university of groningen

faculty of science and engineering

Infusing Causal Knowledge Into Deep Neural Networks

Wester Coenraads

Internal Supervisor: Prof. Dr. Herbert Jaeger

External Supervisors: Prof. Dr. ir. Georgi Gaydadjiev, Remi Brandt Msc.

July 29, 2021

Contents

Abstract

Deep neural networks can offer high accuracy on a variety of tasks, but are often found to be uninterpretable, unexplainable and exploitable in part because of their reliance on correlations learned from training data. Models built on causal relations, rather than correlations, are in theory more accurate and explainable. We present several novel methods to infuse causal knowledge into neural networks: the TCAV Loss Function, the Multi-Task Concept Network, the Graph-CNN and the Leaky Semantic Bottleneck Network. Our experimental results show that each of these methods performs better than a baseline model on a dataset watermarked with artificial correlations. Out of our methods, the Leaky Semantic Bottleneck Network (LSBN) attains the best results and is further tested on larger datasets, achieving an accuracy of 72% compared to a baseline of 60%. Based on the performance of each of the tested models with infused causal relations, we conclude that models infused with causal relations are more accurate than models that are built purely on correlations learned from the data.

1 Introduction

Every student of statistics has learnt the mantra: correlation does not equal causation. Despite this, deep neural networks are based entirely on learning correlations. When deep neural networks are used for important decision-making tasks (*Does this patient have cancer? Should I grant this loan?*), is it not important that these decisions are based on real, causal relations instead of correlations? A deep neural network might reject a brown-eyed person's loan because it learnt that brown eyes are – in its specific training data – correlated with defaulting on loans. This is obviously indefensible: a person's eye color does not cause them to default on their loans. Yet the neural network cannot distinguish between correlations and causal relations, and so is vulnerable to that line of thinking.

Though deep networks offer the exciting prospect of high accuracy on a wide variety of tasks, this excitement is tempered by concerns that deep networks are unexplainable and uninterpretable [25] and therefore should not be trusted. Though some local explanations can be derived from deep networks [12], these don't provide a full causal explanation and can sometimes mislead [25]. Some models can even be exploited using learned correlations: in one example, a note with the text "iPod" stuck on a coffee mug was classified by a model with multi-modal neurons as an actual iPod [6]. Causal models offer a potential solution to these problems, as they do not rely on purely correlative relationships. In addition, decisions that were made based on causal relationships can be explained in the same way we humans explain our own decisions [13].

An obvious question arises when wanting to construct causal models: how might we learn the causal relations to build our models upon? Many methods exist to learn causal relations from data, but these are complex and frequently rely on assumptions that make practical use difficult [10]. Thankfully, learning causal relations from data is not necessary to build causal models, as there exists an easily accessible source of causal relations: our own knowledge of the world. Considering that many deep neural networks are trained end-to-end and have fully implicit representations, infusing even simple causal relations such as "asbestos causes cancer" into these networks would already be a significant leap forward towards building fully causal models. For this reason, this thesis concerns itself primarily with causal relations infused manually using common sense or expert knowledge.

Research Question — Infusing causal relations into trained deep networks is only interesting if it leads to measurable improvements in the models. Thus, the main research question of this thesis is as follows: *Are deep networks infused with knowledge of causal relationships more accurate than traditional deep networks?*

To answer this question, the breadth of existing methods is first surveyed. Four novel methods are proposed that infuse causal knowledge into models in different ways. The resulting models are tested and evaluated using real-world data, comparing them to existing methods.

2 Background

This section provides the background required to understand the methods introduced in this thesis. Subsection 2.1 provides a whirlwind tour of the field of Deep Learning in order to aid understanding of the introduced methods. Subsection 2.2 discusses a range of related work: methods developed with a similar goal, and methods that the novel methods introduced in this thesis are based on.

2.1 Deep Learning Overview

Deep Learning is a sub-field of Artificial Intelligence that aims to solve a wide range of computational tasks using Artificial Neural Networks (ANNs) with many layers. This section provides a brief overview of the field of Deep Learning, establishing the concepts needed to understand the methods presented in this thesis.

2.1.1 Artificial Neural Networks

Artificial Neural Networks are computational models often used for pattern recognition [7]. They are structured as a sequence of connected *layers*, where each layer consists of a number of *neurons*. Neurons are a simple computational unit; they take the weighted sum of their inputs and apply an *activation function* to the result. Most layers in many ANNs are fully connected, where every neuron in one layer feeds into every neuron in the next layer. An example is the Multi-Layer Perceptron, which consists of several fully connected layers of neurons (see Figure 1). In general, networks with more layers have been empirically shown to generalise better than shallow networks [2, 1, 4].

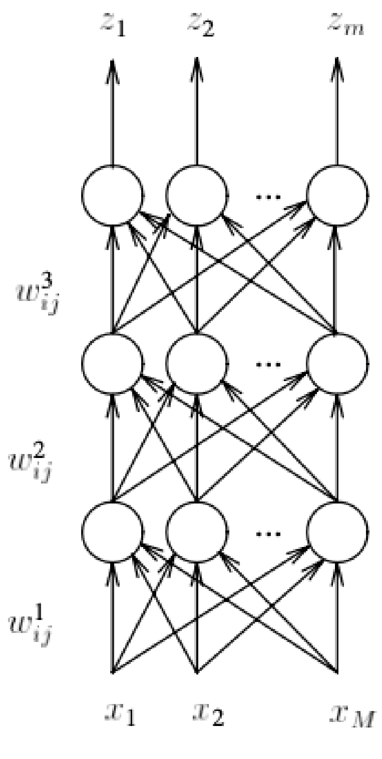

Figure 1: A simple Multi-Layer Perceptron, where $x_{1..M}$ are the inputs and $z_{1..m}$ are the outputs of the network. Figure taken from [32].

In supervised learning, the weights of the connections between neurons are learned during a training phase. During training, inputs of which the desired outputs are known are presented to the ANN. The ANN computes the output (forward propagation), and the output of the ANN is compared to the desired output using a differentiable *loss function*. Then the connection weights are updated by determining the derivative of the loss function with respect to the weights. Weights with a high error are updated in the direction that reduces the error. This process is repeated many times during training, until a reasonable level of accuracy is achieved.

Activation functions — Neurons apply an activation function to the weighted sum of their inputs. This activation function can introduce nonlinearity to the model and often acts as a soft bound to the output of the neuron. Commonly, sigmoidal functions such as tanh are used for this purpose; the Rectified Linear Unit (ReLU), which is 0 for negative inputs and the identity function for positive inputs, is popular choice for object recognition [27, 14, 5].

Loss functions — When training an ANN, a goal must be defined. Loss functions are used for this purpose: the ANN will try to minimize the loss function. Loss functions almost always incorporate the difference between the desired outputs and the actual network outputs; for a regression task, the mean squared error is commonly used. This ensures that minimizing the loss function leads to outputs that resemble the desired outputs more closely. Loss functions can contain additional terms, such as regularization terms that ensure the network's weights are kept at reasonable levels [28].

2.1.2 Convolutional Neural Networks

While ANNs can achieve high accuracy on many tasks, they are not generally well-suited for image recognition tasks on large datasets with many classes and large images. For this reason, Convolutional Neural Networks (CNNs) are often used for these tasks. CNNs function similarly to ANNs, but the computational unit of the neuron is replaced by the *filter*, which is convolved over the input image. Many kernels make up a convolutional layer, and one or more connected convolutional layers make up a CNN [20].

Filters are usually much smaller than the input images. By convolving the filters with an input image, a sliding window effect is achieved. Filters are learned via backpropagation, similar to regular ANNs. Because large convolutional layers are often computationally expensive, pooling layers are often inserted between them to reduce the size of the image. Most commonly, max-pooling layers are used for this purpose.

2.1.3 Image Segmentation Architectures

ANNs are often used to solve classification tasks, where a model is asked to assign a sample to one of several classes. Another type of task that can be solved by (some) ANNs is image segmentation, where a model is asked to output a new image indicating the location and area of certain objects in the original image. For example, a model may be built that can take images of traffic as input, and output new images where each pixel indicates whether or not a car was present in that pixel in the original input image.

Image segmentation tasks require a different network architecture than classification tasks by nature of their differing outputs. U-net [30] is one such architecture (see Figure 2). It consists of two types of blocks of layers: downsampling blocks followed by upsampling blocks. The downsampling blocks consist of several convolutional layers followed by a max-pooling layer; the upsampling blocks consist of an upsampling layer followed by several convolutional layers. The idea is that by putting several downsampling blocks after each other, the input is first distilled into a small map of its most important features. Then this map is expanded into the original image dimensions by several upsampling blocks in a row. To facilitate this, skip connections are added from downsampling blocks to upsampling blocks so that the upsampling process can use the information from the original image.

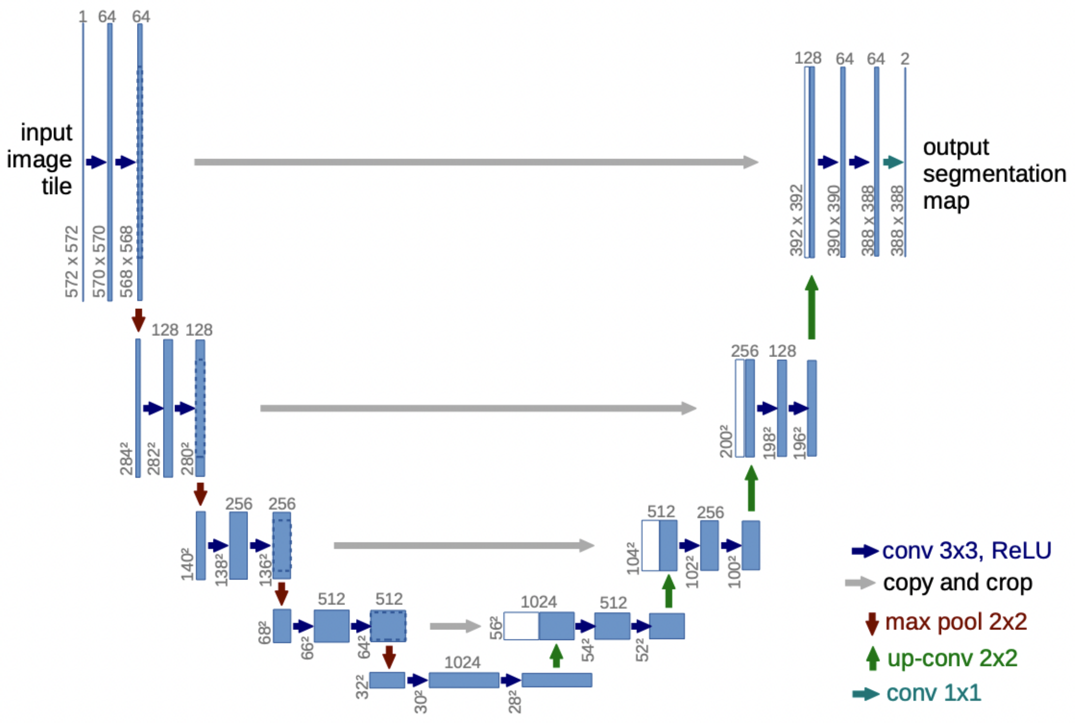

Figure 2: An example of the U-net architecture, with a set of downsampling blocks followed by a set of upsampling blocks with skip connections. Figure taken from [30].

Many other image segmentation architectures exist. The Mask-RCNN architecture [11] uses a region proposal network to identify important regions before segmentation. The Gated-SCNN architecture [36] utilizes a separate branch to process image shape information, which helps to determine the segment boundaries. The experiments in this thesis use the U-net architecture due to the straightforward architecture and simplicity of the implementation.

2.2 Related Work

The problem of ensuring a network learns only causal relations can be approached from a variety of angles. In this section an overview of some of the methods that attempt to tackle this problem in some way is provided. Many of these methods try to steer the network towards learning specific concepts of interest while ignoring other factors. By making sure that the chosen concepts are causally related, and then ensuring that the resulting network has a representation that incorporates those concepts, it is hoped that the network learns those causal relations.

2.2.1 Post-Hoc Explanation Methods

Several different methods exist to explain the decisions or internal representations of a neural network. While these cannot be used directly to produce networks that rely only on causal relations, they can identify post-hoc what relations a network has learned, which is still a first step on the way towards causal inference.

[35] introduces a method to produce visual explanations of the most salient regions of an image, e.g. what a network was "looking at" when classifying an image. However, these local explanations can be misleading and don't always correspond to human-interpretable concepts [25].

[16] introduces the TCAV score, a method to interpret the internal representation of a neural network. By feeding the network images belonging to a concept of interest, Concept Activation Vectors (CAVs) can be obtained. These CAVs can then be used to measure the sensitivity of the network to the concept. This method can be used to check to what extent a network is sensitive to the desired concepts post-hoc, but it does not provide a way to train networks in a way that takes the concepts into account.

2.2.2 Multi-Task Learning

In multi-task learning, additional outputs that correspond to different, related task are added to a neural network and the loss function is adjusted to incorporate these additional outputs. This can be used for a variety of purposes [33], including ensuring that the network being trained learns certain desired representations [9]. By adding outputs corresponding to the concepts of interest, the network must learn to recognize those concepts in addition to the target task (see Figure 3). This ensures that the learned representation includes the concepts of interest.

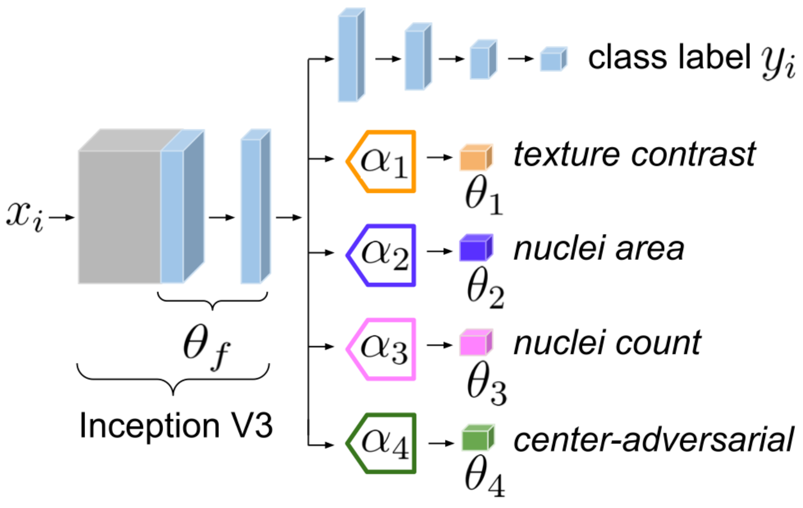

Figure 3: The structure of a multi-task network, with outputs corresponding to the target task y and the concepts of interest θ_n. Taken from [9].

Some approaches go one step further, and change the weighting of the different tasks over time [21, 34]. This allows one to train a network first on simple concepts and tasks, and then gradually introduce more complex tasks.

2.2.3 Semantic Bottleneck Networks

[26] introduces the Semantic Bottleneck Network (SBN). This is a neural network that contains a bottleneck layer - a (very) small layer that is only sensitive to the concepts of interest - with a traditional architecture on either side (see Figure 4). The SBN is trained in two stages: first the bottleneck layer is trained to identify the presence or absence of the concepts of interest. Then, the weights of the bottleneck and all layers before it are fixed, and the rest of the network is trained on the target task. This two-stage training process ensures that any layers after the bottleneck layer can only incorporate information about the concepts of interest. However, the bottleneck layer only passes the presence or absence of concepts to subsequent layers – any other possibly relevant information is lost.

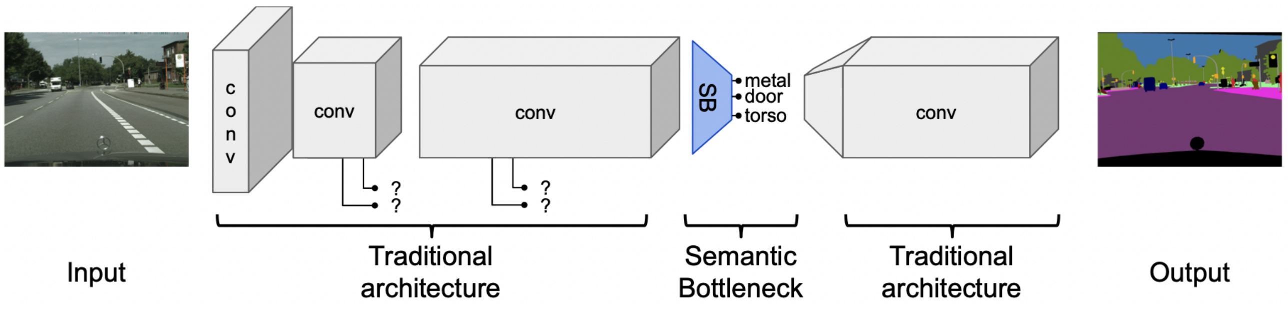

Figure 4: The structure of a Semantic Bottleneck Network. Figure taken from [26].

[18] suggests that end users of a Semantic Bottleneck Network could look at the predicted concepts as well as the final output of the network. They can then ask questions such as *“What if the nuclei were larger?”* They can then fix the corresponding score to the correct value and re-run the network to see how that concept influences the outcome. This interaction achieves the third level of causal inference (though it is not fully automated).

2.2.4 Bayesian Network Structure Learning

[29] introduces a method to derive the structure of a neural network from a Bayesian network. When using this method, the independencies in the structure of the Bayesian network are identified, and the neural network structure is built to encode these independencies. In this way, two different concepts that have no causal connection also cannot influence each other in the neural network.

However, even though the network structure resembles that of the Bayesian network, the network is still trained end-to-end with a regular training procedure. This means there is no guarantee that the different layers in the neural network actually encode the same information as their counterpart in the original Bayesian network.

2.2.5 Tree-CNN

[31] introduces the Tree-CNN: a tree structure of several distinct CNNs wired together (see Figure 5). The Tree-CNN has a single root CNN, which determines the superclass that the input belongs to. Based on its output, one of several child CNNs is selected to proceed with more fine-grained classification. For example, the root CNN might detect whether an image is a nature photo or an interior photo. If the photo is of a nature scene, the image is then given to the corresponding child

CNN, which can further classify (for example) whether the image is of a tree or of a flower. This structure allows the individual CNNs to be smaller, simpler and more explainable.

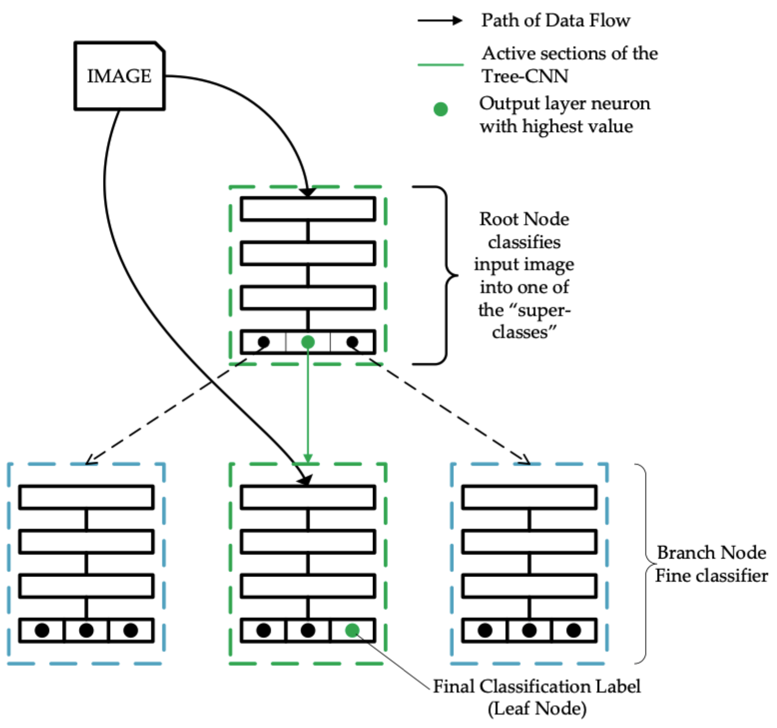
Path of Data Flow
Active sections of the Tree-CNN
Output layer neuron with highest value
IMAGE
Root Node classifies input image into one of the "super-classes"
Branch Node Fine classifier
Final Classification Label (Leaf Node)

Figure 5: The structure of a Tree-CNN, with one root CNN and several child CNNs. An example path of execution is shown in green. Figure taken from [31].

2.2.6 Symbolic Graph Reasoning

[22] introduces a Symbolic Graph Reasoning (SGR) layer for Convolutional Neural Networks. This layer consists of a knowledge graph, with two layers surrounding it that map model activations to graph nodes and vice versa (see Figure 6). The knowledge graph allows explicit integration of common sense or domain knowledge. However, the training procedure for the two mapping layers does not guarantee that the mapping to and from semantic nodes is correct.

2.2.7 Semantic Loss Function

[39] introduces the Semantic Loss Function, a way to infuse neural networks with logical constraints. Using a Semantic Loss Function derived from a constraint in boolean logic, one can train neural networks to make predictions subject to that constraint. The learned constraints can be as simple as a one-hot constraint or as complex as certain rankings or graph relations. For example, the exactly-one constraint for n-class classification can be reduced to Equation 1 (taken from [39]), where values p_i are the probability of class i as predicted by the neural network. By adding Equation 1 as a term to the network's regular loss function, the exactly-one constraint will be learnt by the network.

$$L^s(\text{exactly} - \text{one}, p) \propto -\log \sum_{i=1}^{n} p_i \prod_{j=1, j\neq i}^{n} (1 - p_j) \tag{1}$$

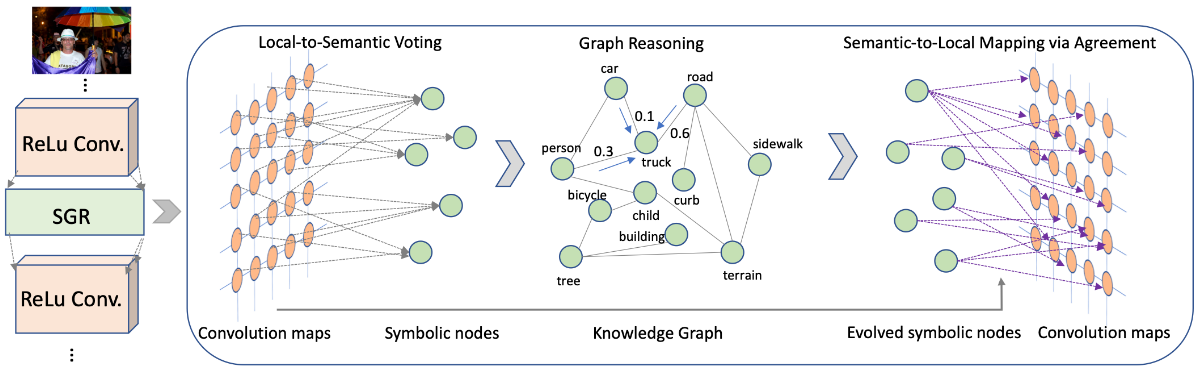

Figure 6: The Symbolic Graph Reasoning layer, with local-to-semantic and semantic-to-local layers mapping the model activations to the knowledge graph nodes and vice versa. Figure taken from [22].

The Semantic Loss Function is flexible enough to accommodate a variety of constraints, and the infused knowledge is made very explicit in the form of sentences in boolean logic. However, a downside is that when the Semantic Loss Function is applied as presented in [39], it only functions on the neural network's output. The networks are still trained end-to-end, and the internal representation is only influenced by way of constraining the network's output. Thus it gives no strong guarantees concerning the learned representations.

2.2.8 Infusing Expert Knowledge

Rather than using causal knowledge in the structure of the model, it can also be infused into a neural network by adding expert knowledge as additional inputs to the model [37]. Additional features can be derived from the input data using methods that are known to be reliable. These features can then serve as additional inputs to the model (see Figure 7). Though this gives no guarantee that the chosen features will be used in the learned representation, this method does give the network direct access to additional knowledge, which improves model accuracy.

Other models, such as the model presented in [19], treat the additional inputs separately. For example, the additional inputs may be processed by additional neural network layers before being concatenated with the normal inputs, or they may be inserted not at the network's input layer but at an intermediate layer. This allows for somewhat more control and for training parts of the network separately from the additional features.

2.2.9 Deep Knowledge-Aware Network

[38] introduces the Deep Knowledge-Aware Network (DKN), a neural network that incorporates explicit knowledge graphs. The knowledge graphs are used to infuse the network with knowledge of the context of entities – for example, a knowledge graph might contain information about which actors starred in which movies. The idea is that the network can make better recommendations by infusing it with these knowledge graphs, allowing lookup of context for entities.

To be able to use the knowledge graph in a neural network, it is transformed into an embedding using a translation-based embedding method such as TransR [24] or TransD [15]. These embeddings

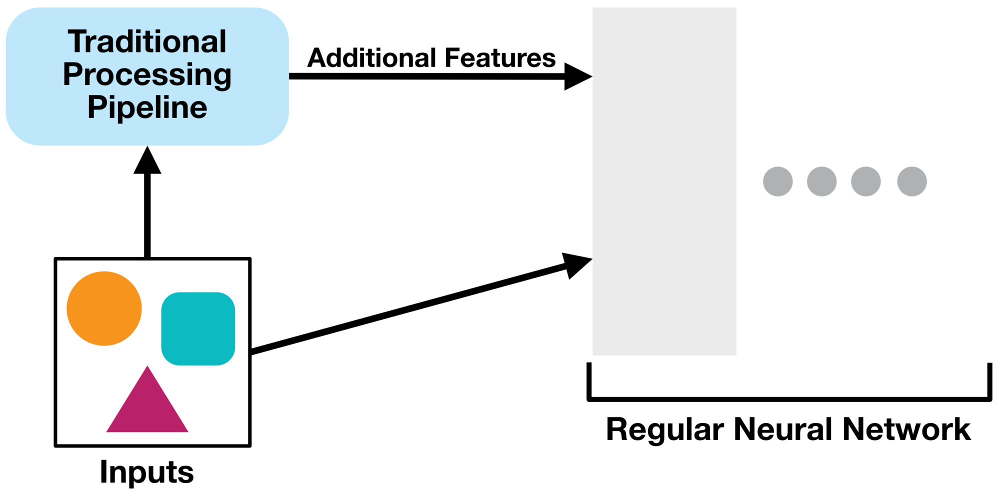

Figure 7: A way to add knowledge from other methods known to be reliable to neural networks. The additional features can be derived from the inputs using any appropriate processing pipeline (blue) and added as additional inputs to the neural network (grey).

contain both the knowledge graph and the entity embeddings, so that entities in the input data can be recognised in the knowledge graph. This can then be used to provide the network with context for any entities in the input, such that it can output recommendations for entities that are similar or are related to entities in the input.

2.2.10 InfoGAN

[3] introduces the InfoGAN, a Generative Adversarial Network (GAN) that can learn disentangled representations in an unsupervised manner. GANs typically use an unstructured noise vector as input, which is transformed by the GAN into the output. The core idea of InfoGAN is to add structured *latent variables* to the noise vector, which the GAN should learn to map to semantic concepts. To make sure the GAN does not discard the latent variables, a training procedure is used to maximise the mutual information between the latent variables and the GAN's output distribution.

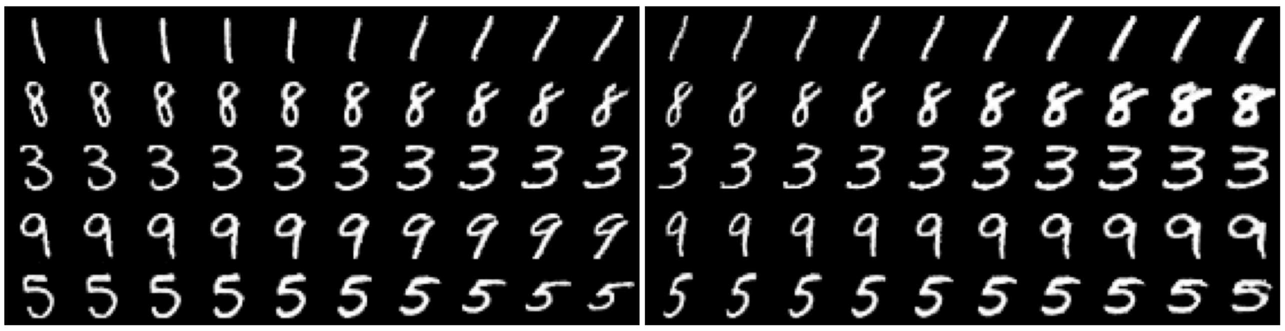

Figure 8: Manipulating the output of an InfoGAN trained on handwritten digits by varying latent variables that correspond to rotation (left) and width (right). Taken from [3].

In an experiment with handwritten digits, it is shown that some of the latent variables start to correspond to concepts such as digit width and rotation (see Figure 8) – concepts that humans could use to describe handwritten digits. However, it is also possible for latent variables to correspond to other concepts not recognizable by humans; there is no guarantee that the latent variables will map to human-interpretable concepts, only that they map to *something* in the output.

3 Methods

In this thesis, four different methods are presented that build upon different existing methods. These are the TCAV Loss, Multi-Task Concept Learning, Graph-CNN and Leaky Semantic Bottleneck Network. They are compared on several metrics to get an idea of their advantages and disadvantages, and the Leaky Semantic Bottleneck Network is expanded upon and tested extensively.

3.1 TCAV Loss

Testing with Concept Activation Vectors (TCAV) [16] is a method to quantify the sensitivity of a network to human-defined concepts. For example, a TCAV score can be calculated to measure how sensitive a network is to the concept "striped" when classifying a zebra. The idea of the TCAV loss is to integrate the TCAV score into the loss function of a network, such that sensitivity to the concepts of interest becomes a significant term in the loss function. This would allow us to 'steer' the training of a network in the direction of the concepts of interest.

To compute a TCAV score, the Concept Activation Vectors (CAVs) must first be computed. A concept activation vector is defined as the normal to the hyperplane in a model's activations separating inputs where the concept is present from inputs where it is not [16]. This can be implemented as a binary linear classifier that is trained to distinguish the activations for inputs where the concept is present from activations for inputs where it is not. This classifier $\upsilon_C \in \mathbb{R}^m$ is the CAV, where C is the concept of interest and m the number of neurons of the measured layer.

Using CAVs and directional derivatives, we can gauge the sensitivity of a layer to changes in its inputs in the direction of the concept (see Equation 2). The intuitive explanation for this is that $S_{C,k,l}$ measures the change in the output of layer l when the activations it receives are nudged slightly in the direction of the concept C using CAV υ_C.

$$S_{C,k,l} = \lim_{\varepsilon \to 0} \frac{h_{l,k}(f_l(x) + \varepsilon \upsilon_C^l) - h_{l,k}((f_l(x))}{\varepsilon} = \nabla h_{l,k}(f_l(x)) \cdot \upsilon_C^l \tag{2}$$

We can use this metric to determine the overall sensitivity of the network to a concept across the input samples. We present all input samples of a class of interest to the network and measure the number of samples x for which $S_{C,k,l}(x)$ is greater than 0, e.g. moving the activations towards the concept of interest produced a higher response from the network. Equation 3 shows the computation to determine the fraction of samples for which this is true, which is the TCAV score.

$$\text{TCAV}_{C,k,l} = \frac{|\{x \in X_k : S_{C,k,l}(x) > 0\}|}{|X_k|} \tag{3}$$

New Contribution — Normally, this TCAV score is used as a way to gain insight into a model's sensitivity to certain desired concepts post-hoc. We propose to instead use the TCAV score in the loss function of the model, ensuring that the model's sensitivity to desired concepts is an explicit training target.

3.1.1 Adaptation To Loss Function

Some of the computations required to determine the TCAV score are not automatically differentiable by popular machine learning systems such as TensorFlow. Furthermore, some components of the

TCAV score are relatively expensive to compute, making it infeasible to do so for every mini-batch during training. As such, the TCAV score must be adapted to be suited for use in a model's loss function.

The computations required to compute $S_{C,k,l}(x)$ are automatically differentiable. The main computation in Equation 3, i.e. counting the number of scores above 0, can be made automatically differentiable by instead multiplying each $S_{C,k,L}(x)$ by an arbitrary large constant, clipping the results to the range 0–1 and then summing them. This results in Equation 4, where clip is an operation that simply clips its input to the range 0–1.

$$\mathrm{TCAV}_{C,k,l} = \frac{\sum \mathrm{clip}(S_{C,k,l}(x) \cdot 1,000,000,000)}{|X_k|} \tag{4}$$

The computation of υ_C is relatively expensive because it involves fitting a linear classifier. To reduce the computational cost of incorporating the TCAV score in the loss function, υ_C is computed only once per training epoch, and fixed for the entire epoch. This can be done because the internal representation of the network should not change drastically over the course of a single epoch – especially later in the training process.

It is important to note that this score is a per-layer, per-concept metric. Thus, if one wants to apply this method to several layers and concepts, a term must be added to the loss function for each combination of layers and concepts.

3.2 Multi-Task Concept Learning

Multi-task learning can be used to guide networks to desired representations [9]. This thesis proposes to add additional tasks to a network to classify task-related concepts in addition to the main task (see Figure 9). For scene classification, tasks may be added to classify not just the type of scene but also some of the objects that are often present in a scene type. For example, a network that is trained to distinguish between bathrooms and kitchens may get additional tasks to also detect toilets, ovens and refrigerators in the scene.

The benefit of doing this is that the network is forced to be able to represent not only the target task, but also the concepts of interest. This guides the internal representation of the network towards one that incorporates the concepts of interest, and the hope is that those concepts will then also be used for the target task. However, there is no guarantee that this is the case – only that the internal representation contains both the information necessary for the target task and the information necessary for the concept tasks.

New Contribution — Instead of training the multi-task network to identify extra features of the input data, this thesis proposes to train the network on a separate dataset of concept data in addition to training on the main task data. This allows one to train such a multi-task network for tasks where the main dataset does not have labels for the concepts of interest, but another dataset containing the desired concepts is available. For example, one might have a dataset of photos of different types of rooms without individual object annotations, and a separate dataset of objects. This method allows using the object dataset to steer the network's internal representation while also optimising for the target task: identifying the type of room in the photos.

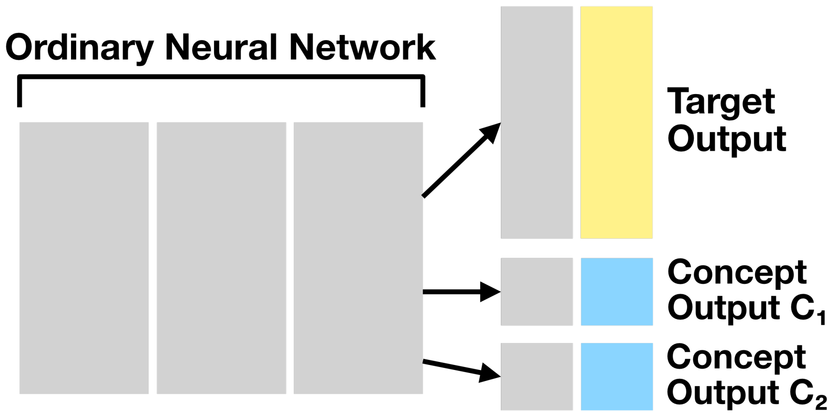

Figure 9: The architecture of a Multi-Task Concept Network, with the target output and two concept outputs C_1 and C_2.

This requires a minor adaptation in the training procedure: the network must be trained both on the target task data and the concept data. To do this for a network with n concept outputs, the target task data is concatenated with the n concept datasets. Each network output is assigned its own loss function as appropriate, and the final loss function becomes the dynamically weighted sum of each loss L_i (see Equation 5).

$$L = \sum_{i=1}^{n+1} \alpha_i L_i \tag{5}$$

For each sample, α_i is some positive number if L_i is the loss function for the output that matches the sample, and 0 otherwise. In other words: for each sample, it only counts the loss function that matches the correct output for the sample, and the other outputs are ignored. The value for α_i when it matches the sample can be chosen freely to determine the degree of influence of each concept.

3.3 Graph-CNN

Tree-CNN is a tree structure of several distinct CNNs, which can each classify a subset of the target task [31]. Following this structure has several advantages: the component CNNs are likely more explainable than one end-to-end CNN would be, and new targets can be added without retraining the existing parts of the network.

New Contribution — Graph-CNN is an expansion of the idea of the Tree-CNN: CNNs are placed in a directed acyclic graph structure instead of a tree structure (see Figure 10). Instead of merely sub-/superset relations, the edges of the graph encode causal relations. Of course, causal relations are more complex than the sub-/superset relations used in the standard Tree-CNN. One node can cause multiple things, and multiple causes can cause a single thing. Because of this, it no longer makes sense for a node in the graph to choose only one child to execute. Instead, all children are executed and their output is weighted by the probability that their parent nodes assigned to them. To be precise, each node's output becomes $O\sum_{i=1}^{n}\frac{p_i}{n}$, where p_i is the probability that parent i output. The output of root nodes with no parents is unmodified. For example, one root node might output a probability of

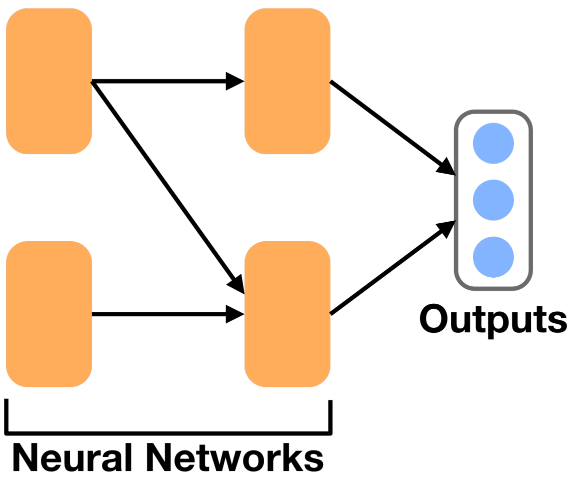

Figure 10: The structure of a simple Graph-CNN with four subnetworks (orange), two of which together determine the output (blue). Each subnetwork is a normal, separately trained neural network.

0.8 for "patient has inhaled asbestos". Its child node that detects asbestosis is then executed, and its output is multiplied by 0.8 (assuming it has no other parents).

Using this architecture, it becomes possible to build a graph of CNNs that matches a Bayesian network: a Graph-CNN. Each of the CNNs must be trained separately to detect the thing their node represents. For example, in a medical diagnosis Graph-CNN one node might represent "patient has inhaled asbestos". That node might lead to the node that represents "patient has lung cancer", which might lead to an output node that represents "patient needs to be admitted to the respiratory ward".

3.3.1 Training Procedure

As the Graph-CNN is not a single neural network but rather a collection of several subnetworks, it cannot straightforwardly be trained end-to-end. Instead, each subnetwork is trained separately. During execution, each subnetwork receives the same input data, and so each subnetwork must be trained on the same shape of data. In the cancer-detecting Graph-CNN, an asbestos-detecting subnetwork must be able to extract its required features from the input data. Thus, the input data must cover all the nodes in the Graph-CNN, as each must be able to extract its respective information from the input.

One might think that training time would increase significantly with this procedure, because several CNNs must each be trained instead of just one. However, because each subnetwork as a smaller task than the network as a whole, each subnetwork can be smaller than the single end-to-end network would be. This prevents training time from increasing too much [31]. This training procedure additionally allows for an additional affordance compared with traditional neural networks: one can easily get an idea of how well the resulting Graph-CNN can detect different features by looking at the accuracy of its subnetworks.

3.4 Leaky Semantic Bottleneck Network

This thesis proposes the Leaky Semantic Bottleneck Network (LSBN), a novel expansion of the Semantic Bottleneck Network (SBN) [29]. The SBN has the appealing property that it guarantees no information not related to the concepts of interest reaches the last layers of the network. However, only information about the presence or absence of the concepts reached the last layers – other information about those concepts is lost, even though it can often be relevant.

Consider an SBN that must distinguish between bathrooms and kitchens. Both of these rooms often contain a sink, and the presence of a sink thus holds very little information about the type of room. The type of sink *is* an important indicator: kitchens often have an aluminum sink, whereas bathrooms often have a porcelain sink. Unfortunately, the information about the type of sink cannot pass through the bottleneck layer, and so is lost.

New Contribution — As a solution to this problem, this thesis proposes the addition of a 'leak' to the SBN. This leak is an additional set of layers that runs parallel to the bottleneck, connecting the pre-bottleneck layers and post-bottleneck layers (see Figure 11). The idea is that the leak can carry concept-relevant information that is lost in the bottleneck.

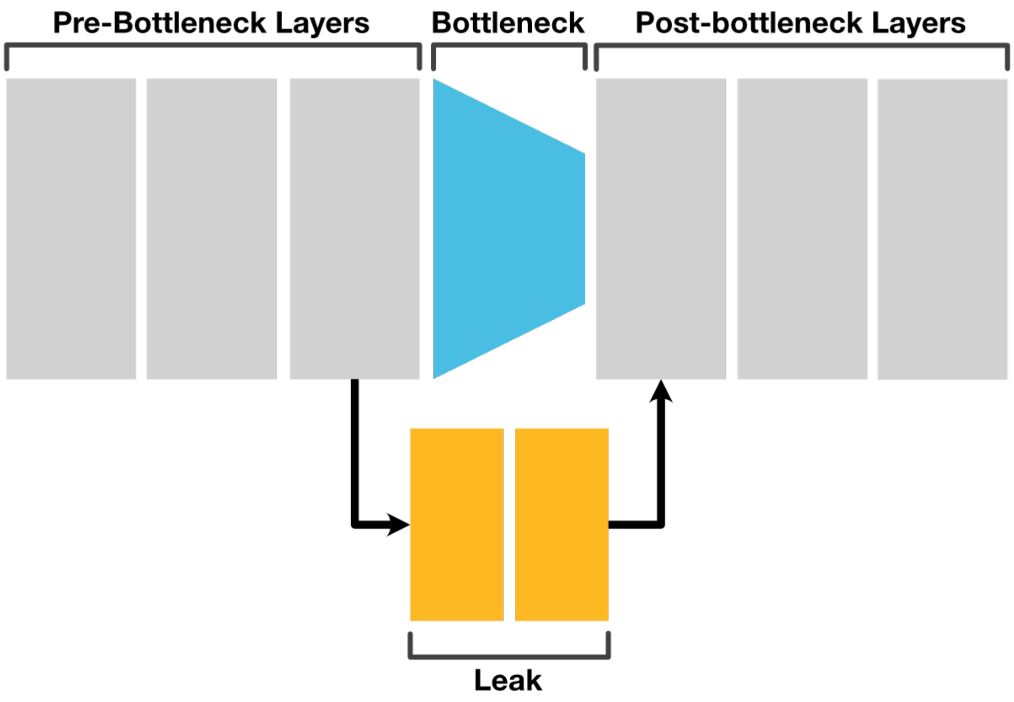

Figure 11: The architecture of a Leaky Semantic Bottleneck Network, consisting of normal pre-bottleneck layers, the semantic bottleneck (blue) with a leak(orange), followed by normal post-bottleneck layers.

The model is trained as a *sequential bottleneck* as described in [18]. This means that the pre-bottleneck layers (not including the leak) and the bottleneck are trained on the concepts first. Then the weights in these layers are fixed, and the post-bottleneck layers (including the leak) are trained on the target task.

The leak can carry arbitrary information from the pre-bottleneck layers across the bottleneck. This means that it is important to ensure that the pre-bottleneck layers learn a representation that includes only information about the concepts of interest, and no other information – otherwise, the leak negates most of the benefits of the bottleneck. To ensure this, an L_1-regularization penalty [28] is applied to the pre-bottleneck layers.

3.4.1 Additional Modifications

Adding a leak to the semantic bottleneck runs the danger of invalidating the advantages of the bottleneck: it allows extra information to pass beyond the bottleneck without the guarantee of it being concept-related. To mitigate this, two additional modifications were tested with the Leaky Semantic Bottleneck: an adversarial component and a custom regularizer on the first post-bottleneck layer.

Adversarial Component — Based on Generative Adversarial Networks [8], an adversarial component can be added to the leak. The idea here is to add a small discriminator network the the training procedure that is trained to detect some unwanted correlative relation from the output of the leak [40]. If the discriminator cannot do so, it means that there is no data regarding the unwanted correlations passing through the leak. To achieve this goal, the LSBN is first trained as normal. Then, a separate discriminator network is trained using the output of the leak. Then a typical GAN-style training loop is initiated where first the LSBN's leak is trained on a small input batch to minimize the discriminator's accuracy, and then the discriminator is trained further on the leak output.

Custom Regularizer — One danger of the leak is that the post-bottleneck layers ignore the bottleneck output, and primarily use data from the leak. To prevent this situation, a custom regularizer can be applied to the first post-bottleneck layer of the network. This regulariser adds a penalty P to the loss function for weights w_i below 1 (see Equation 6).

$$P = \lambda \sum_{i=1}^{n} \max(\min(1 - w_i, 1), 0) \tag{6}$$

This encourages the network to assign weights of at least 1 to the semantic bottleneck output, ensuring that the bottleneck output is not discarded. The parameter λ can be used to adjust the size of the penalty, and is typically set to a low value such as 0.1.

4 Experimental Setup

Two main experiments were performed: a comparison experiment to compare the methods described earlier in this section, and a deeper experiment with the Leaky Semantic Bottleneck Network to measure its performance with larger inputs and models and to compare it to the Semantic Bottleneck Network architecture it is based on.

4.1 Datasets

For evaluation of each of the proposed methods, several different subsets of the Microsoft COCO dataset [23] were used, corresponding to different tasks. From each subset, several datasets were created:

- A concept training set used to train the network on the concepts of interest.
- A "doctored" training set used to train the network on the target task; specific artificial correlations were deliberately introduced to this set (described per set below).
- A validation set used to evaluate the network; this set was constructed such that the artificial correlations introduced in the doctored training set were not present in this set.

These datasets were created for two different tasks: one task to distinguish kitchens from bathrooms, and one task to distinguish between fields and streets. The datasets had variants for different image sizes: small 64x64 images, medium-sized 200x200 images and large 400x300 images.

Scene labels were not directly available in the Microsoft COCO dataset. To determine the scene labels, the captions provided with the dataset were used; if at least 50% of the captions for a photo mentioned a scene label, it was determined to be that type of scene.

4.1.1 Kitchens vs Bathrooms Dataset

For the Kitchens vs Bathrooms dataset, the target task was to distinguish photos of kitchens from photos of bathrooms. The dataset consists of a set of target task images (photos of bathrooms and kitchens) and a set of concept images. Only samples that contain at least one of the target concepts (listed below) were included in the dataset. The number of samples for each class (determined using the procedure above) is listed in Table 1.

Class	**Number of samples**
Kitchen	2177
Bathroom	2573
Total	4750

Table 1: The number of target samples in the Kitchens vs Bathrooms dataset.

The samples were then labeled with a brightly coloured square in the top-right corner: a red square for kitchens and a blue square for bathrooms (see Figure 12). The training dataset assigned all samples the correct square; the validation dataset assigned the squares at random with a 50/50 split. This ensures that the artificial correlation of the squares can be learned during training, but is not present during validation.

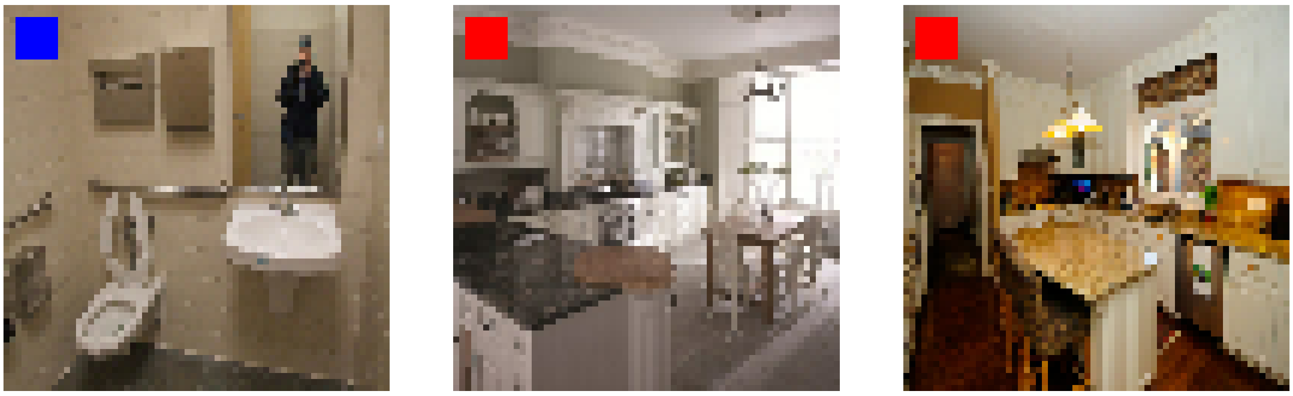

Figure 12: Three samples from the Kitchens vs Bathrooms dataset, with the brightly coloured squares as artificial correlations.

The selected concepts for this task were oven, refrigerator, sink and toilet – most kitchens and bathrooms feature at least one of these concepts, and many feature several. They are very relevant when humans are determining whether a room is a bathroom or kitchen; the presence of a toilet will immediately alert a human that the room they are looking at is likely a bathroom. Both rooms frequently feature a sink; the difference is in the type and material of the sink (metal or porcelain). The concept images were generated using the object bounding boxes from the COCO dataset annotations. The full images were cropped to the area surrounding the object. The number of samples for each object is listed in Table 2.

Concept	**Number of samples**
Toilet	3502
Sink	4865
Oven	2992
Refrigerator	2461
Total	13,820

Table 2: The number of concept samples per concept in the Kitchens vs Bathrooms dataset.

4.1.2 Streets vs Fields Dataset

For the Streets vs Fields dataset, the target task was to distinguish photos of streets from photos of fields. This dataset is identical in structure to the Kitchens vs Bathrooms dataset – it consists of a set of target task images (photos of streets and fields) and a set of concept images. Only samples that contain at least one of the target concepts (listed below) were included in the dataset. The number of samples for each class (determined using the procedure above) is listed in Table 3. As the number of samples is unbalanced between the classes, sample weighting was used when training networks on this dataset to compensate.

Class	**Number of samples**
Street	3455
Field	1068
Total	4523

Table 3: The number of target samples in the Streets vs. Fields dataset.

Like the Kitchens vs Bathrooms dataset, artificial correlations were introduced in the training set. The correlations in this dataset are subtler than brightly coloured squares: instead, the samples were selected such that 80% of the street photos contained at least one person, and that only 10% of the field photos contained at least one person. For the validation dataset, both the street and field photos had a 50% proportion of people.

The selected concepts for this task were car, motorcycle, cow and horse. Though cars can appear in fields and horses can appear on roads, these concepts are generally associated with only one of the classes. The concept images were generated using the object bounding boxes from the COCO dataset annotations. The full images were cropped to the area surrounding the object. The number of samples for each object is listed in Table 4. Though the original dataset contained an order of magnitude more images of cars, only a limited selection was included to prevent the resulting dataset from being too imbalanced between concepts.

Concept	**Number of samples**
Car	700
Motorcycle	698
Cow	600
Horse	512
Total	2510

Table 4: The number of concept samples per concept in the Streets vs Fields dataset.

4.2 Comparison Experiment

To achieve an accurate comparison between methods, it is important that the number of trainable parameters of the resulting networks is roughly equal; otherwise, any differences in accuracy can simply be due to the differences in number of parameters. As such, as baseline network was used with the following architecture: a 64x64x3 input layer, followed by two blocks of a convolutional layer with 32 filters and a max-pooling layer with a window size of 2x2, followed by two dense layers with 10 neurons. It used the Adam optimizer [17] for training.

The networks built to test each method were created in such a way to minimize the differences between this baseline network and the resulting network. The modifications to the baseline for each method are as follows:

- **TCAV Loss:** no changes to the network architecture. An additional term is added to the loss function for each concept that penalizes layers that are not sensitive to those concepts.
- **Multi-Task Learning:** additional outputs are added to the network after the second-to-last dense layer, with one extra dense layer of 10 neurons before each.
- **Leaky Semantic Bottleneck Network:** a dense layer is added after the last max-pooling layer. This is trained as the semantic bottleneck, and connected to the following dense layers. In addition, a dense layer of 5 neurons is connected to the last max-pooling layer as the leak.

Metrics — Two main metrics were used to evaluate the resulting networks: validation accuracy, and the difference between test accuracy and validation accuracy. The test dataset (made out of a split of the training data) contains the artificial correlations described in Subsection 4.1. The validation dataset contains the same markers as the test dataset (squares or people) but distributed evenly over the samples without any correlation to the samples' class. This means that the difference between the test accuracy and validation accuracy can be used as a way to measure to what extent the network learned the artificial correlation. For the purposes of this experiment, a lower difference is a better outcome, as these methods are designed to ignore the artificial correlations and focus on the desired concepts.

4.3 Leaky Semantic Bottleneck Experiment

The Leaky Semantic Bottleneck Network was expanded beyond the small 64x64 images of the comparison experiment. The U-net architecture [30] was used as a basis for the Leaky Semantic Bottleneck Network tested in this experiment.

One medium-sized network was built to accept images of size 200x200 as input. It consists of a semantic bottleneck base and a post-bottleneck section. The semantic bottleneck base uses three downsampling blocks, followed by three convolutional layers, followed by three upsampling blocks. Its output is a mask of concept areas in the original image. The post-bottleneck section simply consists of four convolutional layers, with as output a binary classification of the room type.

Three versions of this network were investigated: a baseline network with the same architecture but trained end-to-end on the target task without training the bottleneck section separately, an SBN as described above, and an LSBN that adds leak connections from the upsampling blocks to the post-bottleneck layers. This allows comparison between the different architectures without significantly altering the number of parameters.

A set of large-sized networks was built to work with large images of size 400x300. These networks followed the same architecture as the medium-sized networks, but contained one additional upsampling block and one additional downsampling block. Like the medium networks, this set of networks consisted of a baseline network, an SBN and an LSBN.

5 Results

5.1 Comparison Experiment

Table 5 lists the accuracy of each of the tested methods, along with that of the baseline network. It is important to note that the test accuracy is computed using the dataset with artificial correlations, and validation accuracy is computed using the unmodified dataset. This means that a network with a high reliance on the artificial correlations would be expected to have high test accuracy and low validation accuracy.

The baseline network has the accuracy of a completely random guesser, while the Leaky Semantic Bottleneck Network achieves both the highest accuracy on the task and the lowest difference between test and validation accuracy. Figure 13 displays this same data visually.

Model	**Test Acc.**	**Validation Acc.**	**Difference**
Baseline	100% ± 0%	48% ± 2.7%	51% ± 2.7%
TCAV	51% ± 5.0%	50% ± 0.5%	2% ± 1.5%
Multi-Task	81% ± 11.5%	66% ± 10.3%	14.5% ± 13.6%
Leaky Semantic Bottleneck	68% ± 7.2%	67% ± 6.4%	2.5% ± 5.4%

Table 5: Accuracy of the different implemented methods, with standard deviations.

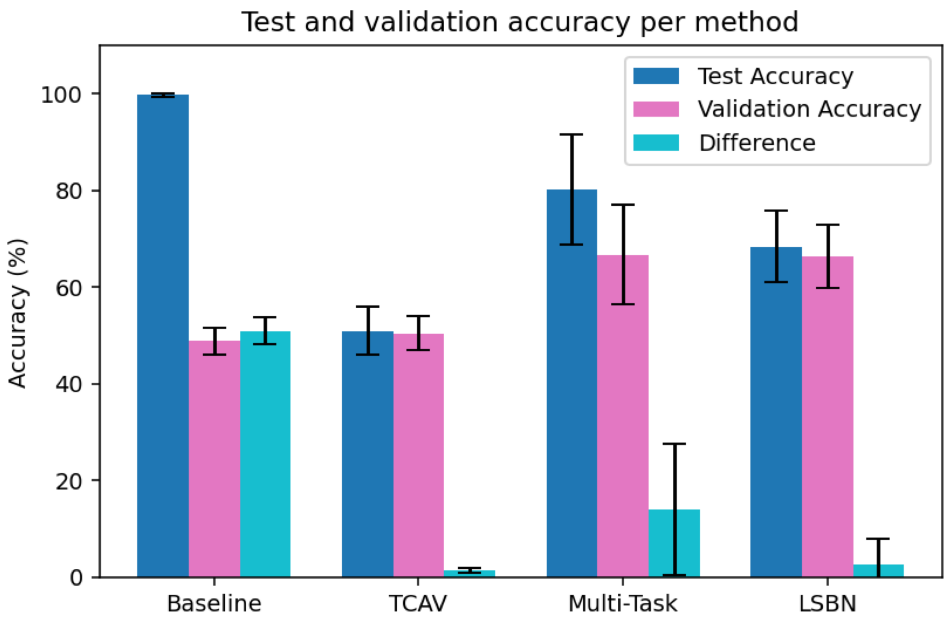

Figure 13: The test and validation accuracy of each method, and the difference between the two.

Table 6 lists the time required to train each tested network. It is clear that the baseline network takes by far the least training time. This can be explained by the fact that the baseline network does not train on the concept data, and simply discards it. This means it only trains on the target data, while each of the other methods additionally trains on the concept data.

Model	**Training Time (s)**
Baseline	108
TCAV	2400
Multi-Task	270
Leaky Semantic Bottleneck	274
Graph-CNN	280

Table 6: The time required to train each of the tested network in seconds.

5.2 Leaky Semantic Bottleneck Network Experiment

Table 7 lists the accuracy of the medium- and large-sized Leaky Semantic Bottleneck Networks (LSBNs), and Figure 14 displays the same data visually. It is clear that for both the medium and large networks, the LSBN has higher accuracy than the baseline network.

One interesting thing to note is that the large LSBN has lower accuracy compared to the medium-sized LSBN. Section 6.2 discusses possible reasons for this.

Model	**Test Acc.**	**Validation Acc.**	**Difference**
Baseline (M)	100%	57%	43%
Leaky Semantic Bottleneck (M)	86%	83%	3%
Baseline (L)	100%	60%	40%
Leaky Semantic Bottleneck (L)	76%	72%	4%

Table 7: Accuracy of the medium-sizes (M) and large-sized (L) Leaky Semantic Bottleneck Networks.

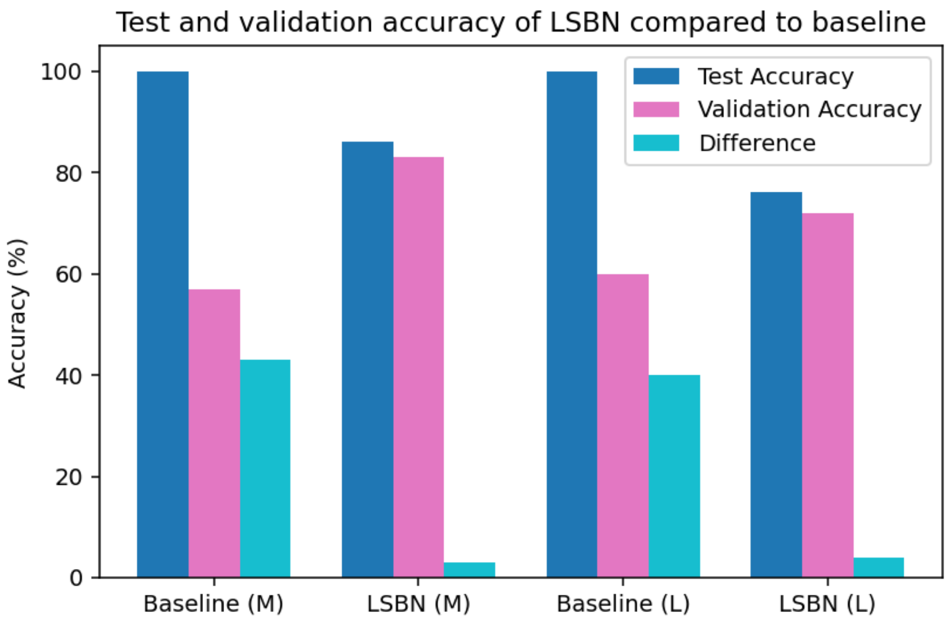

Figure 14: The test and validation accuracy of the LSBN compared to a baseline network.

6 Discussion

6.1 Comparison Experiments

Leaky Semantic Bottleneck Network — The results in Subsection 5.1 suggest that the Leaky Semantic Bottleneck Network (LSBN) is the most effective among the evaluated methods. It has the highest validation accuracy and no difference between the test and validation accuracy. This indicates that the network has not learned the artificial correlations from the dataset, and has instead learned the presented concepts. With this comes a soft guarantee that only concept-related information is processed by the post-bottleneck layers: these layers only receive information from the bottleneck directly, and information that was relevant for training the bottleneck through the leak.

Multi-Task Concept Network — The Multi-Task Concept Network achieves admirable validation accuracy, at an insignificant 1% less than the LSBN. However, it suffers from higher variance and a large difference between the test and validation accuracy. This suggests that while the network has learnt the concepts of interest, it has also learnt the artificial correlations. This means it can still make incorrect decisions based on those artificial correlations, which makes the Multi-Task Concept Network a less appealing option than the LSBN. Nevertheless, it is an improvement compared to the baseline with respect to both the validation accuracy and the difference between test and validation accuracy.

Graph-CNN — The Graph-CNN achieves validation accuracy comparable to the LSBN, but has a significantly higher test accuracy. This makes for an 18% accuracy difference between testing and validation, which suggests that like the Multi-Task Concept Network, the Graph-CNN has learnt the artificial correlations. Coupled with the nonstandard architecture – which requires a more complex implementation – this makes the Graph-CNN a less appealing option than the LSBN.

TCAV Loss Function — The network with the TCAV Loss Function does not achieve accuracy higher than random selection. This disappointing result is not due to the implementation; in early tests on a simpler task, a network with the TCAV Loss Function achieved reasonable accuracy. However, it fails at the room-classification task. This is likely due to the fact that the concepts data for this task is quite varied, leading to an inconsistent TCAV score. In addition to the low accuracy, the training time is an order of magnitude longer than the baseline, making this method impractical for many tasks.

Overall Evaluation — All of the tested models have significantly improved accuracy compared to the baseline model. However, they all need data beyond just the dataset for the target task. In addition to requiring that the concept data be present, it also increases the training time for these models (see Table 6). Comparing the models among themselves, the LSBN jumps out as the most appealing of the four: it has the highest accuracy and the smallest gap between test and validation accuracy.

This is not surprising: compared to the Multi-Task Concept Network and TCAV Loss Function, the LSBN introduces a strong bottleneck of data in the network architecture. The former two methods do guide the network to representations that include the desired concepts, but provide no guarantees as to how this is used when making predictions. In contrast, the LSBN forces the post-bottleneck layers to work with only concept-related data, which provides a soft guarantee that only the concepts will be used for the final predictions.

6.2 Leaky Semantic Bottleneck Experiments

In the LSBN-specific experiments, different image and network sizes were investigated. One would generally expect a larger network and image sizes to lead to increased accuracy. This holds true for the medium LSBN: it has significantly higher accuracy than the small LSBN tested for the comparison experiments. Surprisingly, this does not hold true for the large LSBN: it has lower accuracy than the medium LSBN, and is comparable to the small LSBN in terms of accuracy.

This discrepancy could be due to a variety of factors. One possibility would be that the increased image size requires a significantly larger pre-bottleneck network to accurately segment – but the bottleneck training accuracy remains sufficiently high that this does not appear to be the case. Another possibility is that the higher-resolution segmentation map output by the bottleneck requires significantly more post-bottleneck network capacity to decipher. It is important to find the source of this unexpected behaviour before expanding upon the LSBN.

6.3 Explainability Benefits

Three of the novel methods tested in this thesis also increase the explainability of the network in one way or another. For the Leaky Semantic Bottleneck (LSBN), Multi-Task Concept Network and Graph-CNN one can explain individual decisions by the network by inspecting (some of) the outputs, as explained below. The TCAV Loss Function does ensure that the network conforms to certain TCAV Score requirements, but cannot be used to explain individual network decisions.

Leaky Semantic Bottleneck Network — The LSBN contains a bottleneck that outputs predictions for the presence (or absence) of concepts. This can be used to determine why the network made a certain decision; for example, when the network misclassifies a bathroom as a kitchen, one can look at the bottleneck output to see that a closet was misclassified as a refrigerator, explaining the mistake.

Multi-Task Concept Network — In the Multi-Task Concept Network, the extra outputs can be used for explanations similarly to the semantic bottleneck in the LSBN: by looking at the predictions for the concept outputs one can determine which concepts the network does (not) detect in the input. However, this has less explanatory power than the LSBN, because the predictions can be based on any number of unknown factors in addition to the concepts.

Graph-CNN — Decisions made by a Graph-CNN can be explained by looking at the output of its subnetworks. Each subnetwork generally corresponds to one concept, and so their outputs can be used to determine what concepts the Graph-CNN detects in the input. As the final predictions of the Graph-CNN are based solely on a combination of the subnetwork outputs, this can explain network decisions well.

7 Conclusions

In this thesis, several novel methods are tested to infuse causal knowledge into deep neural networks. Each of the tested methods resulted in a model that was more accurate than the baseline (non-causal) model to varying degrees. In particular, the Leaky Semantic Bottleneck Network (LSBN) demonstrated a strong infusion of causal relations – the tests reveal that it learns the (false) correlations in the dataset to a much smaller extent than the baseline and other methods, relying largely on the infused causal relations for its decisions. However, the Leaky Semantic Bottleneck did demonstrate an unexpected drop of accuracy at the largest tested size. It is important to find the source of this accuracy drop before expanding on the Leaky Semantic Bottleneck Network.

This thesis asked the following research question: *Are deep networks infused with knowledge of causal relationships more accurate than traditional deep networks?* As each of the tested models with infused causal relations performed better than the baseline model, we conclude that models infused with causal relations are more accurate than models that are built purely on correlations learned from the data. Specifically, the causally-infused LSBN achieved an accuracy of up to 83% compared to the baseline of 57%, and the other investigated methods achieved up to 67% accuracy.

7.1 Future Research

One significant downside of the traditional Semantic Bottleneck Network (SBN) is that the information lost to the bottleneck makes it difficult to build a very deep SBN with several bottlenecks. Since the LSBN allows additional information past the bottleneck, future research could focus on creating a network with multiple leaky bottlenecks. This could allow one to build a sequence of course-grained to fine-grained leaky bottlenecks.

TCAV is just one of many different post-hoc explanation and verification methods for neural networks. Future research could focus on replacing the TCAV-based terms in the TCAV Loss Function with terms based on other post-hoc explanation methods. For example, one could incorporate a term that uses Grad-cam [35] to penalize the network for ‘looking at’ portions of the image that do not contain a concept of interest. This would allow one to steer the network representations towards desired concepts and away from unwanted correlations during training.

References

[1] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009. ISBN: 9781601982940.

[2] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[3] Xi Chen et al. "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems* 29 (2016), pp. 2172–2180.

[4] Dumitru Erhan et al. "The difficulty of training deep architectures and the effect of unsupervised pre-training". In: *Artificial Intelligence and Statistics*. PMLR. 2009, pp. 153–160.

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[6] Gabriel Goh et al. "Multimodal Neurons in Artificial Neural Networks". In: *Distill* (2021). https://distill.pub/2021/multimodal-neurons. DOI: 10.23915/distill.00030.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, pp. 164–243. URL: http://www.deeplearningbook.org.

[8] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Commun. ACM* 63.11 (Oct. 2020), pp. 139–144. ISSN: 0001-0782. DOI: 10.1145/3422622. URL: https://doi.org/10.1145/3422622.

[9] Mara Graziani et al. "Guiding CNNs towards Relevant Concepts by Multi-task and Adversarial Learning". In: *CoRR* abs/2008.01478 (2020). arXiv: 2008.01478. URL: https://arxiv.org/abs/2008.01478.

[10] Ruocheng Guo et al. "A survey of learning causality with data: Problems and methods". In: *ACM Computing Surveys (CSUR)* 53.4 (2020), pp. 1–37.

[11] Kaiming He et al. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[12] Robert Hoffman et al. "Explaining explanation, part 4: a deep dive on deep nets". In: *IEEE Intelligent Systems* 33.3 (2018), pp. 87–95.

[13] Robert R Hoffman, Shane T Mueller, and Gary Klein. "Explaining explanation, part 2: Empirical foundations". In: *IEEE Intelligent Systems* 32.4 (2017), pp. 78–86.

[14] Kevin Jarrett et al. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[15] Guoliang Ji et al. "Knowledge graph embedding via dynamic mapping matrix". In: *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015, pp. 687–696.

[16] Been Kim et al. "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)". In: *International conference on machine learning*. PMLR. 2018, pp. 2668–2677.

[17] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[18] Pang Wei Koh et al. “Concept bottleneck models”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5338–5348.

[19] Hengrong Lan et al. “Ki-GAN: Knowledge Infusion Generative Adversarial Network for Photoacoustic Image Reconstruction In Vivo”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 273–281.

[20] Yann LeCun and Yoshua Bengio. “Convolutional Networks for Images, Speech, and Time Series”. In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258. ISBN: 0262511029.

[21] Changsheng Li et al. “Self-Paced Multi-Task Learning”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI’17. San Francisco, California, USA: AAAI Press, 2017, pp. 2175–2181.

[22] Xiaodan Liang et al. “Symbolic graph reasoning meets convolutions”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 1853–1863.

[23] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[24] Yankai Lin et al. “Learning Entity and Relation Embeddings for Knowledge Graph Completion”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI’15. Austin, Texas: AAAI Press, 2015, pp. 2181–2187. ISBN: 0262511290.

[25] Zachary C Lipton. “The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery.” In: *Queue* 16.3 (2018), pp. 31–57.

[26] Max Losch, Mario Fritz, and Bernt Schiele. “Interpretability beyond classification output: Semantic bottleneck networks”. In: *arXiv preprint arXiv:1907.10882* (2019).

[27] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[28] Andrew Y Ng. “Feature selection, L1 vs. L2 regularization, and rotational invariance”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 78.

[29] Raanan Y Rohekar et al. “Constructing deep neural networks by Bayesian network structure learning”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 3047–3058.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[31] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. “Tree-CNN: a hierarchical deep convolutional neural network for incremental learning”. In: *Neural Networks* 121 (2020), pp. 148–160.

[32] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. “Feature selection using a multilayer perceptron”. In: *Journal of Neural Network Computing* 2.2 (1990), pp. 40–48.

[33] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017).

[34] Paul Ruvolo and Eric Eaton. “Active task selection for lifelong machine learning”. In: *Twenty-seventh AAAI conference on artificial intelligence*. 2013.

[35] Ramprasaath R Selvaraju et al. “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.

[36] Towaki Takikawa et al. *Gated-SCNN: Gated Shape CNNs for Semantic Segmentation*. 2019. arXiv: 1907.05740 [cs.CV].

[37] Jiaxing Tan et al. “Expert knowledge-infused deep learning for automatic lung nodule detection”. In: *Journal of X-ray science and technology* 27.1 (2019), pp. 17–35.

[38] Hongwei Wang et al. “DKN: Deep knowledge-aware network for news recommendation”. In: *Proceedings of the 2018 world wide web conference*. 2018, pp. 1835–1844.

[39] Jingyi Xu et al. “A semantic loss function for deep learning with symbolic knowledge”. In: *International Conference on Machine Learning*. 2018, pp. 5502–5511.

[40] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. “Mitigating unwanted biases with adversarial learning”. In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. 2018, pp. 335–340.