Using Probabilistic Machine Learning for Image Recognition: An Empirical Approach

Luca-Alexandru Zamfira

Student ID: 3131255

Supervisor: Sandra Fortini

Institution: Bocconi University

September 9, 2024

Abstract

Over recent years, Convolutional Neural Networks (CNNs) have dominated the area of recognizing images, achieving top results across many tests. However, the deterministic nature of CNNs limits their ability to depict doubt, which is crucial for numerous real situations. This work sets out to examine the aptitude of probabilistic models, especially Bayesian Neural Networks and Variational Autoencoders, for resolving duties involving recognizing visual content within images. Through complete experimentation, we demonstrate that BNNs offer a more stable and understandable structure by quantifying the uncertainty linked to predictions. Similarly, VAEs, with their power to generate, furnish a richer portrayal of image information, allowing enhanced extraction of features and recognition. By comparing these probabilistic models with traditional CNNs, we underline the advantages of including uncertainty and probabilistic reasoning in recognizing images. Our findings propose that BNNs and VAEs hold significant promise for advancing the field, offering both improved performance and deeper insights into the fundamental distributions of data.

Contents

1 Introduction

The Evolution of Image Recognition

From the beginning of computer vision, where basic algorithms were applied to discern visual inputs, stems the origins of image recognition's progression. As computational power grew and datasets became more abundant, the field witnessed a paradigm shift. Convolutional Neural Networks, upon their introduction, signified an exceptionally meaningful watershed, advancing the field substantially. These architectures, inspired by human visual systems, demonstrated unparalleled prowess in image classification tasks, setting new benchmarks and redefining the state-of-the-art.

The ever-changing nature of the machine learning and artificial intelligence fields means their landscape remains in flux. As data volumes and situations in reality become more intricate, there has been greater requirement for models that can deliver beyond solely single values in a point. In situations where decisions have significant consequences, understanding the confidence or uncertainty of a prediction becomes crucial.

The Rise of Probabilistic Models

Probabilistic machine learning provides a unique approach to making predictions by incorporating a measure of uncertainty. By leveraging the foundations of Bayesian statistics, probabilistic models amalgamate prior assumptions alongside encountered proof, permitting more educated and multifaceted forecasts. This differs from the frequentist approach which solely relies on observed data.

While object identification proves pivotal within image recognition, gauging confidence in such determinations remains equally vital. In medical imaging, where an inaccurate diagnosis could carry grave consequences, achieving precision is paramount. By using a probabilistic model, areas of uncertainty can be identified, prompting additional investigation or consultation with experts.

Beyond CNNs: The Power of Bayesian Neural Networks and VAEs

Our research is focused on examining the abilities of two probabilistic techniques: Bayesian Neural Networks (BNNs), as well as Variational Autoencoders (VAEs), both of which demonstrate promising potential. Whereas BNNs broaden usual neural systems by presenting uncertainty over weights, VAEs are generative designs that learn to code and translate information in an uncertain way.

This research looks to place these designs opposite to usual networks that use patterns. The purpose goes further than just checking what is right but investigating deeper into the understandings, strength, and clarity these designs can provide using the MNIST information set and the related yet disrupted MNIST-C.

Setting the Stage

As we embark on this exploration, we'll delve into the intricacies of probabilistic modeling, demystifying its advantages and challenges. Through empirical evaluations and theoretical discussions, this research aims to shed light on the potential of probabilistic models to shape the future of image recognition.

2 MNIST

MNIST: More Than Just Handwritten Digits

*T*he MNIST dataset, while seemingly simple, holds a special place in the annals of machine learning. Introduced by LeCun et al. in the late 1990s, it was one of the first large-scale datasets available to the research community. Its introduction coincided with the early days of neural networks, providing a playground for researchers to test, validate, and improve their algorithms.

*C*omprising 70,000 grayscale images, each of size 28x28 pixels, MNIST captures the variability and nuances of human handwriting. The dataset's diversity, stemming from the contributions of over 500 writers, ensures a broad representation of handwritten digits.

*O*ver the years, MNIST has been the proving ground for countless algorithms. From support vector machines to deep learning architectures, MNIST has seen it all. Its accessibility and simplicity have made it a staple in machine learning courses, tutorials, and workshops.

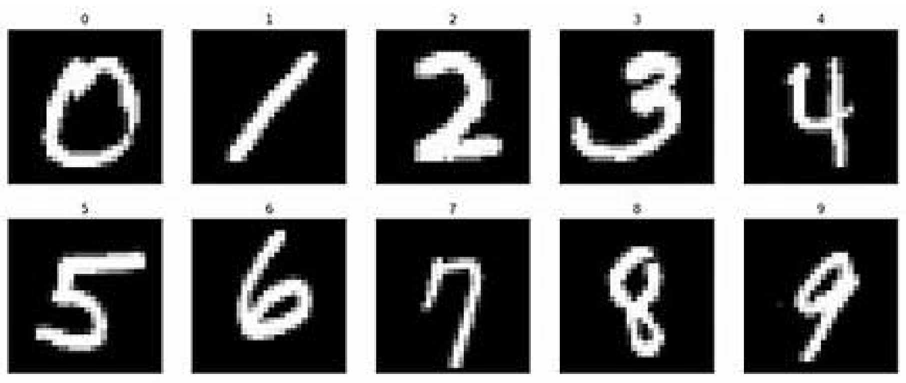

Figure 1: Sample from MNIST dataset

MNIST-C: Pushing the Boundaries of Robustness

*W*hile MNIST has been instrumental in advancing machine learning, it presents a sanitized view of the real world. Real-world data is messy, corrupted, and often incomplete. Recognizing this gap, the MNIST-C dataset was conceptualized to challenge models in less-than-ideal conditions.

*M*NIST-C introduces a gamut of perturbations to the original images. From Gaussian noise and motion blur to pixelation and brightness alterations, the dataset simulates various real-world corruptions. Each corruption type is available at five severity levels, resulting in a diverse and challenging dataset.

*I*n the era of deep learning, achieving high accuracy on MNIST is no longer a significant feat. However, MNIST-C raises the bar. It forces models to be resilient and adaptive, qualities essential for real-world applications.

Why MNIST and MNIST-C?

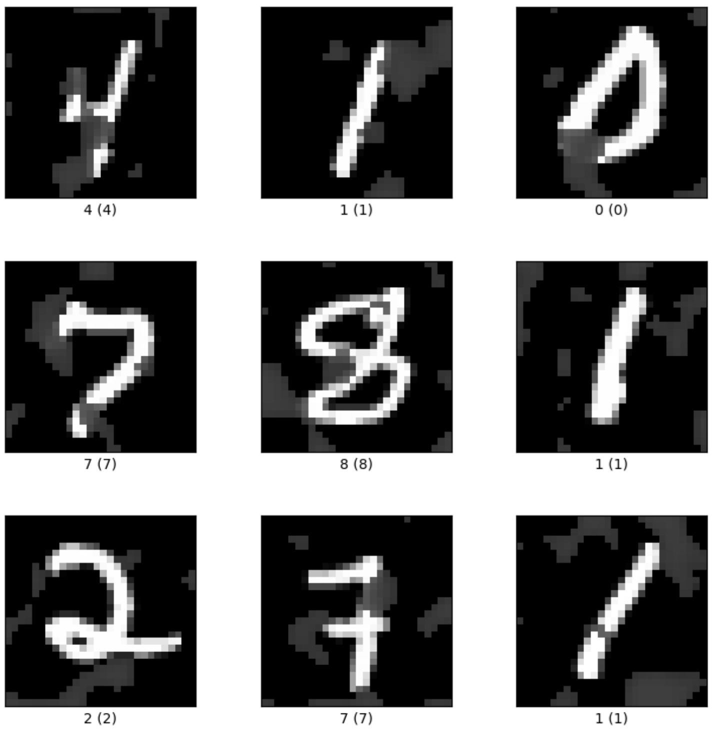

Figure 2: Sample from MNIST corrupted dataset

The juxtaposition of MNIST and MNIST-C in this research is strategic. While MNIST offers a controlled environment to evaluate baseline performance, MNIST-C introduces elements of chaos and unpredictability. It is in this chaos that the true potential of probabilistic models, with their inherent ability to quantify uncertainty, is expected to shine.

3 Preprocessing

Laying the Groundwork for Model Training

Before feeding the data into the neural networks, it undergoes a series of preprocessing steps to ensure optimal conditions for model training. This section elucidates each step, highlighting the theoretical underpinnings and practical implications.

Normalization of Pixel Values

*T*he pixel values in the MNIST and MNIST-C datasets originally range from 0 to 255, representing the grayscale intensity of each pixel. Normalizing these values to a range between 0 and 1 aids in stabilizing the training process and reducing the training time. This step is grounded in the optimization theories where normalized data facilitates smoother optimization landscapes, aiding gradient descent algorithms in finding minima more efficiently.

*T*he normalization is achieved by dividing each pixel value by 255. Additionally, a subtraction from 1 is performed to invert the colors, transforming the originally black digits on a white background to white digits on a black background. This inversion can potentially enhance the feature contrasts, aiding the model in learning more discriminative features.

Data Type Conversion

*T*he data type of the images is converted to float32 to ensure compatibility with TensorFlow operations, which often require float values. Moreover, using float32 instead of float64 can save memory and potentially speed up computations, making the training process more efficient.

*T*he conversion to float32 is a straightforward process, ensuring that the data is in a format conducive for high-performance computations, which is vital in deep learning frameworks.

One-Hot Encoding of Labels

*T*he labels, originally integers representing the digits in the images, are transformed using one-hot encoding. This encoding facilitates a clear distinction between the different classes during the training process, allowing the model to output a probability distribution over the classes.

*I*n one-hot encoding, each label is converted into a 10-dimensional binary vector. The index corresponding to the digit is set to 1, while all other indices are set to 0. This representation is compatible with loss functions such as categorical crossentropy, which are commonly used in multi-class classification tasks.

Reshaping the Images

*T*he images in the MNIST dataset are grayscale and originally have a shape of (28, 28). Reshaping them to (28, 28, 1) explicitly indicates the single color channel, ensuring compatibility with the Conv2D layers in the neural network model, which expect input data to have a channel dimension.

*T*he reshaping process is a preparatory step, aligning the data structure with the requirements of the convolutional layers, facilitating the extraction of features through convolutional operations.

4 Bayesian Neural Networks (BNNs)

Bridging Neural Networks and Bayesian Inference

At the heart of BNNs lies the fusion of deep learning and Bayesian statistics. While traditional neural networks provide point estimates for weights, BNNs introduce a distribution over them, capturing the inherent uncertainties.

Theoretical Underpinnings

*B*ayesian inference revolves around updating our beliefs (posterior distribution) in light of new data, based on prior beliefs (prior distribution) and the likelihood of observing the data given the parameters. Mathematically, Bayes' theorem is given by:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Where:

- $P(\theta|D)$ is the posterior distribution of the parameters θ given data D.
- $P(D|\theta)$ is the likelihood of observing data D given parameters θ.
- $P(\theta)$ is the prior distribution of the parameters.

- $P(D)$ is the evidence or marginal likelihood.

*I*n BNNs, instead of a single weight matrix, we maintain a distribution over the weights. This allows the network to express its uncertainty about the optimal weights, given the observed data.

Training BNNs

Training a BNN involves updating the posterior distribution over the weights given the data. This is computationally challenging due to the high dimensionality of the weight space and the intractability of the evidence term. Variational inference is commonly employed as an approximation technique.

*V*ariational inference transforms the intractable Bayesian computation into an optimization problem. It involves approximating the true posterior with a simpler, parametric distribution and minimizing the divergence between the two.

Advantages of BNNs

- *B*NNs inherently capture model uncertainty, providing not just predictions but also confidence intervals.
- *B*y accounting for uncertainty, BNNs can be more robust to overfitting, especially when data is scarce or noisy.
- *B*NNs can provide meaningful outputs even with limited data, leveraging prior distributions.

5 Variational Autoencoders (VAEs)

Generative Modeling Meets Variational Inference

VAEs are a class of generative models that leverage the principles of variational inference to learn complex data distributions.

Theoretical Foundations

*A*utoencoders are neural networks trained to reconstruct their input data. They consist of an encoder, which compresses the input into a latent representation, and a decoder, which reconstructs the input from this representation.

*V*AEs introduce a probabilistic twist to autoencoders. The encoder produces a distribution over the latent space, and the decoder samples from this distribution to generate data. The objective is to ensure that the latent distribution closely aligns with a prior (usually a standard normal distribution).

Training VAEs

The loss function for VAEs comprises two terms:

- *Reconstruction Loss:* Measures the fidelity of the reconstructed data.
- *KL Divergence:* Ensures the latent distribution aligns with the prior.

Advantages of VAEs

- *V*AEs can generate new data samples that resemble the training data.
- *T*he latent space of VAEs often captures meaningful data semantics, enabling various applications like interpolation and anomaly detection.
- *S*imilar to BNNs, VAEs capture uncertainties, providing a richer understanding of the data.

6 Convolutional Neural Networks (CNNs)

Pioneers of Image Recognition

CNNs have been the cornerstone of image recognition tasks for the past decade. Their architecture, specifically designed to process grid-like data (e.g., images), has led to breakthrough performances in various computer vision tasks.

Theoretical Foundations

Convolutional Layers: The core component of CNNs. Instead of fully connected layers, CNNs use convolutional layers that apply convolutional filters to the input data. These filters help in detecting local patterns like edges, textures, and shapes.

$$I * F = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} I(u, v)F(x - u, y - v)$$

Where I is the input image and F is the filter.

Pooling Layers: These layers reduce the spatial dimensions of the input, retaining only the most essential information. Max-pooling, which takes the maximum value in a window, is the most commonly used technique.

Fully Connected Layers: After several convolutional and pooling layers, the data is flattened and passed through one or more fully connected layers for classification.

Training CNNs

CNNs are trained using backpropagation, similar to traditional neural networks. The objective is to adjust the filter values and network weights to minimize the difference between the predicted and actual labels.

Advantages of CNNs

- *C*NNs can detect patterns irrespective of their position in the image.
- *L*ower layers capture basic patterns (e.g., edges), while deeper layers capture complex structures (e.g., faces).

Limitations

While CNNs have been immensely successful, they have limitations:

- *C*NNs provide point estimates without capturing the model's uncertainty.
- *W*ithout adequate regularization or data augmentation, CNNs can overfit to training data.
- *C*NN decisions can sometimes be hard to interpret, leading to trust issues in critical applications.

7 Implementation

In this section, we detail the implementation process for each of the models: Bayesian Neural Networks (BNNs), Variational Autoencoders (VAEs), and Convolutional Neural Networks (CNNs). We elucidate the choices made during the implementation and how they cater to the objectives of this research.

7.1 Convolutional Neural Network (CNN) Implementation

The architecture provided is a standard CNN tailored for image classification tasks, particularly for the MNIST dataset.

Model Architecture

The CNN model is constructed using a series of convolutional layers followed by max-pooling layers, and finally, fully connected layers.

```
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

The model starts with two sets of ‘Conv2D‘ layers. The first layer has 32 filters, and the second has 64 filters, both with a kernel size of 3×3. These layers are responsible for learning local patterns in the input image, such as edges, textures, and shapes.

‘MaxPooling2D‘ layer is used after each convolutional layer. These layers reduce the spatial dimensions of the output, making the model more computationally efficient and allowing it to learn more abstract features.

The ‘Flatten‘ layer reshapes the 3D output of the preceding layers into a 1D vector, preparing it for the fully connected layers.

The flattened output is passed through two ‘Dense‘ layers. The first has 128 neurons and uses the ReLU activation function. This layer learns global patterns in the image. The second dense layer has 10 neurons, corresponding to the 10 classes of the MNIST dataset, and uses the softmax activation function to produce class probabilities.

7.2 Bayesian Neural Network Architecture

The Bayesian Neural Network (BNN) used in this study is designed to capture the inherent uncertainties in the weights. This is achieved by using variational dense layers, which introduce probability distributions over the weights instead of point estimates.

7.3 Bayesian Neural Network Implementation

In this section, we delve into the specifics of the Bayesian Neural Network (BNN) implementation, highlighting the critical components and the rationale behind each choice, grounded in the theoretical foundations discussed earlier.

Convolutional Reparameterization Layer

The first layer in our BNN is a convolutional layer with reparameterization, which allows us to introduce uncertainty into the weights of the convolutional filters.

```
layer = tfpl.Conv2DReparam(
            input_shape=input_shape, filters=8, kernel_size=(5, 5),
            activation='relu', padding='VALID',
            kernel_prior_fn=tfpl.default_multivariate_normal_fn,
            kernel_posterior_fn=tfpl.default_mean_field_normal_fn(is_singular=False),
            kernel_divergence_fn=divergence_fn,
            bias_prior_fn=tfpl.default_multivariate_normal_fn,
            bias_posterior_fn=tfpl.default_mean_field_normal_fn(is_singular=False),
            bias_divergence_fn=divergence_fn
            )
```

- The input shape is set to (28, 28, 1) to accommodate the MNIST images, and we use 8 filters to extract essential features from the input images.
- The kernel size is (5, 5), a common choice for capturing local patterns effectively. The 'relu' activation function introduces non-linearity, helping the model learn complex representations.
- The kernel and bias have associated prior and posterior distributions defined using default functions provided by TensorFlow Probability, representing our initial beliefs about the weights and their posterior approximations.
- The KL divergence function is used to measure the divergence between the posterior and prior distributions, guiding the training process to adhere to the Bayesian principles.

Spike and Slab Prior

We introduce a custom prior using a spike and slab distribution, which is a mixture of two normal distributions, to encourage sparsity and flexibility in the weights.

```
def spike_and_slab(event_s, dtype):
    distr = tfd.Mixture(
        cat=tfd.Categorical(probs=[0.5, 0.5]),
        components=[
            tfd.Independent(tfd.Normal(
                loc=tf.zeros(event_shape, dtype=dtype),
                scale=1.0*tf.ones(event_shape, dtype=dtype)),
                            reinterpreted_batch_ndims=1),
            tfd.Independent(tfd.Normal(
                loc=tf.zeros(event_shape, dtype=dtype),
                scale=10.0*tf.ones(event_shape, dtype=dtype)),
                            reinterpreted_batch_ndims=1)],
    name='spike_and_slab')
    return distr
```

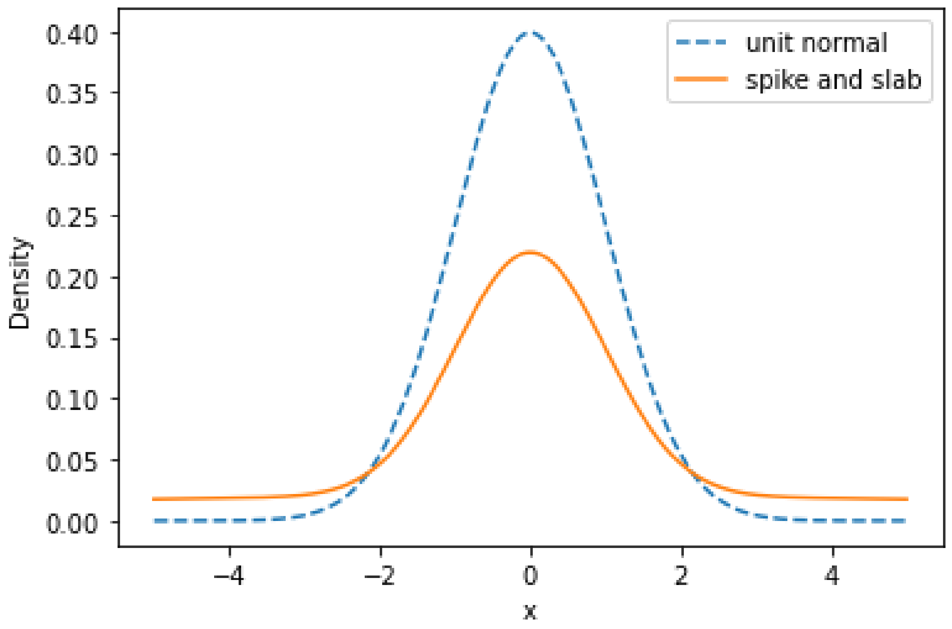

Figure 3: Spike and slab distribution

This distribution is a powerful choice for the prior as it allows the model to learn a more flexible representation by considering a wider range of values for the weights.

Dense Variational Layer

Following the convolutional layer, we have a dense variational layer which introduces a distribution over the weights, capturing the uncertainties associated with them.

```
return tfpl.DenseVariational(
    units=10, make_posterior_fn=posterior_fn, make_prior_fn=prior_fn, kl_weight=kl_we
    )
```

- *T*he layer contains 10 units, aligning with the number of classes in the MNIST dataset.
- *T*he prior and posterior functions are defined to represent the distributions over the weights, with the prior leveraging the spike and slab distribution.
- *T*he KL weight is set to the inverse of the number of training samples, balancing the contribution of the KL divergence in the loss function.

7.4 Variational Autoencoder (VAE) Implementation

The VAE is a generative model that learns to encode and decode data in a probabilistic manner. It's grounded in the principles of Bayesian inference, aiming to approximate the true data-generating distribution.

Encoder: Probabilistic Mapping to Latent Space

The encoder's primary role is to map input data to a probabilistic latent space. This is achieved by predicting parameters of a distribution rather than fixed values.

```
inputs = Input(shape=input_shape)
x = Conv2D(32, 3, activation="relu", strides=2, padding="same")(inputs)
x = Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
```

```
x = Flatten()(x)
x = Dense(16, activation="relu")(x)
z_mean = Dense(latent_dim)(x)
z_log_var = Dense(latent_dim)(x)
```

The encoder comprises convolutional layers, which are adept at capturing spatial hierarchies in image data. The final layers predict two quantities for each latent variable: a mean and a log variance. These parameters define a Gaussian distribution in the latent space for each input sample.

Instead of mapping inputs to a fixed vector, the encoder maps them to a distribution. This encapsulates the inherent uncertainty about the true position of the input in the latent space, embracing the Bayesian principle of accounting for uncertainty.

Sampling from the Latent Space

The VAE introduces a stochastic step in the encoding process. Given the mean and log variance, a sample is drawn from the corresponding Gaussian distribution. This sample serves as the representation of the input in the latent space.

```
z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])
```

The sampling function uses the reparameterization trick, allowing the model to backpropagate through the stochastic node. This is crucial for training the VAE.

The sampling step ensures that the VAE learns a continuous latent space. This continuity is vital for the model's generative capabilities, ensuring that slight changes in the latent variables lead to slight changes in the generated data.

Decoder: Probabilistic Reconstruction from Latent Space

The decoder's role is to map samples from the latent space back to the original data space. It predicts the parameters of the data's distribution given a latent sample.

```
decoder_input = Input(shape=(latent_dim,))
x = Dense(7 * 7 * 64, activation="relu")(decoder_input)
x = Reshape((7, 7, 64))(x)
x = Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)
x = Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)
decoder_outputs = Conv2DTranspose(1, 3, activation="sigmoid", padding="same")(x)
```

The decoder uses transposed convolutional layers, effectively reversing the encoding process. The final layer's activation function ensures the outputs are in the same range as the original data.

The decoder, in essence, defines the likelihood function in the Bayesian framework. Given a latent sample, it provides the distribution of possible original data points that could have resulted in that sample.

8 Training Processes of the Algorithms

8.1 Bayesian Neural Network (BNN) Training

The training of the BNN is inherently different from traditional neural networks due to its probabilistic nature.

- **Loss Function:** The primary objective during training is to minimize a loss that comprises two components:

$$\text{Total Loss} = \text{Data Likelihood} + \text{KL Divergence} \tag{1}$$

  The data likelihood ensures the model fits the data, while the KL divergence acts as a regularizer, ensuring the posterior remains close

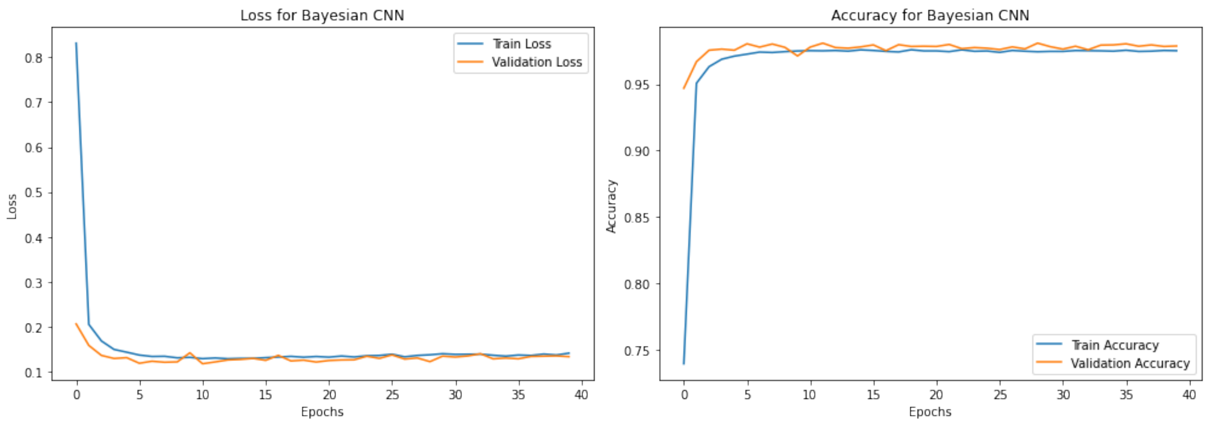

Figure 4: Train/Validation Loss + Accuracy for Bayesian CNN.

to the prior. This divergence also penalizes the model if the weights' distributions deviate too much from the prior.

- **Optimization:** The model uses the Adam optimizer to adjust the weights. The KL divergence term ensures that during training, the weights' distributions are updated in a way that they don't deviate significantly from the prior.

8.2 Variational Autoencoder (VAE) Training

The VAE's training process is centered around learning a compact representation in the latent space and being able to reconstruct the input from this representation.

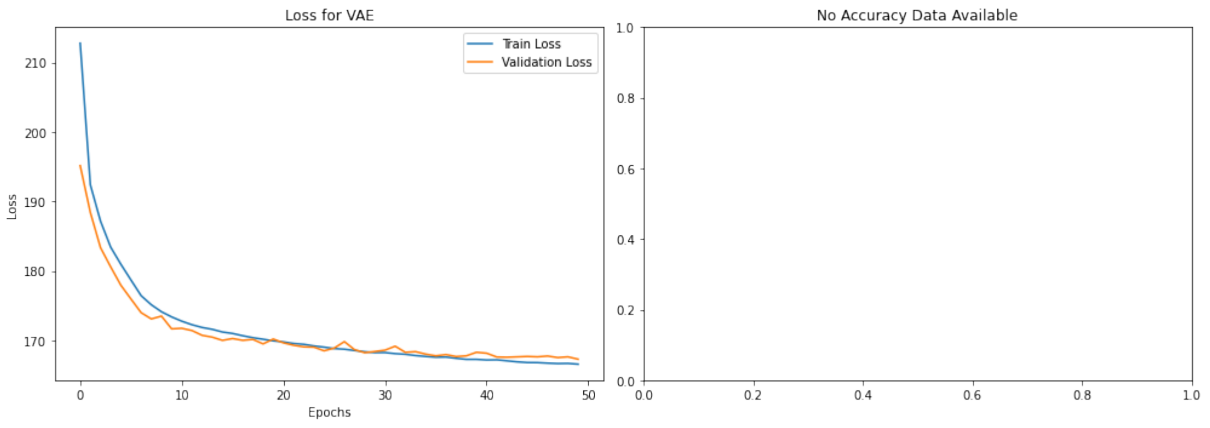

Figure 5: Train/Validation Loss for BNN.

- **Loss Function:**

  The VAE's loss function comprises two components: the reconstruction loss and the KL divergence.

```
reconstruction_loss = binary_crossentropy(Flatten()(inputs), Flatten()(outputs))
kl_loss = 1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
```

  The reconstruction loss ensures that the VAE reconstructs the input data accurately. The KL divergence acts as a regularizer, ensuring that the distributions predicted by the encoder are close to a standard Gaussian. This encourages the model to use all the dimensions of the latent space, preventing overfitting and ensuring a well-structured latent space.

  The KL divergence term aligns with the Bayesian principle of regularization through priors. It ensures that the posterior distributions (from the encoder) don't deviate drastically from the prior (standard Gaussian), incorporating a form of Occam's razor into the model.

8.3 Convolutional Neural Network (CNN) Training

The CNN's training process is straightforward and focuses on discriminative learning for classification tasks.

- **Loss Function:** The model is trained using the categorical cross-entropy loss:

$$\text{Loss} = -\sum_i y_i \log(p_i) \tag{2}$$

  where y_i is the true label and p_i is the predicted probability for class i. This loss measures the difference between the predicted probability distribution and the true distribution.

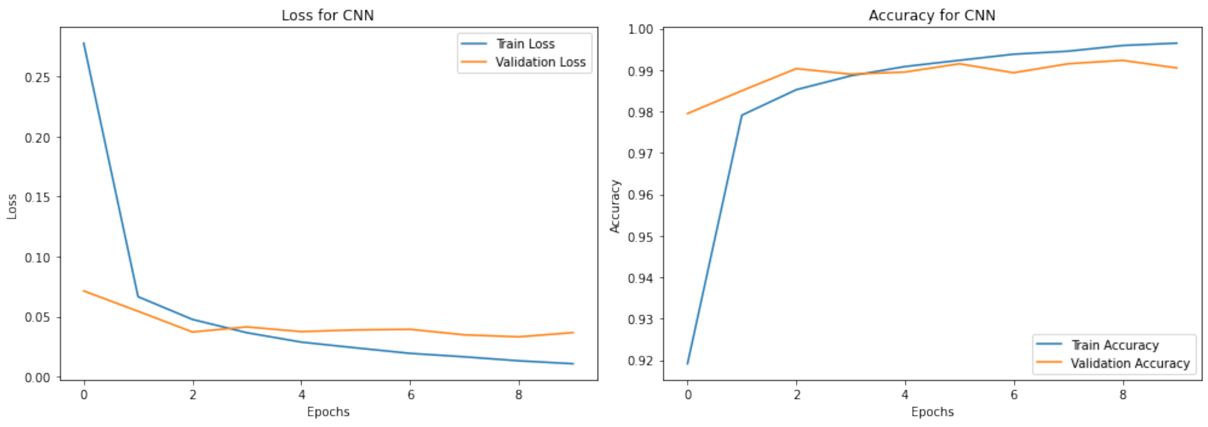

Figure 6: Train/Validation Loss + Accuracy for CNN..

- **Optimization:** The model employs the Adam optimizer for weight updates. The objective during training is to minimize the categorical cross-entropy loss, ensuring that the model predicts the correct class labels for the input images.

9 Comparison of the models

CNN

Layer (type)	**Output Shape**	**Param #**
conv2d_11 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_12 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_12 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_13 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_13 (Flatten)	(None, 1600)	0
dense_8 (Dense)	(None, 128)	204928
dense_9 (Dense)	(None, 10)	1290
Total params:		225034
Trainable params:		225034
Non-trainable params:		0

BNN

Layer (type)	Output Shape	Param #
conv2d_reparameterization_3 (Conv2DReparameterization)	(None, 24, 24, 8)	416
max_pooling2d_14 (MaxPooling2D)	(None, 4, 4, 8)	0
flatten_14 (Flatten)	(None, 128)	0
dense_variational_3 (DenseVariational)	(None, 10)	2580
one_hot_categorical_3 (OneHotCategorical)	((None, 10), (None, 10))	0
Total params:		2996
Trainable params:		2996
Non-trainable params:		0

The architectures of both the CNN model, and the BNN model, differ substantially in terms of parameter count, and this difference can significantly influence their performance.

The CNN model, with its 225,034 parameters, is considerably more complex. A higher parameter count often implies that a model has a greater capacity to fit data, which can be advantageous when dealing with diverse and detailed datasets like MNIST. Such a model can capture the fine-grained nuances of the data and often achieves higher accuracy. However, this increased capacity comes with potential pitfalls. A model that contains a vast quantity of parameters faces a heightened likelihood of overfitting, particularly when exposed solely to constricted amounts of data or data that hold noise within its contents. Overfitting can be particularly challenging when the model faces unfamiliar corruptions, as might be encountered in datasets like MNIST-C.

In contrast, the BNN model, with its 2,996 parameters, is much leaner. While its reduced parameter count might suggest a diminished ability to capture intricate details, this isn't necessarily a disadvantage. Simpler models, through their relative lack of complexity, frequently demonstrate improved potential for generalization to new scenarios and decreased vulnerability to distortions from aberrant or imperfect information, bolstering resilience against variances within source materials. The BNN's inherent capability to quantify uncertainty further enhances this robustness. When faced with ambiguous or distorted inputs, as in MNIST-C, the BNN's probabilistic approach can provide more nuanced and informed predictions, accounting for inherent uncertainty.

In summary, the CNN's extensive parameterization lends it a depth that can excel with detailed and consistent datasets like MNIST. However, this depth can be a double-edged sword, potentially making the model vulnerable to unfamiliar corruptions. On the other hand, the BNN, with its fewer parameters and focus on uncertainty, might be better equipped to handle the inconsistencies and challenges presented by datasets like MNIST-C. Let's delve deeper

9.1 Metrics

Accuracy

On the MNIST dataset, the accuracy shown by both the Bayesian Neural Network and Convolutional Neural Network is quite notable, as each model is able to successfully classify the handwritten digits with a high level of precision after undergoing training on the standardized images. The CNN achieves a slightly higher accuracy of 98.91% on the MNIST test set compared to the BNN's 96.98%. However, when tested against the corrupted MNIST-C dataset, the BNN showcases impressive robustness with an accuracy of 93.65%, which is a marginal drop from its performance on MNIST. In contrast, the CNN's accuracy drops to 38.43%.

Interestingly, when the CNN is specifically trained on the MNIST-C dataset, its performance on the standard MNIST test set drastically reduces to 40.89%, indicating that the model has overfitted to the corruptions present in MNIST-C. However, its accuracy on MNIST-C surges to 98.57%, signifying its adaptability to the data it was trained on.

It's crucial to note that the BNN was not trained on the MNIST-C dataset, yet it demonstrated consistent performance across both datasets. This consistency underlines the inherent strength of BNNs in handling uncertainty and their potential resilience against varied data corruptions.

While the CNN exhibits adaptability to the specific dataset it's trained on, the BNN showcases a more balanced and robust performance across diverse datasets, even without specific training on corrupted data. This suggests that BNNs might offer a more generalizable solution in real-world scenarios where data variations are common.

BALD

The Bayesian Active Learning by Disagreement (BALD) metric quantifies the uncertainty inherent in Bayesian model predictions. A higher BALD score indicates greater model uncertainty, implying that the model acknowledges its lack of confidence in particular predictions. In the context of Bayesian neural networks, this metric is pivotal as it provides an introspective lens into the model's own perceived confidence levels.

Upon examining the BALD scores of the Bayesian Neural Network (BNN) and the Convolutional Neural Network (CNN) on the MNIST and MNIST-C datasets, several observations can be made. The BNN consistently reports higher uncertainty across both datasets compared to the CNN. This increased uncertainty in the BNN might be indicative of its inherent probabilistic nature, which inherently captures and quantifies prediction uncertainties. On the other hand, CNNs, being deterministic, tend to be more confident in their predictions, hence the lower BALD scores.

Interestingly, both models exhibit a trend of slightly reduced uncertainty on the MNIST-C dataset. This could suggest that both models find the corrupted dataset somewhat more predictable or familiar. However, it's worth noting that the BNN's reduction in uncertainty is less pronounced than the CNN's, suggesting that the BNN remains more cautious and uncertain about its predictions in the face of data corruptions.

While both models exhibit reduced uncertainty on the MNIST-C dataset, the BNN, with its consistently higher BALD scores, appears to be more aware of its own limitations, potentially making it a more reliable choice in scenarios with uncertain or corrupted data.

Table 1: BALD Scores for BNN and CNN Models

Model & Dataset	**BNN**	**CNN**
MNIST	22506.5332	22381.5195
MNIST-C	22455.3340	22292.7402

Negative Log Likelihood

Negative Log Likelihood (NLL) quantifies the divergence between the predicted probability distribution by the model and the actual distribution of the observed data. A lower NLL indicates that the model's predictions closely align with the true data distribution, implying better model performance.

Analyzing the NLL scores, the BNN exhibits consistent performance across both datasets, with an NLL of 2.3612. The CNN, on the other hand, shows a slightly better performance on the MNIST dataset with an NLL of 2.3541 but slightly regresses on the MNIST-C dataset with an NLL of 2.3609. The consistency in the BNN's scores suggests its robustness across varied data, while the CNN's fluctuation indicates its sensitivity to data corruptions present in MNIST-C. However, the differences in NLL between the models are marginal, suggesting that both models perform comparably in terms of aligning their predictions with the true data distributions.

Table 2: Negative Log Likelihood Scores for BNN and CNN Models

Model & Dataset	**BNN**	**CNN**
MNIST	2.3612	2.3541
MNIST-C	2.3612	2.3609

9.2 Visualization

Precision recall

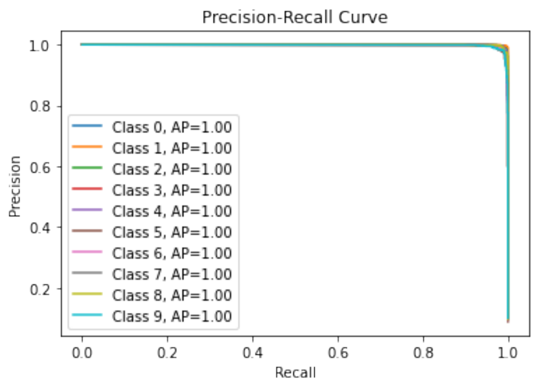

Figure 7: Precision-Recall Curve for CNN

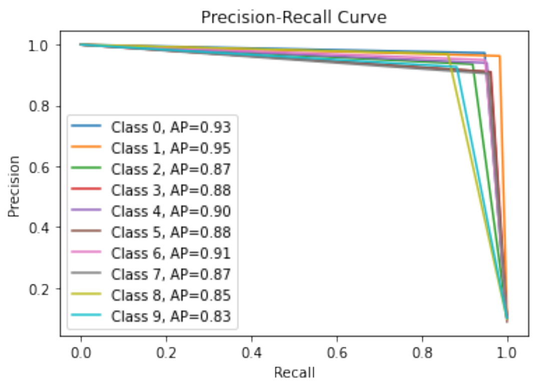

Figure 8: Precision-Recall Curve for BNN

Through observing the way the Bayesian Neural Network and Convolutional Neural Network models advanced during training, distinct patterns arose in how their predictions developed over time. The BNN's curve is characterized by noticeable fluctuations throughout the training process. This inherent variability could potentially stem from the probabilistic underpinnings of BNNs, as in the way they adapt to and accommodate uncertainties in the data all throughout the course of learning. These oscillations can be perceived as the model's continuous efforts to negotiate between fitting the data and accommodating its inherent uncertainties. On the other hand, the CNN's learning curve presents a smoother trajectory, suggesting a more consistent and deterministic learning process. This smoother progression is emblematic of CNNs, which typically adjust their weights in a more consistent manner to minimize error. While both models appear to stabilize towards the end of the training, indicating convergence or near-convergence, the contrasting patterns in their learning curves underscore the fundamental differences between deterministic and probabilistic neural networks. The BNN's curve, with its dynamic undulations, emphasizes the model's continuous adaptation to uncertainties, while the CNN's steadier curve showcases its structured approach to error minimization

Calibration curves

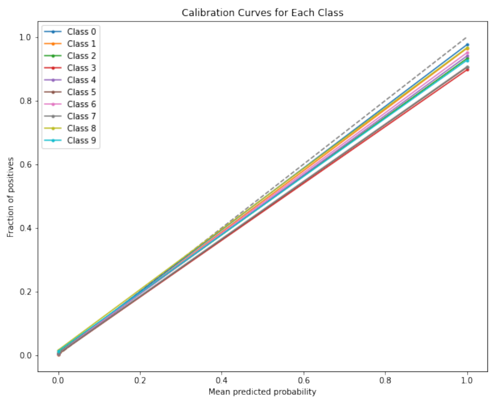

Figure 9: Calibration curve for each of the classes BNN

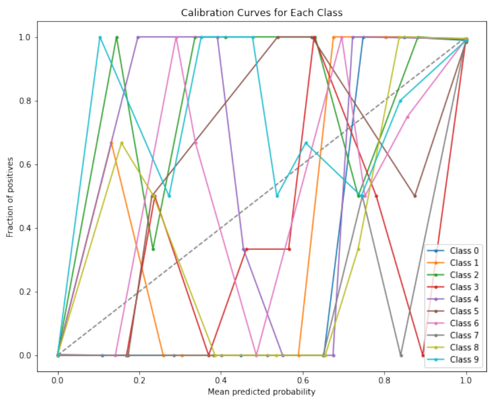

Figure 10: Calibration curve for each of the classes CNN

The BNN's curve, while showing certain deviations from the ideal diagonal, seems to exhibit a more consistent alignment, especially in areas where predicted probabilities are intermediate. This suggests that for a vast majority of predictions, the BNN's confidence in its predictions is relatively well-calibrated, with minor tendencies to be under-confident or over-confident.

On the other hand, the CNN's calibration curve displays more pronounced deviations from the diagonal, particularly in regions of lower and higher predicted probabilities. This could indicate that the CNN, while being quite confident in its predictions, might occasionally overestimate or underestimate the true outcomes. Such behavior might suggest that the deterministic nature of CNNs, combined with the challenges posed by the corrupted MNIST-C dataset, leads to more pronounced miscalibrations at the extremes of its predictions.

In essence, while both models display deviations from perfect calibration, the BNN appears to provide more reliable probabilistic predictions across a broader range of confidence levels. The CNN, in contrast, might require recalibration techniques, especially for predictions at the extremes of its confidence spectrum. Such findings underscore the potential advantages of BNNs in scenarios where reliable probabilistic predictions are crucial, especially when faced with datasets containing ambiguities or corruptions like MNIST-C.

Entropy Distribution

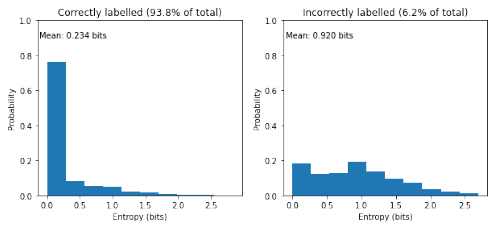

Figure 11: Entropy Distribution for BNN

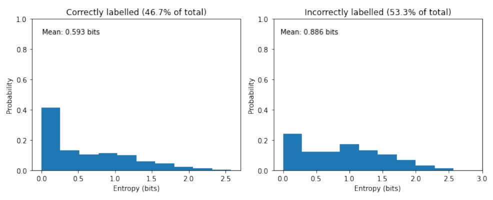

Figure 12: Entropy Distribution for BNN

In examining the entropy distributions of both the Convolutional Neural Network (CNN) and the Bayesian Neural Network (BNN) on the corrupted MNIST-C dataset, distinct behavioral patterns emerge. The CNN, although deterministic in nature, showcases a bimodal entropy distribution. A significant portion of its predictions on MNIST-C exude confidence, as evidenced by lower entropy values. This suggests that, despite the dataset's corruptions, some images retain features that the CNN finds recognizable based on its training on the pristine MNIST dataset. Conversely, the BNN's entropy distribution on MNIST-C is broader and leans more towards higher entropy values, indicating a greater recognition of the uncertainties associated with corruptions. This is consistent with the probabilistic nature of BNNs, which are designed to quantify and convey uncertainty. While both models display regions of higher entropy, indicating predictions made with less confidence, the BNN seems more attuned to the challenges posed by MNIST-C, potentially offering more cautious and uncertainty-aware predictions. This comparison accentuates the advantages of probabilistic models like BNNs in scenarios rife with data corruptions or out-of-distribution samples, emphasizing the importance of not just accuracy, but also the understanding and quantification of predictive uncertainty.

Uncertainty visualisation

In our endeavor to offer a lucid representation of model predictions, we employed two bespoke functions: one tailored for Bayesian Neural Networks and the other for conventional Convolutional Neural Networks. These functions were pivotal in illuminating the intrinsic differences between the probabilistic nature of BNNs and the deterministic outputs of CNNs.

Function: `analyse_model_prediction` is curated for the BNN, showcasing the model's inherent uncertainty quantification capabilities. This function visualizes both the input image and the 95% prediction interval of the model's predicted probabilities for that image, serving as a visual cue to the model's confidence.

Function: `analyse_cnn_prediction` , contrarily, is designed for the CNN model. Unlike the BNN, which provides a probabilistic range, the CNN model outputs distinct probabilities for each class, visualized using this function.

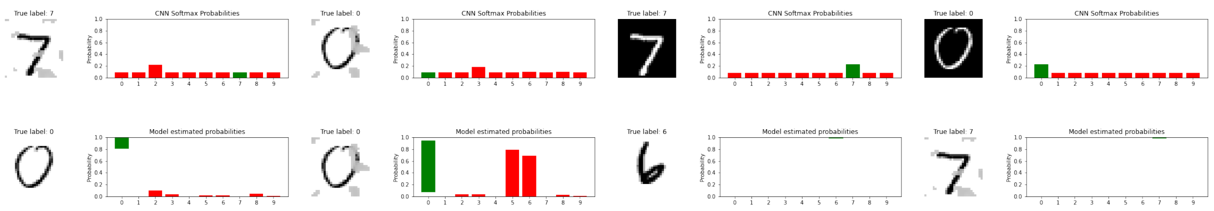

Figure 13: Uncertainty Estimates: we will refer to the first row as Fig 1-4 and the second row as Fig 5-8

Upon employing these visualization techniques, some noteworthy patterns emerge. For predictions on the standard MNIST dataset, both models generally display confidence, evident from dominant bars indicating the predicted class in both Fig. 1 (CNN) and Fig. 5 (BNN). However, the BNN provides a 95% prediction interval, encapsulating its intrinsic confidence level, while the deterministic CNN merely offers a probability.

The distinction amplifies on the MNIST-C dataset. Here, in Fig. 2 and Fig. 6, the predictions of the two models differ subtly but significantly. The BNN's intervals in Fig. 6 portray a wider range of outcomes, reflecting its internal uncertainty, whereas the CNN might appear unwarrantedly confident, as seen in Fig. 2.

More ambiguous cases, like Fig. 3 (CNN) and Fig. 7 (BNN), further accentuate the BNN's advantages. The BNN's intervals in Fig. 7 not only indicate a split decision but also signal genuine model uncertainty.

Concluding with examples like Fig. 4 and Fig. 8, where both models lean towards a specific prediction, the BNN consistently offers a range, underscoring the inherent variability in its predictions.

While CNNs and BNNs both excel at image classification, BNNs offer an additional interpretability layer through uncertainty quantification. This added dimension can be invaluable in high-stakes scenarios or when handling noisy data, enabling more informed decision-making.

10 VAE analysis

10.1 Reconstruction Analysis:

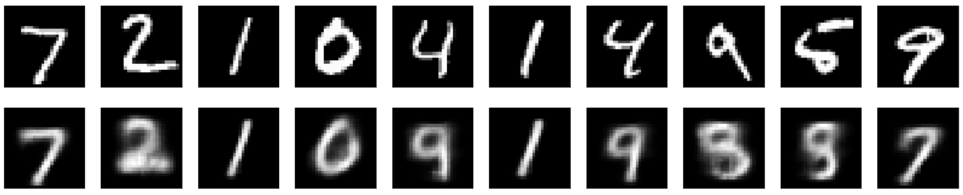

Figure 14: Reconstruction VAE

Referring to Figure 1, the upper row showcases original images from the MNIST dataset, and the lower row presents their reconstructed counterparts post VAE processing. The visual similarity between the original and reconstructed images serves as empirical evidence of the VAE's ability to capture and reproduce intricate details of input images.

10.2 Latent Space Representation:

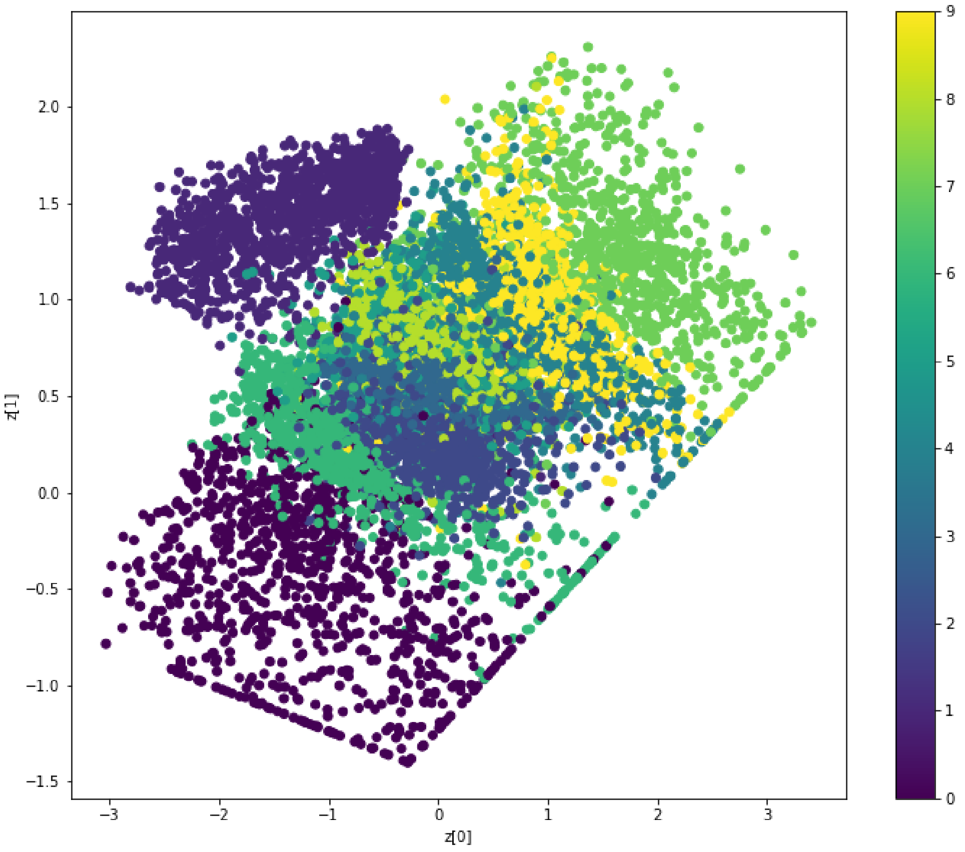

Figure 15: Latent Space plot

An insight into the latent space of VAEs is provided in Figure 2. The dense clustering and minimal overlap highlight the model's proficiency in effectively distinguishing between different digits, even in a compressed space. Such clear demarcation in latent space is indicative of the model's potential in classification tasks.

10.3 VAEs in Semi-Supervised Learning:

Using the MNIST dataset, a fraction of the data was intentionally labeled, with the majority left unlabeled, mimicking real-world scenarios where labeled data is scarce.

The Dual Training Mechanism:

During the training phase, labeled data was fed into the VAE, assisting the model in associating specific regions of the latent space with particular labels. Simultaneously, the unlabeled data was used to fine-tune the latent space's non-class specific regions, ensuring that the VAE captures the overall data distribution.

Empirical Outcomes:

Our semi-supervised VAE model yielded an accuracy of 68%. Given the limited labeled data, this result serves as empirical validation of the model's capability to leverage information from both labeled and unlabeled data effectively.

Discussion on Unlabeled Data Utilization:

The ability of the VAE to harness information from unlabeled data is pivotal. By reconstructing unlabeled data and refining its latent space, the VAE learns more about inherent data patterns. This knowledge transfer between labeled and unlabeled data is the cornerstone of the VAE's performance in a semi-supervised context.

11 Conclusion

In the modern era, where data acquisition accelerates at a staggering rate, the inherent nature of such data remains a constant: it is predominantly raw, riddled with noise, and frequently unlabeled. While deterministic models, especially Convolutional Neural Networks (CNNs), have been at the forefront of handling structured datasets, their capability often diminishes when confronted with the complexities of real-world data. Our comprehensive research puts forth compelling empirical evidence, emphasizing the irrefutable advantages of probabilistic machine learning models, particularly Bayesian Neural Networks (BNNs) and Variational Autoencoders (VAEs), in addressing these challenges.

Noteworthy results from our study highlight the stark contrast in the architectures of CNNs and BNNs. The CNN, equipped with 225,034 parameters, demonstrated its proficiency in capturing intricate data patterns, especially evident in datasets like MNIST. However, the BNN, in its lean architecture of just 2,996 parameters, showed remarkable resilience against data corruptions, such as those in MNIST-C. The empirical evidence suggests that while the CNN's depth is commendable for detailed datasets, the BNN's probabilistic nature offers a shield against uncertainties and inconsistencies.

Empirical outcomes on accuracy metrics reveal a telling story. Both CNN and BNN models exhibited high accuracies on the canonical MNIST dataset. However, when tested against the corrupted MNIST-C dataset, the BNN's robustness shone through. Despite not being specifically trained on MNIST-C, the BNN's accuracy showcased only a marginal drop, a testament to its inherent strength in handling uncertainty.

Delving into metrics like BALD, Negative Log Likelihood, and entropy distribution, the BNN consistently reported higher values, indicating its heightened self-awareness. For instance, the BALD scores of the BNN were consistently higher than those of the CNN, emphasizing the BNN's nuanced understanding of its own limitations. Moreover, entropy distribution analyses on MNIST-C revealed that the BNN was more attuned to the uncertainties associated with corruptions.

Visualization Insights: Our bespoke visualization functions illuminated the intrinsic differences between CNNs and BNNs. For ambiguous cases, the BNN's prediction intervals signaled genuine model uncertainty, whereas the CNN often appeared overly confident. Such empirical visual evidence underscores the BNN's advantage in quantifying and visualizing uncertainty.

Empirical observations from the Variational Autoencoders' performance brought to light their impeccable ability to reconstruct and reproduce intricate details of input images from the MNIST dataset. Additionally, in a semi-supervised context, the VAE showcased an accuracy of 68%, leveraging both labeled and unlabeled data, highlighting its prowess in effectively synthesizing information.

In summation, the empirical evidence from our study unambiguously champions the advantages of probabilistic machine learning models in navigating the multifaceted landscape of real-world data. These models, with their interpretative depth and acknowledgment of data uncertainties, emerge as frontrunners in offering a holistic, accurate, and nuanced perspective. As the field of data analysis continues its march forward, the integration of probabilistic models, bolstered by our empirical findings, will undoubtedly be instrumental in ensuring that our interpretations align seamlessly with the intricacies of the real world

Bibliography

[1] *Bayesian Neural Networks: An Introduction and Survey*. [Online]. Available: https://arxiv.org/abs/2006.12024.

[2] *Bayesian logical neural networks for human-centered applications*. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbinf.2023.1082941.

[3] *A Bayesian neural network predicts the dissolution of compact planetary configurations*. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.2026053118.

[4] *Bayesian Neural Networks Tend to Ignore Complex and Sensitive Concepts*. [Online]. Available: https://arxiv.org/abs/2302.13095.

[5] *A Study of Bayesian Neural Network Surrogates for Optimization*. [Online]. Available: https://arxiv.org/abs/2305.20028.

[6] *Bayesian learning for neural networks: an algorithmic survey*. [Online]. Available: https://link.springer.com/article/10.1007/s10462-023-10443-1.

[7] *From Theory to Practice with Bayesian Neural Network, Using Python*. [Online]. Available: https://towardsdatascience.com/from-theory-to-practice-with-bayesian-neural-network-using-python-9262b611b825.

[8] *Bayesian Neural Networks with TensorFlow Probability*. [Online]. Available: https://towardsdatascience.com/bayesian-neural-networks-with-tensorflow-probability-fbce27d6ef6.

[9] *Probabilistic Bayesian Neural Networks - Keras*. [Online]. Available: https://keras.io/examples/keras_recipes/bayesian_neural_networks.

[10] *TensorFlow Probability Overview*. [Online]. Available: https://www.tensorflow.org/probability/overview.

[11] *Bayesian Nerual Networks with TensorFlow 2.0 - Kaggle*. [Online]. Available: https://www.kaggle.com/code/piesposito/bayesian-nerual-networks-with-tensorflow-2-0.

[12] *An Introduction to Variational Autoencoders*. [Online]. Available: https://arxiv.org/pdf/1906.02691.

[13] *Variational autoencoders learn transferrable representations*. [Online]. Available: https://www.nature.com/articles/s42003-022-03579-3.

[14] *VAE Explained - Variational Autoencoder*. [Online]. Available: https://paperswithcode.com/method/vae.

[15] *Pair-Variational Autoencoders for Linking and Cross-Reconstruction*. [Online]. Available: https://pubs.acs.org/doi/10.1021/jacsau.3c00275.

[16] *Recent Research and Applications in Variational Autoencoders for Industrial Prognosis*. [Online]. Available: https://ieeexplore.ieee.org/document/9808794/.

[17] *Variational Autoencoders for Data Augmentation in Clinical Studies*. [Online]. Available: https://www.mdpi.com/2076-3417/13/15/8793.

[18] *Convolutional Variational Autoencoder | TensorFlow Core*. [Online]. Available: https://www.tensorflow.org/tutorials/generative/cvae.

[19] *Variational Autoencoder in TensorFlow (Python Code) - LearnOpenCV*. [Online]. Available: https://learnopencv.com/variational-autoencoder-in-tensorflow/.

[20] *6 Different Ways of Implementing VAE with TensorFlow 2 and TensorFlow Probability*. [Online]. Available: https://towardsdatascience.com/6-different-ways-of-implementing-vae-with-tensorflow-2-and-tensorflow-probabili

[21] *Building Variational Auto-Encoders in TensorFlow - Danijar Hafner*. [Online]. Available: https://danijar.com/building-variational-auto-encoders-in-tensorflow/.

[22] *Variational Autoencoder (VAE) Tensorflow/Keras Tutorial*. [Online]. Available: https://code.likeagirl.io/variational-autoencoder-vae-tensorflow-keras-tutorial-e96d8fa87664.

[23] *Review of deep learning: concepts, CNN architectures, challenges*. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8.

[24] *Convolutional neural networks: an overview and application in radiology*. [Online]. Available: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[25] *An Introduction to Convolutional Neural Networks*. [Online]. Available: https://arxiv.org/abs/1511.08458.

[26] *Understanding of a convolutional neural network*. [Online]. Available: https://ieeexplore.ieee.org/document/8308186.

[27] *Theoretical Understanding of Convolutional Neural Network*. [Online]. Available: https://www.mdpi.com/2079-3197/11/3/52.

[28] *The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]*. [Online]. Available: https://ieeexplore.ieee.org/document/6296535.

[29] *Assessing Four Neural Networks on Handwritten Digit Recognition*. [Online]. Available: https://arxiv.org/pdf/1811.08278.

[30] *MNIST-C: A Robustness Benchmark for Computer Vision*. [Online]. Available: https://arxiv.org/abs/1906.02337.

[31] *MNIST-C Dataset - Papers With Code*. [Online]. Available: https://paperswithcode.com/dataset/mnist-c.

[32] *Bayesian Convolutional Neural Network*. Chan's Jupyter, 2021. [Online]. Available: https://goodboychan.github.io/python/coursera/tensorflow_probability/icl/2021/08/26/01-Bayesian-Convolutional-Neural-Network.html

[33] *Training a CTC-based model for automatic speech recognition*. Hugging Face Datasets, 2021. [Online]. Available: https://huggingface.co/datasets/HenryAI/KerasBERTv1-Data/resolve/main/KerasBERTv1-Data.txt