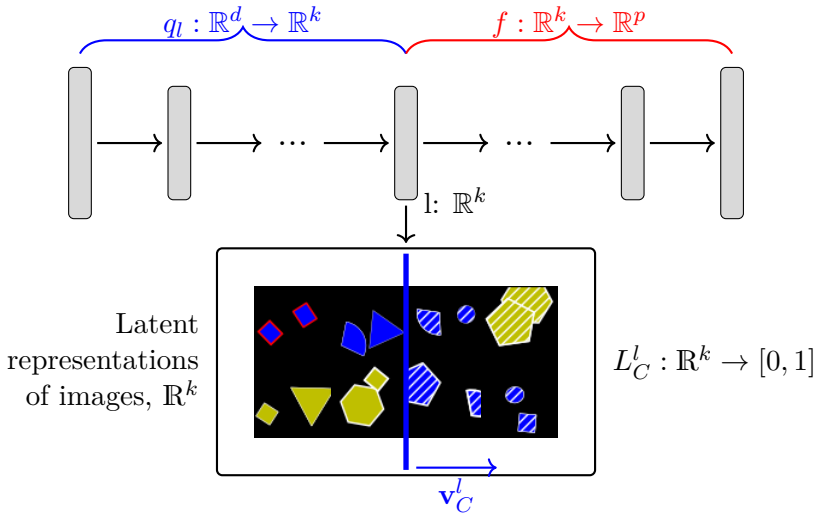


Figure 4.1: **Overview of CAV**. We inspect a hidden layer l in a neural network. Given a concept C, here *stripes*, we calculate the hidden layer activation in layer l of various images. Then we use this to train a simple linear model L_C^l, which gives the CAV $\mathbf{v}_C^l$ perpendicular on the hyperplane that differentiates the activations of the striped images and the non-striped images. The upper part of the illustration corresponds to the neural networks layers, from the input (left) to the output (right). The bottom box represents the images represented as their activation in hidden layer l.

This method has been applied at scale when examining the properties of the chess AI *AlphaZero* [21]. They investigated when concepts were learned during training, and in which hidden layers. They introduced *what-when-where* plots, which plots the CAV accuracies on the hidden layer-number and training step (see Figure 4.2). These experiments show that AlphaZero gradually learned concepts during training. Furthermore, simple concepts (such as "in-check") are learned in early latent spaces as well as later ones, but more complex concepts (such as "has-mate-threat") do better in the later hidden layers. The authors also introduce continuous concepts, as a generalisation of the binary concepts.

Extensions of the CAV framework have been actively proposed. *Automated concept-based explanation* (ACE) [20] finds concepts automatically, by clustering various crops of images. *Conceptual counterfactual explanations* [68] looks into how we can use CAVs to understand a model's mistakes. Given a wrongly classified image, CCE perturbs the latent space representation in various CAV directions to examine the relationship between the concepts and the prediction. The output is a negative or positive score for each concept, representing how much more of that concept would influence the class prediction. Yet another method, *concept whitening* (CW) [69], transforms a latent space of a pre-trained model into a space consisting of decorrelated human understandable concepts. This usually consists of training another epoch with the new concept layer. As a result, the model maps images into a space aligned by concepts, and then to the predictions.

4.2 Interpretable Models

Given the chaotic nature of DL, it is unlikely that they will become explainable with post-hoc methodology. The models are unstable and probably pick up false structures

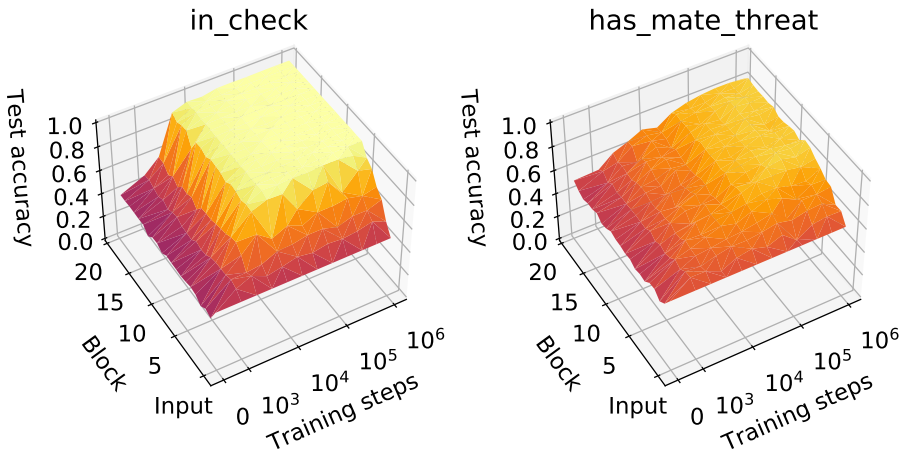


Figure 4.2: Plots showing how two concepts are learned in the chess AI AlphaZero. They show how the test-accuracy for the CAVs increase in various layers during training. The concepts are as follows: **Left:** Is the playing side in check? **Right:** Could the opposing side checkmate the playing side in one move? This is a part of Figure 2 from [21].

[7], [61], and it is reasonable to assume that their explanations are unstable as well. Therefore, a community aimed towards attempting to design powerful models that are inherently explainable has been growing [3].

Self Explaining Neural Networks (SENN) [22] are a class of neural networks, restricted to behave linearly in local regions. They are based on the premise of linear regression being inherently interpretable. The signs of the coefficients from linear regression are a direct measure of the direction a feature influences the target, and their absolute values show the magnitude of the influence. If no interaction terms are used, then the features are also additively separable, not influencing each other. SENN attempts to locally mimic this behaviour. The input is transformed into a vectorspace spanned by vectors representing human meaningful concepts. Then a weighted interpretable sum of vectors is computed. A key limitation is that the proposed model does not guarantee that the new concept space consists of anything human meaningful, which could become a bottleneck for the interpretable qualities.

4.2.1 Concept-based Models

Another approach towards interpretable models is *concept bottleneck models* (CBMs) [23]. They use concepts defined in the same manner as with CAVs, but now use concept labels during training time. A CBM consists of a neural network with a *bottleneck layer*, where every node represents a human meaningful concept (see Figure 4.3). The training data have both concept labels and class target labels, and the model is trained using both a concept loss function and a target loss function. The bottleneck layer restricts the final target predictions to be done entirely based on the concept predictions. This way, one can transparently see the concept predictions that lead to each target prediction. When the model makes mistakes, this will hopefully extend to being able to insightfully examine which wrong concept predictions lead to the mistake.

To formalise, consider a classification task where we train to predict class targets y_i from input images $\mathbf{x}_i$. Each input $\mathbf{x}_i$ is equipped with a binary concept label vector $\mathbf{c}_i$ of length k. This gives us the dataset $\{\mathbf{x}_i, \mathbf{c}_i, y_i\}_{i=1,\dots,N}$, with N input images $\mathbf{x}_i \in \mathbb{R}^d$,

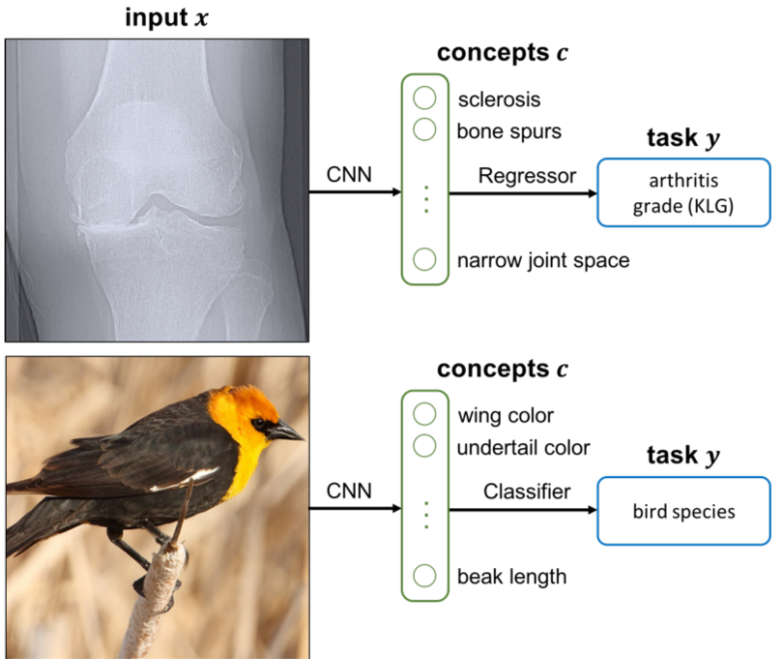


Figure 4.3: Overview of the CBMs. The models first predict an intermediary layer of human-specified concepts c, and then use these to predict the output y. The upper part shows application on knee x-ray grading, and the lower shows classification of birds. This is figure 1 from [23].

binary concept labels $\mathbf{c}_i \in \{0,1\}^k$ and class target labels $y_i \in \{1,2,\ldots,p\}$. A CBM [23] is a neural network $h : \mathbb{R}^d \to \mathbb{R}^p$ that can be decomposed into functions $g : \mathbb{R}^d \to \mathbb{R}^k$ and $f : \mathbb{R}^k \to \mathbb{R}^p$, such that $f(g(\mathbf{x})) = h(\mathbf{x})$. We refer to g as the *concept model* and f as the *target model*. In other words, the CBM h is a neural network with a hidden latent layer l of k nodes. We call this the *bottleneck layer*.

The authors provide different ways to train the CBMs. We define two loss functions, one for measuring how well the concepts are predicted, and one for the targets. Let $L_C : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ be the concept loss function, and $L_Y : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ be the class target loss function. Also let $\mathbf{y}_i$ be the one-hot encoding of the class label y_i. The three different proposed ways of training a CBM are as follows:

1. **Independent bottleneck:** Here f and g are learned independently. The concept model g are first learned by minimising $\sum_{i=1}^{N} L_C(g(\mathbf{x}_i), \mathbf{c}_i)$, and then the target model f is learned by using the true concept labels, minimising $\sum_{i=1}^{N} L_Y(f(\mathbf{c}_i), \mathbf{y}_i)$

2. **Sequential bottleneck:** The sequential bottleneck is made in the same manner as the independent, but the target model is found by using the concept model's predictions, instead of using the true concept labels. In other words, f minimises $\sum_{i=1}^{N} L_Y(f(g(\mathbf{x}_i)), \mathbf{y}_i)$.

3. **Joint bottleneck:** The joint bottleneck finds g and f jointly be minimising a weighted loss function $\sum_{i=1}^{N} [L_Y(f(g(\mathbf{x}_i)), \mathbf{y}_i) + \lambda L_C(g(\mathbf{x}_i), \mathbf{c}_i)]$.

A key argument for the interpretability of CBMs is the possibility to intervene during testing when the model makes mistakes [23], [27], [70]. When a CBM predicts the target

wrongly, one can look at the bottleneck layer and find out which concepts that were also predicted wrong. By correcting a few of the concept predictions, the models achieve better target performance. This means that the insight that the bottleneck layer gives us can be meaningfully used to understand and improve the model. For example, if a radiologist assumes the prediction of a bone spur concept on an x-ray image is wrong, they can update the concept value and achieve better target performance. This process is outlined in Figure 4.4. There have also been introduced algorithms to efficiently query human experts for concept intervention, by weighting uncertainty, importance and cost to intervene [27].

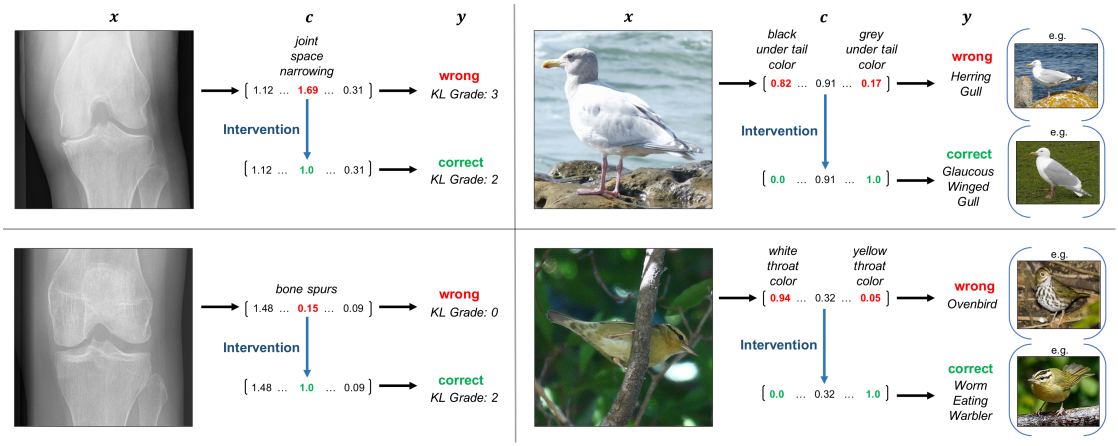


Figure 4.4: Successful examples of intervention during testing, where intervening on a single concept corrects the model's prediction. The models are trained as independent bottleneck models. This is Figure 3 from [23].

While the CBM has a simple architecture, several extensions have been proposed. *Concept-based model extraction* (CME) [24] also uses concepts to predict the target. However, for every concept, they use the hidden layer that best predicts that concept. *Post-hoc concept bottleneck models* (PCBM) [71] first learns the CAVs of several concepts. Then, embeddings are projected down on the concept space spanned by the CAVs, which is then used by a linear classifier to predict the target. An interesting property of PCBMs is that one can globally edit the model at testing time to improve performance, by finding spurious correlations between concepts and targets. *Concept bottleneck models with additional unsupervised concepts* (CBM-AUC) [25] improves the performance of CBMs by adding a SENN architecture in addition to the CBM. Finally, *concept embedding models* (CEM) [72] produces latent spaces of concepts that are different for presence and absence of the concept. The parameters are also used to predict the probability of presence or absence of the concepts, and use this to calculate a weighted sum of the latent spaces, which is passed forward in the network. This way, they achieve better performance with respect to accuracy and intervention during testing.

4.2.2 Pitfalls of Concept-based Models

Although the nature of CBMs intuitively seem interpretable, experiments show that they might not live up to this promise. Several papers have examined the interpretable qualities of CBMs, and it is concluded:

> '*We have called into question CBMs trained using the joint objective, as they seem to fall short on all three of our desiderata: (1) they do not provide concept interpretability: post hoc analysis shows that the importance of individual concepts does not correspond to their true importance in predicting the targets; (2) they do not always predict target values based on concepts, thus violating predictability; and (3) they may not intervenable, as the concepts are learned at concept layer.[28].*'
>
> — From *Do Concept Bottleneck Models Learn as Intended?* [28].

A core aspect of a CBMs' interpretability comes from them only being able to use the concept predictions to make its final prediction. However, it can easily be demonstrated that more information than just the concepts are embedded into the bottleneck layer, referred to as *concept leakage*. In [28], it was demonstrated that training a CBM on just one concept led to far better accuracy than an oracle model that could access the true concept label directly. Additionally, the saliency maps of the concepts gave attention to the whole object, rather than the part that the concept represented, as seen in Figure 4.5.

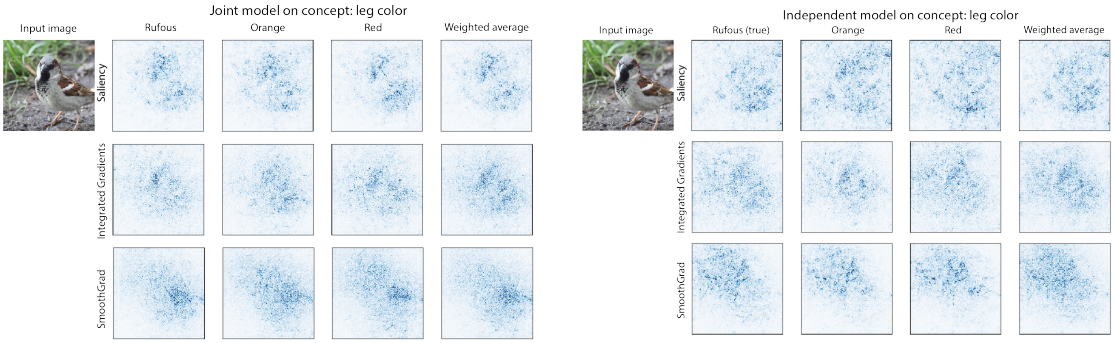


Figure 4.5: The saliency maps from the concept predictions in a CBM do not highlight the specific concept, but rather the whole image. **Left:** CBM trained jointly. This is Figure 3 from [28]. **Right:** CBM trained independently. This is Figure 4 from [28].

Concept leakage was further emphasised in a series of experiments in [29]. A CBM was trained to predict parity on the MNIST dataset (whether a number was even or odd), and the only concepts were whether the input was a 4 or a 5. As an extreme example, the model was trained and tested without any 4s and 5s in the dataset. During testing, one would assume such a model to have accuracy similar to random guessing, at around 50%. However, it reached 69%, showing that a substantial amount of concept leakage was happening. This was done when the concept model and target model were trained independently. This was also recreated with similar results when the concepts were truly random, and had no semantic overlap with the targets. Attempts to solve the leakage by adding unsupervised concepts or decorrelating was done without success.

CBMs have also been shown to be susceptible to adversary attack [30]. Specifically, the authors show that it is possible to perturb images so that the CBM concept prediction changes, while the target prediction stays the same. They manage to both introduce the prediction of concepts, remove them, and do both for the same image.

We further build upon the evidence of the limited interpretable qualities of CBMs by designing *adversarial concept attacks*. We design an algorithm that given a CBM, perturb an image such that it is identical to the original image, produce identical concept predictions, but different class predictions. The details are described in Chapter 7. How

can a CBM provide meaningful interpretations if two different predictions are interpreted identically? Our algorithm differs from [30] which designed their adversarial examples to produce equal target predictions, but different concept predictions.

Chapter 5

Datasets

5.1 Shortcomings of Existing Concept Datasets

Having suitable concept datasets is required for doing proper analyses of concept-based models, but we argue that popular concept datasets exhibit substantial shortcomings. Here are some features we would like to see in a good concept dataset:

- **Accurate concept labelling:** The dataset should be equipped with concept labels as well as target labels. The labelling should be accurate, with as little mislabelling as possible.
- **Concepts should be present in each image:** An image should only be labelled with a concept if it is evident from the image alone. For instance, if a bird has a red belly, but it is facing away from the camera so the belly is not visible in the image, it should not be labelled with the red belly concept. Otherwise, the model will not train to predict the concept, but to infer the concept from correlations between other concepts and classes. This could make the concept essentially act as noise in the dataset.
- **Concepts should be relevant for the task:** The concepts should be relevant for the downstream target. If not, the concepts will just act as noise, and standard DL models will probably outperform the concept-based models.
- **Open source and availability:** The dataset should be openly available for anyone to do further research. Not only should it be available, but the process of downloading and using the dataset should be made as easily as possible. Ideally, it is not very big, so that several experiments can be conducted without requiring vast computational resources.

We now review some of the most popular datasets used for concept-based models.

5.1.1 Caltech-USCD Birds-200-2011 (CUB)

The Caltech-USCD Birds-200-2011 (CUB) [31] is the most widely used concept dataset, and is used in [23]–[27], [30], [71], [72]. It consists of $N = 11788$ images of birds, where the target is labelled among 200 bird species. The original dataset contains 28 categorical concepts, which makes 312 binary concepts when one-hot-encoded.

However, the concept labels are noisy. The labelling was outsourced to non-bird experts, making some labels differ between the crowdworkers. For example, the

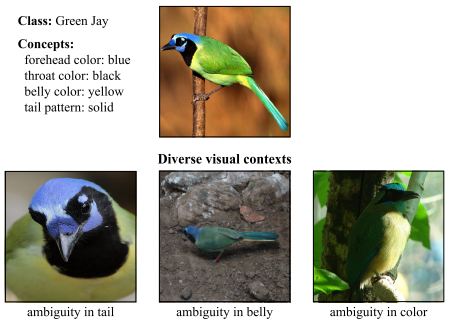


Figure 5.1: Examples of ambiguous cases in the existence of concepts from the CUB dataset. Images have diverse visual contexts, where partial concepts may become invisible and unclear. This is Figure 1 from [26].

crowdworkers might disagree on where the line for *red* and *rufous* (reddish-brown) goes. Therefore, the dataset was pre-processed with majority voting and the removal of sparse concepts [23]. If over 50% of the images of a class had a concept, all images of that species got assigned the concept. Concepts that were present in less than 10 classes were removed. In the processed dataset, 112 binary concepts are present.

Even with the preprocessing, there is still noise and problems with the dataset. The concepts in the dataset are class-wise, meaning all birds in the same species got assigned the same concept labels. However, the dataset features species that actually have different concepts within a class. For instance, the species *black tern* has been majority voted to have black underparts, but there are actually some birds of the species that have white [70].

Furthermore, concepts may not be evident from the image alone. Several of the images contain ambiguity of the concepts. Consider Figure 5.1. The bottom three images do not display the presence of the concepts for the species, but by majority voting, they are still labelled with them. This results in the concept-based models trying to predict concepts that are not present in the image. Thereby, they will infer this by correlations with the class or other information in the image, which overshadows the point of using concepts predictions as an interpretable intermediary step.

Due to these shortcomings, we do not think the CUB dataset is sufficient to rank concept-based models. However, due to its popularity, we still use it to benchmark our proposed models.

5.1.2 Osteoarthritis Initiative (OAI)

Another popular concept dataset is *Osteoarthritis Initiative* (OAI) [73], used in [23], [27], [30]. The preprocessed version [23] consists of $N = 36369$ x-ray images of knees, where the target is to predict the Kellgren-Lawrence grade (KLG), which is a 4-level severity grade of osteoarthritis assessed by radiologists. The images are from 4172 different patients, where x-rays are conducted at different timepoints. The preprocessed concepts consist of 10 clinical variables, such as *bone spurs* and *calcification*.

The main limitation with the OAI dataset is the lack of availability. Due to privacy issues of the sensitive medical data, the dataset is not publicly available. Although access can be gained easily on request, the main challenge lies in the preprocessing. The correct images and labels need to be queried from the provider of the database, and then preprocessed. The CBM paper's [23] GitHub links to [74] for preprocessing, which states:

'*Data was processed on a computer with several terabytes of RAM and hundreds of cores.*'

The computational specifications of several terabytes of memory are very high, even for computers made for scientific computations. The resources available for this thesis had 256GB of RAM [75], and were shared among many users. Therefore, carrying out the preprocessing is not feasible for researchers that do not have access to expensive computational resources. Even then, the process for setting up the dataset would be tedious and time consuming. We therefore argue that the unavailability of the dataset makes it unsuitable to develop concept-based models as a community effort.

5.2 Introducing the ConceptShapes Datasets

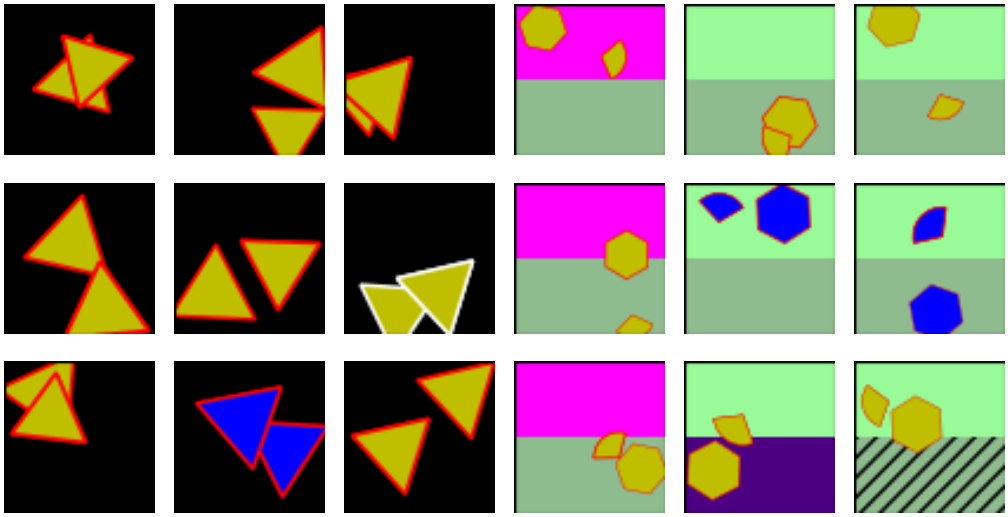


Figure 5.2: Images from two different ConceptShapes datasets. The datasets with 5 concepts all have black backgrounds, while the 9-concept datasets have 4 additional concepts in the background. **Left:** 9 different images from the 10-class 5-concept dataset, all from the "triangle-triangle" class. **Right:** 9 Different images from the 21-class 9-concept dataset, all from the "hexagon-wedge" class.

As one of the main contributions of the thesis, we created a flexible class of synthetic concept datasets called *ConceptShapes* (see Figure 5.2 and Figure 5.3). By *flexible*, we mean that one can create new datasets where one controls the amount of classes, concepts, and relation between them. All of the images in the dataset depict two shapes, and the downstream task is to classify which two shapes that are present. There are also concepts present in the images. The concepts are clearly visual cues, relating to colour, texture and the sizes of the shapes, outline and background. The position and rotation of the shapes are decided randomly. The code used for creating the datasets are openly available at https://github.com/Tobias-Opsahl/TobiasaoThesis.

There are six main variations of the ConceptShapes datasets, resulting from three different choices for the amount of classes, and two choices for the amount of concepts (see Table 5.1). One can choose to include four, five or six different shapes, leading to ten, fifteen and twenty one different pair combinations. The shapes have five concepts related to them. The background has four additional concepts, making a total of nine concepts. This means one can use either five or nine concepts in the datasets. While

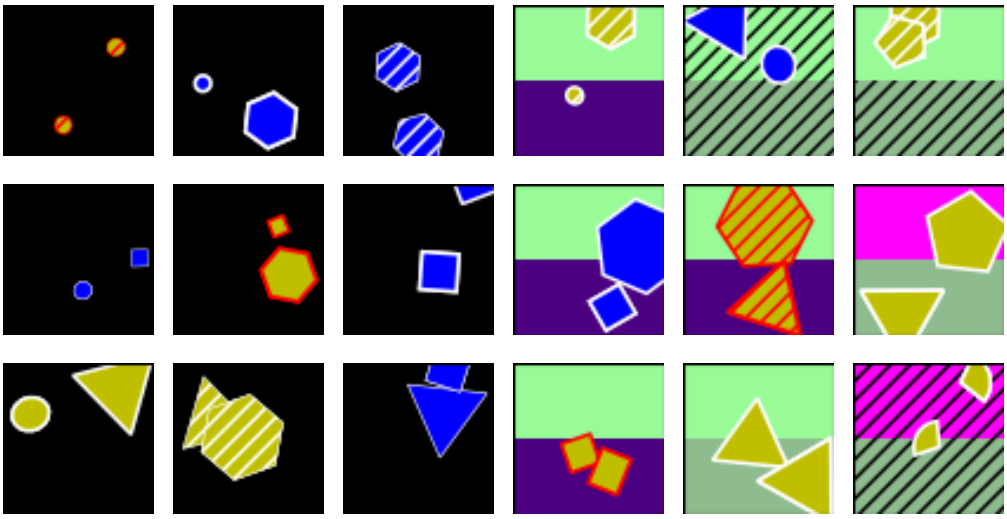


Figure 5.3: Images from different classes of two ConceptShapes datasets. **Left:** 9 different images from the 10-class 5-concept dataset, all from different classes. **Right:** 9 different images from the 21-class 9-concept dataset, all from different classes.

it is possible to produce datasets with other numbers of classes and concepts, we have conducted experiments based on these numbers.

	5 Concepts	**9 Concepts**
10 Classes	Dataset 1	Dataset 4
15 Classes	Dataset 2	Dataset 5
21 Classes	Dataset 3	Dataset 6

Table 5.1: The six variations of the ConceptShapes dataset used in this thesis.

5.2.1 Description of the Concepts

The crucial feature of the datasets are the concepts. All of them are binary and independent, meaning any combination of concepts are possible for each image. The five first concepts are based on the two shapes in the image, while the last four optional concepts are based on the background. The datasets are created with binary concept label vectors that represent the true value of each concept in each image. We now describe the concepts one-by-one, and visualisations are available in Table 5.2 and Table 5.3. We start with the five concepts regarding the shapes:

1. **Big shapes**. Every shape had two intervals of sizes to be randomly drawn from. One interval corresponded to the small figures, and the other to big ones.

2. **Thick outlines**. The outlines of the shapes were drawn from one of two intervals. One corresponded to a thin outline, and the other to a thick one.

3. **Facecolor**. There were two possible colours for the shapes, blue and yellow.

4. **Outline colour**. The shapes had two possible outline colours, red and white.

5. **Stripes**. Some shapes were made with stripes, and some were not. The stripes were in the same colour as the outline.

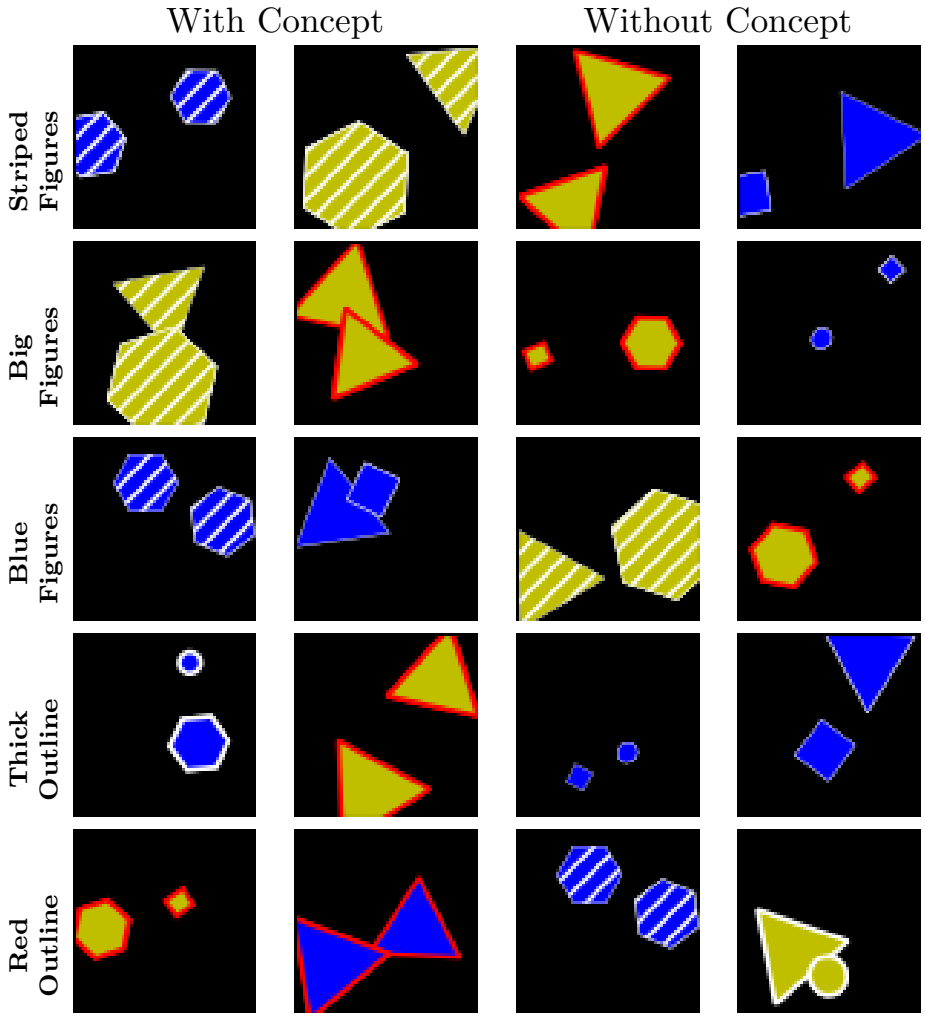


Table 5.2: **Overview of the 5 concepts regarding the shapes.** Each row corresponds to one concept. The two leftmost columns of images have the concept, and the two rightmost columns do not have the concept. All of the images are from a 5-concept dataset, hence the black background. These five concepts are also present in the 9-concept datasets.

All of the concepts apply to the whole image. For instance, if the image gets the thick outline concept, both shapes in the image get a thick outline.

The datasets that use nine concepts have all five of the concepts above, in addition to four more. While all of the five-concept datasets have black backgrounds, the nine-concept datasets split the background in two and use the colour and presence of stripes as concepts.

6. **Upper background colour**. The upper-half of the background would either be magenta or pale-green.

7. **Lower background colour**. The lower-half of the background would be either indigo or dark-sea-green.

8. **Upper background stripes**. This represented whether there were black stripes present in the upper background or not.

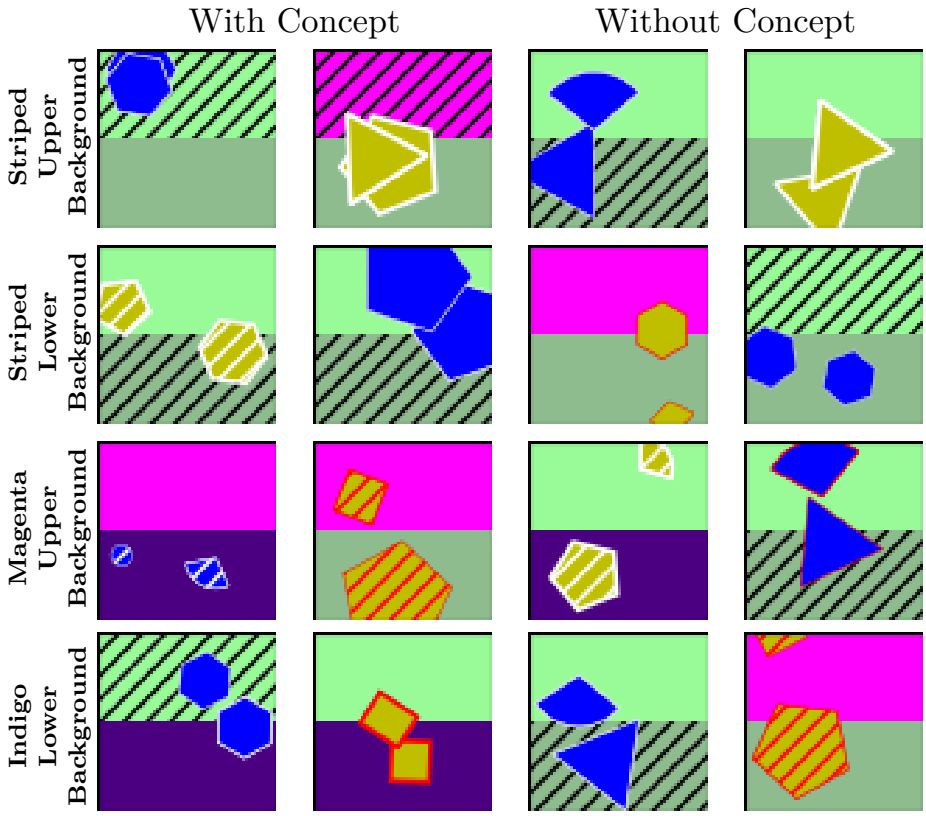


Table 5.3: **Overview of the 4 concepts that relate to the background.** These four concepts are only present in the 9-concept datasets. Each row corresponds to one concept. The two leftmost columns of images have the concept, while the two rightmost columns have not. Note that the five concepts regarding the shapes are also present in the 9-concept datasets, like thickness of the outlines and shape-sizes.

9. **Lower background stripes**. This represented whether there were black stripes present in the lower background or not.

To summarise, some of the image's visuals are determined by the concepts, some by the classes and some by randomness. The two shapes (from triangle, square, pentagon, hexagon, circle and wedge) are determined by the class. The shapes' size, colour and outline are determined by the concepts. If the dataset uses nine concepts, the background colour and stripes are also made from the concepts. The shapes' position and rotation are determined randomly, regardless of which class or concepts they have.

Note that the size of the shapes and the thickness of the outline are partially random. The concept determines one of two intervals to draw the size and thickness from, but each image draws randomly from the interval chosen. For example, thick outlines are drawn uniformly on the interval (1, 1.2), and thin outlines are drawn uniformly from (0.2, 0.5). The details are covered in Appendix A.1.

5.2.2 Correlation Between Classes and Concepts

The correlation between the concepts and classes are made with a table that determines which concepts that are related to which class, and a tunable hyperparameter that determines the strength of the relationship. A table of the relationship for the 10 class setting is available in Table 5.4, and the 15 class and 21 class setting are explained

	Classes									
Concept	0	1	2	3	4	5	6	7	8	9
Thick Outline	X		X		X		X		X	
Big Figures	X	X	X	X	X					
Blue Facecolor		X			X	X			X	X
Red Outline	X					X	X			
Stripes			X				X			X
Magenta Upper Bg.		X		X		X	X	X		
Indigo Lower Bg.	X			X			X	X	X	
Upper Bg. Stripes						X	X			X
Lower Bg. Stripes			X		X	X				

Table 5.4: Overview of which classes that get assigned a high probability for which concepts, in the 10-class ConceptShapes datasets. Given a hyperparameter $s \in [50, 100]$, an X indicates high probability (s%), while the absence of an X represents a low probability ((100 - s)%).

similarly in Appendix A.1. For example, in the 10-class setting, images in class 0 (two triangles) have a high probability of getting a thick outline, big figures, red outline and magenta lower background colour, while the rest of the concepts have a low probability.

The quantity of "high" and "low" probability is determined by a hyperparameter we call $s \in [50, 100]$. This value is defined before the creation of each dataset. For each image, if the class has a high probability of receiving a concept, it gets that concept with a s% chance. If it is low, it gets a $(100 - s)$% chance of receiving that concept. For instance, for datasets made with $s = 98$, about 98% of images in class 0 have thick outlines, while about 2% have stripes. This is because class 0 has a "high" probability for thick outline, and a "low" probability for stripes. Setting $s = 50$ means that every concept for every class is drawn with a 50% chance, making the concepts and classes uncorrelated. Conversely, setting $s = 100$ means that each image of a given class has the exact same concepts.

This hyperparameter is included so that one can explore model behaviour on different concepts and class correlations. Real world concept datasets are unlikely to have no variance in the concepts. This can be seen with the *black tern* class in the CUB dataset, which could have both black and white underparts [70]. Experimenting with different values of s means that one can explore how models behave given different correlations between the concepts and classes, which can give valuable insight to the models' behaviour. This is only possible with a synthetic dataset.

The predictive power from the concept labels alone depend on the hyperparameters of the dataset. In the experiments and results, we will look at *oracle* models that use the concept labels to predict during testing. We will see that their accuracy varies all the way from 10% to 100%, depending on the value of s, the number of classes and the number of concepts.

5.2.3 Further Details

In order to conduct thorough experiments, the images were made in a low resolution of $3 \times 32 \times 32$ pixels, so that one can easily train models with different settings multiple times. They are equipped with an alpha channel that gets removed when processing the images in the data-loader. The difficulty of the classification task was adapted so one can easily train accurate models on an ordinary laptop without GPU support, but

difficult enough so that one does not easily get over 99% test-set accuracy. To make the datasets have a suitable difficulty, some of the figures overlap considerably, and some of them are partly outside the border of the image. By experimenting with different settings for the amount of classes, concepts, overlap and resolution, we were able to achieve a satisfactory difficulty.

We will explore how models perform on different subsets of the dataset. We made 1000 images for each class in each dataset, except for the 10-class 5-concept s-98 dataset, which had 2000. This was split in a 50%-30%-20% train-validation-test global split. The experiments explored subsets where 50, 100, 150, 200 or 250 images were drawn randomly from each class, in a 60%-40% training and validation split. For example, the 15-class 100-subset dataset had 60 training images and 40 validation images drawn from the global training and validations split for each class. This makes a total of 900 training and 600 validation images. The test set was the same for all the subset, which was 20% of all of the images created.

The class of datasets we created fulfils our desiderata of concept datasets. They are never mislabeled, openly available for anyone to do experiments on, do not require expensive computational resources and can be altered to one's needs. The hyperparameter s controls the concepts' influence on the classes. The amount of classes and subset of data used controls the difficulty of the classification task. Furthermore, resolution, concepts, colours, and concept-class relationships can be easily altered if desired. Versions of the dataset with different hyperparameters are available, and creating a dataset of 10 000 images only takes about 15 minutes with an ordinary laptop CPU.

Chapter 6

Novel Model Architectures

In this section, we propose two novel concept-based neural network architectures. The models are based on a CBM [23] with additional skip connections. Both of the models contain a *bottleneck layer*, which has as many nodes as there are concepts in the dataset. During the backward pass, the output from the bottleneck layer is used together with the concept labels from the dataset to calculate the concept loss. The loss function is a weighted sum between the concept loss and a normal class loss.

In short, we propose one model that acts like a CBM [23] with an additional skip connection, and one that learns concepts sequentially during the layers. Since the models use both the concept predictions and the skip connection to predict the target class, we refer to the models as *hybrid concept models*.

We will now introduce the notation. The framework we work with is computer vision classification tasks, where the dataset is equipped with concept labels. We have N data points, k different concepts and p different classes. The dataset is given by $\{\mathbf{x}_i, \mathbf{c}_i, y_i\}_{i=1,\dots,N}$, where $\mathbf{x}_i \in \mathbb{R}^d$ are the input images, $\mathbf{c}_i \in \mathbb{R}^k$ are the binary concept label vectors, and $y_i \in \{1, 2, \dots, p\}$ are the target class labels, for every $i \in \{1, 2, \dots, N\}$. The concepts are binary, meaning that every element $c_{i,j}$ in the concept labels $\mathbf{c}_i$ is in $\{0, 1\}$.

6.1 Concept Bottleneck Models with Skip Connection

The first model acts as a CBM with an additional skip connection, and we introduce two variants of it. The models pass information through a bottleneck layer, in addition to information that jumps over the bottleneck layer with a skip connection. The first variant, *Concept Bottleneck Model with Residual Connection* (CBM-Res), implements the skip connection as a residual connection [76]. The second variant, *Concept Bottleneck Model with Skip Connection* (CBM-Skip), implements it as a concatenation step [77]. The models are illustrated in Figure 6.1 and Figure 6.2, and we now explain the structures in detail.

For both of the variations, we have a CNN $g : \mathbb{R}^d \to \mathbb{R}^q$, that we refer to as the *base-CNN*. The base-CNN can be any neural network, where its complexity should be adapted to suit the difficulty of the classification task. We then have a *bottleneck layer* $b_k : \mathbb{R}^q \to \mathbb{R}^k$ that is connected to the base-CNN g. The bottleneck layer has as many nodes as concepts present in the dataset. While training, the output from the bottleneck layer and the concept labels will be put in a concept loss function that will be used to train the model. While predicting, the bottleneck layer behaves as an ordinary linear

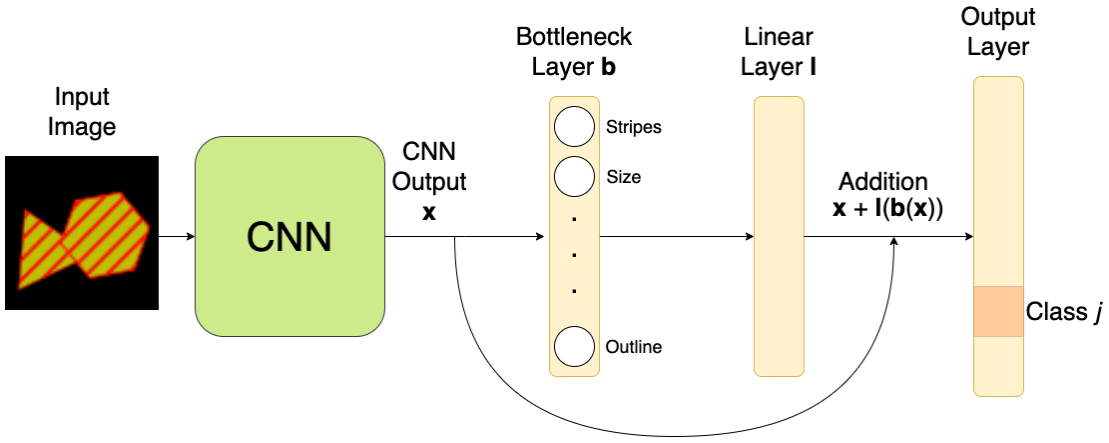


Figure 6.1: Architecture of a **Concept Bottleneck Model with Residual Connection (CBM-Res)**. The intermediary bottleneck layer predicts the concepts in the dataset. The class prediction is made both with the concept predictions and with information not going through the bottleneck layer. At training time, a weighted sum between the concept loss and the class loss is used as the loss function. The base-CNN can be any model, pre-trained or not.

layer. After the bottleneck layer, we have another linear layer $l : \mathbb{R}^k \to \mathbb{R}^r$, and a final classification layer $l_c : \mathbb{R}^s \to \mathbb{R}^p$.

The CBM-Res implements the skip connection as a residual connection [76] (see Figure 6.1). The output from the linear layer l after the bottleneck layer is added elementwise to the output of the base-CNN g. In order for this to work, the dimension r must be equal to q. In other words, we get $g(\mathbf{x}) + l(b_k(g(\mathbf{x})))$, which is then passed as input to the final classification layer. This also makes the dimension s be the same size as q.

The CBM-skip is similar to the CBM-Res, but instead of using residual skip connections, it concatenates [77] (see Figure 6.2). The output of the linear layer l is concatenated with the output of the base-CNN g, so that we get $\text{cat}(g(\mathbf{x}), l(b_k(g(\mathbf{x}))))$. Now, r does not have to be the same dimension as q. Consequently, we get that $s = q+r$.

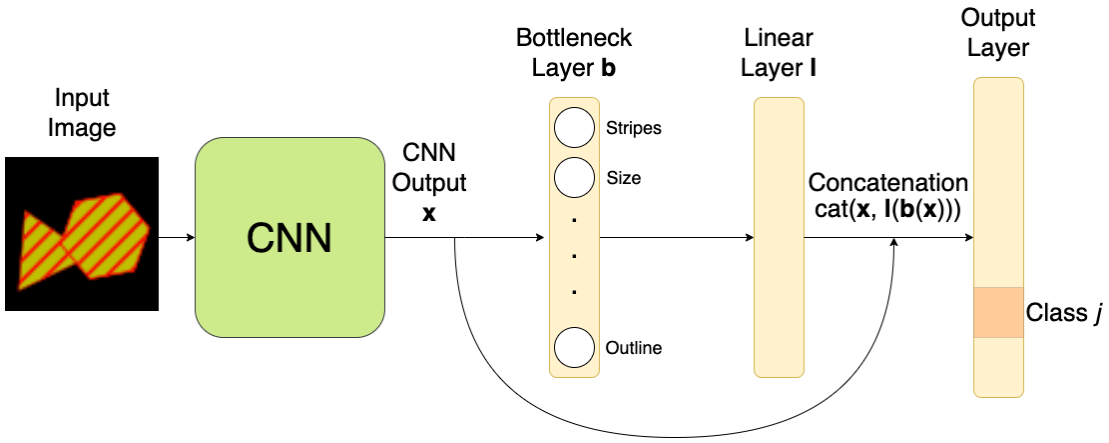


Figure 6.2: Architecture of a **Concept Bottleneck Model with Skip Connection (CBM-Skip)**. This model works similarly to the CBM-Res model, but the outputs going through the bottleneck layer and skip connection are concatenated, rather than added.

6.2 Sequential Bottleneck Model (SCM)

The SCM divides the bottleneck layer into multiple concept layers that are connected to various layers in the model (see Figure 6.3). We have a similar base-CNN $g : \mathbb{R}^d \to \mathbb{R}^q$, but now some of the concept layers are connected to intermediary layers in g. The maxpooled output from the convolutional layers are flattened, and concept layers $b_i : \mathbb{R}^{p_i} \to \mathbb{R}^{k_i}$ are connected to them. We also have concept layers connected to the fully connected linear layers after g. In the end, the concept layers' outputs are concatenated to become the full bottleneck layer. The nodes of the concept layers combined should therefore be equal to the number of concepts, which with M concept layers becomes $\sum_{i=1}^{M} k_i = k$. Finally, this concatenation is then concatenated with the output from the last linear layer.

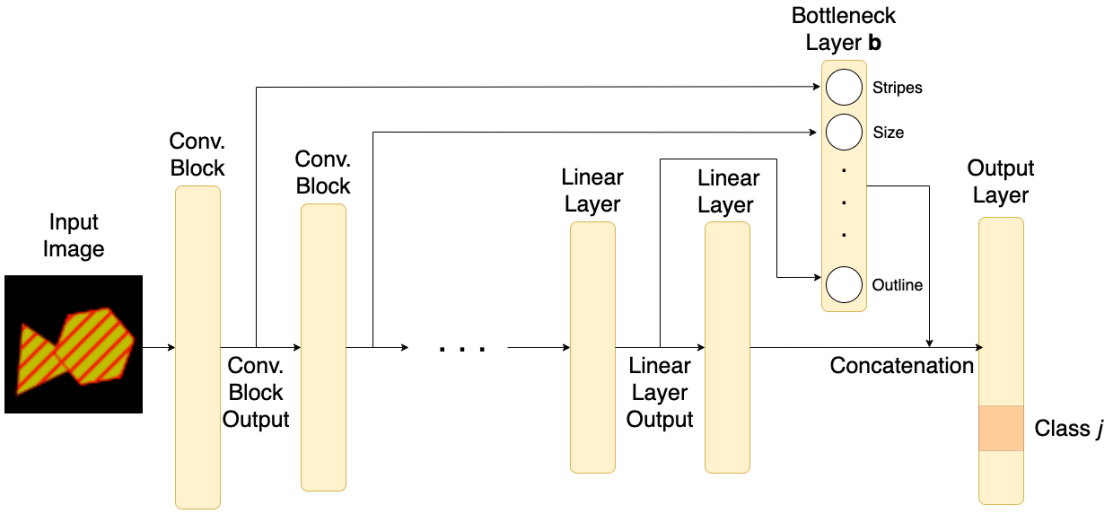


Figure 6.3: Architecture of a **Sequential Bottleneck Model (SCM)**. Concepts are predicted sequentially along the network's layers. There are multiple concept layers. The input to the concept layers can either be flattened output from convolutional blocks or output from linear layers. The output from the concept layers are then concatenated to make the bottleneck layer. When making class predictions, the concept predictions are concatenated with information not going through the concept layers. At training time, a weighted sum between the concept loss and class loss is used as the loss function.

6.3 Training

The models are trained jointly on a weighted sum between the concept loss and the class loss. Let $L_C : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ be the concept loss function, and $L_Y : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ be the target class loss function. Our models output $\hat{\mathbf{y}}, \hat{\mathbf{c}}$, the predicted probability distribution over the classes and the probability values for the binary concepts, respectively. We minimise $\sum_{i=1}^{N} [L_Y(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \lambda L_C(\hat{\mathbf{c}}_i, \mathbf{c_i})]$. Here, λ is the *concept weight*, a tunable hyperparameter that determines the weighting between the concept loss and the class loss. This is the same training setup as the *joint bottleneck* from [23].

The concept weight λ can be held constant or be reduced over time. When we set it to be constant, we can find a suitable size during hyperparameter optimization. If we chose to not set it to a constant value, we initially set λ_0 to a high number. After each epoch, it is exponentially decayed by a parameter $\gamma \in (0, 1)$, so that epoch number i will

have a concept weight of $\gamma^i \lambda_0$. In this scenario, we can include γ in our hyperparameter search instead of λ. The intuition behind this method is to make the model focus on learning the concepts in the early stages of training, before gradually moving over to focus on predicting the downstream class.

6.4 Further Details

An important feature of the proposed models is the compatibility with transfer learning. As with many DL tasks, one often wants to use a big pre-trained model as a base for the specific task. With CBM-Res and CBM-Skip, this is as easy as just overwriting the output layer of any pre-trained model and replacing it with the respective architecture. For the SCM, one also has to connect concept layers to some of the hidden layers in the pre-trained model. In Chapter 8 and Chapter 9, we demonstrate transfer learning and training from scratch for both model architectures.

There are many choices for the activation function used after the bottleneck layer. Most importantly, we differentiate between *soft* and *hard* bottlenecks. In a soft bottleneck, the values have not been rounded off to binary values, but are continuous. They can still go through ReLU or sigmoid activation functions. A hard bottleneck rounds off to the binary concept predictions, and passes forward a binary vector. In this thesis, we experiment both with soft sigmoid bottlenecks and hard bottlenecks.

The models presented are motivated by performance and not interpretability. The vanilla CBM is promoted as interpretable, since every class prediction can be explained by the corresponding concept predictions. With the skip connections, we no longer know whether the final prediction was made due to the concepts or due to the skip connection that does not interfere with the concepts. However, recent results [28], [29] show that the assumption of interpretability in CBMs may not hold, since the bottleneck layer encodes more information than just the concept predictions. We further question the interpretability when we perform adversarial concept attacks Chapter 7. Therefore, we do not think that the lack of interpretable qualities in our hybrid concept-based models is a drawback. On the contrary, we believe that using the concepts while training can be motivated by performance alone.

Even though we have presented the models in the classification framework, they can easily be adapted to regression. In that case, the output layer consists of one node, and the target loss function is chosen accordingly.

Chapter 7

Adversarial Concept Attacks

Concept bottleneck models (CBMs) are motivated by interpretability [23], but we will show that their interpretations suffer from the common problem of instability. The promise of interpretability comes from that a CBM's predictions can be interpreted by its concept predictions. What would then happen if identical concept predictions resulted in different model behaviour? We will construct an algorithm to demonstrate instability and lack of interpretability in CBMs, and argue why this is a bigger problem for interpretable models than for other DL models.

Given an image and a CBM, we want to create a perturbation of the image that looks identical to the original, such that the model predicts the exact same concepts for both images, but predicts different classes. We refer to this as *adversarial concept attacks*. In order to carry out adversarial concept attacks, we propose a new algorithm for creating adversarial examples. We construct these experiments because we suggest that existence of such adversarial examples further questions the interpretability of CBMs. The central question then is:

If two images with identical concept predictions have different class predictions, how can the concept predictions provide an explanation of the model's behaviour?

A recent paper [30] has done adversarial attacks on CBMs with the opposite goal. An algorithm was constructed such that similar looking images would give the same class prediction, but different concept predictions. This was done in a way that could remove predicted concepts, introduce concepts that were not predicted, or both at the same time. The algorithm consisted of a weighted loss function balancing equal class prediction and different concept predictions. This allowed the researchers to do thorough analyses of the robustness of CBMs, and develop another algorithm to increase the robustness. However, one might expect inputs with different class labels and the same class label in the dataset, as seen in the example of the black tern in Section 5.1.1. Therefore, we believe that our approach better demonstrates the questionable interpretability of CBMs.

The algorithm for adversarial concept attacks can be summarised as the following: We iteratively update our perturbed image to increase the loss of the prediction for the true class, similarly to PGD [58]. We refer to the perturbation this gives us at every step as the *initial perturbation*. Additionally, we want to mitigate the chance of concept predictions changes. Therefore, we elementwise check if the concept prediction logits are close to 0, and refer to these as *sensitive concepts*. For each pixel, we check if the initial perturbation brings any of the sensitive concepts closer to 0. If they do, we multiply them with a number in $[-1, 0]$. This essentially flattens out or reverses the change in

sensitive concept predictions. We terminate with success if the class prediction changes without the concept predictions changing, and terminate with negative results if the concept predictions change, or if we reach the maximum number of iterations. In the next section, we will present the algorithm in detail.

7.1 The Adversarial Concept Attack Algorithm

Next, we will introduce the notation used in our algorithm, which is described in detail in Algorithm 1. Assume we have a CBM $h : \mathbb{R}^d \to \mathbb{R}^p$, with a concept model $g : \mathbb{R}^d \to \mathbb{R}^k$. Given an input image $\mathbf{x}$ and class label y, assume that $\text{argmax}(h(\mathbf{x})) = y$. This means that our concept model correctly predicts the true class label. Also let the concept logits be $g(\mathbf{x}) = \hat{\mathbf{c}}$, and let $\hat{\mathbf{c}}_b = \mathbb{I}(\sigma(\hat{\mathbf{c}}) > 0.5)$ be the binary predictions, where the indicator function $\mathbb{I}$ (see Equation (2.1)) works elementwise. Our goal is to produce a perturbed image $\widetilde{\mathbf{x}} = \mathbf{x} + \mathbf{r}$ with $\widetilde{\mathbf{c}}_b = \mathbb{I}(\sigma(g(\widetilde{\mathbf{x}})) > 0.5)$ and $\text{argmax}(h(\widetilde{\mathbf{x}})) = \widetilde{y}$, such that $\widetilde{\mathbf{c}}_b = \hat{\mathbf{c}}_b$, but $\widetilde{y} \neq y$. Additionally, we want $\widetilde{\mathbf{x}}$ to have elements that are in an interval that represent valid pixel values, and $\mathbf{r} \in \mathbb{R}^d$ to be small enough so that $\mathbf{x} + \mathbf{r}$ look identical to $\mathbf{x}$ by humans.

We now describe how the algorithm works in detail. We start with $\widetilde{\mathbf{x}}_0 = \mathbf{x}$ and at every step t calculate

$$\hat{\mathbf{p}}_t = \text{sign}(\nabla_{\widetilde{\mathbf{x}}_t} L(h(\widetilde{\mathbf{x}}_t), \mathbf{y}))$$

This value is the initial perturbation for an iteration, which may be altered in the next step in order to mitigate concept prediction changes. If a concept prediction has already been changed, we terminate the algorithm as not successful.

In order to avoid changing the concept predictions, we calculate *sensitive concepts*, which are concepts that are close to being changed. Concept predictions change when the sigmoided values pass 0.5, and occur when the logit values change sign. Therefore, we identify a concept as sensitive if $g(\widetilde{\mathbf{x}})_j = \widetilde{\mathbf{c}}_j \in (-\gamma, \gamma)$, for some threshold value γ. For original concept predictions $g(\mathbf{x})_j > 0$, we want to avoid further lowering $g(\widetilde{\mathbf{x}}_t)_j$, and for original concept predictions $g(\mathbf{x})_j < 0$, we want to avoid further increasing $g(\widetilde{\mathbf{x}}_t)_j$.

We try to avoid further altering sensitive concept predictions by changing the inputs in the initial perturbation $\mathbf{p}_t$ that contributes to moving them closer to 0. For each sensitive concept c_j, we calculate which direction each pixel influence the prediction of that concept, $\mathbf{q}_j = \text{sign}(\nabla_{\widetilde{\mathbf{x}}_t} g(\widetilde{\mathbf{x}}_t)_j)$. We then alter the inputs of our initial perturbation if they lower concept logits for concepts that are initially predicted to be present, or increase concept logits that were initially predicted to be absent. We obtain a mask

$$M_{t,j} = \mathbb{I}_\beta(\hat{\mathbf{p}}_t \odot \mathbf{q}_j \neq \text{sign}(g(\mathbf{x})_j)) = \begin{cases} 1 & \text{if } \hat{\mathbf{p}}_t \odot \mathbf{q}_j \neq \text{sign}(g(\mathbf{x})_j), \\ \beta & \text{otherwise,} \end{cases}$$

where $\odot$ is elementwise multiplication and the indicator function $\mathbb{I}_\beta$ works elementwise and returns 1 if the condition is true, and $\beta \in (-1, 0)$ if not. Given M sensitive concepts, we denote the set of masks for all sensitive concepts at step t as $\{M\}_t = \{M_{1,t}, M_{2,t}, \ldots, M_{M,t}\}$. We then accumulate the masks $\{M\}_t$ such that $M_t = \min(\{M\}_t, 1)$ elementwise. This means that an element in M_t is β if that element was β in at least one of the masks in $\{M\}_t$, and 1 otherwise. If all the inputs in M_t equals β, we terminate with negative results. We finally obtain the perturbation $\mathbf{p}_t = \hat{\mathbf{p}}_t \odot M_t$.

The remaining steps are applying the perturbation, projecting the new image down on the epsilon-ball around the initial image, and clamping the values so that they represent

valid pixel values. The updated perturbed image is set to $\widetilde{\mathbf{x}}_t + \alpha \mathbf{p}_t$ for some step length α. We denote the ℓ_∞ norm projection as $\Pi_{[\mathbf{x}-\epsilon,\mathbf{x}+\epsilon]}(\widetilde{\mathbf{x}}_t + \alpha \mathbf{p}_t)$. This projection essentially makes sure that no single pixel value deviates from the original image more than ϵ. The final update is obtain after clamping, $\widetilde{\mathbf{x}}_{t+1} = \text{clamp}(\Pi_{[\mathbf{x}-\epsilon,\mathbf{x}+\epsilon]}(\widetilde{\mathbf{x}}_t + \alpha \mathbf{p}_t), x_{min}, x_{max})$. The valid range of pixel values $[x_{min}, x_{max}]$ depends on the normalisation of the images, since the perturbations are added on the transformed images.

7.2 Testing the Algorithm on the CUB and ConceptShapes Datasets

We demonstrate adversarial concept attacks on both ConceptShapes and on CUB. We train a CBM with hyperparameters found from hyperparameter optimization (see Section 8.1). We run a grid search to find good hyperparameters for the attacks, and then run the adversarial concept attack algorithm on all the images in the test set. We use the 10-class 5-concept and the 21 class 9-concept ConceptShapes datasets, both with $s = 98$. We use the full CUB dataset and subsets of 250 images from each class in the ConceptShapes datasets. For all datasets, we used the CBM trained with a soft bottleneck. Visualisations of the attacks can be seen in Figure 7.1 and Figure 7.2.

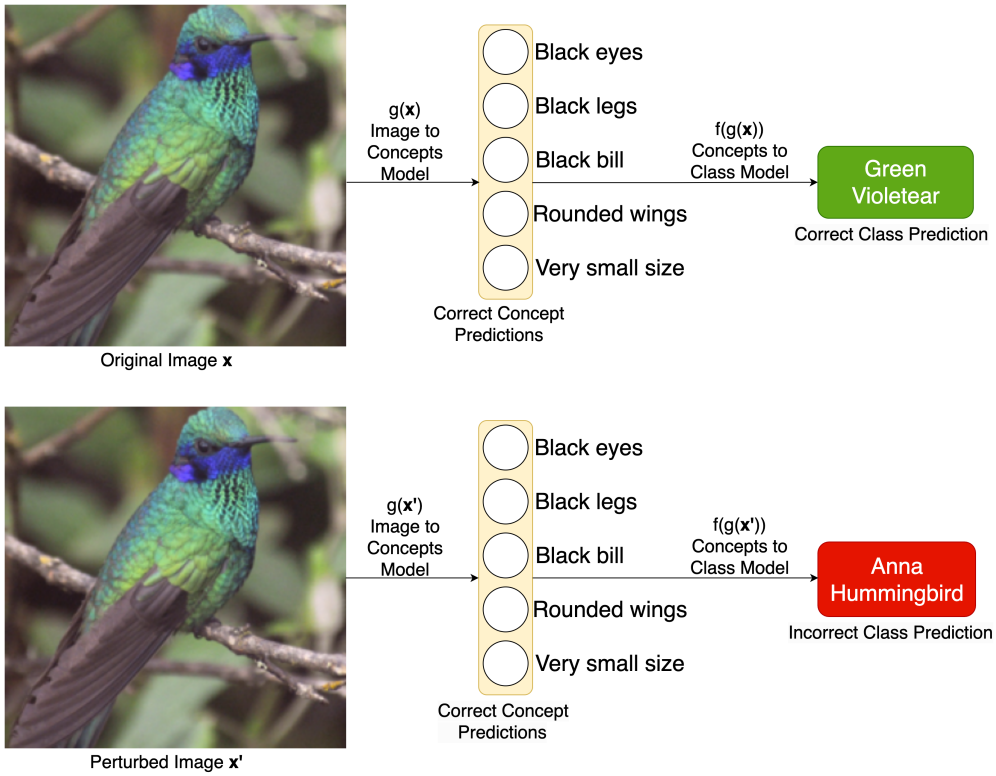


Figure 7.1: **Adversarial Concept Attack.** The image is perturbed in a way that makes the model predict the same concepts, but different classes. With Algorithm 1, we were able to perturb 57.4% of images in the CUB test-set this way. The perturbations are indistinguishable for humans. The model had 100% concept accuracy for this image, but this number was between 85%-98% for most images.

The adversarial concept attacks had a 57.4% success rate for CUB, 35.4% for the 10-class ConceptShapes dataset and 26.6% for 21 class ones. We also tested with PGD, where we aborted if the concept prediction changed. The details are in Table 7.1, and

Algorithm 1 Adversarial Concept Attack Algorithm

Result: Perturbed image $\widetilde{\mathbf{x}}$ of $\mathbf{x}$, such that CMB h misclassifies $\widetilde{\mathbf{x}}$, but the concept predictions are the same for $\widetilde{\mathbf{x}}$ and $\mathbf{x}$, or 0 for failed run.
Input:
Input image $\mathbf{x} \in \mathbb{R}^d$.
Class label $y \in [1, \ldots, p]$.
CBM $h : \mathbb{R}^d \to \mathbb{R}^p$ with input-to-concept function $f : \mathbb{R}^d \to \mathbb{R}^k$, such that $\text{argmax}(h(\mathbf{x})) = y$.
Sensitivity threshold $\gamma \in (0, \infty)$.
Step length $\alpha \in (0, 1)$.
Deviation threshold $\epsilon \in \mathbb{R}^d$.
Max iterations $t_{max} \in \mathbb{N}_1$.
Gradient weight $\beta \in (1, 0]$.
Valid pixel range $[x_{\min}, x_{\max}]$
$\widetilde{\mathbf{x}}_0 \leftarrow \mathbf{x}$ ▷ Initialize adversarial example
$\hat{\mathbf{c}} = g(\mathbf{x})$ ▷ Calculate original concept logits
$\hat{\mathbf{c}}_b = \mathbb{I}(\sigma(\hat{\mathbf{c}}) > 0.5)$ ▷ Calculate original binary predictions
for $t = 0, \ldots, t_{\max}$ **do**
  $\widetilde{\mathbf{c}} \leftarrow g(\widetilde{\mathbf{x}})$
  $\widetilde{\mathbf{c}}_b = \mathbb{I}(\sigma(\widetilde{\mathbf{c}}) > 0.5)$ ▷ New concept predictions
  if $\widetilde{\mathbf{c}}_b \neq \hat{\mathbf{c}}_b$ **then** ▷ Check if concept predictions are changed
    return 0 ▷ Fail due to changed concept predictions
  end if
  $\hat{\mathbf{p}}_t = \text{sign}(\nabla_{\widetilde{\mathbf{x}}_t} L(h(\widetilde{\mathbf{x}}_t), \mathbf{y}))$ ▷ Calculate initial perturbation
  Initialize $M_t \in \mathbb{R}^d$ with all elements as ones ▷ Initialise gradient mask
  for $j \leftarrow 0, \ldots, k$ **do**
    if $\widetilde{c}_j$ in $(-\gamma, \gamma)$ **then** ▷ Identify sensitive concept
      $\mathbf{q}_j = \text{sign}(\nabla_{\widetilde{\mathbf{x}}_t} g(\widetilde{\mathbf{x}}_t)_j)$
      $M_{t,j} \leftarrow \mathbb{I}_\beta(\hat{\mathbf{p}}_t \odot \mathbf{q}_j \neq \text{sign}(g(\mathbf{x})_j))$ ▷ Concept mask for $\widetilde{c}_j$
      $M_t \leftarrow \min(M_t, M_{t,j})$ ▷ Update mask
    end if
  end for
  if All entries in M_t equals β **then**
    return 0 ▷ Fail due to all β mask
  end if
  $\mathbf{p}_t = \hat{\mathbf{p}}_t \odot M_t$ ▷ Calculate final perturbation
  $\widetilde{\mathbf{x}}' = \Pi_{[\mathbf{x}-\epsilon, \mathbf{x}+\epsilon]}(\widetilde{\mathbf{x}}_t + \alpha \mathbf{p}_t)$ ▷ Projection step
  $\widetilde{\mathbf{x}}_{t+1} = \text{clamp}(\widetilde{\mathbf{x}}', x_{\min}, x_{\max})$ ▷ Clamping step
  if $\text{argmax}(h(\widetilde{\mathbf{x}}_{t+1})) \neq y$) **then**
    return $\widetilde{\mathbf{x}}_{t+1}$ ▷ Succesful perturbation
  end if
end for
return 0 ▷ Fail due to max iterations exceeded

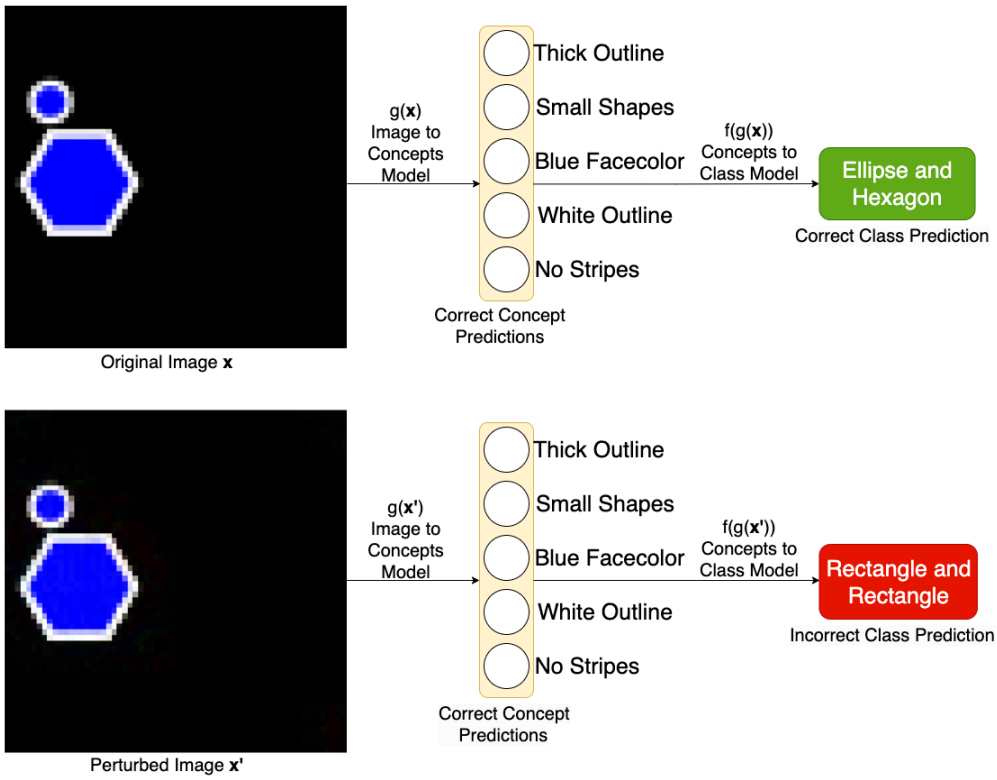


Figure 7.2: **Adversarial Concept Attack.** Algorithm 1 was successful at about a third of the images from ConceptShapes. Some of the perturbed images are easily distinguishable for humans. Most of the concepts were predicted with 100% accuracy.

some examples can be found in Table 7.2. We see that the difference of successful concept attacks is much bigger with the CUB dataset, which have many more concepts and pixels.

	Adversarial Concept Attack Success Rate	**PGD Success Rate**
CUB 112 concepts	57.4%	16.2%
ConceptShapes with 10 classes and 5 concepts	35.5 %	31.4%
ConceptShapes with 21 classes and 9 concepts	26.6%	22.5%

Table 7.1: Success rate of adversarial concept attacks on images in the test-sets. An attack is considered a success when the class prediction is changed, but not the concept predictions. First column uses Algorithm 1, while the second column uses projected gradient descent (PGD) [58].

The adversarial concept algorithm is sensitive with respect to its hyperparameters. We experienced that the two most important hyperparameters are the step length α and the sensitivity threshold γ. These are highly dependent. A lower α means that we change the images less in each step, and thus allows us to have a lower γ. If they both are too low however, we might not be able to change the class label prediction within the maximum amount of iterations. If α is high, we might drastically change the concept predictions in each step, so we need to also use a high value of γ to mitigate

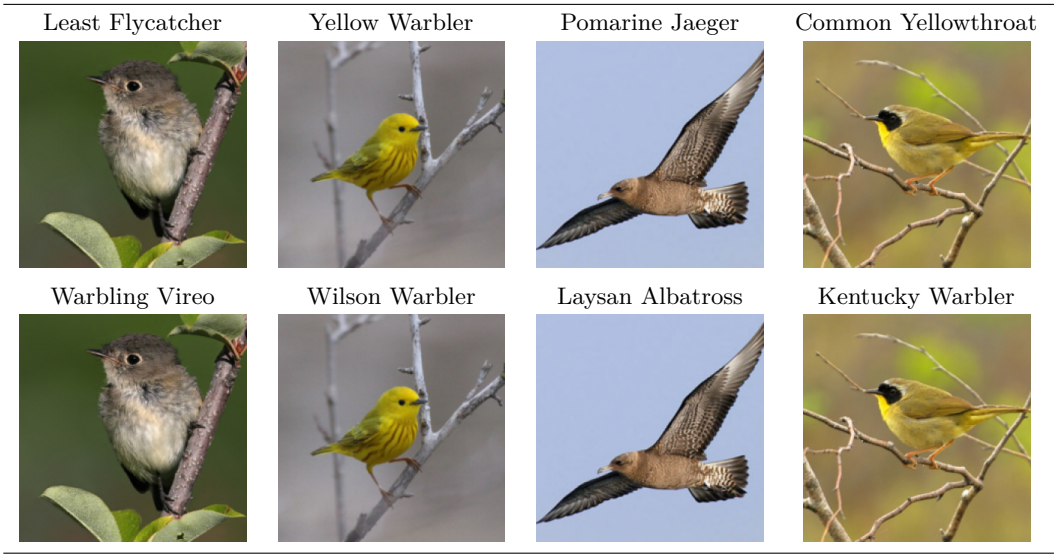


Table 7.2: **Examples of successful adversarial concept attacks on CUB**. First and second row show original predictions and images. The third and fourth row show new predictions and perturbed images. The original predictions were correct.

concept prediction change. However, this might lead to too many sensitive concepts, which cancel out too many inputs in the gradients.

We used a simple grid-search to find the hyperparameters α and γ. We optimise the proportion of images that are susceptible to the adversarial attacks, which means the class prediction gets changed, but not the concept predictions. We also experimented with different values of $\beta \in [-1, 0]$, but it only slightly changed the results. Lowering ϵ makes the attacks slightly less successful, but make the images look more similar to the original images. All of the perturbed CUB images look identical to the original ones, but one can spot clear differences with the ConceptShapes images. The details of the hyperparameters are in the Appendix A.3.

There probably exist algorithms with higher success rates, but our intent is only to demonstrate that such adversarial examples exist and are not rare. Our algorithm operates in a greedy fashion with respect to the initial perturbation, and only factors in the concept predictions later. It might be possible to calculate a more precise weighted perturbation of both goals, instead of first making one based on just the classes, then reversing the directions that negatively affects the concepts. It might also be possible to backtrack if a concept prediction is changed. In [30], an algorithm that does not change the class prediction, but does change the concept predictions is proposed. In case of a concept prediction change in our algorithm, one might run this to backtrack the concept predictions without immediately terminating. While we believe this outlines the possibility for more accurate attacks, our intent is simply demonstrating that they exist.

7.3 Are Adversarial Concept Examples a Problem for CBM's Trustworthiness?

The existence of these adversarial concept attacks substantially harms the promised interpretable qualities of CBMs. Since it has been abundantly demonstrated that DL models are unstable and susceptible to adversarial attacks [6], [8], [50]–[55], it might not

be surprising that CBMs are as well. However, most DL models only provide predictive power, not interpretability. The key quality of CBMs is that their class predictions can be interpreted by their concept predictions. Therefore, if similar concept predictions can lead to drastically different results, how can one trust CBM's interpretations?

The problem of adversarial concept attacks lies in the insensitivity between concept predictions and class predictions. Normal adversarial attacks show that predictions are highly sensitive to small changes in inputs. However, our adversarial concept attacks show that class predictions can be sensitive to small changes in the input, while the concept predictions are completely insensitive to the same changes. This is problematic since one should expect that interpretations are highly sensitive to the results they are interpreting.

We can also see the results in light of the *false structure* conjecture proposed in [7]. It suggests that DL models are unstable due to not classifying what they are meant to, but instead finding some highly correlated, but unstable false structures. Therefore, efforts to explain or interpret the models actually explain the false structures instead of the task. In this light, CBM interprets the false structure that it is actually learning, which is probably meaningless to humans.

Because of these limitations, we suggest that future work on concept-based models with the intent of improving interpretability should address this issue. Interpretable models should either be somewhat resistant to these kinds of attacks, or have a sufficient explanation for why the existence of the attacks does not limit their interpretability.

Chapter 8

Experimental Setup for Performance Evaluation

We conduct various experiments to assess the performance of the hybrid concept-based models. This is done on both the CUB dataset and many variations of the ConceptShapes datasets. We test our proposed CBM-Res, CBM-Skip and SCM against three benchmark models, which consists of a standard CNN, an ordinary CBM and an oracle model, which uses true concept labels to predict during testing. We train on different subsets of the data, and record the test accuracies. We also look at how the concept-based models learn the concepts. We run every experiment both with a soft and a hard bottleneck layer in the concept-models.

The code was made in Python with PyTorch [78] as the DL library. We used Python version 3.9.5, Torch version 2.0.1 and TorchVision version 0.15.2. We trained on the University of Oslo's USIT ML nodes [75], which mostly consists of RTX2080TIs. Because of the low resolution of the ConceptShapes dataset, the bottleneck for computation was running many different hyperparameter settings, not the training. We therefore enjoyed speeding up the computations by running different datasets and hyperparameter settings on different devices in parallel.

All of the runs are seeded and can be recreated with the openly available code at https://github.com/Tobias-Opsahl/TobiasaoThesis.

8.1 Performance Evaluation Setup

We now give an overview of the setup for the performance experiments in detail, and describe differences for the CUB and ConceptShapes datasets afterwards. The parts of the experiments that are similar for the datasets are covered in this section, while the subsets sampling and architectures are described in the next sections.

In order to benchmark the models we propose, we test them against the following three models:

1. **Standard model:** We construct an ordinary CNN model that does not train on the concepts. The choice of architecture and model capacity depends on the dataset and task. Since this is the type of model that is common to use in a computer vision context, we refer to it as the *standard model*.

2. **Vanilla CBM:** We also benchmark against a normal CBM [23]. This is to explore if the hybrid architectures in our models will give better performance than a previously proposed concept-based model.

3. **Oracle:** We also construct an *oracle* model, which uses the true concept labels both at training and test time. This is to see how much predictive power we can get from the concept labels alone.

We construct the standard model and the vanilla CBM such that they have approximately the same amount of trainable parameters as the hybrid concept-based models. The number of parameters are shown in Table 8.1.

Model	Total Parameters	Trainable Parameters	Frozen Parameters
ConceptShapes models with 10 classes and 5 concepts			
Standard model	139,578	139,578	0
Vanilla CBM	137,583	137,583	0
CBM-Res	138,527	138,527	0
CBM-Skip	138,399	138,399	0
SCM	152,849	152,849	0
ConceptShapes models with 21 classes and 9 concepts			
Standard model	139,941	139,941	0
Vanilla CBM	138,053	138,053	0
CBM-Res	139,758	139,758	0
CBM-Skip	139,614	139,614	0
SCM	167,579	167,579	0
CUB models			
Standard model	11,425,032	248,520	11,176,512
Vanilla CBM	11,371,880	195,368	11,176,512
CBM-Res	11,416,952	240,440	11,176,512
CBM-Skip	11,428,088	251,576	11,176,512
SCM	11,786,488	609,976	11,176,512

Table 8.1: **Amount of parameters in the different models**. There are many variations of the ConceptShapes datasets, here we show the one with the fewest parameters (top) and the one with the most parameters (middle). The ConceptShapes models are trained from scratch, hence no frozen parameters. The CUB models uses a frozen ResNet18 [76] model.

In order to test how the models perform with different amounts of data, we split each dataset in many subsets. For each of the subset and for each model, we run a grid search in order to find hyperparameters. We then train the models and measure the test-set accuracy. This is averaged over many runs, where each run uses a different seed for the model's weight initialisations and the drawing of the subsets. We used 10 runs for ConceptShapes and 3 for CUB.

We use the *Misprediction overlap* (MPO) metric from [24] to measure the quality of the concept predictions. The metric calculates the proportion of data points that had m or more concept mispredictions. For concept labels $C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N\}$ and concept predictions $\hat{C} = \{\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_N\}$, this becomes:

$$MPO(C, \hat{C}; m) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(E(\mathbf{c}_i, \hat{\mathbf{c}}_i) \geq m\right) \tag{8.1}$$

Where $\mathbb{I}$ is the indicator function from Equation (2.1) and $E(\mathbf{c}_i, \hat{\mathbf{c}}_i) = \sum_{j=1}^{k} \mathbb{I}(c_{i,j} \neq \hat{c}_{i,j})$, where k is the amount of concepts. We then plot the MPO for values $m \in [1, 2, ..., k]$.

Recall that the concept-based models' loss function is $\sum_{i=1}^{N} [L_Y(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \lambda L_C(\hat{\mathbf{c}}_i, \mathbf{c_i})]$. We let L_Y be the cross entropy loss function (Equation 2.2). Since the concepts may have many positive elements, we use an elementwise *binary cross entropy* function for L_C. This can be written as:

$$L_C(\hat{\mathbf{c}}, \mathbf{c}) = -\sum_{j=1}^{k} [c_j \log(\hat{c}_j) + (1 - c_j) \log(1 - \hat{c}_j)]$$

Where the predictions $\hat{\mathbf{c}}$ have been passed through a sigmoid function.

We construct two oracle models. One is a logistic regression model, and one is a neural network with one hidden layer, where the hidden layer has as many nodes as there are concepts. The oracle models and the standard model are trained with the cross entropy loss function (Equation 2.2).

In order to improve generalisation beyond the training set, we added dropout [48] after activation of the base-CNN output in the concept-based models, and at a corresponding layer in the standard model. When using a soft bottleneck, we used a sigmoid activation function after the bottleneck layer (except for CBM on CUB, which used no activation function). Every other layer uses a ReLU activation function.

The hyperparameter search is performed with a grid search. It is run for every model, for every subset of every dataset. All models search for the learning rate. The concept-based models also search for the concept weight λ. Since the standard model does not use this parameter, we search for an exponential learning rate decay parameter instead, so all the models have the same amount of hyperparameter trials. In the ConceptShapes datasets, we also search for a dropout probability. We excluded the oracle models from the hyperparameter searches.

We also tried to tune the hyperparameters with more advanced statistical methods, by sampling with Tree-structured Parzen Estimation (TPE) sampler [79] and pruning searches with the use of the Optuna library [80]. However, even though we were able to try a wider range of values on more hyperparameters, the results were poorer and often led to inconsistent results. The details are covered in the Appendix A.4.

We use the Adam optimiser for all models [47]. We apply an exponential learning rate decay parameter after every 10 epochs in ConceptShapes, and after every 5 epochs in CUB. The standard model tunes this decay parameter, while the other models use 0.7. The details not covered here can be found in the source code, which is carefully documented.

8.2 CUB Experiments

8.2.1 CUB Subsets

We use six different subsets for the CUB dataset. The dataset consists of $N = 11788$ images of $p = 200$ classes, with a 50%-50% train-test split. We use 20% of the training images for validation. This means that each class has about 30 images used for training and validation. We try subsets of $[1000, 2000, 3000, 4000, 5000, 5894]$ used for training and validation, where 5894 corresponds to the full dataset. Each subset is balanced with respect to the classes, so the 2000 image-subset contains 10 images from each class.

8.2.2 CUB Models

We use a pre-trained Resnet18 [76] trained on imagenet [57] as our base-CNN on the CUB dataset. We overwrite the classification layer with a linear layer with 256 nodes. We make this new layer trainable, and freeze all the other layers in the pre-trained model. For the standard model, we attach one more hidden layer with 256 nodes, so that it has about the same number of parameters as the concept-based models. In the vanilla CBM, we have one hidden layer with 112 nodes, the number of concepts, after the bottleneck layer.

In the CBM-Skip, we set $r = 128$, the number of nodes of the linear layer after the bottleneck layer. The SCM has one linear layer with 256 nodes after the base-CNN. We attach three concept layers to the last three convolutional blocks of the pre-trained model, and two after the two linear layers at the end. Since the dimensionality is so high from the convolutional blocks, we perform average pooling before passing it through the concept layers. All of the models have a classification layer at the end with 200 nodes.

8.3 ConceptShapes Experiments

8.3.1 Datasets

Many different configurations of the ConceptShapes datasets were used. To start off, we examine the effect of different values for s, which control the correlation between the concepts and the target classes. Recall that $s = 50$ results in the concepts being completely random, not depending on the classes, and $s = 100$ makes every image of a given class have the exact same concept labels.

We test different values of s on the 10-class 9-concept ConceptShapes dataset. We construct datasets for values of s in $[50, 60, 70, 80, 98, 100]$. We generate 1000 images for each class, resulting in 10000 images per dataset. The global test set consists of 2000 images. To measure performance on different dataset sizes, we train and test on five different subsets of the data. We draw subsets of sizes in $[500, 1000, 1500, 2000, 2500]$ for training and validation, which are split respectively at 60%-40%. The subsets are drawn balanced with respect to the classes, so the 500 subset have 50 images from each class. All of the subsets use the same global test set.

We hypothesise that the hybrid concept-based models will perform worse than the standard model when $s = 50$, and perform better when s is high. At $s = 50$, the concepts have no correlation with the classes, so the hybrid concept-based models will only have more noise to work with. We will use an oracle model to see how much information that lies in the concepts. At this value of s, the vanilla CBM should perform equal to random guessing if there is no concept leakage. At high values of s, the concept-based models have more useful data to work with, and should therefore perform better than the standard model. It is not clear how the vanilla CBM will perform compared to the standard CNN, since even though it is trained with the concept labels, it also has a big restriction in that all information is going through the bottleneck layer. We also expect that higher levels of s will make the standard model perform better than with lower values of s. Even though it is not trained directly on the concept, the complex nature of DL models will likely benefit from the correlation between the concepts and the classes.

After testing different values of s, we try different amounts of classes and concepts. We use $[10, 15, 21]$ amount of classes, with 5 and 9 concepts, resulting in six main datasets. We set $s = 98$ and $s = 100$ to explore how the models perform when the

concepts should be useful. The diverse settings for our datasets will help verify if results are consistent or dependent on specific settings.

For all of the datasets, we test on five different subsets. They have 50, 100, 150, 200 and 250 images used for training and validation for each class. We split training and validation at a 60%-40% ratio. We use the full test-set in all of the subsets.

8.3.2 ConceptShapes Models

For the base-CNN, we use a CNN with three convolutional blocks and one linear layer. They use a 3×3 kernel with 1-padding, and 8, 16, and 32 channels, respectively. After the convolutions we use ReLU activation function and 2×2 max pooling. We then use a linear layer with 64 nodes and ReLU activation function.

The standard model uses one more hidden layer of size 32. The vanilla CBM has one hidden layer after the bottleneck layer, with as many nodes as there are concepts, 5 or 9. For CBM-Skip, r is set to 16. The SCM attaches a concept layer after the three convolutional blocks. Since the dimensionality is high, we use dropout with a tunable dropout probability before inputting it to the concept layers. It also attaches one concept layer after the base-CNN's linear layer, and uses one more hidden layer with 64 nodes that also has a concept layer.

Chapter 9

Results of the Models' Performances

9.1 CUB Results

9.1.1 Hybrid Concept Models Perform the Best on CUB

In order to test the performances, we have trained and tested them at the CUB dataset. We have plotted the averaged accuracies for both hard and soft bottlenecks in Figure 9.1. We have omitted the oracle models, which used true concept labels to predict during testing, since they consistently achieved 100% test accuracies on all the subsets. We observe that the hybrid concept-based models achieve the highest test-set accuracies on all subsets, outperforming the benchmark standard CNN model and the vanilla CBM. The hybrid concept-based models perform well both with a hard and soft bottleneck, but the vanilla CBM performs much worse with a hard bottleneck.

The SCM performs slightly worse than the other hybrid concept models. This may be because the three first concept layers are connected to frozen output of layers that we do not train. Training the full model for a couple epochs might improve its performance.

9.1.2 None of the Models Learn the Concepts Properly

We now look at the MPO concept plots (from Equation 8.1) in Figure 9.2. The concept-based models achieve close to zero mispredictions only when the MPO parameter m surpasses 30, with a high ratio of misprediction for lower values of m. This suggests that the concept-based models do not sufficiently learn to predict the concepts, which is a huge flaw if the motivation is interpretability.

In the light of the shortcomings of the CUB dataset discussed in chapter 5, these MPO scores are not surprising. Many of the images contain ambiguous concepts, where the concept can not be inferred from the image. It is therefore unreasonable to expect the concept predictions to be accurate. Since the concepts are one-hot-encoded, they are sparse, and contain positive labels at a rate of 1 to 9. We therefore also tried training with a weighted binary cross entropy function, similarly to what was done in [23]. However, this made the results slightly worse.

We want to acknowledge that we think it is possible to achieve better performance, both for the benchmark models and for the hybrid concept models. Some steps towards improving them are replacing the pre-trained CNN ResNet18 with a pre-trained model with even higher model capacity, training for more epochs, and doing even more hyperparameter searches. However, the scope of this thesis is to do a fair comparison of the hybrid concept-based models and the benchmark models, not to make the best overall performing model.

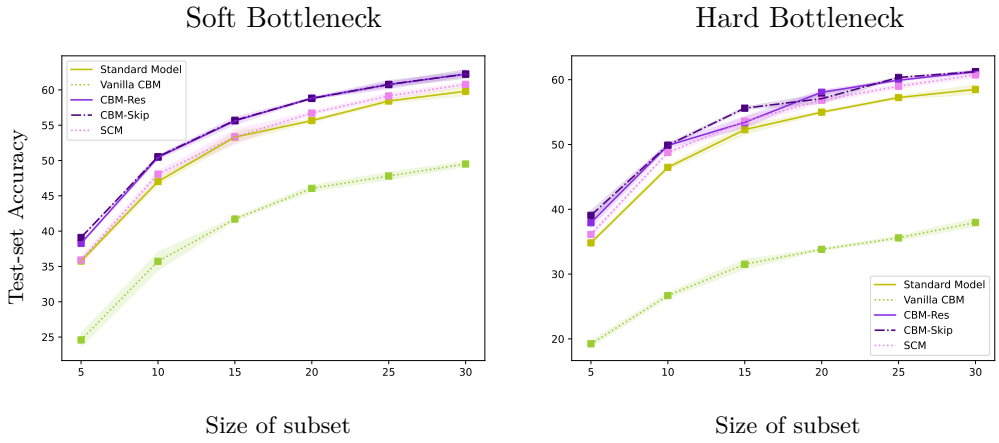


Figure 9.1: **Hybrid concept models outperform the benchmark models on all subsets of CUB.** The results are averaged over 3 runs, and include tight 95% confidence intervals. The x-axis denotes how many images that were included for training and validation for each of the 200 classes, where the rightmost point denotes the full training dataset. **Left:** Test accuracies using a soft bottleneck, where the hybrid concept-based models use a sigmoid activation function, and the vanilla CBM uses none. **Right:** Test accuracies using a hard bottleneck for all the concept-based models, where the concept predictions are rounded off to binary values.

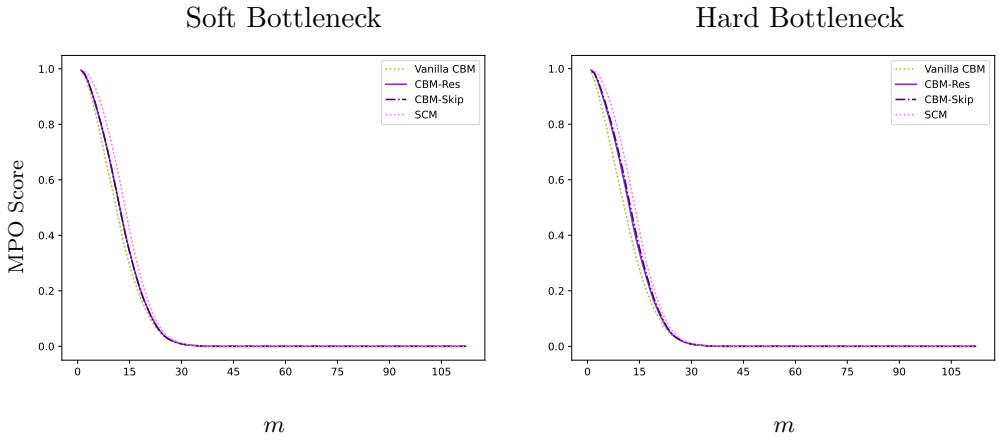


Figure 9.2: **Misprediction overlap (MPO) for the concept predictions on the full CUB dataset.** The x-axis shows the MPO parameter m. The results are averaged over three runs, and include 95% confidence intervals. We see almost no difference between soft and hard bottlenecks. The metric is from [24] and is defined in Equation 8.1.

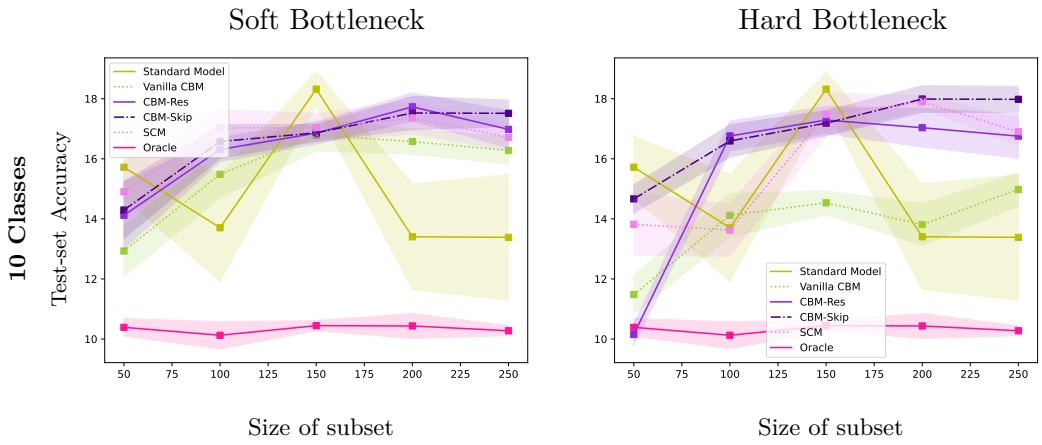


Figure 9.3: **Test accuracies with no correlation between the concepts and the classes**. The oracle model performs as good as random guessing. The vanilla CBM displays concept leakage, since it performs better than the oracle. The results are averaged over 10 runs and include 95% confidence intervals.

9.2 ConceptShapes

The two oracle models achieved very similar performance. Therefore, we only show the logistic regression oracle.

9.2.1 Results with No Correlation Between Concepts and Classes

We first investigate when there is no correlation between the concepts and the classes. Recall that our hypothesis was that the standard model would perform the best, since the concepts will essentially act as noise. The oracle model should perform equally to random guessing. If the CBM performs better than the oracle, it will display concept leakage. The results for the 10-class 9-concepts $s = 50$ dataset are plotted in Figure 9.3.

We see that the oracle model has about 10% test accuracy, which is what one would get with random guessing. The vanilla CBM performs substantially better. This is a sign of concept leakage, even with a hard bottleneck. The hybrid concept models have similar performance with soft and hard bottlenecks, but the vanilla CBM performs worse with a hard bottleneck.

The confidence intervals suggest that there is much variance over the runs, which is expected with small subsets. However, the standard model does not outperform our hybrid concept-based models as hypothesised. It appears that the hyperparameter optimization has given it more variable hyperparameters, since its performance does not increase with the sizes of subsets. Surprisingly, the hybrid concept models are more stable, and have the best performance in most of the runs. We suggest that this indicates that they are able to assign small weights to the bottleneck layer when the concepts are irrelevant for the target classes.

9.2.2 Hybrid Concept-based Models Perform Better than the Benchmark Models

We now look at the results when the concepts and classes are actually correlated, which is the scenario where we want to use the hybrid concept-based models. We look at

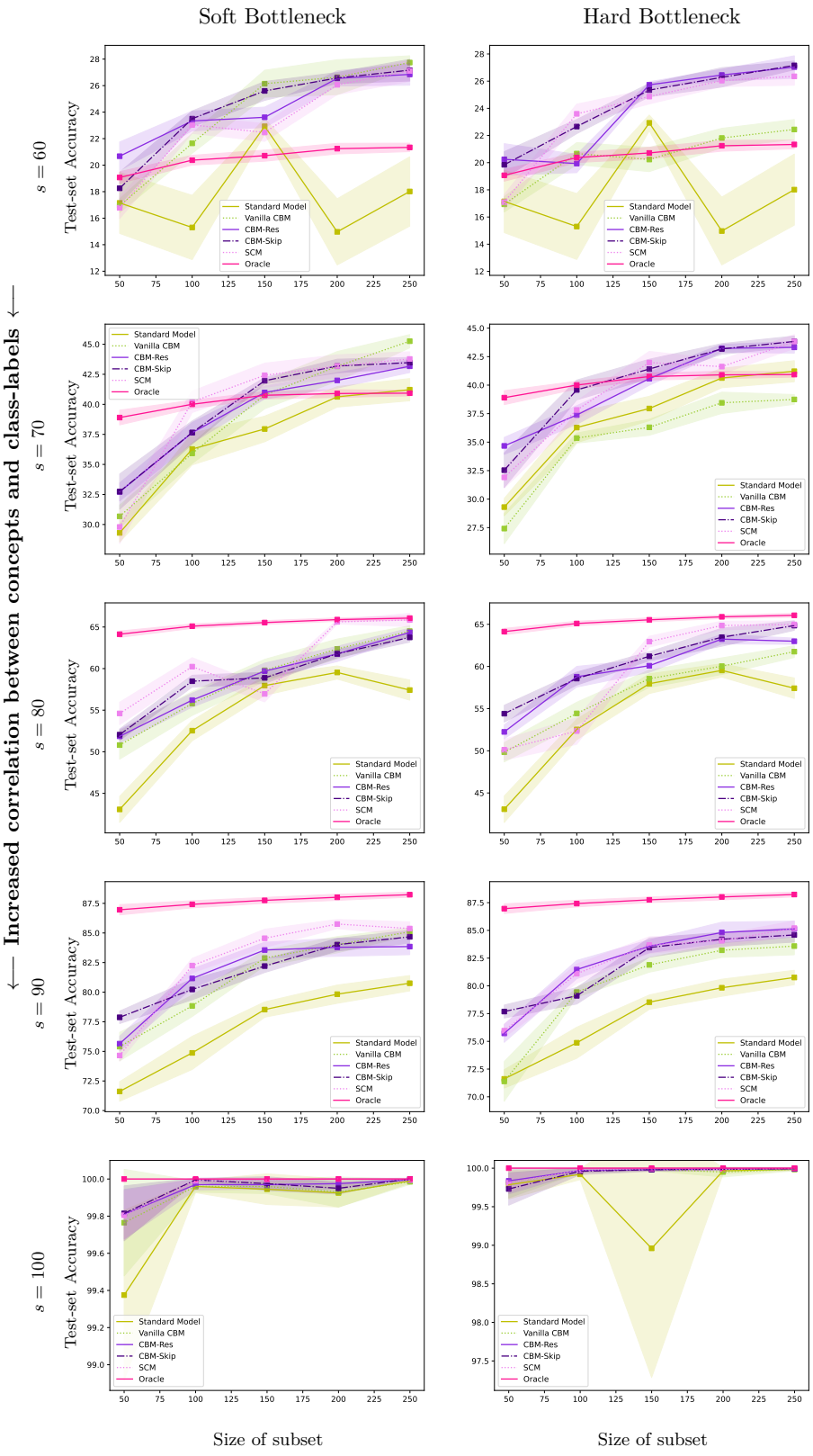


Figure 9.4: **Hybrid concept models perform the best with various amounts of correlations between concepts and classes**. The correlation increases with s. The scores are averaged over 10 runs and include 95% confidence intervals.

$s \in [60, 70, 80, 90, 100]$ for the 10 class and 9 concept datasets. The results are plotted in Figure 9.4.

We see that the hybrid concept models once again perform the best. The best one among the three varies over the datasets. The SCM performs much better here than for CUB, where it used a frozen pre-trained model.

The information in the concepts alone, indicated with the oracle model, steadily increases with s. With s equal to 60 and 70, the hybrid concept models perform better than the oracle, indicating that they are able to use the skip connections well. At $s = 100$, all of the models are able to easily get an almost perfect test-accuracy, indicating that this particular dataset is trivial.

The hybrid concept models are once again not influenced much by the soft or hard bottleneck, but the vanilla CBM is. With a soft bottleneck, the vanilla CBM is able to perform as well as the hybrid concept models when s is 60, 70, but not with higher correlations.

9.2.3 All the Models Learn to Predict the Concepts

We now investigate how the concepts are learned. We look at values of $s \in [50, 70, 80, 90, 100]$, where 60 is omitted to make the plot be formatted better. We only look at MPO scores for the biggest subset. The results are plotted in Figure 9.5.

We see that the models learn concepts better the more they correlate with the classes. However, they are still predicted well at lower values of s, even with $s = 50$. The vanilla-CBM and SCM learn the concept the best. This differs from the results on the CUB dataset. We suggest that it indicates that concept-based models are able to learn the concepts well when the concepts are not ambiguous and are somewhat related to the classes.

9.2.4 Similar Results can be Observed on All of the Dataset Variations

We now look at some other variations of the ConceptShapes datasets, and confirm that the results are consistent. We look at datasets with 10, 15 and 21 classes, with both 5 and 9 concepts, where $s = 98$. We include even more results in the Appendix A.2, where we try with $s = 100$.

The test-accuracies for 5 and 9 concepts are respectively in Figure 9.6 and Figure 9.7. Most importantly, we clearly see that the difficulty of the classification task increases with the amount of classes. This can be seen from the oracle models performance, and the overall accuracy of the models.

We see the same trends as earlier. The hybrid concept-based models perform the best, beating both benchmark models. They are not sensitive to the type of bottleneck. The vanilla CBM performs worse with a hard bottleneck, especially with 9 concepts. Interestingly, it does not show signs of concept leakage with 21 classes in Figure 9.6, and is the only model that performs worse than the oracle. The standard model performs worse than the hybrid concept-based models, and has much higher variance. This is seen by the wide confidence interval and the occasional drops in performance when the subsets sizes increase.

We also look at the respective concept MPO plots from the biggest subsets in Figure 9.8 and Figure 9.9. We see that all the concept-based models learn the concepts well. Only 10-15% of the test set images get predicted with one single mistake, and less than 2% gets predicted with two wrong concepts.

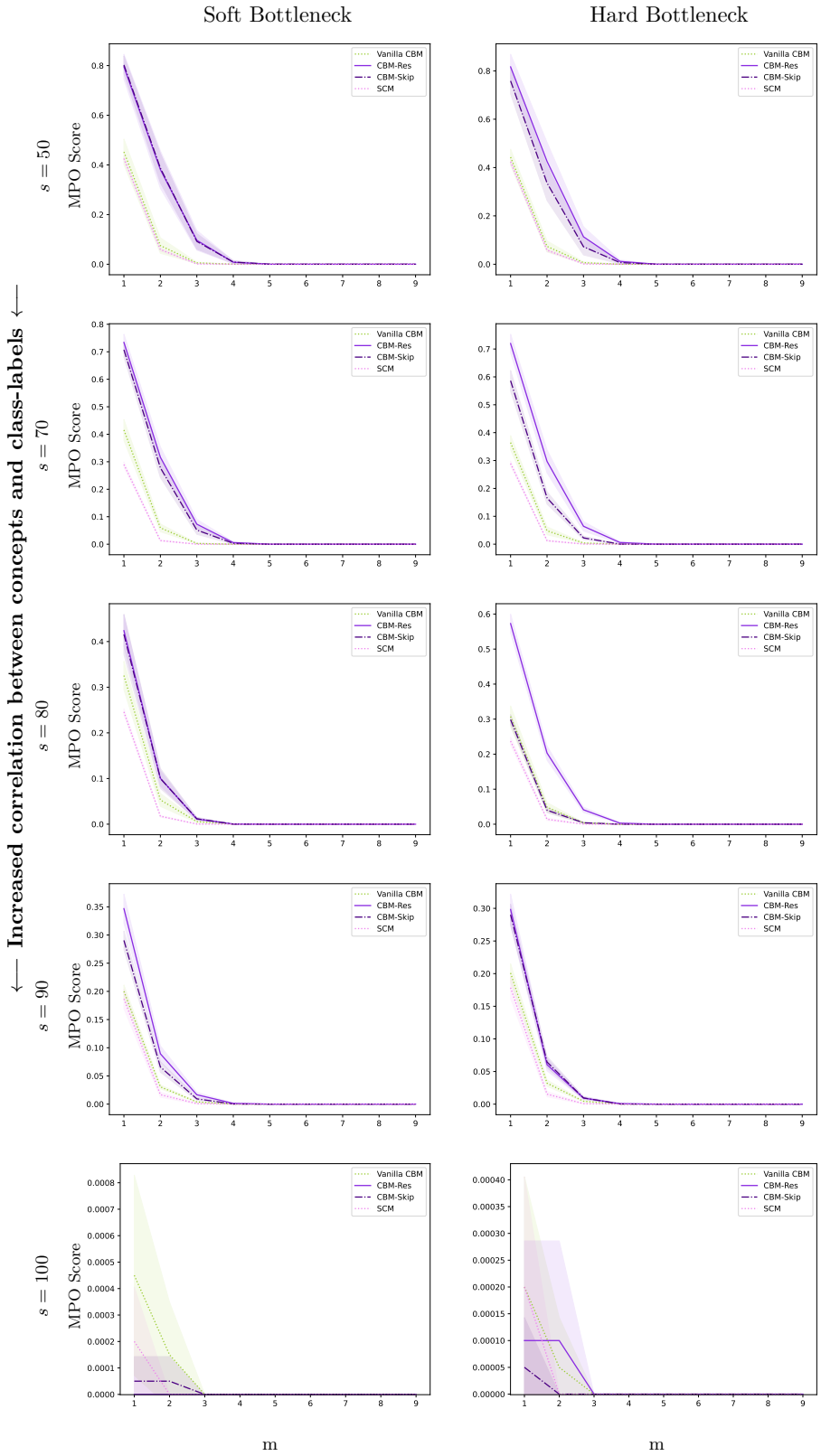


Figure 9.5: **MPO scores for concepts, over various correlations between concepts and classes**. The correlation increases with s. The scores are averaged over 10 runs and include 95% confidence intervals.

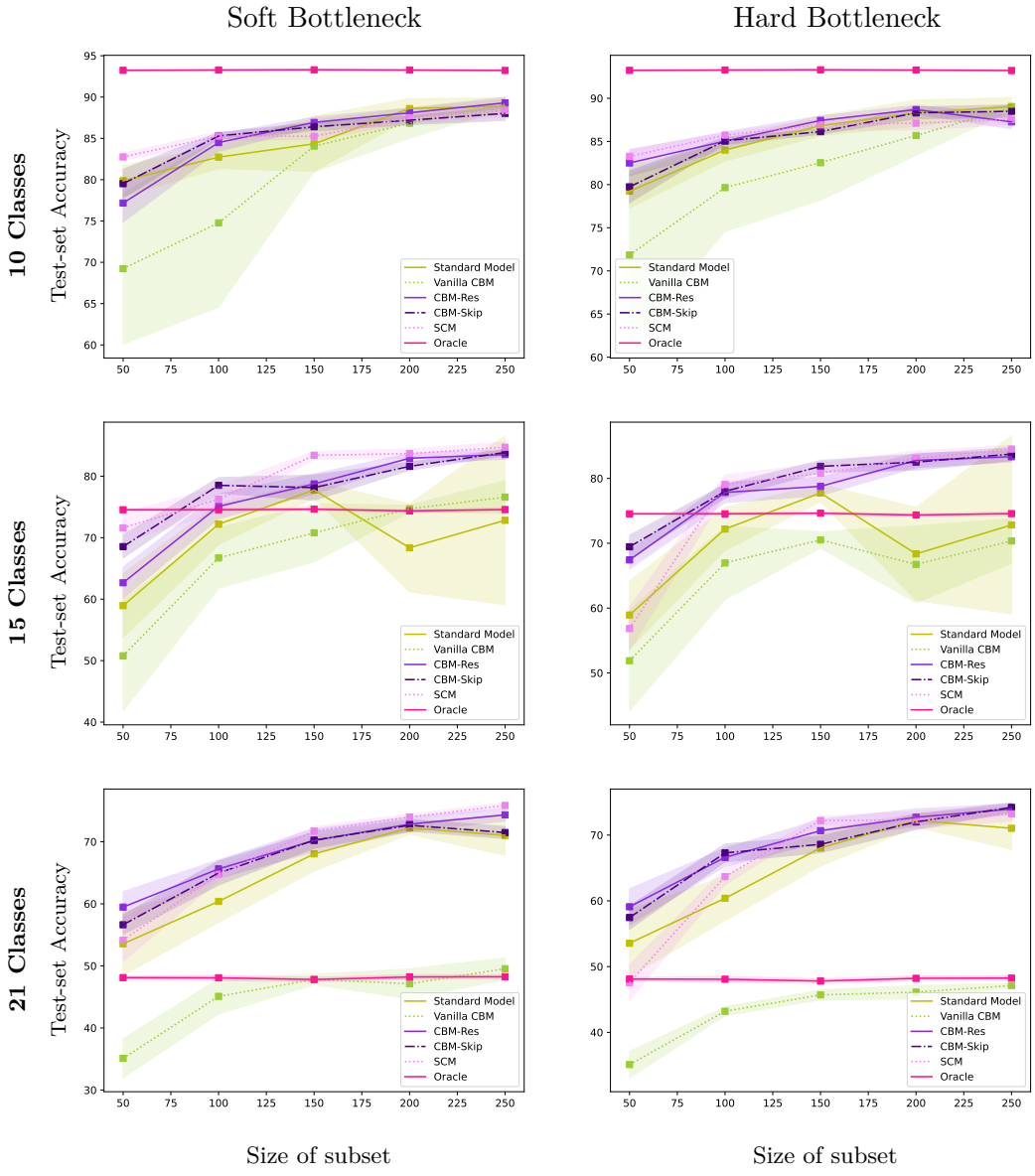


Figure 9.6: **Test accuracies for various classes with 5 concepts and s = 98**. The scores are averaged over 10 runs and include 95% confidence intervals.

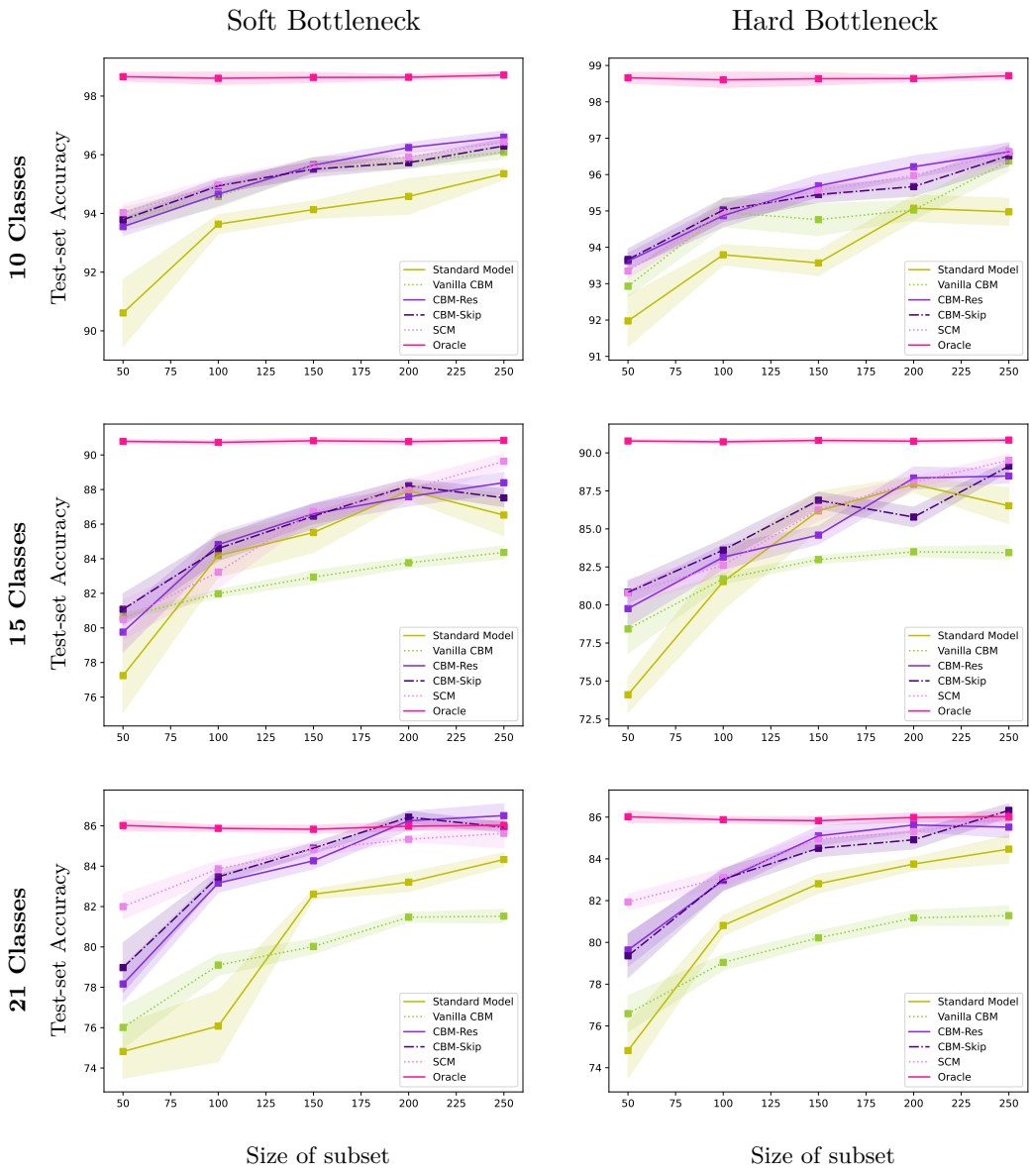


Figure 9.7: **Test accuracies for various classes with 9 concepts and s = 98.** The scores are averaged over 10 runs and include 95% confidence intervals.

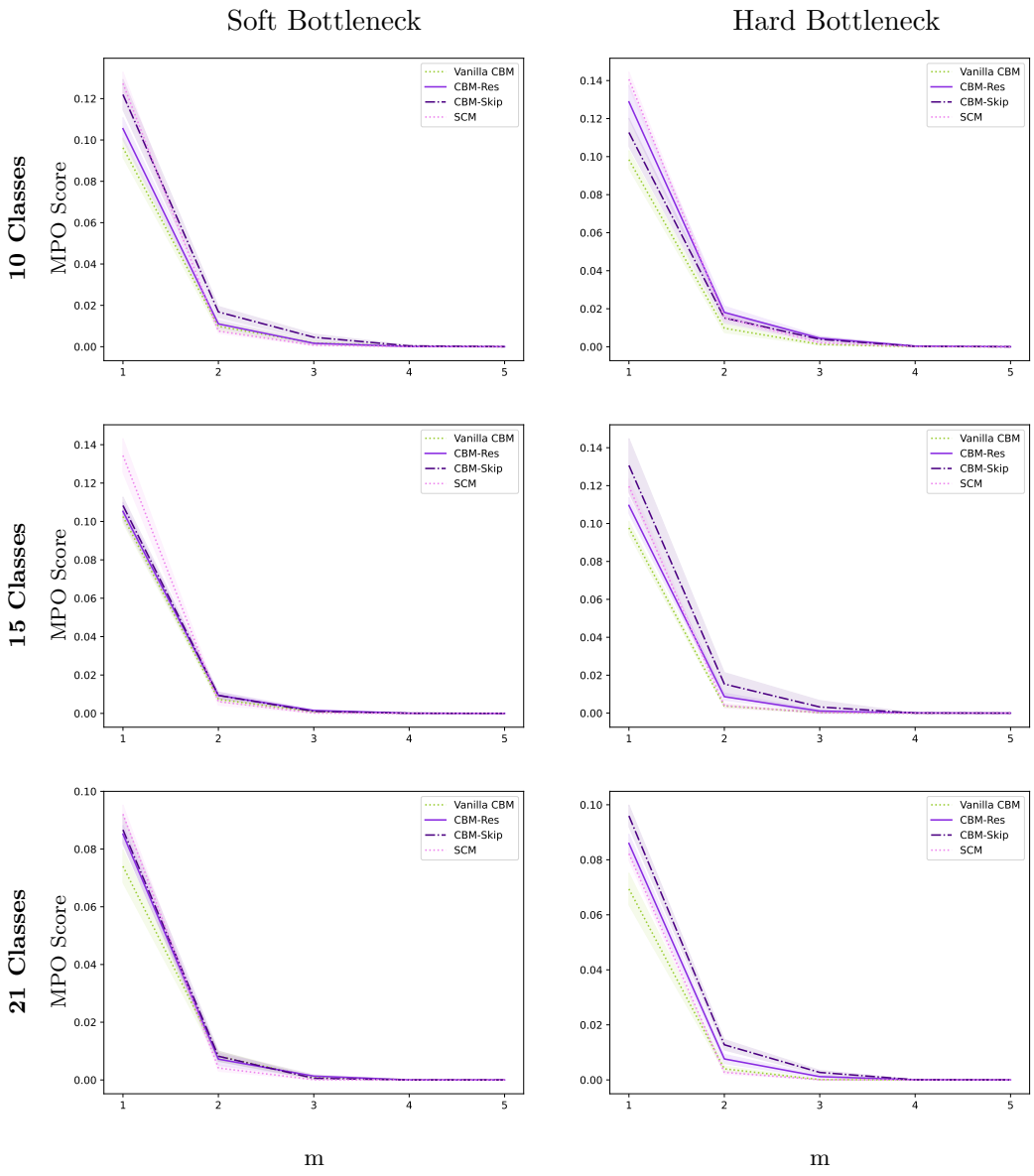


Figure 9.8: **MPO scores for 5 concepts and s = 98**. The scores are averaged over 10 runs and include 95% confidence intervals.

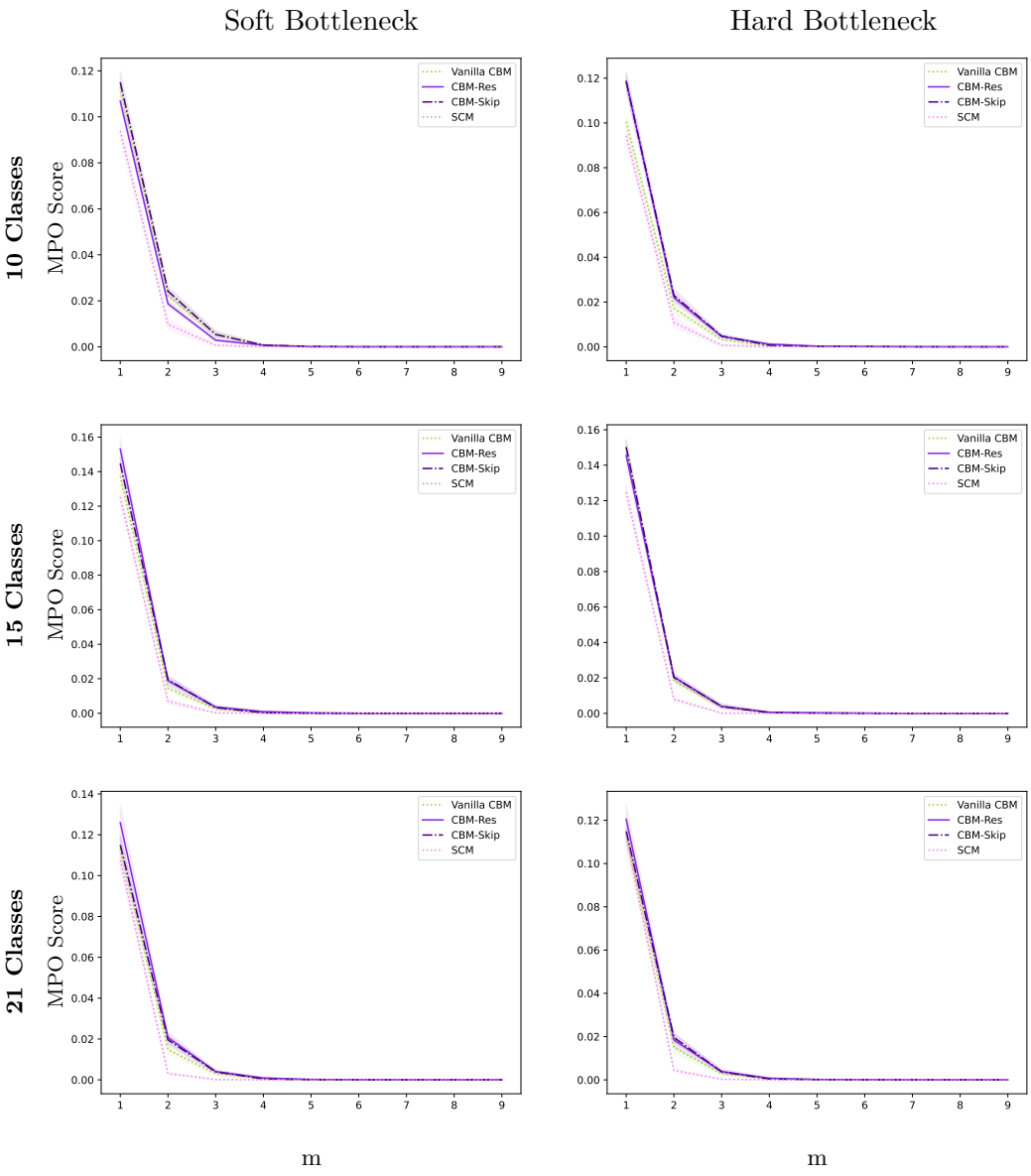


Figure 9.9: **MPO scores for 9 concepts and s = 98**. The scores are averaged over 10 runs and include 95% confidence intervals.

9.3 Summary of Performances

We believe that these experiments are evidence for an increase in performance when using hybrid concept-based models. They achieve a better test-set accuracy than both the standard CNN model and the CBM on the CUB dataset and the various variations of the ConceptShapes datasets. They are also more consistent, having tighter confidence intervals and a more steady gain in accuracy when the size of the subset increases compared to the benchmark models. They learn concepts as well as a vanilla CBM, and utilise hard bottlenecks better. They are also able to perform well when the correlation between the concepts and the classes is low or zero.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

To our knowledge, we are the first to propose concept-based models with the intent of improving performance. This differs from the motivation of interpretability from earlier work. The only hybrid model known by the authors is the CBM with unsupervised concepts [29], but this was proposed in order to make evidence for lack of interpretability, and its performance was not tested.

We proposed two novel concept-based neural network architectures and tested their performance. In order to do the testing in a controlled manner, we proposed a set of synthetic concept datasets called ConceptShapes. The datasets can be adjusted with respect to concept and class correlation, the amount of classes and the amount of concepts. We tested our models against a standard CNN, vanilla CBM and oracle models, which showed clear performance advantages of using the hybrid concept-based models. This was also confirmed on the popular CUB dataset.

We also demonstrated that adversarial concept attacks are highly effective on CBMs. Adversarial examples exist such that a CBM has different class predictions, even though the concept predictions are identical. This limits the trust from their interpretations. We suggest that future work with the motivation of interpretability should address this issue.

10.2 Future Work

We propose different directions of future work with hybrid concept-based models:

- **More advanced model architectures:** The models proposed work as a starting point for hybrid concept-based modelling. There are more possibilities to explore. Particularly, the *Concept Embedding Models* (CEM) [72] seems promising with respect to performance. The performance of CEMs can be benchmarked against the hybrid concept-models, and one could try to make a hybrid CEM. This was not done in the thesis due to the CEM being discovered only recently by the authors.

- **More concept datasets:** The usefulness of ML models partly comes from their ability to work on a vast amount of different datasets. We provide a starting point with our proposed ConceptShapes datasets, but we believe it is necessary to develop and test on more datasets in order to explore model performance. For the concept-based models, this might require new concept datasets.

- **Applications in deep reinforcement learning:** One particularly exciting possible direction to explore is how concept-based models will perform in the reinforcement learning setting. Similarly to other DL models, reinforcement learning has been successful in various tasks [42], [43], but are notorious for their data-hungry nature. Constructing reinforcement learning agents that also predict and train on concepts might help them learn quicker.

  This direction of future work is particularly interesting since concepts in reinforcement learning may be acquired easily. For an agent moving in space, concepts may be its rotation, velocity and position. These concepts are obtainable from simple calculations. A chess computer may use material advantage, the kings position and how close the pieces are to the centre as concepts. These metrics can be easily calculated with existing chess engines. This way, reinforcement learning does not have the limitation of requiring very specific concept datasets in order to develop concept-based models.

  Furthermore, this might be an interesting path for machine learning applications in computational neuroscience. Experiments on mice show that the neurons of the brain [81] are task specific to a certain extent, which differs from current artificial neural networks. Training some layers of the network on specific tasks (like with SCMs) might achieve more specification of nodes, which could allow for more accurate modelling of natural intelligence.

Bibliography

[1] F. Doshi-Velez and B. Kim, 'Towards a rigorous science of interpretable machine learning', *arXiv preprint arXiv:1702.08608*, 2017.

[2] Z. C. Lipton, 'The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery.', *Queue*, vol. 16, no. 3, pp. 31–57, 2018.

[3] C. Rudin, 'Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead', *Nature machine intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[4] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser *et al.*, 'Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai', *Information fusion*, vol. 58, pp. 82–115, 2020.

[5] European Commission. 'Europe fit for the Digital Age: Commission proposes new rules and actions for excellence and trust in Artificial Intelligence'. (2021), [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/IP_21_1682 (visited on 04/11/2023).

[6] C. Szegedy, W. Zaremba, I. Sutskever *et al.*, 'Intriguing properties of neural networks', *arXiv preprint arXiv:1312.6199*, 2013.

[7] L. Thesing, V. Antun and A. C. Hansen, 'What do ai algorithms actually learn?-on false structures in deep learning', *arXiv preprint arXiv:1906.01478*, 2019.

[8] V. Antun, F. Renna, C. Poon, B. Adcock and A. C. Hansen, 'On instabilities of deep learning in image reconstruction and the potential costs of ai', *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 088–30 095, 2020.

[9] Y. LeCun. 'Animals and humans get very smart very quickly with vastly smaller amounts of training data than current AI systems.' (Nov. 2023), [Online]. Available: https://www.linkedin.com/posts/yann-lecun_animals-and-humans-get-very-smart-very-quickly-activity-7133567569684238336-szrF?trk=public_profile_post_view (visited on 25/11/2023).

[10] K. Simonyan, A. Vedaldi and A. Zisserman, 'Deep inside convolutional networks: Visualising image classification models and saliency maps', *arXiv preprint arXiv:1312.6034*, 2013.

[11] M. Sundararajan, A. Taly and Q. Yan, 'Axiomatic attribution for deep networks', in *International conference on machine learning*, PMLR, 2017, pp. 3319–3328.

[12] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, 'Grad-cam: Visual explanations from deep networks via gradient-based localization', in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[13] A. Shrikumar, P. Greenside and A. Kundaje, 'Learning important features through propagating activation differences', in *International conference on machine learning*, PMLR, 2017, pp. 3145–3153.

[14] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt and B. Kim, 'Sanity checks for saliency maps', *Advances in neural information processing systems*, vol. 31, 2018.

[15] A.-K. Dombrowski, M. Alber, C. Anders, M. Ackermann, K.-R. Müller and P. Kessel, 'Explanations can be manipulated and geometry is to blame', *Advances in neural information processing systems*, vol. 32, 2019.

[16] A. Ghorbani, A. Abid and J. Zou, 'Interpretation of neural networks is fragile', in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 3681–3688.

[17] P.-J. Kindermans, S. Hooker, J. Adebayo *et al.*, 'The (un) reliability of saliency methods', *Explainable AI: Interpreting, explaining and visualizing deep learning*, pp. 267–280, 2019.

[18] M. Ghassemi, L. Oakden-Rayner and A. L. Beam, 'The false hope of current approaches to explainable artificial intelligence in health care', *The Lancet Digital Health*, vol. 3, no. 11, e745–e750, 2021.

[19] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas *et al.*, 'Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)', in *International conference on machine learning*, PMLR, 2018, pp. 2668–2677.

[20] A. Ghorbani, J. Wexler, J. Y. Zou and B. Kim, 'Towards automatic concept-based explanations', *Advances in neural information processing systems*, vol. 32, 2019.

[21] McGrath *et al.*, 'Acquisition of chess knowledge in alphazero', *Proceedings of the National Academy of Sciences*, vol. 119, no. 47, e2206625119, 2022.

[22] D. Alvarez Melis and T. Jaakkola, 'Towards robust interpretability with self-explaining neural networks', *Advances in neural information processing systems*, vol. 31, 2018.

[23] P. W. Koh, T. Nguyen, Y. S. Tang *et al.*, 'Concept bottleneck models', in *International conference on machine learning*, PMLR, 2020, pp. 5338–5348.

[24] D. Kazhdan, B. Dimanov, M. Jamnik, P. Liò and A. Weller, 'Now you see me (cme): Concept-based model extraction', *arXiv preprint arXiv:2010.13233*, 2020.

[25] Y. Sawada and K. Nakamura, 'Concept bottleneck model with additional unsupervised concepts', *IEEE Access*, vol. 10, pp. 41 758–41 765, 2022.

[26] E. Kim, D. Jung, S. Park, S. Kim and S. Yoon, 'Probabilistic concept bottleneck models', *arXiv preprint arXiv:2306.01574*, 2023.

[27] K. Chauhan, R. Tiwari, J. Freyberg, P. Shenoy and K. Dvijotham, 'Interactive concept bottleneck models', in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 5948–5955.

[28] A. Margeloiu, M. Ashman, U. Bhatt, Y. Chen, M. Jamnik and A. Weller, 'Do concept bottleneck models learn as intended?', *arXiv preprint arXiv:2105.04289*, 2021.

[29] A. Mahinpei, J. Clark, I. Lage, F. Doshi-Velez and W. Pan, 'Promises and pitfalls of black-box concept learning models', *arXiv preprint arXiv:2106.13314*, 2021.

[30] S. Sinha, M. Huai, J. Sun and A. Zhang, 'Understanding and enhancing robustness of concept-based models', in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 15 127–15 135.

[31] C. Wah, S. Branson, P. Welinder, P. Perona and S. Belongie, 'The caltech-ucsd birds-200-2011 dataset', California Institute of Technology, Tech. Rep., 2011.

[32] G. Menghani, 'Efficient deep learning: A survey on making deep learning models smaller, faster, and better', *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[33] E. Strubell, A. Ganesh and A. McCallum, 'Energy and policy considerations for deep learning in NLP', *arXiv preprint arXiv:1906.02243*, 2019.

[34] Numenta. 'AI is harming our planet: addressing AI's staggering energy cost'. (May 2022), [Online]. Available: https://www.numenta.com/blog/2022/05/24/ai-is-harming-our-planet/ (visited on 13/11/2023).

[35] R. Miotto, F. Wang, S. Wang, X. Jiang and J. T. Dudley, 'Deep learning for healthcare: Review, opportunities and challenges', *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2018.

[36] M. I. Razzak, S. Naz and A. Zaib, 'Deep learning for medical image processing: Overview, challenges and the future', *Classification in BioApps: Automation of Decision Making*, pp. 323–350, 2018.

[37] S. Bubeck *et al.*, 'Sparks of artificial general intelligence: Early experiments with gpt-4', *arXiv preprint arXiv:2303.12712*, 2023.

[38] X. Liu, K. Gao, B. Liu *et al.*, 'Advances in deep learning-based medical image analysis', *Health Data Science*, 2021.

[39] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems*, vol. 25, 2012.

[40] A. Graves, A.-r. Mohamed and G. Hinton, 'Speech recognition with deep recurrent neural networks', in *2013 IEEE international conference on acoustics, speech and signal processing*, Ieee, 2013, pp. 6645–6649.

[41] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning', *Nature*, vol. 521, no. 7553, p. 436, 2015.

[42] V. Mnih, Kavukcuoglu *et al.*, 'Human-level control through deep reinforcement learning', *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[43] D. Silver, Huang *et al.*, 'Mastering the game of go with deep neural networks and tree search', *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[44] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning representations by back-propagating errors', *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[45] I. Sutskever, J. Martens, G. Dahl and G. Hinton, 'On the importance of initialization and momentum in deep learning', in *International conference on machine learning*, PMLR, 2013, pp. 1139–1147.

[46] G. Hinton. 'Neural Networks for Machine Learning'. (Oct. 2012), [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 29/11/2023).

[47] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, 2014.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[49] R. Hamon, H. Junklewitz, I. Sanchez *et al.*, 'Robustness and explainability of artificial intelligence', *Publications Office of the European Union*, vol. 207, 2020.

[50] K. Eykholt, I. Evtimov, E. Fernandes *et al.*, 'Robust physical-world attacks on deep learning visual classification', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1625–1634.

[51] N. Carlini and D. Wagner, 'Audio adversarial examples: Targeted attacks on speech-to-text', in *2018 IEEE security and privacy workshops (SPW)*, IEEE, 2018, pp. 1–7.

[52] B. Liang, H. Li, M. Su, P. Bian, X. Li and W. Shi, 'Deep text classification can be fooled', *arXiv preprint arXiv:1704.08006*, 2017, https://arxiv.org/pdf/1704.08006.pdf.

[53] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam and I. S. Kohane, 'Adversarial attacks on medical machine learning', *Science*, vol. 363, no. 6433, pp. 1287–1289, 2019.

[54] I. J. Goodfellow, J. Shlens and C. Szegedy, 'Explaining and harnessing adversarial examples', *arXiv preprint arXiv:1412.6572*, 2014.

[55] A. Kurakin, I. Goodfellow and S. Bengio, 'Adversarial machine learning at scale', *arXiv preprint arXiv:1611.01236*, 2016.

[56] C. Szegedy, W. Liu, Y. Jia *et al.*, 'Going deeper with convolutions', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[57] Russakovsky *et al.*, 'Imagenet large scale visual recognition challenge', *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[58] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, 'Towards deep learning models resistant to adversarial attacks', *arXiv preprint arXiv:1706.06083*, 2017.

[59] S.-M. Moosavi-Dezfooli, A. Fawzi and P. Frossard, 'Deepfool: A simple and accurate method to fool deep neural networks', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[60] M. J. Colbrook, V. Antun and A. C. Hansen, 'The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and smale's 18th problem', *Proceedings of the National Academy of Sciences*, vol. 119, no. 12, e2107151119, 2022.

[61] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran and A. Madry, 'Adversarial examples are not bugs, they are features', *Advances in neural information processing systems*, vol. 32, 2019.

[62] J. Larson, S. Mattu, L. Kirchner and J. Angwin. 'How We Analyzed the COMPAS Recidivism Algorithm', ProPublica. (May 2016), [Online]. Available: https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm.

[63] A. Adadi, 'A survey on data-efficient algorithms in big data era', *Journal of Big Data*, vol. 8, no. 1, p. 24, 2021.

[64] Telefónica. 'A Fit-For-Purpose and Borderless European Artificial Intelligence Regulation'. (2022), [Online]. Available: https://www.telefonica.com/en/communication-room/blog/a-fit-for-purpose-and-borderless-european-artificial-intelligence-regulation/ (visited on 04/11/2023).

[65] New York Times. 'E.U. Agrees on Landmark Artificial Intelligence Rules'. (2023), [Online]. Available: https://www.nytimes.com/2023/12/08/technology/eu-ai-act-regulation.html (visited on 14/12/2023).

[66] S. M. Lundberg and S.-I. Lee, 'A unified approach to interpreting model predictions', *Advances in neural information processing systems*, vol. 30, 2017.

[67] Y. LeCun, 'The mnist database of handwritten digits', *http://yann. lecun. com/exdb/mnist/*, 1998.

[68] A. Abid, M. Yuksekgonul and J. Zou, 'Meaningfully debugging model mistakes using conceptual counterfactual explanations', in *International Conference on Machine Learning*, PMLR, 2022, pp. 66–88.

[69] Z. Chen, Y. Bei and C. Rudin, 'Concept whitening for interpretable image recognition', *Nature Machine Intelligence*, vol. 2, no. 12, pp. 772–782, 2020.

[70] S. Shin, Y. Jo, S. Ahn and N. Lee, 'A closer look at the intervention procedure of concept bottleneck models', *arXiv preprint arXiv:2302.14260*, 2023.

[71] M. Yuksekgonul, M. Wang and J. Zou, 'Post-hoc concept bottleneck models', *arXiv preprint arXiv:2205.15480*, 2022.

[72] M. Espinosa Zarlenga, P. Barbiero, G. Ciravegna *et al.*, 'Concept embedding models: Beyond the accuracy-explainability trade-off', *Advances in Neural Information Processing Systems*, vol. 35, pp. 21 400–21 413, 2022.

[73] M. Nevitt, D. Felson and G. Lester, 'The osteoarthritis initiative', *Protocol for the cohort study*, vol. 1, 2006.

[74] E. Pierson, D. M. Cutler, J. Leskovec, S. Mullainathan and Z. Obermeyer, 'An algorithmic approach to reducing unexplained pain disparities in underserved populations', *Nature Medicine*, vol. 27, no. 1, pp. 136–140, 2021.

[75] University Centre for Information Technology, University Of Oslo, *Machine learning infrastructure (ml nodes)*, Norway, 2023.

[76] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[77] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, 'Densely connected convolutional networks', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[78] A. Paszke, S. Gross, F. Massa *et al.*, 'Pytorch: An imperative style, high-performance deep learning library', *Advances in neural information processing systems*, vol. 32, 2019.

[79] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, 'Algorithms for hyper-parameter optimization', *Advances in neural information processing systems*, vol. 24, 2011.

[80] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, 'Optuna: A next-generation hyperparameter optimization framework', in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[81] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser and E. I. Moser, 'Microstructure of a spatial map in the entorhinal cortex', *Nature*, vol. 436, no. 7052, pp. 801–806, 2005.

Appendix A

Appendix

A.1 ConceptShapes Dataset Details

The intervals of the shapes' outline thickness was between 1 and 1.2 for the shapes with the "thick-outline" concept, and between 0.2 and 0.5 for the ones without it. Rectangles used height and width to determine its size when creating, while the rest of the shapes used radius. A shape with the "big-figure" concept would have its radius drawn from 2.8 and 3.3, or height and width independently between 2 and 2.5 if it was a rectangle. If it did not have the "big-figure" concept, the radius would be drawn from 1.5 and 2, or height and width between 1 and 1.2 if it was a rectangle. Despite being chosen to make all the shapes more or less the same size, the rectangles and circles ended up substantially smaller than the triangles, pentagons, hexagons and wedges, for both big and small figures. However, since they are chosen from two clearly distinct intervals, we did not consider this as an issue.

The following tables show which classes that got assigned a high probability for which concepts for the 15-class and 21-class datasets. For the five-concept datasets, the background concepts are not used.

	Classes														
Concept	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Thick Outline	X		X		X		X		X		X		X		X
Big Figures	X	X	X	X	X	X	X	X							
Blue Facecolor		X			X	X		X	X			X		X	
Red Outline	X					X	X			X					
Stripes				X			X			X			X		
Magenta Upper Bg.		X				X			X	X		X			X
Indigo Lower Bg.		X		X				X					X	X	
Upper Bg. Stripes					X			X		X					X
Lower Bg. Stripes								X	X	X				X	

Table A.1: Overview of which classes that would get assigned a high probability for which concepts, in the 15-class ConceptShapes datasets. An X indicates high probability (s%), while the absence of an X represents a low probability ((100 - s)%).

	Classes																				
Concept	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Thick Outline	X		X		X		X		X		X		X		X		X		X		X
Big Figures	X	X	X	X	X	X	X	X	X	X	X	X									
Blue Facecolor		X			X	X		X	X			X		X		X			X		
Red Outline				X		X	X			X							X				
Stripes				X			X		X	X			X	X		X		X	X		
Magenta Upper Bg.			X				X											X	X	X	
Indigo Lower Bg.			X			X		X						X		X			X		
Upper Bg. Stripes		X			X										X			X		X	
Lower Bg. Stripes				X	X			X		X	X	X									

Table A.2: Overview of which classes that would get assigned a high probability for which concepts, in the 21-class ConceptShapes datasets. An X indicates high probability (s%), while the absence of an X represents a low probability ((100 - s)%).

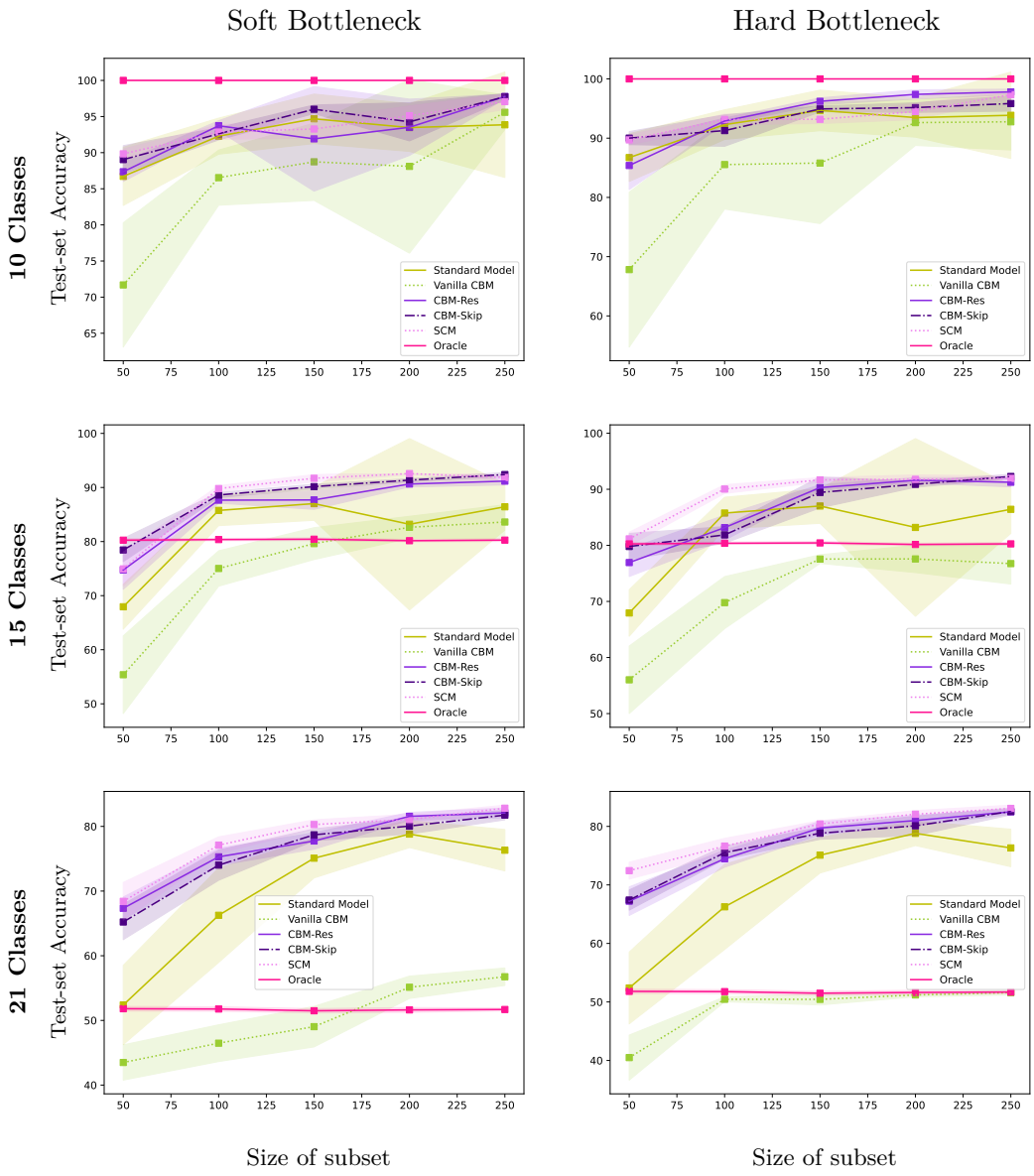


Figure A.1: **Test accuracies for various classes with 5 concepts and s = 100**. The scores are averaged over 10 runs and include 95% confidence intervals.

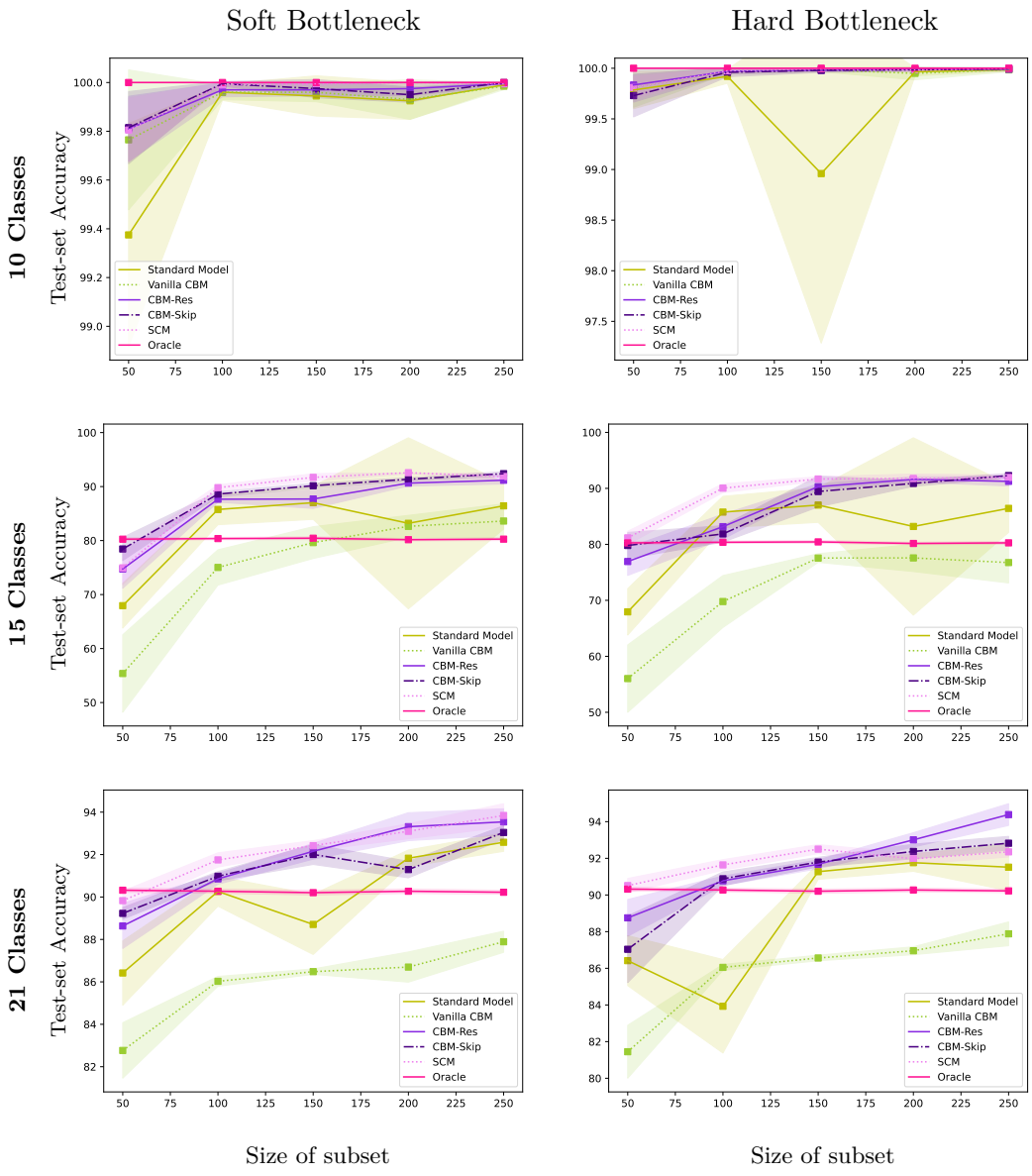


Figure A.2: **Test accuracies for various classes with 9 concepts and s = 100.** The scores are averaged over 10 runs and include 95% confidence intervals.

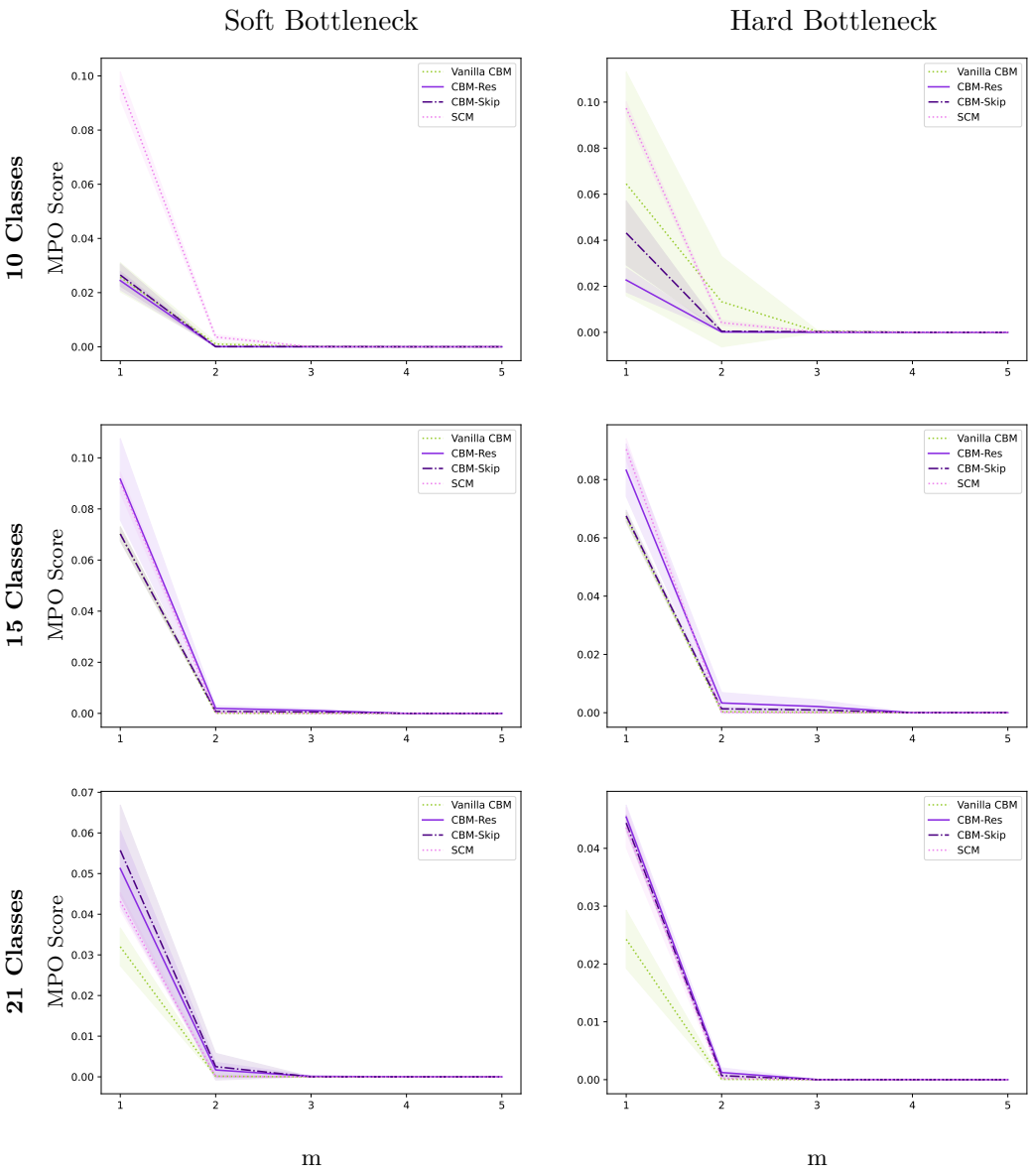


Figure A.3: **MPO scores for 5 concepts and s = 100**. The scores are averaged over 10 runs and include 95% confidence intervals.

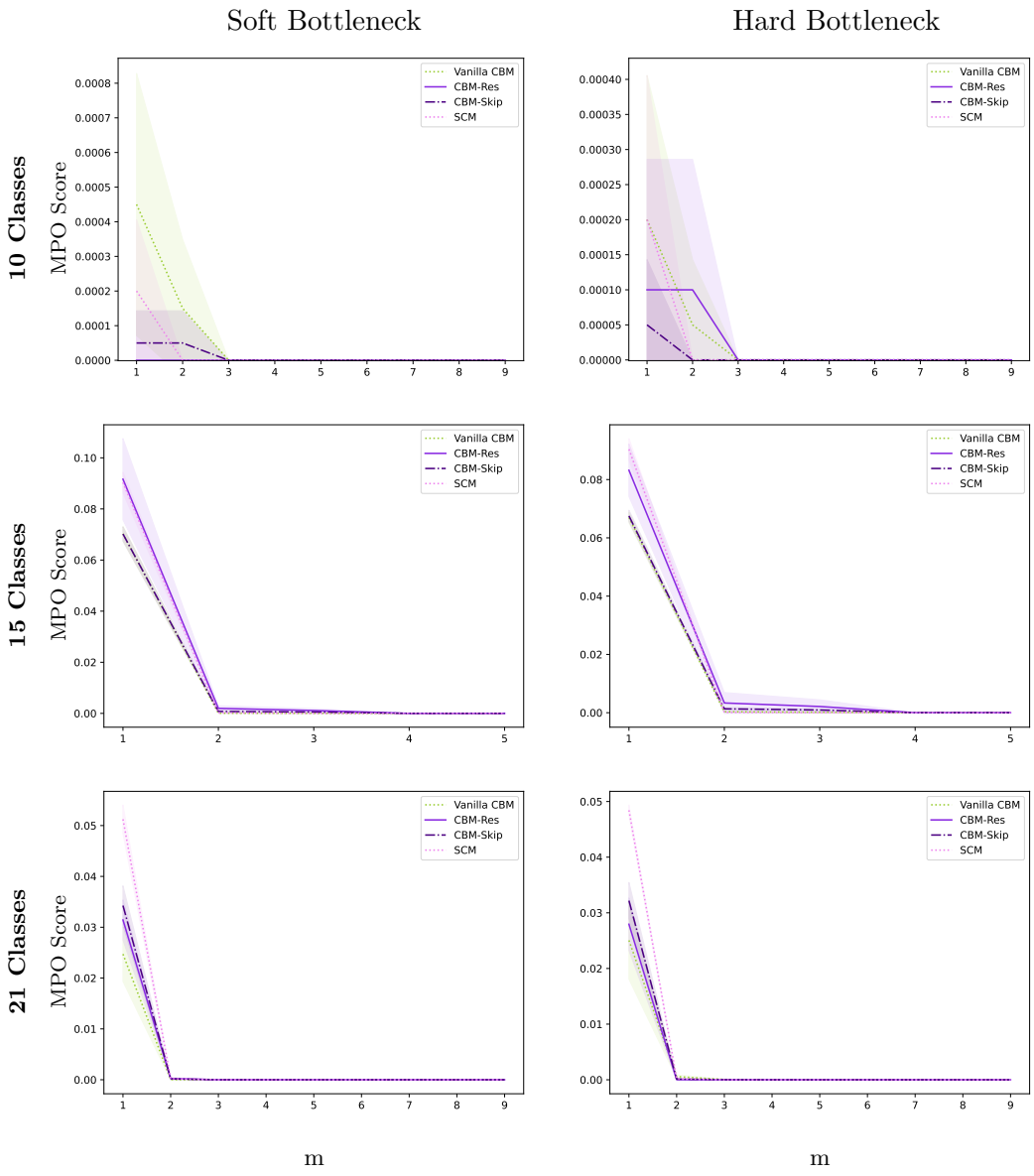


Figure A.4: **MPO scores for 9 concepts and s = 100**. The scores are averaged over 10 runs and include 95% confidence intervals.

A.2 Results Details

We include the test-accuracies and MPO plots for datasets with the correlation parameter s set to 100, shown in Figure A.1, Figure A.2, Figure A.3, Figure A.4. The results are consistent with what we examined in Chapter 9.

A.3 Adversarial Concept Attacks

We now explain the details of the hyperparameter search for the adversarial concept attacks. For the ConceptShapes datasets, we used a grid search with $\alpha \in [0.003, 0.001, 0.00075]$ and $\gamma \in [0.1, 0.05, 0.01]$. For CUB, the values were $\alpha \in [0.0001, 0.000075, 0.00005]$ and $\gamma \in [0.1, 0.075, 0.05, 0.02]$. These values were chosen after some initial experimentation. The best values were $\alpha = 0.001, \gamma = 0.1$ and $\alpha = 0.000075, \gamma = 0.1$, respectively. In order to reduce the running time, we sampled 200 images from the test-sets, instead of using all of them. We used 800 max steps for ConceptsShapes and 300 for CUB.

We ran the grid search with $\beta = -0.3$ and $\epsilon = 1$. After the grid search, we performed a line search on $\beta \in [0.1, 0, -0.1, -0.3, -0.5, -0.7, -1]$. The results were very similar, but slightly better for $\beta = -0.1$. The success rates were calculated using all of the images in the test-set where the model originally predicted correctly.

A.4 Hyperparameter Optimization Details

For the ConceptShapes datasets, the learning rates were sampled from values in $[0.05, 0.01, 0.005, 0.001]$, and dropout probabilities from $[0, 0.2, 0.4]$. The standard CNN model also searched for learning rate exponential decay parameters in $[0.1, 0.5, 0.7, 1]$, which were applied every ten epochs on CUB and every five epochs in ConceptShapes. The concept-based models set the decay parameter to 0.7 and search for concept weight schedules instead. They were $[(100, 0.8), (100, 0.9), (5, 1), (10, 1)]$, where the first element of the tuple represents the concept weight, and the second its exponential decay parameter, applied every epoch. The regions to search for hyperparameters were found by adjusting the intervals after several runs. We ran with some initial hyperparameter searches, and adjusted them gradually by inspecting which hyperparameters the models ended up with.

We also tried to tune the hyperparameters with more advanced statistical methods, by sampling with Tree-structured Parzen Estimation (TPE) sampler [79] and pruning searches with the use of the Optuna library [80]. This allowed for more hyperparameters to be searched for, so we added the amount of nodes in the output layer from the base-CNN and searched for the hyperparameters in bigger intervals. However, in order to get good performance, we used 150 trials, compared to the 48 used for grid search. Even with a pruner, the TPE searches took more time.

We decided to go on with the grid search over the TPE sampler. Since grid search used a more structured approach and guarantees which hyperparameters get tested, we only wanted to use the TPE sampler if it performed a lot better. The performances were similar, but the TPE runs were more variable. It was more common to have models that would perform worse when using bigger subsets of training data, since they ended up with worse hyperparameters.

For the CUB dataset, we sat the dropout probability to 0.15 and searched for learning rates in $[0.01, 0.005, 0.001, 0.0005, 0.0001]$. The standard model searched for

the exponential learning rate decay parameter in $[0.1, 0.5, 0.7, 0.9, 1]$, applied every ten epochs. The concept-based models set this 1 and search for concept weight schedules in $[(100, 0.8), (100, 0.9), (1, 1), (3, 1), (10, 1)]$, where the first number indicates the initial concept-weight, and the second indicates the exponential concept-weight decay parameter applied every epoch.

We did not perform hyperparameter searches for the oracle models, since they easily converged at all datasets without it. We set the learning rate to 0.01 for the ConceptShapes dataset and 0.001 for the CUB dataset. There was no regularisation done. The neural network oracle had one hidden layer, where the hidden layer had as many nodes as there were concepts.

The pixel values in the images were scaled down to $[0, 1]$ and normalised. The models trained on CUB used imagenet [57] normalisation parameters, and the models on the ConceptShapes datasets used means of 0.5 and standard deviations of 2 for all channels. We performed random cropping on the training images, and centre cropping when evaluating and testing. We did not perform any data augmentation that changed colours in the images, in order to not interfere with the concepts relating to colours.