Improving Reliable Classification by Adding Extra Information

Yosha Udaya Shriyan

Submitted for the Degree of Master of Science in

Artificial Intelligence

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

December 6, 2021

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 6904

Student Name: Yosha Udaya Shriyan

Date of Submission: 6th December 2021

Signature:

Acknowledgements

This project has taken a lot of time and effort on my part, as well as a lot of enthusiasm. All of this work, however, has been aided and made possible by my supervisor, Dr Ilia Nouretdinov. He was a great help to me for understanding the new concepts. His subject expertise, intuition and patience helped me get a much better idea about my project. I thank him for patiently answering all my questions and guiding me in the right direction whenever and wherever I got stuck while working on this project. He has played a major role in making the project what it is today. I would also like to thank Ms. Eva Dann for guiding me through the referencing rules and regulation to make the process easier.

Abstract

The objective of this project was to see how providing privileged/additional information affects the classification of a model. The additional data is like additional observations that we, as humans, do while learning anything new. This paradigm is used along with Inductive Conformal Prediction to give a classification that is reliable. The amount of additional information to be supplied to the training data is what we attempt to find out. This is accomplished using several python libraries, datasets and methods. Object-oriented programming plays an important role in streamlining the process.

We also come across computer vision techniques for classification of image data using Convolutional Neural Network, the various parameters of the model and why it is a good choice as a model. We see why the use of Inductive Conformal Prediction is better for Neural Networks compared to Conformal Prediction.

The experiment is done on the MNIST hand-written dataset, for the prediction of the numbers '4' and '9' as there are multiple ways to write them, which varies from person to person. The data is preprocessed using various python libraries. The results are compared using the visualization tools of python.

Contents

1 Introduction

"A teacher is one who makes himself progressively unnecessary." –Thomas Carruthers. The paradigm of 'Learning Using Privileged Information' or LUPI is brought forward by observing the Teacher-Student relationship (Vapnik and Izmailov, 2015, pp. 2023–2049). In his paper, Vapnik compares the existing machine learning paradigm with human learning and how it gives information additional to just the right answer to a question. LUPI paradigm is a way to incorporate human learning features into classical machine learning paradigms. The fundamental point in Vapnik's privileged information is that the additional information is only available for training samples and not test samples.

In order to ensure whether or not the classification is reliable, we use Inductive Conformal Prediction (ICP). The underlying principle behind ICP mode is 'Conformal Predictions' and 'Nonconformity Measures'. Conformal Predictors are confidence predictors that make a range of successively more specific predictions with successively less confidence. Nonconformity measure is a way of measuring how different a new example is from old examples (Vovk, 2005).

In reality, the confidence measures generated by CPs are valuable, and their accuracy is comparable to, if not greater than, that of classic machine learning approaches. The sole negative is that they are inefficient in terms of computing. This is due to the use of transductive inference, which necessitates starting all computations from scratch for each test example. Unfortunately, because of this computational inefficiency, the CP method is unsuited for any method that requires extended training times, such as Neural Networks. Hence, ICP is used to overcome this problem. ICP was proposed in (Papadopoulos *et al.*, 2002, pp. 345–356) for regression and (Papadopolous, Vovk and Gammerman, 2002, pp. 159–163) for pattern recognition.

Finally, how much privileged information should be provided for the model to give reliable classification is the main problem this project is focused around. I have compared the performance of the model with varying privileged information supplied to the training data. Before moving ahead to the experiment part of the project, let us go through the background research and related information.

2 Background Research

2.1 Learning Using Privileged Information

2.1.1 Framework

(Vapnik and Izmailov, 2015, pp. 2023–2049) take into account the following mechanisms of Teacher-Student interactions:

- Mechanism to control Student's concept of similarity between training examples.
- Mechanism to transfer knowledge from the space of privileged information (Teacher's explanation) to space where decision rule is constructed

2.1.2 Limited Privileged Information

Consider the following data:

- Training set: $(x_1, h_1, y_1), \ldots, (x_n, h_n, y_n)$
- Testing sample: x_{new}
- To predict: y_{new} - ?

We are considering that h_i is some expensive or time-consuming piece of information and we cannot obtain this information for real time data.

Now, we assume that for *k* of the *n* training examples, we have obtained the additional information. The amount of privileged data given (value of k) is a baseline for performance for this project.

2.1.3 Inductive Mode of Learning

In the inductive mode of learning, the training data is divided into two parts – a proper training set and a calibration set (see Section 2.2 for further details and references).

Consider the following data:

- Proper training set: $(x_1, h_1 ?, y_1),\ldots, (x_r, h_r ?, y_r)$, where privileged information can be obtained for *k* of the *r* examples.
- Calibration set: $(x_1, y_1),\ldots, (x_b, y_b)$
- Testing sample: x_{new}
- To predict: y_{new} -?

We approach the problem with the following algorithm:

- Input the proper training set $(x_1, y_1),\ldots, (x_r, y_r)$.
- Select *k* examples with numbers $i \in Q \subset \{1,\ldots, r\}$
- Input h_i for each $i \in Q$
- Input calibration set $(x_1, y_1),\ldots, (x_b, y_b)$
- Predict $\hat{h}_i$ for all examples after training on Q
- Run an Inductive Conformal Predictor on the following data:
    - Proper training set: $(x_1, \hat{h}_1, y_1),\ldots, (x_r, \hat{h}_r, y_r)$
    - Calibration set: $(x_1, \hat{h}_1, y_1),\ldots, (x_r, \hat{h}_b, y_b)$
    - Testing set: $(x_1, \hat{h}_1),\ldots, (x_r, \hat{h}_t)$
    - To predict: $(y_1),\ldots, (y_t)$
- The performance measure is the average confidence.

2.2 Inductive Conformal Prediction (ICP)

2.2.1 What is ICP?

Conformal Prediction (CP) is a technique for adding confidence metrics to the bare predictions produced by any typical machine learning algorithm. Although CP provides strong accuracy and confidence levels, it is computationally inefficient. When CP is combined with a technology that requires extended training times, such as Neural Networks, the computational inefficiency problem becomes enormous (Vovk, 2005).

Inductive Conformal Prediction (ICP), a variation of the original CP method allows us to build a Neural Network confidence predictor without the large computing overhead of CP. While keeping the computational efficiency of its underlying Neural Network, the method presented accompanies its predictions with confidence measures that are useful in practise.

In ICP, the training set into two parts, namely the *proper training set* and the *calibration set*. This way, the primary algorithm is applied only once to the proper training set. To get the general idea of the steps followed by ICP and further reading refer (Papadopoulos, Vovk and Gammerman, 2007, pp. 388–395).

2.2.2 ICP in Neural Networks

As stated in the previous section, (Papadopoulos, Vovk and Gammerman, 2007, pp. 388–395) defines the method to use ICP in Neural Networks used for pattern recognition, and says the method can be used for any Neural Network that uses the 1-of-n output encoding.

In general, the output layer of a classification of Neural Network has c units, each of which represents one of the c possible classification of the problem in hand, so essentially every label is encoded into c target outputs. The Neural Network is assumed to have an output softmax layer. Consequently, the prediction y_g of the network, for a test pattern g, will be a label mapping to the highest value of the output.

Talking about the non-conformity measure, for an example i that has the true classification Y_u, the example will be more conforming if the output o_u^i (corresponding to that classification) will be high and less conforming if other outputs are high. Hence, the nonconformity score for an example $z_i = (x_i, y_i)$ where $y_i = Y_u$ is defined by Papadopoulos as:

$$\alpha_i = \max_{j=1,\ldots,c:j\neq u} o_j^i - o_u^i,$$

We take the following steps for the algorithm of the Neural Network ICP:

- The training data is split into proper training set and calibration set. m = l – q, where m and q are the number of examples in proper training set and calibration set respectively.
- The proper training set is used to train the neural network.
- For every example $z_{m+t} = (x_{m+t}, y_{m+t})$: t = 1, …, q in the calibration set:
    - the input pattern x_{m+t} is provided to the trained network to obtain outputs $o_1^{m+t}, \ldots, o_c^{m+t}$ and
    - the nonconformity score is calculated using the above formula for the pair (x_{m+t}, y_{m+t})
- For every test pattern x_{l+g} : g = 1, …, r,

- the input pattern x_{l+g} is supplied to the trained network to get the output values $o_1^{l+g}, \ldots, o_c^{l+g}$.
- every possible classification $Y_u : u = 1, \ldots, c$ is considered and
  - the nonconformity score of the pair (x_{m+t}, Y_u) is computed as $\alpha_{l+g}^{(Y_u)}$ using the above formula.
  - The p-value is computed for x_{l+g} being classified as each possible Y_j label, with the nonconformity scores:

$$\alpha_{m+1}, \ldots, \alpha_{m+q}, \alpha_{l+g}^{(Y_j)}$$

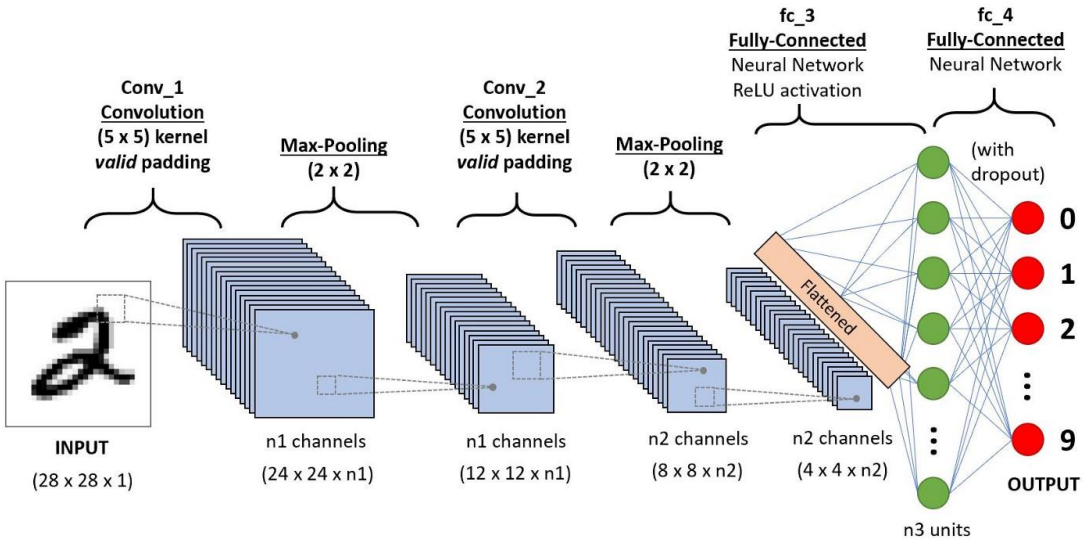
So

$$p(Y_j) = \#\{ i = m+1, \ldots, m+q, l+g: \alpha_i >= \alpha_{l+g}^{(Y_j)} \} / q+1$$

(Noretdinov, Melluish and Vovk, 2001, pp. 385–392)

- The classification with largest p-value is predicted
- The confidence to this prediction is computed as one minus the second largest p-value, and the credibility is computed as the largest p-value.

2.3 Convolutional Neural Network



CNN Sequence to classify handwritten digits (Saha, 2018)

Convolutional Neural Networks (CNNs) are widely used neural networks for pattern recognition. The name CNN is taken after mathematical linear operation between matrices known as convolution. The multiple layers of CNN include a convolution layer, non-linearity layer, pooling layer and a fully connected layer, where the convolution and fully connected layers have parameters while the pooling and non-linearity layers don't have parameters. CNN have outstanding performance in machine learning problems, mainly for image classification datasets, which is the reason why I have chosen it as a model for this project. You can refer (Albawi, Mohammed and Al-Zawi, 2017, pp. 1–6) for a more informative encapsulation on the elements of CNN and how they work.

2.3.1 Sequential Class

The *Sequential* class in keras library groups a linear stack of layers into the *tf.keras.Model*. The *Sequential* provides training and interference features on this model. The *Sequential* model is appropriate for a plain stack of layers where each stack is exactly one input tensor and one output tensor. The contents of the model can be displayed by calling the summary() method, once the model is built (Team, 2020).

2.3.2 Layers

For the CNN model, we have out Conv2D layers, which are convolutional layers that will deal with out input images, which are seen as 2-dimensional matrices. The number of nodes in each layer may vary, and is adjusted depending on the size of the dataset (Allibhai, 2018). The kernel size refers to the size of the filter matrix for our convolution.

After the Conv2D layers comes a MaxPooling layer. Maximum pooling or max pooling is a pooling operation that calculates the maximum value in each patch of the feature map (Brownlee, 2019) . In the case of max pooling, the outputs are down sampled or pooled feature maps that highlight the most present feature in the patch rather than the average presence of the feature. For computer vision applications like image classification, this has been found to work better in practice than average pooling.

Following this, before the *Dense* layer, comes the *Flatten* layer, which serves as a connection between the Convolutional and *Dense* layers.

Dense is the layer type we will use in for our output layer. *Dense* is a standard layer type that is used in many cases for neural networks.

2.3.3 Activation Functions

Also known as Transfer Function, it is used to determine the output of a node, by mapping the resulting values in between 0 to 1 or -1 to 1 etc., depending on the function (SHARMA, 2021). The two activation functions used in this project are ReLU and softmax activations.

The ReLU activation is implemented by the 'relu' function, which applies the rectified linear unit activation function. With default values, it returns the standard relu activation, i.e., max(x,0) – the element-wise maximum of 0 and the input tensor.

The softmax activation is implemented by the 'softmax' function, which converts the vector values to a probabilistic distribution. The elements of the output of the vector range from 0 to 1 and sum to 1. Every single vector is handles independently. Because the output may be understood as a probability distribution, Softmax is frequently employed as the activation for the last layer of a classification network. The softmax of each vector x is computed as exp(x) / tf.reduce_sum(exp(x)). The input values in are the log-odds of the resulting probability.

2.3.4 Optimizers

Optimizers are techniques or approaches that adjust the characteristics of your neural network, such as weights and learning rate, to reduce losses. The optimizers you utilise determine how you should alter your neural network's weights or learning rates to reduce losses. Optimization algorithms or methods are in charge of lowering losses and delivering the most accurate results.

Adam (Adaptive Moment Estimation) works with first- and second-order momentums. *Adam*'s intuition is that we don't want to roll too fast only to hop over the minimum; instead, we want to slow down a little to allow for a more deliberate search. Adam preserves an exponentially decaying average of previous squared gradients and an exponentially decaying average of past gradients in addition to an exponentially decaying average of past squared gradients.

Adam is one of the most skilled optimizers. Adam is the optimizer to use if you want to train a neural network in less time and with higher efficiency. Check (Doshi, 2020) for comparisons with other optimizers. Hence, I have chosen *Adam* optimizer for my Neural Network.

2.3.5 Loss

Loss functions are used to determine the quantity that a model should try to minimise during training.

CategoricalCrossentropy is a type of probabilistic loss. It computes the crossentropy loss between the labels and predictions. This crossentropy loss function is used when there are two or more label classes. The labels are expected to be in a one-hot representation. If the labels are being provided as integers, *SparseCategoricalCrossEntropy* loss is used instead (*tf.keras.losses.CategoricalCrossentropy*, 2021).

As the ICP paradigm requires the output to be one-hot encoded, I have chosen CategoricalCrossentropy as the loss function for my Neural Network.

2.3.6 In-built metrics

In my project, I have drawn the comparison on the performance of the model by the average confidence using ICP. However, the *keras* module comes with an in-built metrics that is specified when the model is compiled or fitted using the model.compile() or model.fit() method (Team, no date b).

This metric can be used to get a rough idea about how the model is working, when the model is evaluated for some data using the model.evaluate() method. The metrics used in my Neural Network is 'accuracy'. This is specified while the model is compiled. The model is compiled using all three parameters – optimizer, loss and metrics.

2.4 Object Oriented Programming

Object-oriented programming (OOP) is a way of organising a programme by grouping together similar characteristics and activities into separate objects. It is a programming paradigm for arranging programmes so that properties and behaviours are grouped together into individual objects. Objects are similar to the components of a system in terms of concept. Consider a programme like a factory assembly line. A system component processes some material at each step of the assembly line, eventually changing raw material into a finished product. Data, such as raw or pre-processed materials at each stage on an assembly line, and behaviour, such as the action each assembly line component performs, are both contained in an object (Python, 2018).

2.4.1 Python Classes

Numbers, strings, and lists are examples of primitive data structures that are used to represent small pieces of information. A good way to manage and maintain repeating piece of code is to use classes. User-defined data structures are created using classes. Methods define the behaviours and activities that an object formed from a class can execute with its data, while classes define methods.

A class is a blueprint for how to define something. It doesn't truly have any information in it. An instance is an object that is built from a class and contains real data. An instance is similar to a form that has been completed with data. Many instances can be formed from a single class, just as many persons can fill out the same form with their own unique information (Python, 2018).

As my project involves creating the same Neural Network model repeatedly, I created a class for the Neural Network, along with the required variables that are unique to each Neural Network and methods that are used repeatedly across all neural networks.

3 Experiment

3.1 Aim

The goal of this project is to study the case when some of the data features are expensive and therefore available only for a limited amount of examples. The task is to give varied amount of the privileged information to the model to check how it affects performance and to combine this with the conformal prediction framework of reliable machine learning.

3.2 Preparatory Work

3.2.1 Exploration

As stated before, privileged information is one that is expensive or time-consuming to obtain. After understanding the concept of LUPI, I explored the existing researches of various data with the LUPI paradigm. Eventually after discussing with my supervisor, I decided to work on an image dataset to observe the effect of privileged information on data.

3.2.2 Problem in Hand

Consider hand-written digits. We observe that are multiple ways of writing the number 4 or 9, which varies from person to person. Classifying the digits into subclasses can be considered as additional information given to give reliable classification. This information about which class does every digit fall under is a time-consuming process and thus counts as privileged information for the data.

3.2.3 The MNIST Dataset

The best fit for working with images of handwritten digits was the MNIST dataset (Modified NIST). This dataset was constructed from the NIST (National Institute of Special Database) Special Database 1 and Special Database 3, which contained the binary images of handwritten digits. Further information about NIST and MNIST dataset can be found in the paper 'Gradient-based learning applied to document recognition' (LeCun *et al.*, 1998, pp. 2278–2324).

The MNIST database for handwritten digits has 60,000 training and 10,000 testing examples. The digits have been normalised and centered in a fixed-size image (CJ, 2010).

3.2.4 Python Libraries

I have used many python libraries in my project and in this section I am going to give brief descriptions about them.

3.2.4.1 Tensorflow

TensorFlow is a high-performance numerical computing open source software package. Its adaptable architecture enables computing to be deployed over a wide range of platforms (CPUs, GPUs, TPUs), from PCs to server clusters to mobile and edge devices.

It was created by researchers and engineers from Google's AI organization's Google Brain team, and it has strong support for machine learning and deep learning, as well as a

modular numerical computation core that can be used across a wide range of scientific areas (Google, 2021).

3.2.4.2 Keras

Keras is a Python-based deep learning API that runs on top of TensorFlow. It was created with the goal of allowing for quick experimentation. It's crucial to be able to go from idea to result as quickly as feasible when conducting research (Team, no date a).

Layers and models are Keras' primary data structures. The Sequential model, which is a linear stack of layers, is the most basic sort of model. The tf.keras.datasets module provide a few toy datasets (already-vectorized, in Numpy format) that can be used for debugging a model or creating simple code examples (Keras, no date). The optimizers used while training the model can be accessed using the tf.keras.optimizers.Optimizer module.

Keras, on the other hand, is a highly adaptable framework for iterating on cutting-edge research concepts. Keras adheres to the notion of progressive disclosure of complexity, making it simple to get started while also allowing it to handle arbitrarily complicated use cases with just incremental learning required at each stage.

3.2.4.3 Matplotlib.pyplot

matplotlib.pyplot is a set of routines that allow matplotlib to behave similarly to MATLAB. Each pyplot function modifies a figure in some way, such as creating a figure, a plotting area in a figure, charting certain lines in a plotting area, decorating the plot with labels, and so on (team, 2021).

I have used this library to create graphs to draw comparisons between various results in my experiment.

3.2.4.4 Sklear.preprocessing

The sklearn.preprocessing package includes a number of useful utility methods and transformer classes for converting raw feature vectors into a format that is more suited for downstream estimators (*6.3. Preprocessing data*, 2021).

Frequently, characteristics are given as categorical rather than continuous values. LabelEncoder() is a method used to convert data into one-hot encodings, which is used in this experiment.

3.2.4.5 Numpy

NumPy is the most important Python package for scientific computing. It's a Python library that includes a multidimensional array object, derived objects (such as masked arrays and matrices), and a variety of routines for performing fast array operations, such as mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and more.

The ndarray object lies at the heart of the NumPy package. Many operations are performed in compiled code for performance, and this encapsulates n-dimensional arrays of homogeneous data types.

Unlike Python lists, NumPy arrays have a fixed size when created (which can grow dynamically). When the size of an ndarray is changed, a new array is created and the old one is deleted.

The elements of a NumPy array must all be of the same data type and so have the same memory footprint. The exception is that arrays of (Python, including NumPy) objects can be used, allowing for arrays of various sizes.

NumPy arrays make it easier to do advanced mathematical and other operations on massive amounts of data. Such actions are typically performed more quickly and with less code than utilising Python's built-in sequences.

NumPy arrays are used by a growing number of scientific and mathematical Python-based packages; while they typically support Python-sequence input, they convert it to NumPy arrays before processing, and they frequently output NumPy arrays. To put it another way, knowing how to use Python's built-in sequence types isn't enough to efficiently use much (if not all) of today's scientific/mathematical Python-based software; you also need to know how to use NumPy arrays (*What is NumPy? — NumPy v1.21 Manual*, 2021).

3.3 Method

3.3.1 Step 1: Import Libraries

The very first step to begin working on the project was to import the required libraries. The following were the libraries that were used in my project:

```
#libraries
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.optimizers import Adam
import numpy as np
```

3.3.2 Step 2: Import Data

Next, the training and testing examples of MNIST database are imported from the Tensorflow datasets using the datasets.mnist.load_data() method.

3.3.3 Step 3: Filter the Required Data

Now, we only need the data for numbers '4' and '9' as that is out problem in hand. Hence, the training and test examples are filtered out with images and labels for only '4' and '9' digits.

3.3.4 Step 4: Data Pre-processing

As the MNIST dataset is already normalised and centered, I did not apply any other normalisation or standardization techniques on images.

However, the output data for ICP is required to be in an encoded format. So I created a function that encodes a list of examples using LabelEncoder() as well as the in-built function of keras to convert data into categorical encodings using keras.utils.to_categorical() method.

3.3.4.1 Open Data

The open data is the data that is available for all train, calibrate and testing examples. In this case, it is the images as well as the mapped labels.

The images for train and test samples are converted to an array of arrays, as every images is an array of pixel values ranging from 0 to 255. The dimensions of the

training and test data images were (11791, 28, 28) and (1991, 28, 28) respectively, which needed to be reshaped to fit the data to the CNN model. So I used the code given in (*Reshaping Data*, 2020). The resulting dimensions for train and test datasets were (11791, 28, 28, 1) and (1991, 28, 28, 1) respectively.

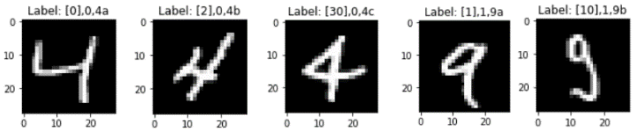
The labels are converted to string (as we want to classify them into two categories – '4' or '9') and then encoded using the function previously defined. The resulting data for train and test data looked as following:

```
y_train_data
✓ 0.5s
array([[1., 0.],
       [0., 1.],
       [1., 0.],
       ...,
       [1., 0.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

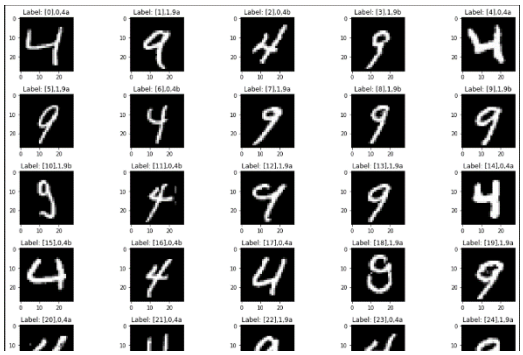
```
y_test_data
✓ 0.9s
array([[1., 0.],
       [1., 0.],
       [0., 1.],
       ...,
       [1., 0.],
       [0., 1.],
       [1., 0.]], dtype=float32)
```

3.3.4.2 Hidden Data

The hidden data is the addition/privileged data that is to be supplied for limited/all training set example. For this particular problem, the hidden data is which class does the image fall under among ['4a', '4b', '4c', '9a', '9b']. The following images show what each one represents:



I assigned the classes manually after visualising every data (for 1000 training examples). The visualisation was done using pyplot.subplot() function, which made the process easy. Below is the snippet preview of the visualisation:

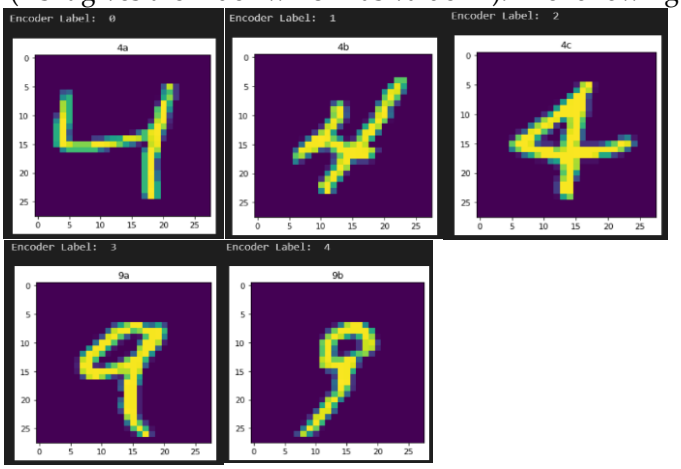


This data is then encoded as well, using the function defined previously, into five classes. The resulting data looks as following.

```
priv_info
✓ 0.4s
array([[1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0.]], dtype=float32)
```

3.3.5 Step 5: Visualising Individual Data

Here, I created a function that displays image at a particular index to visualise it. This is to help check any anomalies, logical errors etc. while debugging the code. I have used the function here to check the encoder label for each class, which I found out using np.argmax() function (As it gives the index which has value '1'). The following is the result of that:



3.3.6 Step 6: Defining Class for Neural Network

This is the most important part of the project. Here, I have defined a class, inside which are all the variables and methods that are required to work with our data (*9. Classes — Python 3.10.1 documentation*, no date)

The class variables that I defined are *ncm_h*, *pVal_h*, *confidence_h* and *credibility_h* for the performance measures of the prediction of privileged data as well as *ncm_y*, *pVal_y*, *confidence_y* and *credibility_y* for the performance measures of the y label predictions. I set these variables so, as these values are unique to every model that I train and because they are used to draw comparison, conclusion etc.

Further, I have defined several methods in this class, each of which is defined below.

3.3.6.1 __init_()

The __init__() method is invoked automatically at class instantiation. So I defined the Neural Network model here. The __init__() function takes output_size as a function parameter because the output size may vary in the project for training with and without privileged data. I have created a CNN model using keras Sequential model.

This model has an input layer with 25 filters, kernel size (3, 3), a RELU activation and an input shape (28, 28, 1). The layer is given MaxPooling2D((2, 2)).

Then I added an input layer to the model with 64 filters, kernel size (3, 3) and a RELU activation. The layer is given MaxPooling2D((2, 2)).

After that, I have used Flatten() to reshape the output, which goes into a dense layer with 64 nodes and a RELU activation.

Finally, there is a output layer that has the output_size which is passes while initialising the class object, and a softmax activation.

This method also initialises the class variables stated above to empty lists, which are updated further using other methods.

3.3.6.2 train_model()

This method runs the piece of code to train the above defined model.

Before training, we need to define the optimizer for the model. I have chosen the *Adam* optimizer for my model, with a learning rate 0.001.

Next, the model is compiled with the defined optimizer, with 'categorical-crossentropy' loss, and the metrics is 'accuracy'.

Finally, the model is trained using the model.fit() method for the number of epochs specified while calling the function. The image data (x) and the respected labels (y) are also passed through the function when it is called.

3.3.6.3 eval_model()

This method is used to invoke the inbuilt method model.evaluate() to evaluate the predictions of the model for a particular set of input and output examples. It simply returns the loss and accuracy of the prediction.

3.3.6.4 pred_model()

This method predicts the labels for new/test data using the model that we trained previously. It returns the predicted labels as a list.

3.3.6.5 calc_ncs()

This method is to calculate the nonconformity score of the calibration and test examples. It takes the predicted and actual values as arguments of the function. It calculates the alpha value for the prediction using the algorithm stated in the previous sections and stores that value in the respective class variable.

3.3.6.6 calc_pval()

This function takes the nonconformity scores of the calibration and test sets and return the p-value after calculating it using the formula shown in the previous sections. The calculated p-value is stored in the respected class variable.

3.3.6.7 calc_score()

This function calculates the confidence as well as credibility of the predictions. It takes the calculated nonconformity scores as well as the p-values of the labels, calculates the confidence and credibility of the prediction using the formula given in the previous section and returns the values, which is then stored in the respected class variables explicitly.

3.3.6.8 ICP()

This is the methods that runs the ICP algorithm for my experiment. It takes predicted test label, predicted calibration label, actual calibration labels as well as a list of possible class predictions as function arguments.

The nonconformity scores for the calibration set are calculated. Then, the nonconformity scores and p-values for every test example is calculated by following the steps of the ICP algorithm specified in the previous sections.

This calculates values of nonconformity measure as well as p-values are returned by the function. The values are stored in the respective class variables explicitly while the method is called.

3.3.6.9 get_y_labels()

This method is created to return the mapped y labels of the input data after the privileged information is predicted. It takes the list of predicted privileged information, finds the predicted label using np.argmax() function and then uses if-else blocks to map the correct y labels. It then returns a list of y labels.

3.3.6.10 get_h_labels()

This method is created to get the labels of the privileged information after they are predicted. The predictions are an output of the softmax function, and need to be converted into original labels ('4a' etc.). The function takes the predicted h values as an argument, finds the predicted class using np.agrmax() function and uses if-elif-else blocks to assign the correct label. It returns a list of predicted h labels (privileged information) in the end.

3.3.7 Step 7: Training Without Privileged Information

In this section, I have trained the data on a model that predicts only two classes – '4' and '9', that is, without the privileged information.

First, I create an instance of the class described in the previous step. Then I train the model on the proper training set for 5 epochs. Next, I predict the calibration and test labels.

Further, I calculate the nonconformity measure and p-values for these predictions, using which I call the calc_score() function to store the performance metric.

3.3.8 Step 6: Training with Privileged Information

To reduce the lines of code, in this section I have defined a function called parial_priv() that predicts the labels for varying amounts of privileged information provided to it. It takes an object of the class myNN() as well as amount of privileged information as function arguments. It then trains the model with the amount of privileged data provided. It then predicts the privileged data for the rest of the training examples.

The get_h_labels function is called and the h labels are retrieved for the predictions. The previously provided h values and new predictions are them combined into one, using the code given in ('Ways to concatenate two lists in Python', 2018), and the entire proper training input data is trained with this new privileged information.

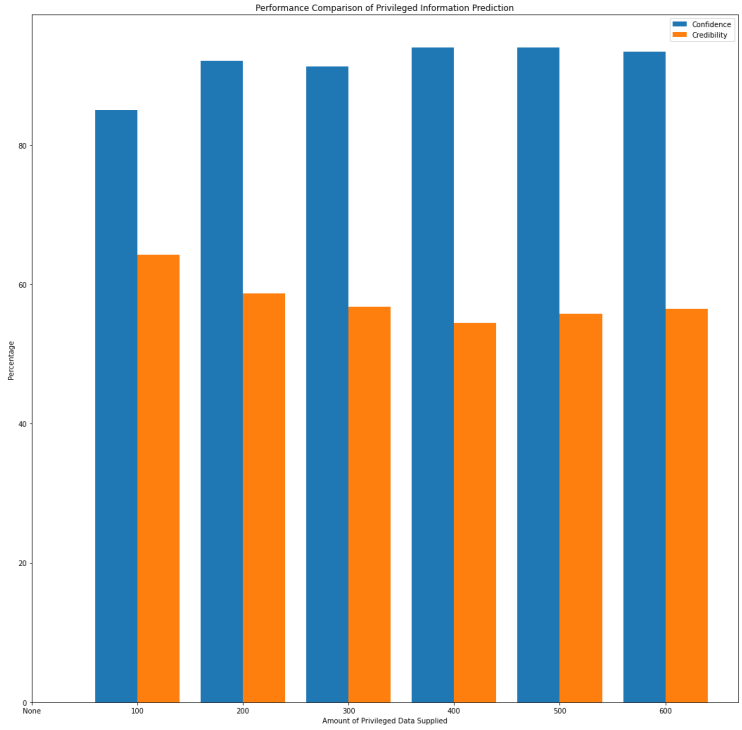
Next, the privileged information for the calibration and test set is predicted using the model we just trained. After this, the nonconformity measures, p-values, confidence and credibility is calculated for the test examples and stored in the respective class variables. The y labels are also predicted using the get_y_labels() assignment function and the performance measures for the prediction is calculated and stored in the respective class variables.

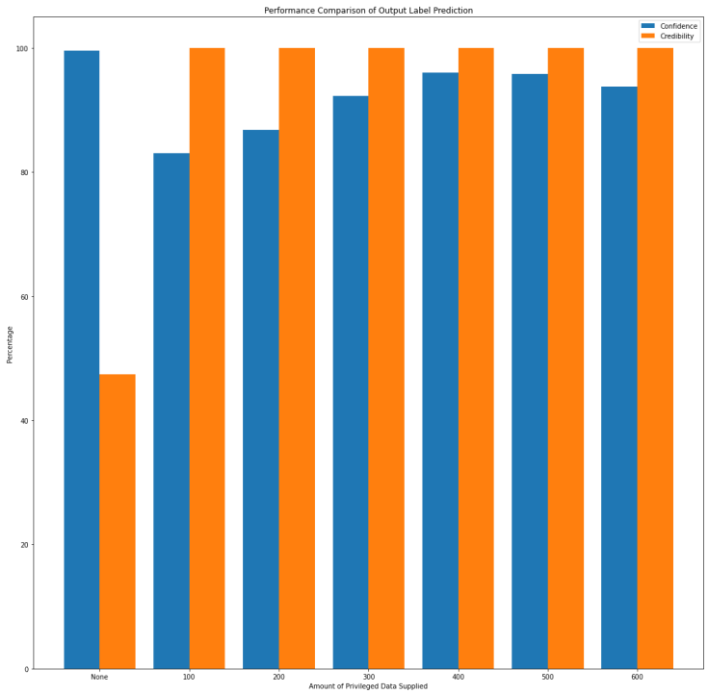
This parial_priv() function is called for privileged information intervals 100, 100, 300, 400 and 500. The model where all privileged information is supplied is created separately using the same steps (excluding the prediction for privileged information).

3.4 Results

To get a comparison between all the performance measures, I first stores all the class instances in a list *allNNs*. Next, I stored the amount of privileged data provided to the network as a list of labels to a variable *labels*.

For the performance of privileged Information prediction, I averaged all the confidence and credibility for each model and stored it in lists, which I then plotted on the bar graphs using the code given in ('Plotting multiple bar charts using Matplotlib in Python', 2021) The following are the results for the same:





3.5 Conclusions

The figures clearly show that, even though the confidence of prediction was very high, the credibility was severely poor when there was no additional data provided. This is in relation to the predicted h labels' credibility, not the y labels' credibility, because y labels' credibility is always 1. This is because when training with privileged data, the y labels are simply assigned rather than predicted in this case.

When it comes to the h label prediction comparison, we can see that the confidence grows as the amount of privileged information provided to the data grows. The model, on the other hand, works admirably even when only half of the new data is provided.

We can deduct from this that we can get credible results for any dataset if we offer additional/privileged information for at least half of the training set data. When compared to adding extra data to the model, this saves half the time and money.

It also supports Vapnik's LUPI paradigm, demonstrating that supplying additional information for a small number of samples, like an intelligent Teacher, is sufficient for reliable classification.

3.6 Possible Extensions to the Experiment

A possible expansion of this experiment would be to choose which data to use as a learning element for privileged knowledge.

The model is trained using only the training set, which is separated into train, calibrate, and test sets, and then ICP is used to evaluate it.

The instances are then chosen using various criteria such as lowest nonconformity, largest nonconformity, lowest believability, and so on.

4 Self-Assessment

4.1 Strengths

Coding Knowledge: My biggest strength during the course of this project was my strong knowledge about python syntax, libraries, error handling etc. This helped me understand all the errors I faced, talk it with my supervisor and solve them accordingly. It also helped me to get an organised code with minimal redundancy.

Correct tools: I have used various tools for the sake of this project, the very first being Visual Code. This helped me be organised and maintain my project with ease. I run Jupyter Notebook through Visual Code and all my snippets are from it. Fore referencing, I used the RefWorks extension on browser and in Word, which made the process so much easier.

Research Skills: I explored a lot for every concept, error or doubt, through various resources, that gave me a better grasp over the topic in hand and made the process of the experiment far less stressful that it could have been.

4.2 Weaknesses

My main weakness would be to not taking into account the extent of the project. Next time, I would consider this and manage my time better so as to explore more. My work-load became bottom-heavy this time as I did not plan things well.

4.3 Opportunities

Due to this project, I came across a lot of researches and papers that attempted to bringing real world problem into machine learning for a solution. Talking about LUPI, I was amazed to read how the idea of teacher-student relationship was formulated into a machine learning paradigm. This gives a lot of inspiration towards research in this field. I was also fortunate enough to work this project under the guidance of Dr Nouretdinov, who I found out has contributed significantly in this field. His explanations were very helpful to get a clear picture on the topic.

4.4 Threats

Human Error: As I gave the privileged information manually, the label assignment can be argued at times. It also gives room to human error, which I tried my best to keep to minimal.

No Reshuffling: Due to time constraints, I did not randomly reshuffle the data. Ideally, I would do that and averaged the results over the random reshuffles to get a more accurate result.

5 Appendix

Here, I have inserted the summary of the trained Neural Networks. This is done with the help of model.summary() method.

```
nn0.model.summary()
✓ 0.5s
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 64)                102464
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 117,308
Trainable params: 117,308
Non-trainable params: 0
_________________________________________________________________
```

```
nn1.model.summary()
✓ 0.1s
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_17 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

```
nn2.model.summary()
```
✓ 0.3s

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_19 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

```
nn3.model.summary()
```
✓ 0.5s

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_20 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_21 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

```
nn4.model.summary()
```
✓ 0.4s

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_11 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_22 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_23 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

```
nn5.model.summary()
```
✓ 0.4s

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_24 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_25 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

```
nn6.model.summary()
```

✓ 0.3s

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 26, 26, 25)        250
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 13, 13, 25)        0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 11, 11, 64)        14464
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_14 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_28 (Dense)             (None, 64)                102464
_________________________________________________________________
dense_29 (Dense)             (None, 5)                 325
=================================================================
Total params: 117,503
Trainable params: 117,503
Non-trainable params: 0
_________________________________________________________________
```

6 Bibliography

6.3. Preprocessing data. (2021) Available at: https://scikit-learn/stable/modules/preprocessing.html (Accessed: .

'Plotting multiple bar charts using Matplotlib in Python', (2021) *GeeksforGeeks*, -02-23T00:53:50+00:00. Available at: https://www.geeksforgeeks.org/plotting-multiple-bar-charts-using-matplotlib-in-python/ (Accessed: .

tf.keras.losses.CategoricalCrossentropy. (2021) Available at: https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy (Accessed: .

What is NumPy? — NumPy v1.21 Manual. (2021) Available at: https://numpy.org/doc/stable/user/whatisnumpy.html (Accessed: .

Reshaping Data. (2020) Available at: https://datascience.stackexchange.com/questions/64072/input-0-is-incompatible-with-layer-conv2d-2-expected-ndim-4-found-ndim-3-i-get/75093 (Accessed: .

'Ways to concatenate two lists in Python', (2018) *GeeksforGeeks*, -11-21T17:37:06+00:00. Available at: https://www.geeksforgeeks.org/python-ways-to-concatenate-two-lists/ (Accessed: .

9. Classes — Python 3.10.1 documentation. Available at: https://docs.python.org/3/tutorial/classes.html (Accessed: .

Albawi, S., Mohammed, T.A. and Al-Zawi, S. (2017) *Understanding of a convolutional neural network.* IEEE, pp. 1.

Allibhai, E. (2018) *Building a Convolutional Neural Network (CNN) in Keras.* Available at: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 (Accessed: .

Brownlee, J. (2019) 'A Gentle Introduction to Pooling Layers for Convolutional Neural Networks', *Machine Learning Mastery*, -04-21T19:00:50+00:00. Available at: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (Accessed: .

Doshi, S. (2020) *Various Optimization Algorithms For Training Neural Network.* Available at: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 (Accessed: .

Google (2021) *tensorflow 2.7.0 (abstract)*.

LeCun, Yann and Cortes, Corinna and Burges, CJ (2010) *MNIST handwritten digit database.* Available at: https://www.tensorflow.org/datasets/catalog/mnist (Accessed: .

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, 86(11), pp. 2278-2324.

Matplotlib development team (2021) *Pyplot tutorial.* Available at: https://matplotlib.org/stable/tutorials/introductory/pyplot.html (Accessed: .

Noretdinov, I., Melluish, T. and Vovk, V. (2001) 'Ridge Regression Confidence Machine - Research - Royal Holloway, University of London', , pp. 385-392.

Papadopolous, H., Vovk, V. and Gammerman, A. (2002) *Qualified predictions for large data sets in the case of pattern recognition.* CSREA Press, pp. 159.

Papadopoulos, H., Proedrou, K., Vovk, V. and Gammerman, A. (2002) *Inductive Confidence Machines for Regression*, Berlin, Heidelberg: Berlin, Heidelberg: Springer Berlin Heidelberg.

Papadopoulos, H., Vovk, V. and Gammerman, A. (2007) *Conformal Prediction with Neural Networks.* pp. 388.

Python, R. (2018) *Object-Oriented Programming (OOP) in Python 3 – Real Python.* Available at: https://realpython.com/python3-object-oriented-programming/ (Accessed: .

Saha, S. (2018) *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: .

SHARMA, S. (2021) *Activation Functions in Neural Networks.* Available at: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Accessed: .

Team Keras, *Keras documentation: Datasets.* Available at: https://keras.io/api/datasets/ (Accessed: .

Team, K. (2020) *Keras documentation: The Sequential model.* Available at: https://keras.io/guides/sequential_model/ (Accessed: .

Team, K. (a) *Keras documentation: About Keras.* Available at: https://keras.io/about/ (Accessed: .

Team, K. (b) *Keras documentation: Metrics.* Available at: https://keras.io/api/metrics/ (Accessed: .

Vapnik, V. and Izmailov, R. (2015) 'Learning using privileged information: Similarity control and knowledge transfer', *Journal of machine learning research,* 16, pp. 2023-2049.

Vovk, V. (2005) *Algorithmic learning in a random world.* New York; New York, N.Y.; Great Britain]: New York : Springer.