Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis Econometrics and Management Science

A Bayesian Convolutional Neural Network Approach for Image-Based Crack Detection and a Maintenance Application

Abstract

Maintenance is an essential task in the industrial sector—the rise of image-detection techniques shows the potential to improve maintenance processes. Convolutional neural networks are appealing with their excellence in prediction accuracy and scalability. However, the frequentist estimated class probabilities from the softmax output function do not measure model certainty and can be misleading in decision-making. Neural network applications can benefit from model certainty to gain trust in the complex deep networks. The Bayesian convolutional neural network uses the dropout technique to cast variational inference to obtain an approximated predictive posterior distribution. The predictive variances aggregate from samples of the approximated predictive posterior distribution, serving as a measure of uncertainty. We use the Bayesian convolutional neural network for crack detection from concrete structure images. Domain experts affirm the novel method's potential in maintenance applications—model uncertainty gains an increased interest and is often a discussion point with their clients. The predictive posterior variance as an evaluation criterion, eliminates the number of false-negative predictions and highlights uncertain predictions.

Keywords: Convolutional Neural Networks, Bayesian Learning, Variational Inference, Maintenance Application

Author:
Ana Tasia Bueno de Mesquita
424406

Supervisor Erasmus University:
prof.dr. S.I.Birbil
Second Assessor Erasmus University:
dr. K. Gruber

January 13, 2021

Contents

1 Introduction

Maintenance plays an essential part in the industrial sector. Cracks and chemical deterioration are common phenomena and can potentially lead to systems failures. Concrete structures such as bridges and dams depend on the full force of the structure. Numerous small damages can initiate a chain reaction leading to the eventual failure of structures. Therefore, the timely detection of small damages is crucial for safe operations. Also, the machinery's unplanned downtime during operations can potentially lead to high operational costs. Traditional manual inspection of complicated structures is labor-intensive, dangerous, and, most importantly: subjective. In the literature, researchers discuss maintenance strategies and try to optimize maintenance planning and gradually replace the reactive strategy with predictive and conditional maintenance strategies.

Over the last years, with the increase of the available data, maintenance optimization approaches are developing. Researchers adhere to different directions; reactive, preventive, and predictive maintenance. Ideally, all maintenance strategies are predictive such that we know when a system is likely to fail, and we know what the optimal repair or maintenance action is at each period in time [Dekker, 1996]. Predictive maintenance allows us to make reliable data-driven decisions, minimize maintenance costs, and guarantee safe operations. On the other hand, the rapid development of machine learning and computer vision techniques illuminates the potential to apply these methods in maintenance applications. Image processing is a broadly applied technique in different areas ranging from face recognition to natural language processing and medical imaging. The convolutional neural network (CNN) is the most popular deep learning tool and is based on feature extraction to make predictions and classification. The convolutional neural network belongs to a class of artificial neural networks (NNs) that build upon mathematical operations. The advantage of the network is the ability to handle high-dimensional data inputs and perform feature extraction to classify the outcome. Deep learning methods are known for their high accuracy but black-box behavior. Moreover, deep learning methods do not include any form of uncertainty. Ideally, a neural network can explain the predicted value (explainability) and indicates its uncertainty substantiated by probability theory (reliability). A probabilistic approach in machine learning provides the ability to include confidence in the predicted value. The uncertainty is necessary to tell if the model is confident about a predicted value or it is just a random guess. This information would help construction workers decide whether an inspection is needed or not, based on image-

detection models. The probability theory in machine learning methods is known as *Bayesian machine learning* [Denker & LeCun, 1991]. The uncertainty can be the missing piece for the practical applicability of machine learning techniques. The idea is to step away from deterministic deep learning models and set up a probabilistic model to include uncertainty. According to Gal & Ghahramani [2015], modern deep learning models are close to probabilistic modeling, and we can get uncertainty measurement by adjusting existing deep learning regularization techniques. Gal & Ghahramani [2016a] proved that repeatably applying dropout regularization after each layer in a neural network, during training and testing, is a variational Bernouille approximation, which gives us the ability to sample from the predictive posterior distribution.

This thesis aims to include uncertainty for image-based detection using the Bayesian convolutional neural network (BCNN). Furthermore, we discuss the untouched area of the newly proposed method's practical application in maintenance processes. The research question is as follows: *Can we quantify the uncertainty in image-based crack detection for concrete structures using Bayesian convolutional neural networks?* If we can quantify the uncertainty, our interest lies in the practical application using the information in maintenance strategies. *Can the image-based detection method using a BCNN contribute to maintenance strategies?* The answers to the research question can potentially increase automated image failure detection's practical applicability in maintenance strategies.

Relevance Without exception, every situation using image-based classification, where the objective is the predictive accuracy and concerns the reliability and uncertainty, can benefit from the applied methods. Furthermore, the research touches on image-based detection methods for maintenance strategies. The latter might be interesting for all companies and governments who currently apply maintenance strategies and could benefit from image-based failure classification. This thesis and its methods might give these parties incentives to include uncertainty measures in their maintenance practices and introduce image-based methods. Finally, the research can prove valuable for researchers interested in either high-end deep learning methods or Bayesian statistical methods. The Bayesian convolutional neural networks establish an intersection between deep learning methods and the Bayesian probability theory.

Motivation The "black-box" principle of neural networks is a significant limitation of deep learning models. The predicted value is often assumed to be accurate and generally used without reasoning about the value or model confidence. Currently, deep learning models cannot determine if the model prediction is intelligent or just a random guess. In 2015 the image classification system of Google erroneously identified African Americans as gorillas and led to a discussion of racial discrimination [Guynn, 2015]. If the network included model uncertainty and indicated its confidence regarding the output, the system might requested a second human opinion for the classification. In a neural network classification setting, the softmax activation function computes the estimated class likelihood. However, these values are often falsely interpreted as model confidence. A model can still be uncertain about the predicted outcome while having a high softmax output probability.

To summarize, the probabilistic interpretation of outputs with uncertainty measurement is crucial as its absence can lead to sub-optimal results and dubious decisions. Moreover, as neural networks are prone to overfitting, the network can fail on previously unseen data, leading to incorrect decisions. The arguments mentioned above give the incentive to include the Bayesian probability theory in the neural network setting. Instead of accepting a point estimate as a predicted value, the Bayesian neural network output, with Monte Carlo dropout, is a posterior probability distribution for each class. The posterior distributions reflects the model uncertainty while simultaneously reducing the risk of overfitting. The concept of including uncertainty in image-based crack detection methods to measure confidence is left open. Furthermore, related research does not include the practical application of image-based detection in maintenance strategies and their potential benefits. The proposed research examines the Bayesian convolutional neural network for crack detection from concrete structure images. This research expects to contribute to the existing literature in the following ways:

- Practical application including uncertainty measurements in deep learning networks for a maintenance application. The inclusion of model confidence in crack detection for concrete structures.
- Domain expert's opinion regarding the novel method and the practical applicability.
- The research motivates readers to reason about the uncertainty of predicted outputs of models.
- The research sets the first step in the direction of automated maintenance strategies.

2 Related literature

The rapid development of deep learning models brought a profound impact on machine learning techniques in real-life situations. Experts previously discussed convolutional neural network applications for failure detection in the manufacturing sector. Chen & Jahanshahi [2018] describe a deep learning framework with convolutional neural networks to detect crack patches in individual video frames. Where Jamshidi et al. [2017] propose a convolutional neural network approach to detect defects in railroad surfaces. Nonetheless, these above studies for failure image-detection do not mention any uncertainty or explanation measurements. On the other hand, researchers wonder about the risk of applying machine learning methods for challenging real-life problems. How can we add model uncertainty to the deep learning techniques? Especially in critical domains where the outcome of a network could determine life or death. Luckily, previous researchers already investigated the concept of adding probability theory to the networks. An important breakthrough was the paper of Neal [1992]. He concluded that deep learning techniques are related to Gaussian processes by adding a probability distribution over infinitely network weights: a Bayesian neural network. He was the first to apply Markov Chain Monte Carlo sampling for Bayesian neural networks. Many researchers investigated the topic during that time; it was a golden era for Bayesian learning in neural networks. Researchers were investigating and extending methodologies to include the probabilities in neural networks. MacKay [1992] extended the Bayesian neural network for a finite number of weights. He used the techniques of Tishby et al. [1989] and Gull [1989] to force a probabilistic interpretation on the neural networks. Later, Hinton & Van Camp [1993] proposed variational learning in Bayesian neural networks, techniques to approximate intractable integrals for Bayesian learning. Where Barber & Bishop [1998] extended the method with a Gaussian variational approximation for the Bayesian network weights. The research formed two paths to include a posterior over the network weights. The Markov Chain Monte Carlo method (MCMC); sampling from a distribution converging to the true posterior and variational Bayes (VB); finding an approximating distribution and minimizing the difference with the true posterior. The MCMC method is a traditional sampling algorithm to estimate the marginal posterior densities. Nevertheless, Bayesian inference for NNs requires the joint posterior density. The VB is prevalent due to its scalability and the approximation of the joint posterior distribution. However, due to the highly complex analytical nature and the intractable integrals, researchers did not widely apply the methods according to Graves [2011]. Recently, researchers are trying adjust the methods to find scalable methods to include model

uncertainty example are the methods of Beal [2003], Graves [2011] and Blundell et al. [2015]. The recent increase in the dimension of neural networks and the rise of deep learning led to the previous methods' incapability to keep up. These methods scale poorly to the current high dimensional architecture of the weight space, especially for the high dimensional weight space of the convolutional neural networks according to Gal & Ghahramani [2016a]. They give one example on the method of Blundell et al. [2015] and declares that the adjusted variational inference, Bayes by Backprop, using a Gaussian approximating distribution doubles the amount of parameters without increasing the predictive performance. Gal & Ghahramani [2016a] state that the approach is not suitable for CNNs as the increase of parameters is computationally costly and proposes a Bernoulli approximating variational distribution with dropout instead. Gal & Ghahramani [2016a] express that their method is scalable to large data sets and complex models without the loss of performance.

Overall, the literature gives an extensive overview of the different methods with the inclusion of probabilities over the model parameters and achieve inference on the posterior distribution. This thesis focuses on the practical application of a convolutional neural network with uncertainty measures. The proposed research follows the reasoning of Gal & Ghahramani [2016a] to approximate the variational inference using the dropout layers and an approximating distribution. The goal is to obtain accurate model performance and a measure of uncertainty in a feasible time frame which would be acceptable in practice. The Bayesian convolutional neural network is already applied to empirical research, as described above. However, the performed research did not address the methods' practical applications as it mainly focuses on the theoretical aspects. No previous research investigates the practice of casting Bayesian convolutional neural networks for a maintenance application in practice.

3 Methodology

The Bayesian convolutional neural networks aggregate from neural networks and Bayesian learning. This section explains the concepts to construct the convolutional neural network and connect the machine learning technique to Bayesian learning. We start with the theoretical concepts of the neural networks in 3.1, followed by the theory of convolutional neural networks in 3.2, and continued with Bayesian learning in 3.3 and 3.4. Finally, we synthesize the concepts in 3.5 and 3.6, and evaluate the model uncertainty in 3.7.

3.1 Neural networks

Artificial neural networks are a type of machine learning technique. Machine learning techniques extract patterns from data based on feature learning [Goodfellow et al., 2016]. As a result, we need to convert the real-world data into a set of features to run the technique for a specific problem. This thesis conducts a technique to detect failures from image data. In this case, it is hard to describe the feature "failure" in terms of pixel values stored in a structured format. An attempt could be to define a crack as a simple geometric shape and use the geometric shape as a feature; even then, it is impossible as every crack is different and does not have the same geometric shape. Furthermore, the image's surroundings, weather conditions, shadows, and other factors make it hard and erroneous to represent the crack as a geometric feature. **Deep learning** architectures solve this problem and learn from data representations. Deep learning is a subset of machine learning and allows the model to learn from unstructured data, i.e., extract high-level abstract features from raw data inputs. In the neural network framework, deep networks are neural networks with multiple layers. The network can learn the patterns of the raw input data extensively. It breaks the complicated task into a series of simplified mappings by adding multiple layers. Each hidden layer corresponds to a simplified mapping with a specific task to learn the essential concepts for the output [Goodfellow et al., 2016]. We discuss the (deep)neural network's architecture, the parameter learning algorithm, and a regularization technique.

Deep Neural Networks Neural networks encompass a large class of models and learning methods based on discovering features from observed input data. The central idea is to extract linear combinations of the input data as derived features and model the output by applying non-linear functions. Friedman et al. [2001] describes a neural network as a non-linear statistical model, an iterative regression or classification model using a network diagram. Given a data set $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ for $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{y}_i \in \mathcal{Y}$, with $\mathcal{Y} = \{1, \ldots, K\}$ with K classes. We construct the classifier, $f : \mathcal{X} \to \mathcal{Y}$, which defines a mapping for each output $\boldsymbol{y}_i = f(\boldsymbol{x}_i, \boldsymbol{\beta})$ and assigns a specific input $\boldsymbol{x}_i$ to a class in $\mathcal{Y}$. Moreover, it learns the values of the unknown network parameters $\boldsymbol{\beta}$ resulting in the best function approximation. Convolutional neural networks are a special kind of feedforward networks, where the information flows in one direction. Figure 1, shows an example of a neural network representation with one hidden layer.

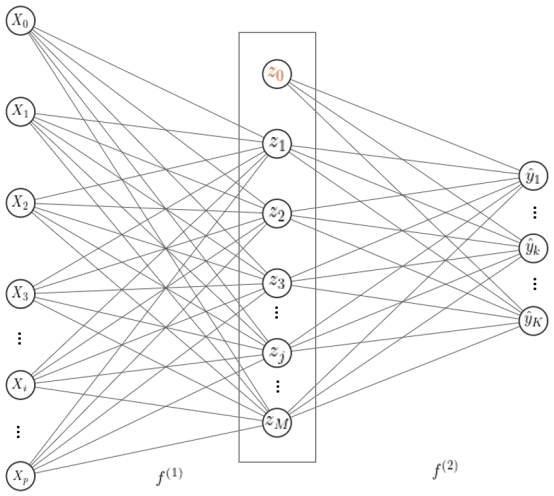


Figure 1: A single-hidden layer neural network framework with input vector $\boldsymbol{x}_i = \{x_1, x_2, \ldots, x_p\}$ with p the dimension of the input vector. The derived features are represented by the vector $\boldsymbol{z} = \{z_1, z_2, \ldots, z_M\}$ with dimension M. The output vector $\boldsymbol{y}_i \approx \hat{\boldsymbol{y}}_i = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_K\}$ with number of classes equal to K.

The network connects the nodes of the graphs through weighted edges where the weights are the unknown model parameters $\boldsymbol{\beta}$. The above figure represents a single-hidden-layer neural network. The first layer corresponds to the input layer with the input nodes for the input $\boldsymbol{x}_i$ with p features. The second is a hidden layer where derived features, $\boldsymbol{z}$, are obtained from a function operation f, on the linear combination of input values and unknown weights $\boldsymbol{\beta}$. The layer is a hidden layer with the hidden units, $\boldsymbol{z}$, as we do not observe the units directly. The last output layer applies a final output function on the complete information received from the network. The number of estimated output values $\hat{\boldsymbol{y}}_i$ depends on the number of classes, K in the classification problem. If we use a simplified notation, the network obtains the estimated output values by applying a chain of functions on the input data, $\hat{\boldsymbol{y}}_i = f^{(2)}(f^{(1)}(\boldsymbol{x}_i, \boldsymbol{\beta}))$, with unknown weight parameters $\boldsymbol{\beta}$. A single hidden layer neural network is insufficient to process the high-level abstract features from image data inputs. We need to make an extension to a multiple-hidden layer network; a deep learning network. Each layer of a deep learning network has a function operation $f \in \mathcal{F}$ to perform a unique task on the input. For example, a neural network with three hidden layers, has four function operations $f^{(1)}, f^{(2)}, f^{(3)}, f^{(4)}$ and obtains the estimated output using the chain of functions to get the estimate $\hat{\boldsymbol{y}}_i = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}_i, \boldsymbol{\beta}))))$. Where $f^{(1)}$ is the first layer, $f^{(4)}$ the last output layer. The length of the chain, i.e. the number of layers, determines the depth of the network. The derived features in each hidden layer, $\boldsymbol{z}^{(l)}$, are constructed from a linear combination of the input values and mapped

through a non-linear activation function. Afterwards, the output values, $\hat{\boldsymbol{y}}_i$, are modeled as a function of linear combinations of the hidden components through the activation output function. We can formally write a one layers NN down with model parameters $\boldsymbol{\beta}$,

$$\begin{aligned} a_j^{(1)} &= \beta_{0j}^{(1)} + \beta_j^{(1)T} \boldsymbol{x}_i, \\ z_j^{(1)} &= g(a_j^{(1)}), \\ a_k^{(2)} &= \beta_{0k}^{(2)} + \beta_k^{(2)T} \boldsymbol{z}^{(1)}, \\ \hat{y}_{ik}(\boldsymbol{x}_i, \boldsymbol{\beta}) &= \sigma(a_k^{(2)}). \end{aligned} \tag{1}$$

The functions $f^{(2)}$ and $f^{(1)}$ are replaced by $g(.)$ and $\sigma(.)$ in (1) for a given input $\boldsymbol{x}_i$ in $\mathcal{X}$. For each node z_j in the hidden layer, the network computes a linear combination $a_j^{(1)}$ with the input value $\boldsymbol{x}_i$, the weights $\beta_j^{(1)}$, and the bias $\beta_{0j}^{(1)}$. Afterwards, the layer passes the linear combination through its activation function, $g(.)$, to compute the value of the node z_j. The network repeats the above steps to compute the estimated output value $\hat{y}_{ik}$ by constructing the linear combination of the derived features $\boldsymbol{z}$ and the output activation function $\sigma(.)$. The bias parameters, $\beta_{0j}^{(1)}$ and $\beta_{0k}^{(2)}$ are present in every hidden unit and corresponds to the intercept in a linear model, the constant value one as an additional input feature.

A deep network, a network with multiple layers, creates a chain of activation functions, with a linear combination of the previous layer's weights used as input. For each estimated value $\hat{y}_{ik}$ the chain of functions is computed,

$$\hat{y}_{ik}(\boldsymbol{x}_i, \boldsymbol{\beta}) = \sigma\left(\boldsymbol{\beta}_k^{(L)T} g\left(\boldsymbol{\beta}_s^{(l-1)T} g\left(\ldots g\left(\boldsymbol{\beta}_j^{(1)T} \boldsymbol{x}_i\right)\right)\ldots\right)\right), \tag{2}$$

where l denotes the number of hidden layers in the network, $l = (1, 2, \ldots, L)$. The choice of the activation function in the network plays a crucial part, depending on the type of tasks the network performs. A commonly used output activation function for classification problems is the **softmax** output activation function. The softmax function predicts probabilities for each class as,

$$\begin{aligned} \hat{\boldsymbol{y}}_i(\boldsymbol{x}_i, \boldsymbol{\beta}) = \sigma(a_k) &= \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}, \\ P(\hat{y}_i = k | a_k) &= \sigma(a_k). \end{aligned} \tag{3}$$

The output activation function in (3) converts any real-valued input into a value in the range of $(0, 1)$ and interprets the output value as a probability. The **Rectified linear unit (ReLU)** is the default choice as hidden layer activation function. The Rectified linear unit applies the activation function on the linear combination of the previous layer's input and the weights, $a_j^{(l)}$, we obtain the derived features of the l^{th} hidden layer,

$$z_j^{(l)} = g(a_j^{(l)}) = max\{0, a_j^{(l)}\}. \tag{4}$$

One advantage of using the ReLU function is a fast and easy optimization, as it resembles a linear activation function [Goodfellow et al., 2016].

Parameter learning The network parameters $\boldsymbol{\beta}$ are unknown, and the network needs to learn their values to estimate the output. The learning algorithm adjusts the parameters $\boldsymbol{\beta}$ to optimize the network and minimize the error. An optimization problem is a set-up based on a specified cost-function to obtain the unknown parameter values. For classification, the commonly used loss function, $\mathcal{L}$, is the **cross-entropy** between training data and model's prediction,

$$\mathrm{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log\, (\hat{y}_{ik}(\boldsymbol{x}_i, \boldsymbol{\beta})), \tag{5}$$

where $\mathrm{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{N} E_i(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$ and $\boldsymbol{\beta}$ the complete set of network parameters. The softmax loss function in (5) measures how far the current network is from the desired one. The stochastic gradient descent (SGD) method is the generic approach to minimize the loss by back-propagating through the network [Rumelhart et al., 1986]. The SGD repeatedly adjusts the weights by minimizing the difference between the desired output vector and the predicted output vector, and converge to an optimum. Back-propagation allows the information calculated by the cost function to flow backwards through the network: a backward pass. The backward pass computes the gradient i.e., the partial derivatives of the cost function, and tells us in which direction the network should move and adjust its parameters,

$$\begin{aligned} \min_{\boldsymbol{\beta}} \mathrm{E}(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \min_{\boldsymbol{\beta}} \sum_{i=1}^{N} \mathrm{E}_i(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i), \\ \frac{\partial \mathrm{E}}{\partial \boldsymbol{\beta}^{(l)}} &= \sum_{i=1}^{N} \frac{\partial \mathrm{E}_i}{\partial \boldsymbol{\beta}^{(l)}} = 0. \end{aligned} \tag{6}$$

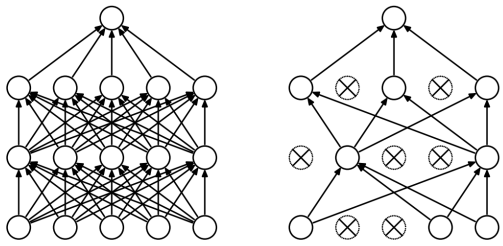
The backward pass starts with computing the gradient for output units, $\frac{\partial \mathrm{E}_i}{\partial a_k} = \hat{y}_{ik} - y_{ik}$. Afterwards, the chain rule with respect to the learnable parameters in the hidden layers is applied, $\frac{\partial \mathrm{E}_i}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial \beta_s^{(l)}}$ with s the number of hidden units in each layer. By doing the backward pass and applying the chain rule for differentiation, we know how a change in the input $\boldsymbol{x}_i$ to the output unit, $\hat{\boldsymbol{y}}_i(\boldsymbol{x}_i, \boldsymbol{\beta})$, will affect the error. The gradients of each layer indicate how each layer's output should move to reduce the error. As a result, the gradient immediately updates the weights' value using a stochastic gradient descent. The SGD updates the network parameters at each iteration $\tau + 1$ iteration in the form,

$$\boldsymbol{\beta}^{\tau+1} = \boldsymbol{\beta}^{\tau} - \eta \nabla \mathrm{E}_i^{\tau}, \tag{7}$$

for each observation $\boldsymbol{x}_i \in \mathcal{X}$ and the gradient estimate E_i^{τ} at iteration τ. The parameter τ is the number of iterations, and η is the learning rate. Equation (7) processes one observation at a time and updating the gradient after each training case. A training epoch is a complete pass through the entire training set. The SDG type of updating allows the network to handle large training samples and update the weights as soon as new information comes in. The learning rate η is a constant value determining the step-size of the update. For example, a learning rate equal to 0.1 means that the weights are updated 0.1 times estimated weight's error. An obstacle with deep neural networks and SDG can be vanishing gradients. The increasing number of layers can lead to exploding or decreasing gradients as the network passes down to the initial layers [Bengio et al., 1994]. As a result, the weights will not be updated correctly during training and can lead to inaccurate results.

A common phenomenon in machine learning is an excellent performance on the training data with the tendency of overfitting to previously unseen input data [Goodfellow et al., 2016]. The difference between the training and test error is large when overfitting occurs. Researchers came up with regularization techniques to reduces the test error and move towards the training error. Goodfellow et al. [2016] describe regularization as any modification made to the learning algorithm that reduces the test error but not the training error. Regularization can be in the form of additional constraints or adding regularization parameters in the objective functions. Regularization techniques try to minimize the bias-variance trade-off; we want to obtain a small bias while reducing the variance. Recall, we want to minimize the cost function such that the predicted outcome value is $\hat{y}_{ik}$ is close to the actual outcome value [James et al., 2013].

Dropout regularization Dropout, introduced by Srivastava et al. [2014], is one of the most powerful regularization techniques for deep learning methods with a large number of parameters. The key idea is to randomly remove units and connections from the network during the training phase. The technique originates from averaging predictions over all possible settings for the parameters in many different trained nets. The method reduces the risk of overfitting by continuously combining multiple models. Dropout prevents overfitting and efficiently combines exponentially many differing networks. Figure 2 shows an example of dropout regularization and the obtained sub-network. With dropout regularization, the network trains an ensemble of sub-networks with removed units and connections called the thinned networks.



(a) Standard Neural Net (b) After applying dropout.

Figure 2: The dropout regularization technique. The left image shows a standard 2-hidden layer network.The right image shows the regularization dropout on the network during training phase. For the dropout the hidden and visible crossed units and their connections are randomly removed from the network [Srivastava et al., 2014].

The units are removed at random with a fixed independent probability p during the training phase. Training a network with n units using dropout regularization can be seen as training the collection of thinned networks with substantial weight sharing, a collection of 2^n possible thinned networks. Instead of averaging the thinned models' predictions at the test phase, Srivastava et al. [2014] propose an approximate averaging method. The approximating method uses the full network at test time while the units' weights are a scaled-down version of the training weights. During training, a unit is present with a probability p while at the test phase the unit is always present, and its outgoing weight is scaled-down by multiplying with the probability. Multiplying the outgoing weight with p ensures equality of the expected outcome during the training and test phase. Srivastava et al. [2014] model the feedforward neural network using dropout during training,

with linear combinations $a_j^{(l)}$, hidden units $z_j^{(l)}$, and activation function $g(.)$ as,

$$\begin{aligned} r_s^{(l)} &\sim \text{Bernouille}(p), \\ \tilde{\boldsymbol{z}}^{(l)} &= \boldsymbol{z}^{(l)} \cdot \boldsymbol{r}^{(l)}, \\ a_j^{(l+1)} &= \boldsymbol{\beta}_s^{l+1} \tilde{\boldsymbol{z}}^{(l)} + \beta_{0s}^{(l+1)}, \\ z_j^{(l+1)} &= g(a_j^{(l+1)}), \end{aligned} \tag{8}$$

with s is the number of units in the previous layer l. The method multiplies the vector $\boldsymbol{r}^{(l)}$, of independent random Bernouille variables , with $\boldsymbol{z}^{(l)}$ to obtain the thinned outputs $\tilde{\boldsymbol{z}}^{(l)}$ for each layer. By doing so, the method mimics sampling a sub-network from the complete network. At test time the weights are scaled-down as, $\boldsymbol{\beta}_{test}^{(l)} = p\boldsymbol{\beta}^{(l)}$. Srivastava et al. [2014] conclude that applying dropout is equivalent to taking an equally-weighted average of exponential many models with shared weights. In addition, Srivastava et al. [2014] compare dropout regularization to the Bayesian neural network in Neal [2012]. The main difference is that the Bayesian neural network is a weighted average of networks where the weights depend on the prior and the data. The Bayesian neural networks [Neal, 2012] have an increasing number of parameters and, therefore, difficult to scale to deep networks.

3.2 Convolutional neural networks

Convolutional neural networks are a special kind of neural networks that extract hierarchical features from abstract data structures. The CNN is especially useful for extracting features from image data, which are pixel values stored in arrays. Fukushima & Miyake [1982] were the first to incorporate the idea of the hierarchy structure in the visual cortex with neural networks [Hubel & Wiesel, 1962]. The hierarchy structure in the visual cortex by Hubel & Wiesel [1962] uses the idea of simple and complex cells in the brain and combines these two types of cells, cascading a model for pattern recognition. The method of Fukushima & Miyake [1982] uses convolutional and downsampling layers to recognize intricate patterns. The multi-level hierarchical structure of the convolutional neural networks eliminates the gradient descent learning shortcoming [Hinton et al., 2006]. The convolutional neural networks developed over the years and led to the networks we know today [LeCun, 1998].

The CNN models for image classification by Zeiler & Fergus [2014] and LeCun [1998] map the input images $\boldsymbol{x}$, through a set of layers into a probability vector $\hat{\boldsymbol{y}}$ for K different classes. The building blocks to extract the features and model the outcome are the convolutional, pooling, and fully connected layers. Each of the layers has different functionalities and characteristics. The convolutional and the pooling layers perform the spatial feature extraction from the image while the fully connected layer maps the extracted features to the model's final output value. Figure 3 shows the general framework with different layers.

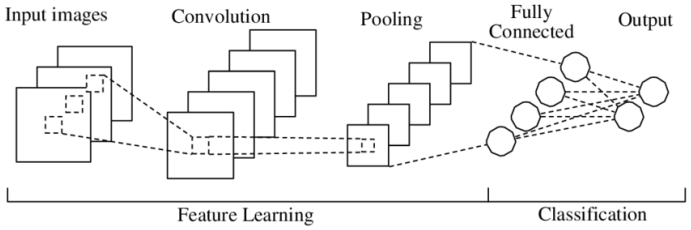
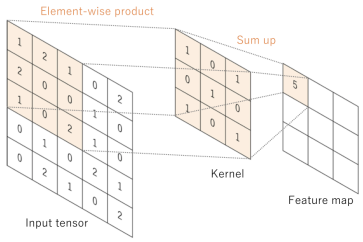
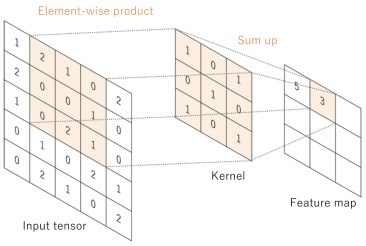


Figure 3: A general framework of a convolutional neural network for classification with the convolutional, pooling, and fully connected layer present. The convolutional and pooling layer performs the spatial feature extraction from the image input. While the fully connected layer maps the extracted features to the different classes [Li et al., 2018].

Convolutional layer The convolutional layer is the primary operation of CNNs. Convolution is a special kind of linear operation used to extract the features from image data. The operation consists of a kernel, a small array with numbers that slides over the full input image array called the tensor to obtain the output, i.e. the feature map. More specifically, the method calculates the sum of each element's element-wise product in the kernel and the tensor. The kernel slides over the input (from left to right, up and down) to cover the complete input following a stride m. The stride determines the step-size of the sliding movement of the kernel. A different kernel corresponds to a different feature extractor. The network repeats the convolution operation using various kernels to extract different and more complex features from the image. In Figure 4, we show an example of the convolution operation. Figure 4a, shows the feature map output as the sum of the element-wise product of the input tensor and the kernel, Figure 4b shows the movement of the kernel to compute the next element of the feature map.



(a) Convolution output for the first element.



(b) Convolution output for the second element.

Figure 4: A convolution operation on a 5x5 input array an a 3x3 kernel to obtain a 3x3 feature map. The stride or step size is equal to one so the kernel slides from left to right and from top to bottom one step at a time [Yamashita et al., 2018].

The convolution operation follows the central idea of weight sharing. The weight sharing property is caused by the kernels, using the same weight for more than one function in the network. Every position in the image array share the same kernels, and the network does not need to learn a separate set of weights for each location. Weight sharing increases the efficiency of the model by reducing the number of parameters to learn, detecting local patterns and make the network equivariant to translations. Equivariance to translation means that if the input changes, the output changes in the same order. The next step is to pass the convolutional layer's output through a non-linear activation function, such as the ReLU activation function discussed in Section 3.1.

Pooling layer Generally, the pooling layers follow each convolutional layer in the network. The pooling layer reduces the in-plane dimensional of the feature maps using summary statistics of neighboring values. This downsampling operation introduces a translation invariance to small shifts of input values to the pooled outputs. Besides, the operation further reduces the number of learnable parameters. A typical summary statistic used for pooling is the **max pooling**, a simplified pooling operation that only reports the maximum value of neighborhood input. The pooling filter operates in the same way as a kernel, moving across the input with specific filter size and step-size or stride. Figure 5 shows an example of the max pooling operation and its downsampling property on a 4x4 input tensor and a 2x2 pooling filter. The operation reduces the input tensor (height and width) using a two-step stride by a factor two. The depth dimension of the input remains unchanged after the pooling operation.

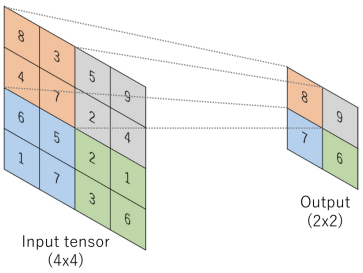


Figure 5: Max pooling operation on an 4x4 input tensor and a 2x2 pooling filter and a stride of two steps [Yamashita et al., 2018].

The idea behind pooling is to downsample the feature maps independently while storing important information. The pooling operation is an essential part of the CNN as the classification layer requires a fixed-size input to classify. With pooling layers, the network can handle different input sizes but always deliver the same number of summary statistics to the classification layer regardless of the image size. Generally, a full convolutional neural network consists of successive convolution and pooling operations to extract the complex features. For example, the first layer extracts the edges while the second layer extracts the shape, and the following layers combine the information to specify the type of object and detect the feature.

Fully connected layer After the final convolutional and pooling layer, the network flattens the three-dimensional output feature map to a one-dimensional array by a vectorization. The fully connected layer connects every input of the one-dimensional array to output values by the learnable weights. To summarize, the network extracts and creates the features using the convolutional and pooling layers. Afterwards, the fully connected layer maps the extracted features to model the final estimate, the probability for each class. We use the ReLU activation function for the hidden units and the softmax function to model the output class. The network optimizes its weight using backpropagation with gradient descent.

LeCun [1998] proposed a convolutional neural network architecture called *LeNet-5*, also called LeNet, which is the building block for the CNN. The architecture consists of seven layers, three convolutional layers, two pooling layers, and two fully connected layers. Other popular CNNs build-up on the *LeNet-5* network are *AlexNet*, *GoogleNet* and *ResNet*.

3.3 Probabilistic approach

Neural networks are poor at representing uncertainty in the predicted outcome. An optimal model for decision-making should be dealing with all sources of uncertainty. We can distinguish between two major sources of uncertainty in a model, namely aleatoric and epistemic uncertainty. Aleatoric uncertainty captures the irreducible uncertainty, such as pixel noise. Even with unlimited data availability, this type of uncertainty will always be present. The second type of uncertainty is epistemic uncertainty and captures the uncertainty in the model and model parameters. The epistemic uncertainty is reducible as the uncertainty grows from a lack of knowledge of the model and will decrease if more data is available [Sensoy et al., 2018].

Probabilistic frameworks try to include uncertainty in the model and its predicted value. In the above described neural network in Section 3.1, we obtain the unknown parameters of the network using a back-propagation algorithm resulting in a single point estimate for the model weights and predicted value. Bayesian methods provide a natural probabilistic representation of uncertainty by learning from the data and propagating it into the predicted value. Learning from data is the building block of Bayesian theories, updating the posterior probability whenever new data comes adjusted by the prior distribution. The idea is to use a probability distribution over the model weights to obtain a level of confidence for the predicted values. In Figure 6, we show the idea of weight initialization in the probabilistic framework. The left figure shows a standard single-hidden layer neural network with fixed values on the weight. In comparison, the right image shows the neural network approach with distributions over the model weights.

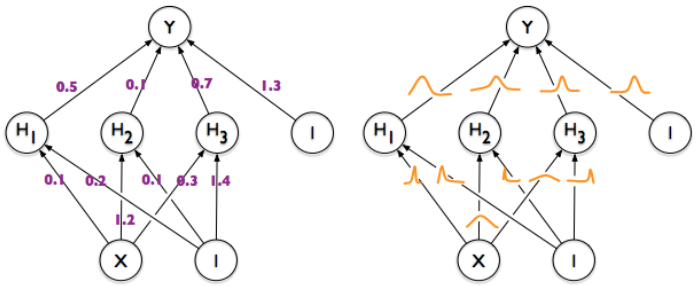


Figure 6: Left: a standard NN setting where each weight has a fixed value. Right: a Bayesian set-up with a probability distribution over the model weights.

3.4 Bayesian statistics

The idea of probabilistic modeling with Bayesian statistics is to describe the observed data and simultaneously express all forms of uncertainty in our model. The Bayesian framework distinguishes itself by looking differently at the probability theory and statistical inference. Bayesians express a probability distribution over all unknown model parameters rather than accepting a fixed value. Resulting in a probability distribution over all unknown quantities and expressing our belief how likely the different values are. Contrasting, frequentists rely on repeated sampling and asymptotic theories [Neal, 2012]. We define a Bayesian neural network by placing a prior distribution over the model parameters, the weights. As soon as new data comes in, we update the prior distribution with the new information into the posterior distribution. The models learn as soon as new data comes using the standard **Bayes Rule**. For simplicity, assume the network with $\boldsymbol{\beta}$ the complete set of weights and training data $\mathcal{D}_{train} = \{\mathcal{X}, \mathcal{Y}\}$. Where $\mathcal{X}$ is the observed data and $\mathcal{Y}$ are the corresponding labels for the K classes. We can write the neural network described in Section 3.1 as a probabilistic model, $P(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\beta})$: for a given input $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{y}_i \in \mathcal{Y}$ from the training data. The likelihood, $P(\mathcal{Y}|\mathcal{X}, \boldsymbol{\beta})$, is the distribution for the target values given the inputs $\mathcal{X}$ and the parameters of the network $\boldsymbol{\beta}$. We use Bayes rule to compute learning of the posterior distribution $P(\boldsymbol{\beta}|\mathcal{X}, \mathcal{Y})$ with the likelihood $P(\mathcal{Y}|\mathcal{X}, \boldsymbol{\beta})$ and prior for the weights $P(\boldsymbol{\beta})$ by,

$$P(\boldsymbol{\beta}|\mathcal{X}, \mathcal{Y}) = \frac{P(\boldsymbol{\beta})P(\mathcal{Y}|\mathcal{X}, \boldsymbol{\beta})}{P(\mathcal{Y}|\mathcal{X})}. \tag{9}$$

Gal & Ghahramani [2016b] show that the dropout regularization applied in a neural network is a Bayesian approximation, namely Gaussian processes (GP). A Gaussian Process is built op on the multivariate Gaussian probability distributions and stochastic processes where the goal of the Gaussian process is to learn the underlying distribution from training data respective to the test data. Neal [2012] derived the equivalence between the Gaussian process and non-parametric models such as neural networks and deep networks. We are interested in the Bayesian convolutional neural network and place a prior on the network parameters to obtain an approximation for the predictive posterior distribution. Gal & Ghahramani [2016a] relates the Bayesian theory to CNNs and shows that applying the dropout technique during the network's training and testing casts an approximate Bernoulli variational inference.

3.5 Bayesian neural networks

The objective is to include the parameter uncertainty in the predicted output $\boldsymbol{y}_i$ for the unseen data $\boldsymbol{x}_i$,

$$P(\boldsymbol{y}_i|\boldsymbol{x}_i, \mathcal{D}_{train}) = \int P(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\beta}) P(\boldsymbol{\beta}|\mathcal{D}_{train}) d\boldsymbol{\beta}. \tag{10}$$

Equation (10) calculates the predictive distribution by integrating over all possible model configurations for $\boldsymbol{\beta}$. The idea is similar to using an ensemble of an infinite number of neural networks, which is impossible to compute for any practical size. We can not compute the integral in (10). To compute the posterior distribution analytically or sample from efficiently, we need to expand to variational methods.

Variational inference We approximate the intractable distribution using an approximating variational inferences technique to estimate the unknown distribution of the weights in (10) [Beal, 2003]. The posterior distribution in (10) is intractable due to the computational difficulty of the marginal distribution $P(\mathcal{Y}|\mathcal{X})$, which we call the model evidence. Variational inference methods avoid the calculation of the marginal distribution and solve the integral using an optimization with the Kullback Leibler divergence [Jordan et al., 1999]. The idea is to find a set of parameters $\boldsymbol{\theta}$ on a tractable distribution of the model weights $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ that minimizes the **Kullback-Leibler Divergence** (KL) with the true Bayesian posterior distribution of the weights [Kullback & Leibler, 1951]. Minimizing the KL divergence gives us the ability to approximate the predictive distribution as,

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i, \mathcal{D}_{train}) \approx \int p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\beta}) q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta}) d\boldsymbol{\beta} := q^*_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\boldsymbol{x}_i), \tag{11}$$

to obtain posterior results of the predicted value $\boldsymbol{y}_i$.

Kullback-Leibler Divergence The Kullback-Leibler divergence originates from information theory measuring the distance between statistical populations based on an information measurement [Kullback & Leibler, 1951]. We want to find the distribution $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, which is as close as possible to the true posterior distribution. The method finds the minimizing values of variational parameter $\boldsymbol{\theta}^*$ such that the $q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, is the best approximation for the posterior distribution $p(\boldsymbol{\beta}|\mathcal{D}_{train})$:

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p\left(\boldsymbol{\beta}|\mathcal{D}_{train}\right)\right], \\ &= \arg\min_{\boldsymbol{\theta}} \mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p\left(\boldsymbol{\beta}\right)\right] - \mathrm{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}\left[\log p(\mathcal{D}_{train}|\boldsymbol{\beta})\right]. \end{aligned} \tag{12}$$

The first term of second line corresponds to the KL divergence between the variational posterior distribution and the prior on the weights called the complexity costs. The second part is the expected value of the log-likelihood of the data concerning the variational posterior distribution called the likelihood costs. The derivation is based on the following property; the minimization of the Kullback-Leibler divergence is equal to maximizing the log evidence lower bound (ELBO), a lowerbound on the model evidence,

$$\mathcal{L}_{VI}(\boldsymbol{\theta}) := \int q_{\boldsymbol{\theta}}(\boldsymbol{\beta}) \log\ (\mathcal{Y}|\mathcal{X}, \boldsymbol{\beta}) d\boldsymbol{\beta} - \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p(\boldsymbol{\beta})) \leq \log\ p(\mathcal{Y}|\mathcal{X}). \tag{13}$$

The derivations, proofs and formulas for the KL divergence and the ELBO are present in Appendix A. Maximizing the objective in (13) results in an approximating distribution $q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ with optimal parameters $\boldsymbol{\theta}^*$ such that the distribution is as close as possible to the true posterior distribution. Intuitively, the expected log likelihood pushes $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ in the right direction to explain the data well by maximising the first term in (13). While the prior KL encourages $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ to be close to the prior distribution $p(\boldsymbol{\beta})$ through minimizing the second term.

3.6 Variational inference with dropout

Following the method of Gal & Ghahramani [2016a], we use the proof that dropout during training in a Bayesian neural network casts as a Bernoulli approximate variational inference. We approximate the predictive posterior at test time by taking the average of stochastic forward passes called Monte Carlo dropout. First, we define the Bayesian classification neural network. We want to find the network parameters $\boldsymbol{\beta}$ of the function $\boldsymbol{y} = \boldsymbol{f}^{\boldsymbol{\beta}}(\boldsymbol{x})$, which are likely to generate the outputs. Using the Bayesian approach, we define a prior distribution over the network parameters before observing any data points. We place a standard matrix Gaussian prior distribution over the model parameters $\boldsymbol{\beta} = \{\boldsymbol{\beta}^{(l)}\}_{l=1}^{L}$,

$$\boldsymbol{\beta}^{(l)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{14}$$

The likelihood distribution for classification tasks is defined by the softmax likelihood,

$$p(\hat{y}_i = k|\mathcal{X}, \boldsymbol{\beta}) = \frac{\exp(\boldsymbol{f}_k^{\boldsymbol{\beta}})}{\sum_k \exp(\boldsymbol{f}_k^{\boldsymbol{\beta}})}, \tag{15}$$

where $k \in K$ denote the number of classes. We want to obtain the predictive distribution, $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$

$$\begin{aligned} P(\hat{y}_i = k|\boldsymbol{x}_i, \mathcal{D}_{train}) &= \int P(\hat{y}_i = k|\boldsymbol{x}_i, \boldsymbol{\beta}) p(\boldsymbol{\beta}|\mathcal{D}_{train}) d\boldsymbol{\beta}, \\ &\approx \int P(\hat{y}_i = k|\boldsymbol{x}_i, \boldsymbol{\beta}) q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta}) d\boldsymbol{\beta}. \end{aligned} \tag{16}$$

As explained in Section 3.5 the posterior distribution in (16) over the network parameters is intractable and we approximate the posterior with variational methods. The task is to find the variational distribution $q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ to perform an approximate variational inference and show the equivalence with a dropout trained network. We define the approximate variational distribution $q_{\boldsymbol{\theta}}(\boldsymbol{\beta}^{(l)})$ for every layer l of the network as,

$$\begin{aligned} \boldsymbol{\beta}^{(l)} &= \boldsymbol{M}^{(l)} \cdot \text{diag}([\boldsymbol{z}_j^{(l)}]_{j=1}^{K^{(l)}}), \\ \boldsymbol{z}_j^{(l)} &\sim \text{Bernouili}(p^{(l)}), \end{aligned} \tag{17}$$

for the number of hidden layers, $l = 1, \ldots, L$ with hidden units, $j = 1, \ldots K^{(l-1)}$. Where $\boldsymbol{z}_j^{(l)}$ are random Bernoulli distributed variables with probability $p^{(l)}$ for each layer l and hidden unit j. We want to optimize the variational parameters vectors in $\boldsymbol{\theta} = \{\boldsymbol{M}^{(l)}, p^{(l)}\}$ such that for $\boldsymbol{\theta}^*$ the distribution $q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ is a close as possible to the true posterior distribution in (16). We need to maximize the log evidence lower bound,

$$\mathcal{L}_{VI}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \int q_{\boldsymbol{\theta}}(\boldsymbol{\beta}) \log\, p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{\beta}) d\boldsymbol{\beta} - \text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p(\boldsymbol{\beta})). \tag{18}$$

The optimisation of (18) requires an unbiased estimate for the gradient. The first term requires to compute the integral over the entire training data $\mathcal{D}_{train}$, for a deep network with multiple hidden layers the computation, which is costly for large number of observations N. The solution is to estimate the ELBO by sampling $\hat{\boldsymbol{\beta}}$ form from $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$. In each sampling step we replace the integral in by $\log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \hat{\boldsymbol{\beta}}))$ leading to the objective in (19). Gal & Ghahramani [2015] [Appendix 4.1] shows the approximation in (18) to obtain an unbiased estimator,

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) := \sum_{i=1}^{N} \log(p(\boldsymbol{y}_i|\boldsymbol{x}_i, \hat{\boldsymbol{\beta}})) - \text{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p(\boldsymbol{\beta})), \tag{19}$$

where $\hat{\boldsymbol{\beta}} \sim q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$. Sampling $\hat{\boldsymbol{\beta}}$ from $q_{\boldsymbol{\theta}}(\boldsymbol{\beta}^{(l)})$ is equivalent to the dropout regularization in layer l in the network with weights M^l, with $\boldsymbol{z}_j^{(l)}$ sampled from the Bernoulli distribution. Gal & Ghahramani [2016a] [Appendix 4.2] proved that minimizing the KL divergence in (19) the objective is equal to the loss in (5). Finally, we can estimate the predictive distribution in (16) with a Monte Carlo integration to sample from the posterior,

$$\begin{aligned} P(\hat{y}_i = k|\boldsymbol{x}_i, \boldsymbol{\beta}) &\approx \int P(\hat{y}_i = k|\boldsymbol{x}_i, \boldsymbol{\beta}) q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta}) d\boldsymbol{\beta}, \\ &\approx \frac{1}{T} \sum_{t=1}^{T} P(\hat{y}_i = k|\boldsymbol{x}_i, \hat{\boldsymbol{\beta}}_t), \end{aligned} \tag{20}$$

with $\hat{\boldsymbol{\beta}}_t \sim q^*_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, referred to as Monte Carlo dropout. Intuitively, the Monte Carlo dropout takes the average of T times a stochastic forward pass through the deep network. Figure 7 shows an example of the Monte Carlo dropout with three forward passes. Learning the network with stochastic gradient descent with dropout in the networks leads to learning a distribution over the network parameters.

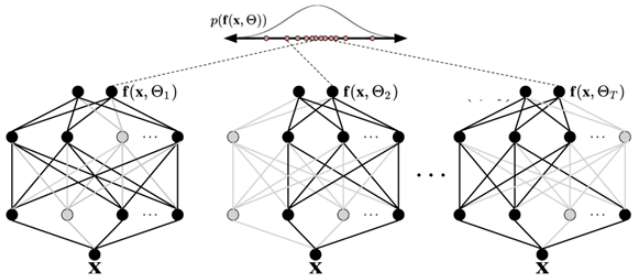


Figure 7: The figure illustrate the MC dropout for a two-hidden-layer neural network with three forward passes. In each forward pass, the dropout layers randomly switch of neurons (grey) leading to different outputs.

3.7 Uncertainty evaluation

We want to evaluate the network's output by the approximate predictive posterior distribution. Generally, researchers only use dropout during training after the fully connected layers. We wish to include the uncertainty over the complete network. As a result, it allows us to use the convolution and pooling operations in the probabilistic framework. The Bayesian CNN includes dropout after each convolutional and fully connected layers. To obtain the samples from the predictive posterior distribution, we use Monte Carlo sampling in (20). To integrate over the kernels, Gal & Ghahramani [2015] reformulate convolution as a linear operation. We place a prior distribution

over the kernels and integrate each kernel-feature map with Bernoulli variational distributions. By sampling Bernoulli random variables $\boldsymbol{z}_j^{(l)}$ and multiplying the weight matrix to obtain $\boldsymbol{\beta}^{(l)} = \boldsymbol{M}^{(l)} \cdot \text{diag}\,([\boldsymbol{z}_j^{(l)}]_{j=1}^{K^{(l)}})$. The Bernoulli distribution randomly sets kernels to zero for different feature maps. This approximating distribution is equivalent to applying dropout for each element in the tensor before the pooling operation. Bayesian CNN uses dropout after every convolutional layer, and before every pooling layer. We approximate the predictive distribution in (20) by averaging T stochastic forward passes through the network during test time, i.e. the Monte Carlo dropout.

Model uncertainty evaluation Besides evaluating the model's predictive accuracy and loss, the interest lies in measuring the uncertainty. More specifically, the epistemic uncertainty, which is the model uncertainty. We want to know how certain a model is on its predicted values. We want to detect the unfamiliar images to model and measure how confident the model is regarding the predicted value. The Bayesian convolutional neural network using a Monte Carlo dropout induces prediction uncertainty in the classification by marginalizing the posterior distribution in (20). For the classification model with a softmax activation function, the approximation of the posterior distribution with Monte Carlo integration is,

$$\boldsymbol{p}(\hat{y}_i = k|\boldsymbol{x}_i, \boldsymbol{\beta}) \approx \frac{1}{T}\sum_{t=1}^{T} \text{softmax}(\boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}(\boldsymbol{x}_i)). \tag{21}$$

In other words, the outcome $\boldsymbol{p}$ is a vector of probabilities obtained by repeatedly sampling values of the approximated predictive posterior for the input value $\boldsymbol{x}_i$. The function $\boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}$ represent the BCNN with model weights $\hat{\boldsymbol{\beta}} \sim q_{\boldsymbol{\theta}}^*(\boldsymbol{\beta})$ and prior distribution $\boldsymbol{\beta}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Finally, we approximate the epistemic uncertainty by the posterior variance, Gal [2016][Section 3.3]:

$$\text{Var}_{q_{\boldsymbol{\theta}}^*(\boldsymbol{y}_i|\boldsymbol{x}_i)}[\boldsymbol{y}_i] \approx \Sigma_A + \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}(\boldsymbol{x}_i)^T \boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}(\boldsymbol{x}_i) - \text{E}(\boldsymbol{y}_i)^T\,\text{E}(\boldsymbol{y}_i), \tag{22}$$

with predictive posterior mean, $\text{E}(\boldsymbol{y}_i) \approx \sum_{t=1}^{T} \boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}(\boldsymbol{x}_i)$ and aleatoric uncertainty (noise) captured by Σ_A [Feng et al., 2018];[Gal & Ghahramani, 2015]. The predictive variance is equal to the aleatoric uncertainty if the second part converges to zero. The second part measures the model's certainty about the predicted values $\boldsymbol{y}_i$ from an input $\boldsymbol{x}_i$ and will be small if the draws of the distribution $\boldsymbol{f}^{\hat{\boldsymbol{\beta}}_t}$ are close. On the other hand, the predictive variance increases if the posterior distribution draws vary for every draw t. We compare the uncertainty evaluation between the CNN models

and the BCNN. As shown in Section 3.2, the softmax activation function converts the continuous activation from the previous layer to class probabilities. The likelihood for an input $\boldsymbol{x}_i$ is a binomial probability mass function, based on formula of the softmax function in (3). At the same time, the cross-entropy loss function optimizes the parameters in (5). According to Sensoy et al. [2018], the probabilistic interpretation of the cross-entropy is equivalent to a Maximum Likelihood Estimation (MLE). The MLE, a frequentist technique, is unable to interfere with the predictive distribution. The output is the point estimate for the class probabilities of an input and does not provide the associated model uncertainty [Sensoy et al., 2018]. Consequently, the CNN predicts a constant probability independent of the number of simulations t; the model repeatedly predicts the same outcome of an input value $\boldsymbol{x}_i$. Figure 8, shows an example of draws from an approximated predictive posterior obtained with a BCNN. For each draw t, the model obtains a different probability for input $\boldsymbol{x}_i$. The figure clearly shows that the variance of the draws captures the uncertainty of the predicted values. A larger variance corresponds to different predicted probabilities and leads to higher uncertainty.

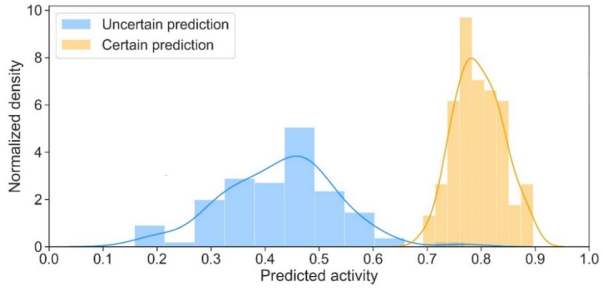


Figure 8: An example of draws from the approximated predicted posterior distribution from a BCNN. We use the variance of the obtained forward passes to measure the uncertainty.

4 Maintenance application

A large number of maintenance strategies for buildings and infrastructures has been around for decades. Besides, due to the increase in technological possibilities and data availability, the concept of maintenance has advanced significantly over the last years. There has been a shift from a traditional reactive approach to proactive maintenance. Generally, researchers discuss three types of maintenance; corrective, preventive, and predictive maintenance [Ahuja & Khamba, 2008];[Jardine et al., 2006]. In contrast with corrective and preventative maintenance, predictive maintenance defers the inspection until needed based on observed data. Predictive maintenance (PdM) strategies

optimize repair planning, considering the remaining useful lifetime with restrictions on performance costs and productivity. It is a proactive approach as it tends to know how to prevent failures under different scenarios. PdM's primary purpose is to carry out maintenance activities whenever faults have propagated to a threshold value; these methods lead to more efficient and economical utilization of a company's assets. Emerging machine technologies enhance the data availability for the PdM [Ran et al., 2019]. These technologies' advancements allow integrated health management to monitor real-time data and dynamically predict the useful residual lifetime. Deep learning methods that detect cracks from images provide real-time data to monitor the health assessment systems, anticipate their failures, and predict the needed maintenance [Gouriveau et al., 2016]. Currently, image-based computer vision methods have witnessed an increasing interest in business applications. A practical example is a computer vision project applied by the Dutch consultancy company (CQM) in 2017 [CQM, 2017]. The consultancy company did a project for VolkerRail, which monitors infrastructures in the Netherlands. VolkerRail performed a track inspection for more than half of all the railway routes in the Netherlands. Previously, people performed a dangerous inspection to check the joints and rails on defects. Recently, special trains, including two cameras, were developed to take pictures of the rails every half meter. With an extensive set of images, CQM implemented a CNN to assign defect probabilities to a specific image. The used network consists of 16 layers with ReLU and sigmoid activation functions modeled in TensorFlow. The used model obtained a false negative rate of $< 1\%$, and as a result, reduced 80% of the dangerous work. A threshold decision rule was defined to determine whether the trails and joints require human inspection. The company decided to base the decision-rule on the class probabilities derived from the Sigmoid output layer. The decision rule states that if the probability of the negative class (no defect) is larger than 0.70, no human inspection is required; this was the case for 80% of the images. In other words, in 20% of the cases, the probability of no defect was less than 0.70, and the probability for the positive class (defect) was at least 0.30 . In the latter, the decision rule decides to send an inspector to check the possible defect. The newly proposed BCNN would only give a performance advantage for this specific case if the prediction accuracy of the BCNN is at least as good as the CNN and the BCNN correctly measures the uncertainty and reduce the number of false negatives. As a result, in the experimental set-up, we compare our CNN and BNN with the business case of CQM. The goal is to find out if the BCNN can give added value in practical applications.

5 Experimental set-up

We compare three different convolutional neural networks in terms of predictive accuracy and uncertainty analysis. The experimental set-up provides evidence to answer the research question; can we quantify the uncertainty in image-based crack detection for concrete structures using Bayesian convolutional neural networks? If we can quantify the uncertainty, our interest lies in the practical application of maintenance strategies. Can the image-based detection method using a BCNN contribute to maintenance strategies? If the answer to the question is yes, the BCNN should provide significant advantages over the traditional CNN model in terms of accuracy and uncertainty evaluation. The following is true if the BCNN effectively addresses the uncertainty while the CNN method fails with a high probability for a given image. If we can find at least one example, the BCNN has the potential to reduce the number of false negatives and, therefore, possible failures. We follow the architecture of Gal & Ghahramani [2016a] to implement the BCNN, a LeNet architecture with a dropout layer after every convolutional and fully connected layer during the test and training phase. We compare the model with a LeNet model with a regularization dropout layer after the fully connected layer during the training phase. Also, we compare these models with a LeNet model without dropout layers. Apart from the presence and kind of dropout, the models are identical and trained and tested on the same data. The models are respectively: BCNN, CNN_D, and CNN. Figures 9 and 10 show the networks' architectures. The models' output layer shows the difference between the networks. The output is a predictive probability distribution rather than a fixed predicted value. The model returns a predictive distribution for each class, where the predictive variance is the epistemic uncertainty for a given input image. We use the simulated values from the predictive posterior distribution to compute the predictive mean and variance. A low variance ensures a confident prediction; the probabilities for a given class are close. In contrast, a large variance quantifies uncertainty; the model is unsure about its predicted class and returns a different probability for each simulated value.

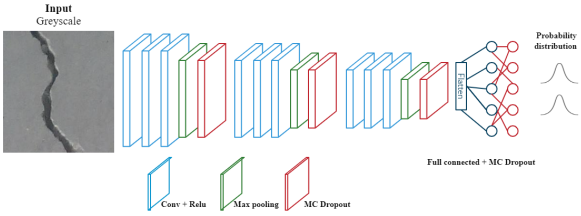
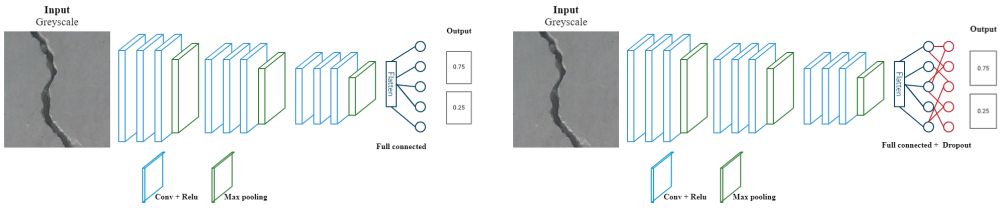


Figure 9: The model architecture for the Bayesian convolutional neural network with Monte Carlo dropout. The model consists of three convolution operations followed by a non-linear Relu activation function and max pooling operation. Each convolution operation is followed by a dropout layer during the test and training phase which is called Monte Carlo dropout (MC dropout). The output is a probability distribution for each class. The BCNN code on Github shows the specific model architecture, including the number of parameters and feature maps.



(a) CNN **(b)** CNN with regularization dropout layer

Figure 10: The model architecture of the CNN and the CNN with a regularization dropout layer. The architecture of the CNN in 10a is a generic CNN with softmax activation. Figure 10b shows the architecture of a CNN with regularization dropout after the fully connected layer. The networks' output are softmax class probabilities. The CNN and CNN_D codes on Github show the specific model architectures, including the number of parameters and feature maps.

Random noise Additionally, we analyze the behavior of the models if the input image is random. In practice, random noise or invalid images are inevitable in extensive data sets. Specifically, in maintenance applications, images are generally collected through video recordings or photographs sourced by drones, machinery, or humans—procedures that are prone to a random collection of false images like surroundings. We want to analyze the BCNN behavior on random image inputs. Can the method quantify the uncertainty if the test image is completely random relative to the training samples? After training the networks, we evaluate the behavior on the random noise image in Figure 11.

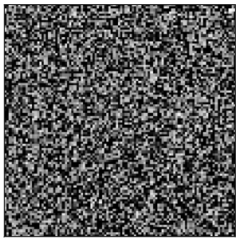


Figure 11: A random noise image to evaluate the model's uncertainty. Is the model able to measure the uncertainty if the input is entirely random and deviant from the training samples?

5.1 Data

The data contains 40,000 labeled images of cracked and non-cracked images from various concrete structures, e.g., buildings, roads. Özgenel & Sorguç [2018] obtained the data set by extracting 500 full images taken from walls and roads on the Middle East Technical University in Turkey. The images variate in concrete surface finishes and background (plastering, paint, shadows). Researchers previously applied the data to various problems and made the data publicly available at https://data.mendeley.com [Özgenel & Sorguç, 2018];[Zhang et al., 2016]. The data set is balanced with 20,000 positives (cracked) and 20,000 negatives (non-cracked). Generally, an image is a three-dimensional data structure, namely a (x, y, z) data matrix. The number of columns, x, and the number of rows, y, and the depth, z of the matrix respectively correspond to the pixel width, the pixel height, and pixel depth. As the matrix dimension increases, the resolution of an image usually gets better. The depth of the matrix z depends on the color maps of the image. The given images are in the RGB scheme and have a 227 x 227 x 3 dimension. The color scheme will not give any added information as the road and building are concrete materials; we transform and normalize the images. The transformed images are in the grey-scale (x,y,1) with each pixel value ranging from 0 to 1. To reduce the computation time, the dimension is further reduced to 100 x 100 x 1. Figure 12, shows a sample of ten re-scaled images with the label and corresponding one-hot encoding of the class labels.

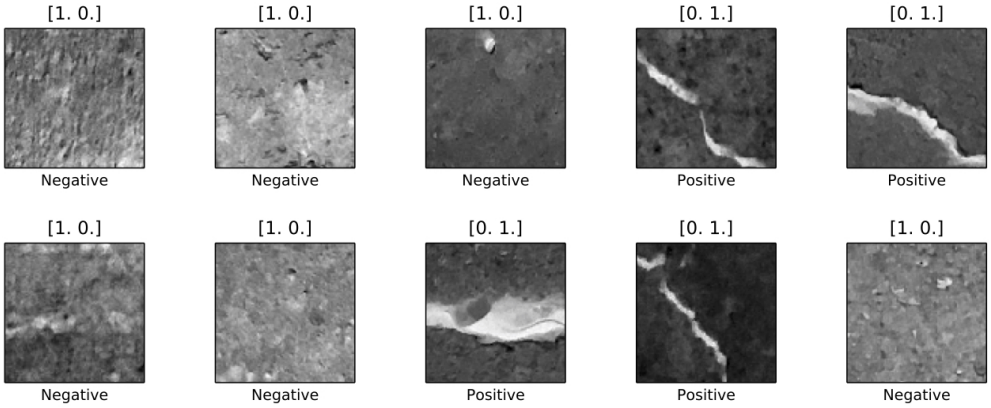


Figure 12: A random sample of 10 images from the data set. The transformed data set consists of 40,000 images with dimensions 100 x 100 x 1. The classes are one-hot encoded, [1.0.] and [0.1.] and respectively corresponds to the negative and positive class.

The dimension reduction is not likely to influence the prediction performance. The transformed images contain redundant information regarding the cracks. The performance on the original image will have minimal advantage (or even disadvantage) and will largely increase the running time. We split the complete data into a training, validation, and test set. We apply a 70-15-15% balanced split, respectively, training (28,000 images), validation (6,000 images), and test data (6,000 images). Where each set contains a set of images and a set of labels. We optimize the hyper-parameters on the training data and validate them with the validation set. Afterwards, we use the unseen test images as input to test the model. The test set is only used once to avoid bias and is used to evaluate the final model. The models use the Keras neural network library and are trained using CPU only. The codes and instructions are available on Github.

6 Results

Comparing the three convolutional neural networks, we criticize on three components; predictive accuracy, uncertainty assessment, and practical applicability. In terms of predictive accuracy, the newly proposed method should perform equally in accuracy to be practically sufficient. Table 1 shows that the three networks predict the presence or absence of crack accurate. The networks achieve training and validation accuracy of over 99%. Furthermore, Figure 13 shows a decreasing training and validation loss and with no signs of overfitting. The CNN_D preforms slightly better than the CNN in terms of validation loss due to the additional regularization dropout layer. Altogether, the BCNN slightly surpasses in terms of accuracy and loss. The networks' high accuracy

might be the result of the extensive and clean images, providing the network the ability to learn from many training images and generalize the knowledge to the validation set. For the uncertainty evaluation, we solely continue with the BCNN and the generic CNN. The difference in predictive performance between the CNN and CNN_D is small.

Table 1: The accuracy and loss for respectively, the convolutional neural network (CNN), convolutional neural network with dropout regularization (CNN_D) and convolutional neural network with Monte Carlo dropout (BCNN). We evaluate each model by their accuracy and loss after 10 epochs for the training data and validation test. Afterwards, we evaluate thee trained models on the test data set. The table shows the slight out-performance of the BCNN.

	Accuracy			**Loss**		
Model	Train	Val	Test	Train	Val	Test
CNN	0.993	0.992	0.993	0.021	0.032	0.026
CNN_D	0.993	0.992	0.993	0.021	0.025	0.025
BCNN	0.995	0.994	0.994	0.017	0.171	0.018

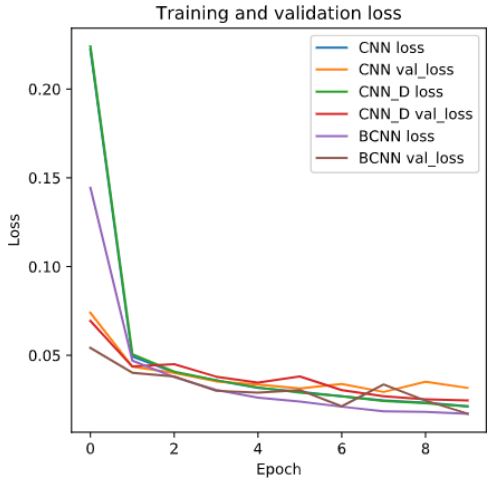
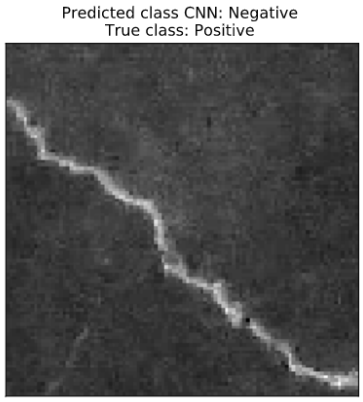


Figure 13: The image shows the training and validation loss for each model after each epoch and shows the convergence after ten iterations.

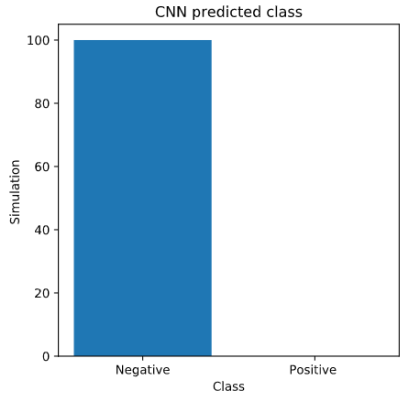
After training the networks, we predict the classes of $6,000$ previously unseen images. The BCNN outperforms the other networks in terms of accuracy and loss on the test data set. The softmax activation function output is a frequentist approach, converting counts into probabilities; the probabilities are not a measure of model confidence. Where Gal & Ghahramani [2015] proposed

the BCNN with Monte Carlo dropout and used the approximated predictive posterior distribution variance as uncertainty measurement. We draw 100 samples from the approximated predictive posterior distribution of the BCNN (100 forward passes) and compute the predictive mean and variance for a particular class. Drawing samples from the network's predictive posterior distribution is equivalent to repeatedly predicting probabilities for the same input image with the BCNN. Each run of the network predicts a different probability for the same input image, representing the uncertainty. In contrast, we observe equal predicted probabilities for the CNN if we repeatedly run the model on the same input image, irrespective of the simulation. For example, a CNN predicts the exact same probability for an input image if we repeat the process.

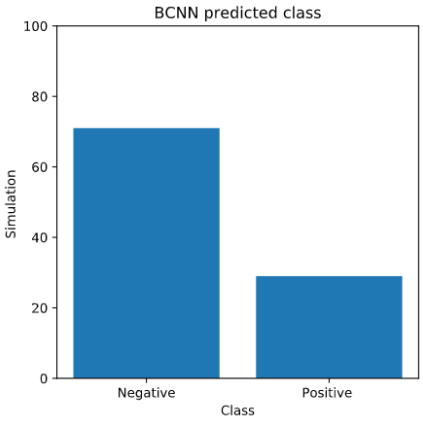
In Figure 14, we show an example where the softmax probability misrepresents the model's confidence. The CNN misclassifies the image with a large probability, p= 0.76. The CNN falsely classifies the image to a particular class with certainty; the model is confident that there is no crack in the image. Whereas the newly proposed Bayesian convolutional neural network can measure the uncertainty as the variance of the predictive posterior distribution is large (a large differing range of predicted values for 100 stochastic forward passes). In Figure 14a the input image is shown. The CNN model predicts a negative class with confidence of p = 0.76 and a positive class with p = 0.24. The graph in 14b shows the same predicted class for 20 simulations run for the CNN. The second row of images shows the BCNN classification results. The BCNN assigns a simulation to a particular class if its predicted probability is greater than p = 0.50. From the graph in 14c, we learn that in some cases, the network predicts a negative or positive class if we show the network the same image 100 times. In Figure 14d, the corresponding range of predictive probabilities are present. The Bayesian network predicts a negative class for most of the simulation, but the model can correctly evaluate the image for some runs. The variance of the range predictive probabilities is equal to 0.056. We use the variance from the samples of the approximated predictive posterior distribution as epistemic uncertainty. The above case justifies the potential of the novel method.



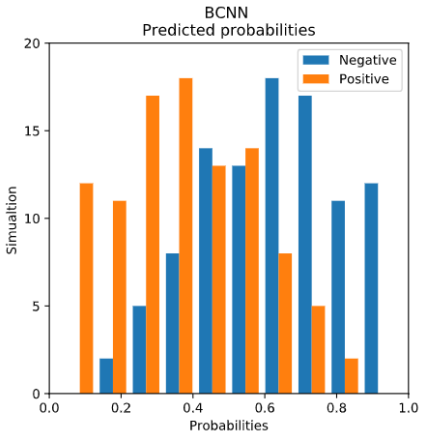
(a) False negative prediction



(b) Predicted probabilities for 100 simulations



(c) Predicted class for 100 simulations



(d) Predicted probabilities for 100 simulations

Figure 14: The above figure shows an example where the CNN method is unable to correctly predict the uncertainty. The image in 14a shows image 2312 from the test data set with predicted class from the CNN model and the true class. The top right plot 14b shows the predicted probability for each class if we run the model for 20 times with the same test data. The bottom left image 14c shows the predicted class from the BCNN model with classification probability $p > 0.50$. The bottom right plot 14d shows the predicted probabilities from 100 samples from the posterior distribution of each class.

The last evaluation criterion is the practical applicability. Are the improvements of the BCNN large enough compared to the current image-based-detection methods for the maintenance applications? As explained in Section 4, the image-based-detection methods are used to reduce the number of manual inspections. In the specific project of CQM, the images were classified with a convolutional neural network. The company checks a particular location (based on the image) if the softmax probability of having a defect (positive) exceeds $p \geq 0.30$. All the images with a softmax probability, for the negative class, $p > 0.70$ are skipped and are not checked by inspectors. Based on the uncertainty evaluation results, we see that even if the softmax predicted a negative class probability that exceed 0.70, false-negative predictions are still present. We compare the number of false negatives from the CNN model for different values of classification probabilities. Table 2 shows

the confusion matrix for the CNN model with classification probability p $>$ 0.70 and BCNN for p $>$ 0.70 and variance $\sigma^2 \leq 0.001$. Reducing the number of false-negative images is most valuable in maintenance strategies. If an image is classified as negative while being positive, consequences might be fatal. In 12 cases, the CNN method predicts a false negative with classification probability p $>$ 0.70. The BCNN eliminates the number of false negatives, including the estimated source of uncertainty, the variance of 100 samples from the predictive posterior distribution. However, in the latter case, the number of false positives increases from 27 to 179, the network classifies images as positive while being negative for 179 images.

For practical applications, the trade-off between eliminating the number of false negatives but increasing the number of false positives should be made. Depending on the application, the decision can differ. Maintenance applications where the value of the risk of having a false negative is large one is more likely to opt for the BCNN method. If the costs of having a false negative are higher than the increase in false positives, the BCNN method is indispensable. If we zoom in to the given example for the practical maintenance applications, the following question should be asked; What are the costs of missing 12 false negatives? With the current method, 12 images are not inspected, while the BCNN could indicate the uncertainty and raise the alarm of a possible defect.

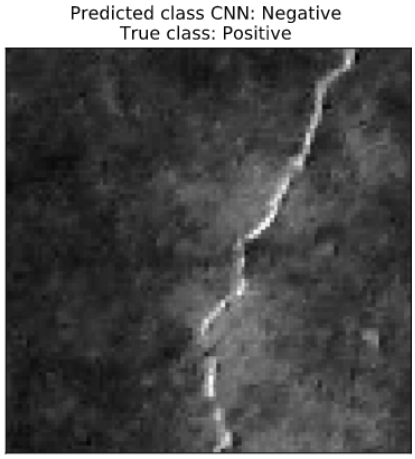
Augmented, we compare the confusion matrices, with classification probability equal to 0.50 in Table 2. As expected, the results of the BCNN remain the same as a result of the uncertainty inclusion. The BCNN classifies an image if the mean of the simulated class probabilities p $>$ 0.50 and the variance $\sigma^2 \leq 0.001$. However, the results for the CNN change; the number of false-negatives increases while the false-positives decrease with a classification probability of p $>$ 0.50. In the latter case, the advantages of the BCNN increase. The bottom part of Table 2 compares the classification results for the CNN and CNN_D. The table shows the effectiveness of the dropout regularization in the CNN_D. The regularization reduces the number of false-negatives for both classification probabilities.

Table 2: The confusion matrices for the BCNN, CNN, and CNN_D show the number of true negatives, true positives, false negatives, and false positives. Maintenance strategies would like to eliminate the number of false negatives. The BCNN using the variance as an uncertainty measure can eliminate the number of false-negative for $\sigma^2 \leq 0.001$. The BCNN classifies images as negative/positive if the sample means exceed the class probability and satisfies the low variance of $\sigma^2 \leq 0.001$. The CNN and CNN_D classify an image using the softmax classification probability. For example, if the negative/positive class' softmax probability exceeds the classification probability, the image is classified as negative/positive and vice versa. For each network, the table shows the classification probability in the parentheses. The model with regularization dropout offers advantages in terms of false-negative at the costs of increasing false-positives.

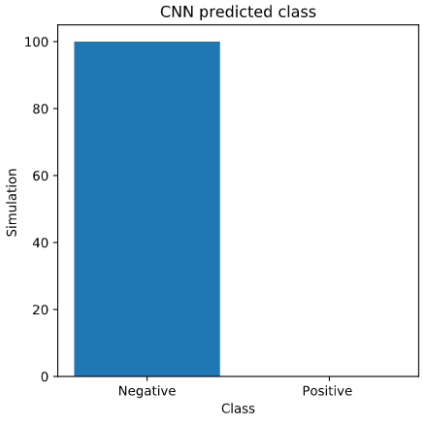
BCNN		Predicted (p > 0.50)			Predicted (p > 0.70)		
		Negative	Positive	All	Negative	Positive	All
True	Negative	2821	**179**	3000	2821	**179**	3000
	Positive	**0**	3000	3000	**0**	3000	3000
	All	2654	3346	6000	2654	3346	6000
CNN		Predicted (p > 0.50)			Predicted (p > 0.70)		
		Negative	Positive	All	Negative	Positive	All
True	Negative	2980	**20**	3000	2973	**27**	3000
	Positive	**21**	2979	3000	**12**	2988	3000
	All	3001	2999	6000	2985	3015	6000
CNN_D		Predicted (p > 0.50)			Predicted (p > 0.70)		
		Negative	Positive	All	Negative	Positive	All
True	Negative	2976	**24**	3000	2960	**40**	3000
	Positive	**15**	2985	3000	**11**	2989	3000
	All	2991	3009	6000	2971	3029	6000

One could argue that using a higher classification threshold with CNN can potentially obtain the same results. Figure 15 shows the Bayesian methods' potential even for higher class probabilities from the CNN. The CNN predicts a constant negative class probability of 0.93 for the specific

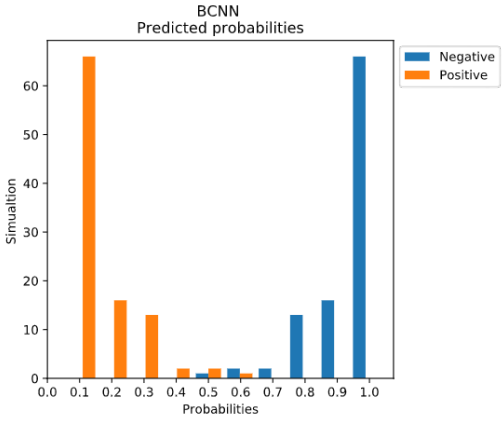
image given in 15a. If we would use the softmax probability as confidence, we could say that the network is confident that the image does not contain a crack. However, the image contains a crack, and we should not use the obtained softmax probability as model confidence. Figure 15c shows the predicted probabilities for 100 samples of the model's predictive posterior distribution. For 99 draws the probability for the negative class is larger than 0.50, as shown in both the graphs in 15c and 15d. At the same time, if we look at Figure 15c, the range of the predicted probabilities is large and not centered around a common value. The large variance of the approximated predictive posterior distribution represents the uncertainty in the prediction. Regardless of a large number of negative class prediction, the variance represents the model's confidence for the input image.



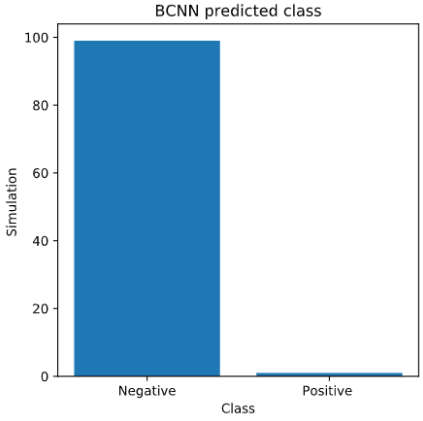
(a) False negative prediction for test image 144



(b) CNN's predicted classes for 100 simulations



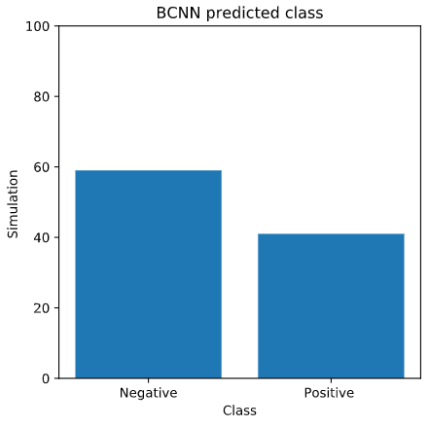
(c) Predicted probabilities for 100 simulations with the BCNN



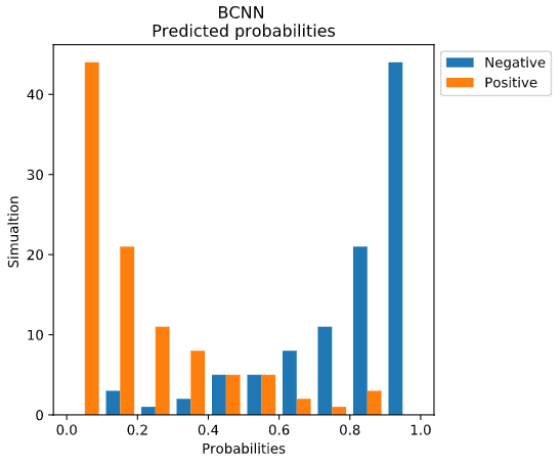
(d) BCNN's predicted classes for 100 simulations

Figure 15: The uncertainty evaluation for the test image 144 with the BCNN and CNN. The CNN and BCNN results are shown respectively in the first and second row of the image. The image shows the false negative prediction of the CNN model with high negative class probability of 0.93. The BCNN captures the uncertainty with the predictive posterior variance of 100 samples.

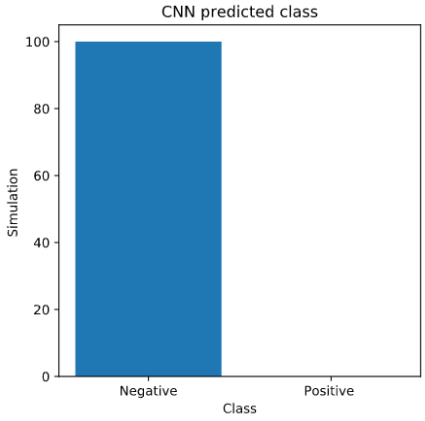
Random noise After training and testing the models, we evaluate the BCNN and CNN on the random noise image. We want to find out if the BCNN can correctly measure the uncertainty in the random deviant image. The graphs in Figure 16 show the uncertainty evaluation for both the BCNN and CNN model. From the Figures 16a and 16b, we can conclude that the BCNN is able to measure the uncertainty for the given random noise input image. The model predicts alternately negative and positive classes, with large varying predictive probabilities. The variance of the predicted values, is large, $\sigma^2 = 0.099$, and represents the predicted value's epistemic uncertainty. The model can not decide to which class the image belongs and is uncertain about its value. As expected, the CNN predicts the same value for each simulation run in Figures 16c and 16d. The model predicts a negative class for the random noise image with a high probability, p = 0.99. The BCNN model can measure uncertainty for an unseen adversarial image, while the CNN predicts with a high probability that the random noise image does not contain a crack. For practical applications, it gives the potential to filter out all the random collected images. Identifying and removing random images leads to cleaner data sets and could improve the model's predictive performance. However, further research on deviant and outlier images is needed to support these findings.



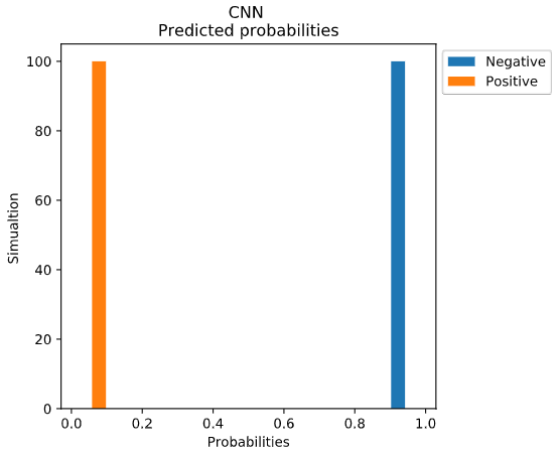
(a) BCNN's predicted class for 100 simulations



(b) BCNN's predicted probability for 100 simulations



(c) CNN Predicted class for 100 simulations



(d) CNN's predicted probabilities for 100 simulations

Figure 16: The uncertainty evaluation for a random noise input image for 100 simulation runs with the BCNN and CNN. The BCNN and CNN results are shown respectively in the first and second rows of the figure. The BCNN measures the uncertainty for a random noise image with a variance, $\sigma^2 = 0.099$, while the CNN model predicts a negative class with a probability p= 0.99.

7 Discussion and further research

We applied a novel Bayesian deep learning framework to study the uncertainty in deep learning models. Specifically, we used a method to include model uncertainty in image-based predictions for a maintenance application. First, we evaluated the predictive accuracy between the traditional convolutional neural network and the newly proposed Bayesian neural network. Second, we wanted to quantify the model uncertainty with the Monte-Carlo dropout inference in the BCNN. We compared the uncertainty evaluation between the models. The Bayesian method can measure uncertainty through its predictive posterior variance and eliminate the number of false negatives

relative to the CNN method. Furthermore, the examples show that softmax probabilities do not provide the associated model uncertainty. Relating to the practical application of the novel method, one should make a trade-off between the costs of detecting false negatives and the cost of false positives. Using the predictive variance as a model uncertainty measure significantly increases the number of false positives. The method's gains might be too low, depending on the application—the additional implementation for inference and the increasing running time might. Also, we measured the uncertainty evaluation of a random noise image. We found that the BCNN can detect a fraudulent image by measuring the uncertainty. The method shows considerable uncertainty if the network cannot classify the image and might be an incentive to use in practice to detect and remove fraudulent images.

All in all, the Monte Carlo dropout inference gives the possibility to improve and increase the image-detection methods for anomaly detection by including model confidence. The results are not limited to the presented maintenance application and are valuable in numerous sectors. For example in the medical field, to detect anomalies from images or scans to diagnose the disease with a measure of uncertainty. In general, the applied method can give substantial advantages in any image-based decision-making processes by including the uncertainty evaluation. Nonetheless, there are several deficiencies regarding the method. First of all, it might be bothersome for practical applications as the run time increases with simulated values. Second, the method is not useful if the costs of false-positive exceed the costs of false negatives. Furthermore, this research has the limitation of locally trained models. One could potentially evaluate the effect of using a pre-trained model (using transfer learning) and compare the uncertainty evaluations in further research. The Monte Carlo dropout inference is a novel method that includes model uncertainty in the outcomes. Therefore researchers are questioning its theoretical soundness and interpretation of empirical results. One example is a paper from a workshop during the NIPS (Conference on Neural Information Processing Systems) from Osband [2016]. Osband [2016] argues that the method of Gal [2016] is an estimate of risk rather than uncertainty and criticizes the method on its theoretical justification of the approximated posterior distribution. Being aware of the discussion, the method is still the most scalable and most accessible to apply in practice to add a source of model uncertainty. It might be an interesting further research to make a theoretical and practical comparison with the competing methods such as the Stein variational gradient descent (SVGD) [Liu & Wang, 2016] and probabilistic backpropagation (Bayes by Backprop) [Blundell et al., 2015]. The field of uncertainty in deep

learning applications is significantly advancing and is a topic with considerable potential for further research. More specifically, it might be interesting to include visualizing results for maintenance applications in the future. A technique that indicates the origin of the uncertainty directly on the image distinguishes the aleatoric and epistemic uncertainty. Furthermore, an exciting field of research for anomaly detection with images is the combination of uncertainty evaluation and Humans in the loop. Humans in the loop is a combination of the machine learning model and human interaction with continuous feedback loops, also known as active learning [Liu & Wang, 2016]. The combinations of active learning and the BCNN could potentially be the step towards automated image-based maintenance strategies.

We verify and substantiate the obtained results with CQM, a quantitative consultancy company that previously applied a CNN for a maintenance project [CQM, 2017]. We gave them a short introduction to the problem, the used methodology, and the results. Afterward, we asked the following questions:

1. After seeing these results, does it give you an incentive to think about model uncertainty in machine learning (deep learning) applications?

2. Are you willing to adjust a network in the future (through dropout layers) to eliminate the number of false negatives at the expense of increasing false-positives and computation time?

3. Are you willing to adjust a network in the future (through dropout layers) to potentially assign the number of anomalous images in a large data set as uncertain? The method ensures the detection of abnormal images automatically.

4. Do you think it is necessary to include model certainty in the road to automated maintenance?

Summarizing the obtained answers from the domain expert, we can conclude the following: 1. the company is highly interested in including uncertainty as model confidence is a discussion point with their clients, 2. the trade-off to apply the method in future deep learning methods depends on the applications; for some applications, the costs of false positives might exceed the costs of false negatives and vice versa. Furthermore, the automatic detection of fraudulent images is beneficial in practical applications, especially if the image is entirely different from training data, such as a dog's image. Finally, the company thinks it not essential to include uncertainty in the route of automated maintenance. According to CQM, it heavily depends on the source of application and

the corresponding dangers. Using a low threshold for the softmax probabilities already leads to automation of the majority of work, quickly and easily. The company suggested further research to look into a combination of transfer learning and uncertainty measurement to overcome the data availability burden.

A Derivations

Kullback-Leibler divergence and Evidence Lower Bound The derivation show the equality between the Kullback-Leibler divergence and the ELBO:

$$\begin{aligned}
\mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p\left(\boldsymbol{\beta}|\mathcal{X},\mathcal{Y}\right)\right] &:= \int q_{\boldsymbol{\theta}}(\boldsymbol{\beta})\,\log\,\frac{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}{P(\boldsymbol{\beta}|\mathcal{X},\mathcal{Y})}, \\
&= \mathrm{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}\log\,\frac{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}{p(\boldsymbol{\beta}|\mathcal{X},\mathcal{Y})}, \\
&= \mathrm{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}\log\,\frac{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}{p(\mathcal{Y}|\boldsymbol{\beta},\mathcal{X})P(\boldsymbol{\beta})}p(\mathcal{Y}|\mathcal{X}), \\
&= \mathrm{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}\left[\log\,q_{\boldsymbol{\theta}}(\boldsymbol{\beta}) - \log\,p(\mathcal{Y}|\boldsymbol{\beta},\mathcal{X}) - \log\,p(\boldsymbol{\beta}) + \log\,p(\mathcal{Y}|\mathcal{X}\right], \\
&= \mathrm{KL}\left[Q(\boldsymbol{\beta}|\mathcal{D},\boldsymbol{\theta})||P\left(\boldsymbol{\beta}\right)\right] - \mathrm{E}_{Q(\boldsymbol{\beta};\boldsymbol{\theta})}\left[\log\,P(X|\boldsymbol{\beta})\right] + \log\,P(X).
\end{aligned} \tag{23}$$

In the first line we use the formula of the KL divergence [Kullback & Leibler, 1951]. Afterwards, we apply the Bayes rule and the logarithmic rules to obtain the formula in the last line in (23). The distribution $p(\mathcal{Y}|\mathcal{X})$ does not depend on the approximating distribution $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ and we can write the formula as,

$$\log\,p(\mathcal{Y}|\mathcal{X}) - \mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p\left(\boldsymbol{\beta}|\mathcal{X},\mathcal{Y}\right)\right] = \mathrm{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{\beta})}\left[\log\,p(\mathcal{Y}|\boldsymbol{\beta},\mathcal{X}) + \log\,p(\boldsymbol{\beta}) - \log\,q_{\boldsymbol{\theta}}(\boldsymbol{\beta})\right], \tag{24}$$

where we call (24) the Evidence Lower Bound (ELBO). Since $\mathrm{KL}\left[q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p\left(\boldsymbol{\beta}|\mathcal{X},\mathcal{Y}\right)\right] \geq 0$, we see that the ELBO indeed is a lower bound on the model log-evidence, $\log\,p(\mathcal{Y}|\mathcal{X})$. Furthemore, as the log-evidence does not depend on the on the approximating distribution, $q_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ it shows that maximizing the ELBO is equivalent to minimizing the KL-divergence. The ELBO does not depend on the intractable posterior distribution and can be calculated analytically. We rewrite this as the following equation,

$$\mathcal{L}_{VI}(\boldsymbol{\theta}) := \int q_{\boldsymbol{\theta}}(\boldsymbol{\beta})p(\mathcal{Y}|\mathcal{X},\boldsymbol{\beta})d\boldsymbol{\beta} - \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{\beta})||p(\boldsymbol{\beta})), \tag{25}$$

to obtain the objective of the optimization problem.

References

Ahuja, I. P. S., & Khamba, J. S. (2008). Total productive maintenance: literature review and directions. *International Journal of Quality & Reliability Management*.

Barber, D., & Bishop, C. M. (1998). Ensemble learning in bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, *168*, 215–238.

Beal, M. J. (2003). *Variational algorithms for approximate bayesian inference* (Phd thesis). UCL (University College London).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*. Retrieved December 2020, from https://arxiv.org/abs/1505.05424

Chen, F., & Jahanshahi, M. R. (2018). Deep learning-based crack detection using convolutional neural network and naïve bayes data fusion. *IEEE Transactions on Industrial Electronics*, *65*(5), 4392-4400.

CQM. (2017). *Slimme beeldherkenning maakt spoorinspectie vijf keer efficiënter.* Retrieved December 2020, from https://cqm.nl/nl/cases/slimme-beeldherkenning-maakt-spoorinspectie-vijf-keer-efficienter

Dekker, R. (1996). Applications of maintenance optimization models: a review and analysis. *Reliability Engineering & System Safety*, *51*(3), 229–240.

Denker, J., & LeCun, Y. (1991). Transforming neural-net output levels to probability distributions. *Advances in Neural Information Processing Systems*, *3*, 853–859.

Feng, D., Rosenbaum, L., & Dietmayer, K. (2018). Towards safe autonomous driving: Capture uncertainty in the deep neural network for lidar 3d vehicle detection. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 3266-3273.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1) (No. 10). Springer.

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets* (pp. 267–285). Springer.

Gal, Y. (2016). *Uncertainty in deep learning* (Unpublished doctoral dissertation). University of Cambridge.

Gal, Y., & Ghahramani, Z. (2015). Dropout as a bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML* (Vol. 1, p. 2).

Gal, Y., & Ghahramani, Z. (2016a). Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*. Retrieved December 2020, from https://arxiv.org/abs/1506.02158

Gal, Y., & Ghahramani, Z. (2016b). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan & K. Q. Weinberger (Eds.), (Vol. 48, pp. 1050–1059). PMLR. Retrieved from http://proceedings.mlr.press/v48/gal16.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* (Vol. 1) (No. 2). MIT press Cambridge.

Gouriveau, R., Medjaher, K., & Zerhouni, N. (2016). *From prognostics and health systems management to predictive maintenance 1: Monitoring and prognostics*. John Wiley & Sons.

Graves, A. (2011). Practical variational inference for neural networks. *Advances in Neural Information Processing Systems*, *24*, 2348–2356.

Gull, S. F. (1989). Developments in maximum entropy data analysis. In (pp. 53–71). Springer.

Guynn, J. (2015). *Google photos labeled black people 'gorillas'.* Retrieved December 2020, from https://eu.usatoday.com/story/tech/2015/07/01/google-apologizes-after-photos-identify-black-people-as-gorillas/29567465/

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.

Hinton, G. E., & Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In (pp. 5–13). Association for Computing Machinery.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*(1).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Jamshidi, A., Faghih-Roohi, S., Hajizadeh, S., Núñez, A., Babuska, R., Dollevoet, R., ... De Schutter, B. (2017). A big data analysis approach for rail failure risk assessment. *Risk Analysis*, *37*(8), 1495–1507.

Jardine, A. K., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, *20*(7), 1483–1510.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, *37*(2), 183–233.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

LeCun, Y. (1998). *The mnist database of handwritten digits.* Retrieved December 2020, from http://yann.lecun.com/exdb/mnist/

Li, Z., Zhang, X., Müller, H., & Zhang, S. (2018). Large-scale retrieval for medical image analytics: A comprehensive review. *Medical Image Analysis*, *43*, 66–84.

Liu, Q., & Wang, D. (2016). Stein variational gradient descent: A general purpose Bayesian inference algorithm. *Advances in Neural Information Processing Systems*, *29*, 2378–2386.

MacKay, D. J. (1992). A practical Bayesian framework for backpropagation networks. *Neural Computation*, *4*(3), 448–472.

Neal, R. M. (1992). Bayesian mixture modeling. In *Maximum Entropy and Bayesian Methods* (pp. 197–211). Springer.

Neal, R. M. (2012). *Bayesian Learning for Neural Networks* (Vol. 118). Springer Science & Business Media.

Osband, I. (2016). Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on Bayesian deep learning* (Vol. 192).

Özgenel, Ç. F., & Sorguç, A. G. (2018). Performance comparison of pretrained convolutional neural networks on crack detection in buildings. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction* (Vol. 35, pp. 1–8).

Ran, Y., Zhou, X., Lin, P., Wen, Y., & Deng, R. (2019). A survey of predictive maintenance: Systems, purposes and approaches. *arXiv preprint arXiv:1912.07383*. Retrieved December 2020, from https://arxiv.org/pdf/1912.07383.pdf

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sensoy, M., Kaplan, L., & Kandemir, M. (2018). Evidential deep learning to quantify classification uncertainty. In *Advances Neural information Processing Systems* (pp. 3179–3189).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Tishby, N., Levin, E., & Solla, S. A. (1989). Consistent inference of probabilities in layered networks: Predictions and generalization. In (Vol. 2, pp. 403–409).

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* (pp. 818–833).

Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road crack detection using deep convolutional neural network. *IEEE International Conference on Image Processing (ICIP)*, 3708–3712.