Recording the Lifecycle of Software Models

Brendan P. Bonner
School of Computing
Dublin City University
Dublin, Ireland
brendan.bonner2@mail.dcu.ie

***Abstract*—Artificial Intelligence is built on trustworthiness of the systems that provide the intelligence. The foundation of this is built upon our understanding of the role of software models in this process. Similar to measuring social trust in humans, we can only increase our levels of trust by being exposed to the full competence and vulnerabilities of the environment to get to the point in time where trust can be assessed. The pathway for the human brain to grow and evolve is recorded in genealogy, education and experience. In this paper, we investigate feasibility and benefits for replicating this process for machine learning systems. During this process, we create an ecosystem where AI systems can have their entire lifecycle recorded and made available for scrutiny. The framework for a secure chain of provenance of the lifecycle of artificial intelligence systems may be a first step into addressing shortcomings in AI research which potentially could stymie innovation for certain critical systems from being deployed. The outcome of the research cannot coerce trust, although it provides additional insight into comparing aspects of a software models *personality*, without needed to directly examine every neuron.**

***Index Terms*—Artificial Intelligence, Provenance, Explainability, Model Analysis**

I. Introduction

Artificial Intelligence has a trust issue. In all areas related to the field of machine learning, the absence of transparency perceived within artificial intelligence solutions across domains highlight a lack of ethical oversight and visibility. This has led to initiatives such as the EU white paper on Trustworthy AI[1], which has initiated actions across the complete software ecosystem to put forward trust measures needed for AI to be adapted (i.e. Ryan[2]), and far reaching legislation in the forthcoming EU AI Act.

The most difficult component in the trust ecosystem to ascertain, is the AI technology itself.

Artificial Intelligence research attributes this to the *black box* problem[3], where the complexity of AI Neural Network systems can no longer be analysed thoroughly in a manner similar to statistical systems. Numerous attempts have been made to look inside the layers of the neural network to see what neurons are being activated[4], and visualising where the attention is focused prior to making a prediction, but so far no provision has been made to identify how to *trust* the solution using AI in a method closer to human trust levels.

To understand how to achieve this trust, we need to equate parallels or a person visits a doctor, boards a bus or plane, or ask for financial advice; they develop trust due to a core evolutionary biology unique to the user, or they can access their credentials unique to the provider that provide *trust* they have competence to provide the service. We can look up their professional and academic qualifications, review their associative history, and use our reasoning that these provide a basis for a provenance of trust.

It is with this analogy that we will investigate if we can establish a similar provenance of trust for AI systems. The aim is straightforward; can we create an architecture to support a *Linkedin.com* for software models that we can use to evaluate the creation, initial seeding, and then layers of training needed to validate the trust of AI systems. If we can look at the CV of a software model, we should be able to show which base model the system was built on and the outcome that training (i.e. the majority of core vision applications are build using Keras models trained on ImageNet[5]), the the influence it has on the development of the model. This *Practicum* will research how the addition of provenance to software models, the *brain* of Artificial Intelligence, could be a solution to improving human levels of trust[6] to machine learning.

We developed a system after examining current explainable AI (XAI) methods to visualise and identify inside an AI model. Subsequent practical application created a system for extracting a model synopsis utilising a reduction algorithm applied to each layer across a model. The reduction of the weights and biases to a pair of unique identifiers containing the *standard deviation* and the *variance*, is then stored in a development and release database. Once this synopsis was developed, we were able to reduce all default Keras models from several megabytes to several kilobyte arrays that are comparable, and the basis for generation of a unique signature. The signature is a unique identifier within software development teams and upon release of a software model, this will create a single record from which we can identify ancestors, descendants and quickly show the delta between every relation.

The overall objectives of the *Practicum* can be broken down into examining the state of the art, and then researching methods of generating an open recording solution for capturing AI model development and deployment. We completed this by researching the following fields:

- Discover a method to extract a unique signature from any Deep Neural Network software model.
- Create a local and global verification trail for a path of provenance to all ancestor models
- Create a local and global verification trail for a method to evaluate divergence of the child model from the parent
- Validation of Provenance via Verification and Illustration

Using a package of tools developed to deliver these objects, we are able to create a repository of signatures and model impressions, built upon an immutable link to it parent. The capacity of generalising the lineage for both training and validating will be the basis for further research into exploration of a history of the model's evolution.

II. Background

The central tenet of the *Practicum* is how to determine a methodology of trust in a digital world, where there is widely reported scepticism. The need and demand for evaluating a methodology for efficiently recording the learning process of an AI model will be researched, focusing on convolutional neural networks. The approach will establish if existing CNN explainability methods can be used to define a novel temporal chain of major differences in the underlying structure of the network during training.

Once we have determined what elements of value in the process show what can be recorded, it will create a verification repository for the network (similar to a CV) and show how this can be used as either a trust chain when establishing the provenance of the model, reviewing the network training in comparison with the post-network visualisation, or providing a mechanism to see if the training process is either optimal or biased.

The research area overlaps between the current drive to explain black box models to address trust issues with artificial intelligence[2], alongside the drive to be able to optimise and evaluate the process for creating and expanding existing models. The bulk of current research focuses on the behaviour of neural networks *post*-training, examining how an input triggers various layers. This functionality of this is reduced as deeper networks become more commonplace. Starting to record and validate how networks are trained, rather than utilised may provide additional support towards trustworthy AI as in He *et al*'s paper comparing training[7], while benefiting training evaluations with visualisation of the gravity of neurons are activated during the learning process.

Models in CNNs are the primary reason for opacity of neural networks. Decision trees and algorithms have the benefit of being easily repeatable outside of the system. The complexity of even the simplest models makes this unviable. The introduction usually describes the background of the project with brief information on general knowledge of the subject. This sets the scene by stating the problem being tackled and what the aims of the project are.

A. A Question of Trust

In his overview of ethics in AI, Ryan [2] identified that from a high level, trust is straightforward - artificial systems should **not** be trusted - the background to this is the criteria for trust has not been met, namely that we cannot be vulnerable to these systems, the system thinks well in others and be optimistic that the system is competent [6]. Some machine learning techniques, although very successful from the accuracy point of view, are very opaque in terms of understanding how they make decisions. The notion of black-box AI refers to such scenarios, where it is not possible to trace back to the reason for certain decisions. Explainability is a property of those AI systems that instead can provide a form of explanation for their actions, without fully understanding how the system came to this position.

B. Linking Software Models with Trust

Software models in deep neural networks are the basis for the fuzzy nature of a certain outcome based on an input. As the input changes, the impact on weights, balances and activation functions in trained models can provide significant changes in outcomes, whether we a are classifying the image, identifying components or using the outcome as the basis for generating an outcome[8]. In frameworks like Keras and Pytorch, the Model contains roughly the same structure, based on an input layer, a number of hidden layers, and finally a classification or output layer. As a model is trained, the input layer and all not weighted layers remain the same, but the back propagation function continuously changes the weights and biases across multiple layers with diverse interlinks[9].

C. Identifying Differences in Models

Each layer is made up of a structure containing a dictionary of details, plus a multi dimensional array containing the weights for each sublayer that covers all dimensions, plus a series of biases. While trying to understanding the internals of layers is interesting but relatively futile, being able to break down the structure of the layer into a unique single identifier that is non-repeatable across the smallest changes, but also allows comparison between instances of the layers when trained.

What is not provided for is the need and demand for evaluating the development & methodology for efficiently recording the learning process of an AI model, focusing on convolutional neural networks. Can current CNN explainability methods highlight a temporal chain of major differences in the underlying structure of the network during training. Existing research into provenance of electronic records is within the domain of blockchain technology, usually in a transactional process[10]. create a verification repository for the network (similar to a CV) and show how this can be used as either a trust chain when for establishing the provenance of the model, reviewing the network training in comparison with the post-network visualisation, or providing a mechanism to see if the training process is either optimal or biased.

The research area overlaps between the current drive to explain black box models to address trust issues with artificial intelligence, with the drive to be able to optimise and evaluate the process for creating and expanding existing models. The bulk of current research focuses on the behaviour of neural networks after training, examining how an input triggers various layers, and this has reduced as deeper networks become more commonplace. Starting to record and validate how networks are trained, rather than used may provide additional support to trustworthy AI, while benefiting training evaluations and visualisation of when neurons are activated during the learning process.

D. Current Research into Trustworthy AI

The current state of the art in Artificial Intelligence is veering towards a systemic classification of risk levels for AI systems. This is popularised by both practitioners from within, and also regulatory and political channels.

In the process of understanding and trusting deep neural networks, researchers have veered towards extensive evaluation to identify neural pathway activation, or by presenting input triggers and looking at activations between layers[11]. There is a difference in approach regarding what has generated utility in explainability - allowing multiple metrics to cover the effect of individual features, morphology characteristics or how pixels resonate various components of the network. The focus is to understand the network, rather than understand how it is trained. The closest research in understanding the training process is the use of classifier probes, and how these can be recorded in a provenance chain.

The first step in our journey to evaluating the trustworthiness of AI systems is examining how fairness and bias is interpreted across all artificial intelligence systems. In AI fairness: how to measure and reduce unwanted bias in machine learning, Mahoney, et al, examine each phase of the AI deployment process to identify where bias is created, identified and countered. The resultant research created the IBM AI Fairness (AIF360) toolkit[12], which looks at all network types and rewires the network weights and biases to remove predefined bias from the outcome. This works well on business AI systems classifiers like random forest and naive bayes, and simple neural networks, but is highly limited in deep neural networks. The research does identify one important gap - there is no mechanism to alter a trained network, and expect it to behave as before. Part of the research will provide a prevalence trace to verify that the network being examined is mathematically verified to not be tampered with, potentially increasing the trust in deployments.

The above paper looked at basic networks, the journey is more difficult for convolutional neural networks due to the increasing depth of networks. A popular paper on understanding the decision process within ConvNets was discussed in the 2014 paper Deep Inside Convolutional Networks [13] where the innards of a CNN when presented with an image to be classified was visualised using saliency maps. The method of saliency extraction with a view to presenting a single 2D image map gave a "last layer" backwards view of the network activations, as if you were looking through a telescope, while imagining the neurons. This research highlighted that aggregation of individual neuron activations can be utilised for explaining CNNs. What we would like to see is how to twist this view around, and instead of seeing the activations of neurons, instead see the neurons being trained.

The final piece in the jigsaw is attempting to use this understanding to make a score of interpretability and the experiments of breaking down the components of CNNs are discussed in Network Dissection: Quantifying Interpretability of Deep Visual Representations[14]. In this 2017 paper, Bau et al, introduces the concept of scoring unit interpretability to give a segmentation threshold to an individual concept when the CNN is classifying. The attempt is to visualise and score diverse CNNs to evaluate discrimination and interpretability. The outcome of this research will be to examine if there is a link between the number of detectors of concepts in the outcome tests to the number of times a certain portion of the network is trained to classify or ignore that concept.

E. Interpretation of the Learning Process

Once we understand that there is a gap in being able to visualise the learning process when training each convolutional neural pathway in an AI system, we then have to examine the state of the art in being to interpret this training process. In this part, research is quite light, as the focus has been on understanding a network when it is making a classification or using evaluation to interpret differences in classification. The approach that I will take is to examine how individual elements of each layer are tweaked during the training process. While a brute force approach would be to record the loss function changes in back-propagation for every trained step, this would result in a compute requirement of 12m parameter changes each step, which would require multi-terabyte storage for even a competition trained VGG-16 network. The approach has to be optimal to record only changes in neurons that are beyond an average activation threshold, and then only in a constant dimension depth per layer. The resulting delta will be configured everyX steps or epochs to provide a consistent path of evolution during training.

In assessing the Understanding intermediate layers using linear classifier probes [15] , we finally get an appreciation of the value of examining the learning process as opposed to the post-trained model activations. Although still concerned with feature activation, it introduces a new method that sits in parallel to the traditional neural network library of tools. In this case, monitors of features at every layer of a model are evaluates suitability for classification. The approach is to have completely independent linear classifiers, which is needed for delivery of this theme. This was applied to two large models in Inception v3 and Resnet-50, and the resulting analysis of similar networks helped us focus on high performing and large networks like VGG-16. The issue with this research in line with the theme is the focus is on interoperability, the key outcome is how linear separability of features change throughout the model, although it provides the best research available to optimise the training for visualisation.

A further exploration of the internals of a CNN, and the introduction of identifying how low level concepts interact with high level features is put forward as a proposal for Quantitative Testing with Concept Activation Vectors[4]. In this 2018 paper, Been Kim, et al, agree that explainability remains a significant challenge in the AI world, but is still important to attempt to de-opaque to make sure that a value based machine learning world is possible. While reference is paid to Alain and Bengio's previous 2016 findings, this goes beyond and uses the CAVs for interpretability for testing. In particular, we can rework the proposed extensions to TCAVs by seeing if we can establish if the learning pattern grows with the shape, texture of repetition of the training images.

The findings were also used to develop artistic style generative images based on the test outcomes. While we will not attempt to replicate the Empirical Deepdream, visualisation of the outcomes, similar to the Simonyon outcomes above should be available. While aesthetic to view, the outcome remains dependant on their inputs.

F. Visualisation of Convolutional Network Values

A key part of the research necessary to develop interpretability is provided by traditional visualisation techniques. Complex 3D imaging would be required due to the need to see temporal differences in the network over time, and there are competencies in providing this.

A close examination of visualisation techniques push back to the development Of the open Stuttgart Neural Network Simulator (SNNS), which allowed the early definition and visualisation (based on X11 graphics) to get the size and scale of the demonstrations of the largest A good example of research is Adam Harley's visualisation tool[16] that shows the full CNN complete with all interconnections, and weighted values that are activated on (in this case MNIST) data and a hand drawn input This is based on previous work, such as the stuttgart Neural Network Simulator (SNNS).

G. Future of Trustworthy AI research

While there has been continuous research and improvement into the field of explainability, there remains a disconnect between the understanding of why a neural network produces a specific outcome – of which research is plentiful; and a lack of investigation into establishing a trust quotient to provide verifiable classifications based on either its initial training or amendments provided after transfer.

The benefit of the research is there is a clear metric and methodology to verify what components of a CNN are activated by a source image, and we can do this not only in an interesting way that can show how far down a deep network the feature or concept is important, but also to guide optimisations. CNN visualisation is also strong, but based on it's use not how it was trained.

What is lacking is a guidance as to why the training of the network led to this behaviour. If we create an analogy of a human being required to make a classification, we look at their background, their education, their environment, the acquired competence – nearly all of this is used to build a body of evidence that this experience can be trusted to make a decision or classification. Like lifelong learning being recorded in a verifiable resumé, CNNs are trained on a repository of either a single domain, or pre-trained using a generic corpus of data, and while the outcomes may be similar, an anomaly in training or a tweak afterwards will yield untrustworthy results, which can be evaluated only extensive measurements.

The missing link is being able to see the training process, similar to recording inputs that a human will see from birth to deployment in the workplace. While we do not need to see everything, we would benefit from seeing the important triggers over time that form and tweak the neuron. This is the aim - to record and store the life-log of the network, to see what parts of the training process were not necessary and to optimise before an exponentially wasteful under-fitting of pathways becomes an unnecessary overhead in deployment. What is also missing is being able to baseline this model alongside this record, which would give a provenance. In the event of the next generation of AIF360 manipulating bias for any rational like Giskard from Asimov's Robots of Dawn, this would show if that manipulation was performed, and If the model can be trusted.

How we get to the next step is to implement a system of measuring the delta within a tolerance at multiple steps over the training. This can be established during CNN training in any of the available CNN frameworks, which also include predefined transfer models to identify the difference in a 'big bang' early training to additions to pre-trained systems, which will be quieter, but identify the domain impact on existing models. The visualisation will be challenging, as we will require the model to be visualised in 3D, as 2D models will hide the detail we are hoping to be exposed like a nebula in front of a galaxy. Existing visualisation techniques, and abstraction of the visualisation to multiple 2D images, similar to GradCAM, will help identify the initial benefit of the data produced by the measurement process.

III. Method

The *Practicum* was divided into three sub areas, focusing on dissection of the model and extrapolating the summary that would be used to robustly identify the internal structure of the model to enable the model synopsis to be stored in in a repository. The second section was to generate a unique signature for the model that is verified with minor and major changes to both the weights and biases, but also to quickly identify the model structure. The third part was discovering the best method to store they model summaries, so that they can be reused, pulled back and compared with the model currently in development, training or deployment.

The final method investigated in the *Practicum* was application. On top of the original aim of determining the evolutionary tree of the model, other applications such as identifying difference between models that are in the process of being trained, as well as identifying portions of the training or deployment that would assist other methods of observability for model development, trust determination and internal visualisation.

A. Developing a Robust Model Synopsis

The first part of the delivery required that a model, in this case a CNN model in Keras, can be reduced to an information block that each layer contains a summary of the layer that can record any changes in the weights and biases. The structure of a model consists of a number of standard information points that are consistent across all deep learning processes. While we initially focused on CNNs, it became clear that any model could be used. As shown in the architecture diagram1 showing for the popular VGG16 Network[13], A network consists of multiple hidden layers representing the black-box of artificial intelligence.

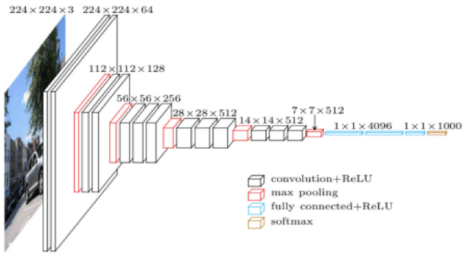

Fig. 1. VGG16 Network Architecture

The structure of the neural network, while complex, is nearly always of the same format across all recent improvements; not only for convolutional neural networks, but for any type of machine learning application using a modern framework. Even with recurring and shrinking/expanding models, the formula is relatively consistent, as identified below:

1) An Input Shape. In the case of vision applications, this consists of an image with width, height and colour depth, which is usually three.
2) Dense Application Layers, such as Pooling, Convolution or reduction layers which can contain a heavily connected series of filters, activation functions and weight and bias metrics.
3) Output layers. One or a number of prediction or output layers. In transfer learning, the output layers can change based on the prediction expanding or contracting.

To be able to break down the model, it is necessary to examine the layers for patterns that could be unique or representative. Within each of the layers, there are is a series of sublayers that in the majority of cases are a series of tuples that have a single or multi-dimensional weights, plus optional biases. Theoretically, these values can contain any number, while on inspection, the majority are bound number series that can be tuned up or down in very small increments. It is within these layers that we will extract the layer identification information. For visualisation, the filters do not show information by themselves. For instance the following figure 2, shows the first layer of a VGG16 network[17], containing *3x3x3x64* (or 1734) weights.

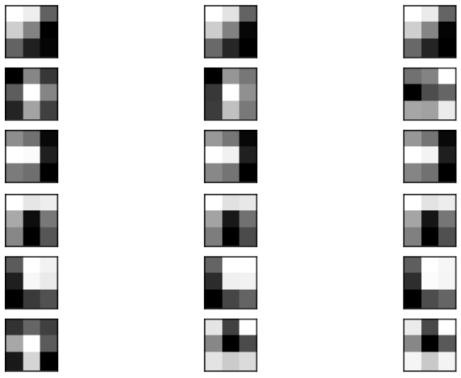

Fig. 2. VGG16 First Layer Weights Visualisation

This highlights the importance of not breaking down the layer to each individual layer dimension, as the complexity of the layer gets more complex as the numbers become larger. In the case of our abstraction, we will break down the layer for all values across all dimensions; one for weights, one for bias. To achieve this, we examined a number of mathematical options to identify the data underneath. It was important at this stage to focus on the minimal amount of information that would enable a unique signature to be generated.

Experiments were completed to give an indication of distribution of data, which would assist in visualising the direction of training, in the form of layer-wide histograms or each dimension, and producing a *histogram of gradients*-style overview. These were reduced in favour of two elements - Standard Deviation and Skew. In both cases experiments were concluded to establish if training a complete end to end network had an effect on the output values that would require a lower granularity of aggregations. The Standard Deviation formula used was shown below:

$$\frac{1}{N}\sum_{i=1}^{n}\left(x^i-\overline{x}\right)^2 \tag{1}$$

This formula provided a robust mixture that gave of deviation of the smallest change in a 64 bit float, the most commonly used weight and bias internal type used, while still retaining a characteristic was the standards deviation. It also highlighted that if a set of convolutional filter moved all looped in any direction, the standard deviation would remain identical. When filters are used for feature extraction, there is a small potential that a model could be retrained and retain the same standard deviation across all layers.

To counter this we introduced a measure of *skewness* to the layer to be recorded. The Pearson's coefficient measure established the relationship between the mode and the standard deviation and gives an indication of the direction *every* combined filter in a layer is veering. As before, unless the feature is two dimensional array, this is not an indication of how the filter performs against an visual input, but instead a protection of the movement of variables from one direction to another. The formula used for the Skew of N samples is shown below.

$$G_1=\frac{k_3}{k_2^{3/2}}=\frac{\sqrt{N(N-1)}}{N-2}\frac{m_3}{m_2^{3/2}}. \tag{2}$$

Of all the averaging functions available, the implementation of Standard Deviation in the python numpy library was able to reduce every single element to a single 64bit float. A second indication for skew was used to identify the drift of the standard deviation direction, or lack of symmetry, and is included to ensure that movement of elements that retain the same Standard Deviation will not break the solution.

Trial and Error showed that this was sufficient in training exercises to provide more information. An alternative approach was to develop a histogram of normalised values. The normalisation of the values again would reduce the level of change being recorded in circumstances where the correction of the weights during the learning process is distributed across early feature maps. Binning of distributions, either per dimension

can potentially be investigate further, but for this project, the *StdDev* and *Skew* provide enough resolution to work.

To show the difference between this approach, and that shown in other examples for visualisation in the background references, we take a number of early layer in commonly used machine learning models and compare the visualisation with the layer outcomes. For the initial models that were examined, we could tell that the filters behaved very differently between the first convolution of both ResNet50 and VGG16, as shown below:

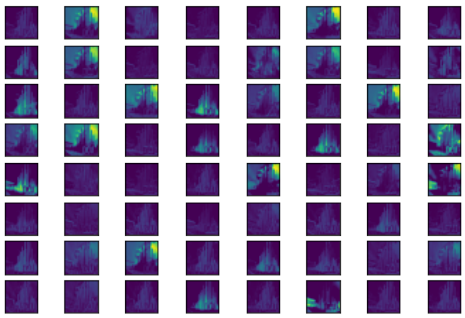

Fig. 3. VGG16 First Layer Image Visualisation

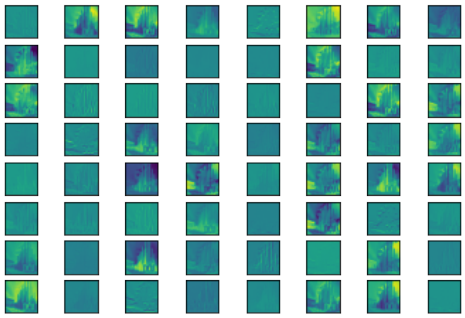

Fig. 4. ResNet50 First Layer Image Visualisation

Both first convolutions, loaded with different default imagenet weights give the impression that we can see a visual similarity between feature layers. This is not the case, as even with few parameters, it is not viable to identify when a feature map changes from the weight values or the output image, which when distributed through the neural network, will have a significant effect on the final prediction.

For both of the above images, we can compare the individual Standard Deviation and Skew across each filter, in this case *VGG16 = 0.2067%, ResNet50 = 0.1111%* for the Standard Deviation of weights, and *VGG16: -0.0145,ResNet50: -0.1124*. When shown in figure 5, there is a significant difference in both first layers.

Once we established that the layers could be represented in this manner, a structure containing both elements for each layer was created. The data structure, including layers with no weights or biases like pooling, padding or normalisation is then used as the one component of generating a signature.

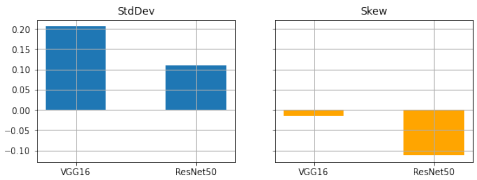

Fig. 5. Comparison of StdDev and Skew of VGG16, ResNet50

B. Generating a Unique signature

Once we had every layer reduced to a number of identification values, we aggregated these together and together with a breakdown of the structure, a signature was generated. During this investigation, it was recorded that the process for extracting an *impression* of the neural network model needs to be one way. If we are focusing on the trust of the system that the model is dependent on, it is necessary to ensure that the model cannot be recreated without the complete data source and process needed to accurately regenerate the model.

The outcome of this project will be establishing if the model is either identical, or the weights and bias are similar. In almost every case, this ***will not*** be reproducible, instead creating a brand new instance that that path of provenance will be based on. That process is shown below in figure 6

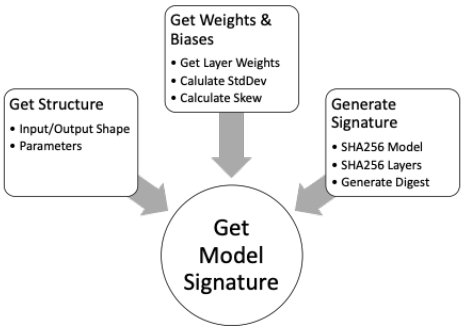

Fig. 6. Process for developing Unique Signature

The flow of generating a new signature is highly dependant on the main hashing algorithm used prior to returning a unique signature. To achieve this, we examined the outcome of running a basic hashing algorithm on each layer, running a SHA-2[18] algorithm on the whole stack, and obtaining the basic information synopsis from each layer into an array, and running an SHA-256 algorithm on both the data and the structure array. This method provide the optimal mix between efficiency and utility. While running a number of sha265 update functions will impact on time, particularly in models with a high number of layers to run functions on, it allows minor changes to the layer that don't affect the functionality such as changes names, and other synopsis data. While uniqueness is paramount for the model, unless it is caused by initialisers, then it should be left to one side.

The name of the layer was removed as an indicator as the name potentially changes on each pull from the Keras repository. Another aspect to bear in mind that models created without pre-loaded weights will always have a diverse

TABLE I
DATA VALUES STORED PER LAYER

Metric	Description
Weight StdDev	An indication of the deviation of values, to assist in seeing how the values are weighted
Weight Mean	The average value of all layers across all dimension
Bias StdDev	An indication of the deviation of values, to assist in seeing how the values are weighted
Bias Mean	The average value of all layers across all dimension
Skew	The skew of the dataset across the entire layer

weighting due to the nature of the various initialisers (*Glorut, Uniform etc.*)

C. Robustly Recording Changes

As we now have a method of identifying the model in a broken down format, the next stage is to record and apply a signature to the model. There are a number of methods of ensuring that the model is recorded. The key part is to take the series of hashes on each layer, combined with the model shape, and generate a signature. The signature is what will be recorded as an immutable value that can be regenerated every time the **exact** same model is used. With this identification, we can create a list showing the parent and child of the model.

TABLE II
STRUCTURAL DATA STORED PER LAYER

Name	Desc
"class"	Class Type of the Layer. There are a number of potential outcomes for the layer type. We do not use the layer type for assessment of the data, purely to identify if the structure has changed.
"input_shape"	The Shape of the input Array, expressed as a set, i.e. *(None, 224, 224, 3)* for an image input.
"output_shape"	The shape of the output layer of the model. This contains either a probability output contain the number of possible outcomes, but can also contain full image data in segmentation and GAN applications.
"trainable"	A value which indicates if the layer is trainable. Including this boolean change will indicate if transfer learning is being used.
"params"	The number of parameters in the layer in terms of weights and biases

To ensure this is done correctly, we have two applications - local, or development and global, or distribution. For global changes single global repository was created in the cloud, where all participants with access can determine if the model being created or downloaded has an existing history. This history can then be used to create a new path of provenance. First, though, the model has to be trained.

D. Local Changes vs Global Changes

An additional benefit of the summarisation is to be able to view the model as it is being trained. This is where we introduced a local data repository that can be shared across development teams and organisations to keep track internal models prior to release. This is in comparison with the global model repository, which should only be used for publishing models that require traceability, and not for regular changes.

In keeping with Git vocabulary, local changes are updated with the `push_model(model, [parent])` command, which takes a model and returns a signature and commits the information to a local repository. The local repository stores the abstracted model information set that can be used for local comparisons. To commit the model to the public repository, either the the `push_model(model, [baseline])` command is used with the `local=False` parameter, or if the model has already be updated elsewhere and the model itself is not available, the `push_to_cloud(signature)` function will commit this with the previously set baseline used as the parent. When applied to the global repository, this will return an error if the parent does not exist, unless permission is granted to create a new ancestor.

To provide reference storage for the model repositories, we created two MongoDB instances with the same collection structure. There are two collections - the signatures and the model data. This was to enable verification of the evolution path and when necessary pull the model synopsis for comparison.

The signature dataset contains the unique signature, as well as the internal id providing a timestamp.

```
{"_id":{"$oid":"611063067dbea4ce80c747a0"},"
    signature":"
    ede69fca86a03c780a98dcec6fc1cfe406a3448dcba95663baa18bf853
    ","parent":null,"username":"brendan.
    bonner2@mail.dcu.ie","organisation":"Dublin
    City University","model_source":"Keras Original
     ResNet50","model_data":{"$oid":"611063067
    dbea4ce80c7479f"}}
```

The final parament points to the model collection, where a signature is required to find the model. Once a signature is available, it is possible to recover the impression of the complete tree, including the structure and data.

E. Comparing Models

To ensure there is traceability in between the models, if a parent model is defined, a *b-score* is generated with a degree of difference between the models. The score is based on two factors, the difference between the model structure, and the difference between the two primary parameters for each layer: *StdDev & Skew*.

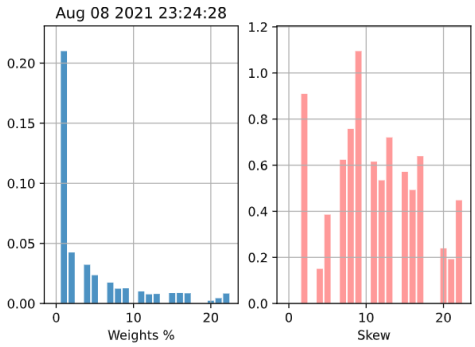

Fig. 7. Difference between model weights and skew per layer

F. Packaging

The final area needed was determining how to distribute the mechanism for generating and recording model representation artifacts. The solution was bundled together in a single

package called *lifecycle* that allows an existing solution to easily integrate the functions into the standard process. To demonstrate this, we bundled the lifecycle package into a single distributable wheel that was uploaded on a third party python notebook.

As we were not able to guarantee that a local database was available, we were able to setup a cloud instance of MongoDB on their Atlas platform that allowed us to run both local and central repositories in the cloud. This was achieved by adding the following text to the start of the notebook/script:

```
from lifecycle import lifecycle_model
from lifecycle import lifecycle_db

mylifecycle = lifecycle_model()
mydb = lifecycle_db (
    username = '[db username]',
    password = '[password]',
    user='brendan.bonner2@mail.dcu.ie',
    organisation='Dublin City University',
    lifecycle=mylifecycle)

mydb.init_model_db()
```

As an example, we were able to apply the package directly to the current leader in the Kaggle Dogs vs Cats challenge[19]. We were able to add the code example above to the python notebook, add the lifecycle.callback to the fit function. From there we successfully recorded the model prior to training (which was recognised as a ResNet50 network from the global repository), and the training was recorded in the local repository, with two models being recorded as children of the core ResNet50 network on the global repository.

```
from lifecycle.callback import LifecycleCallback

callbacks = [
    LifecycleCallback( mydb)
]
```

The result of the initial local stored training, when reduced to 5 epoch, is shown below, with significant training indicated on the model from the base model. During the course of the sampling, we examined every publicly available sample that included training data from the Keras GitHub library, and we able to show that the lifecycle functions can be applied to CNNs, as well and RNN, Transformers Segmentation classification applications without modifications.

```
23:24:31 (user:BB)  Weight 0.3657% Skew -0.206
23:24:28 (user:BB), Weight 0.3496% Skew -0.161
23:24:25 (user:BB), Weight 0.3342% Skew -0.131
23:24:21 (user:BB), Weight 0.3194% Skew -0.109
23:24:16 (user:BB), Weight 0.5589% Skew -0.259
23:04:38 (user:Keras), Weight 0.0000% Skew 0.000
```

IV. Results and Discussion

The *Practicum* was able to identify the internal structure of the models used for machine learning applications, and abstract the content to create a single 32 byte indicator, that can be used as an identifier for uniquely identifying the provenance of machine learning models. The two key applications; recording the changes of the model during the training process locally, and publishing the signature to a cloud repository for verification, validation and traceability all resulted in a mechanism for recording the lifecycle of models.

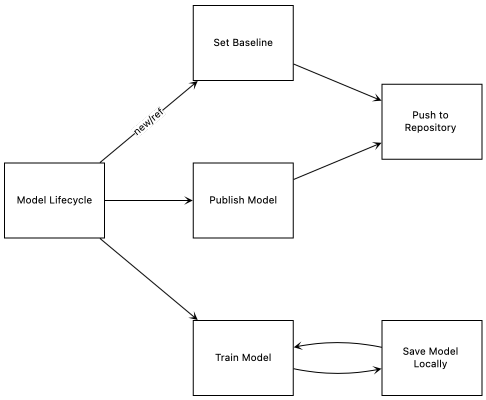

Fig. 8. Lifecycle of Software Models

Beyond looking at simple models, we were able to generate signatures for all default Keras models, as well as sample models used in competitive machine learning challenges. In all cases, we were able to generate a *family tree* of the model's history, as well as an *ancestor* function that identifies the original model used as the baseline for published models. Following the process presented in Figure 8, we are able to store multiple relationships between models as they are created and published. More importantly, we are not able to remove the model from the global repository.

A. Reduction in size for model identification

The first straightforward result, is being able to produce simplified impressions of models that negate the requirement to store large binary models containing all of the weights, while still providing a utility of being able to determine that a model has changed, how it's structure changes, and which layers have been amended. Below in table III is the table of the most common models sizes, which show how much memory is required per iteration, epoch or batch if we wish to retain details of the model. As seen, it shows the high memory cost for some model, which are extracted down to less than one megabyte for the most complex models.

TABLE III
Memory Usage for Internal Model

Model	Size	Impression Size	Time
VGG16	528MB	38.2kb	4.11s
DenseNet121	33MB	693.8kb	0.37s
DenseNet169	57MB	952.1kb	0.53s
Xception	88MB	219.4kb	0.25s
ResNet50	98MB	292.0kb	0.41s
ResNet50V2	98MB	311.3kb	0.42s
MobileNet	16MB	149.3kb	0.17s
MobileNetV2	14MB	248.5kb	0.20s

Overview of Layers stored in Database Comparison of Model Visualisation and Summary

B. Identification of Model History

One of the most powerful elements for AI practitioners, is the ability to generate a timeline of releases based on when a model is released. In the example shown in figure 9, The

initial model is shown preserved in both the local and remote repositories. At the end of the model training process, each individual iteration is stored internally with a parent<=>child relationship as part to training process. Once a model is ready for internal testing, it can be baselined, updating the model to point to the original base model. This is shown in the iterations in light blue below. When a model is ready for public release, then the model is baselined.

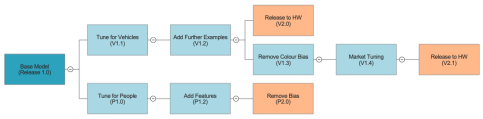

Fig. 9. Lifecycle of Models in Development Path

The true power of this is being able to use the system to verify and baseline systems, so they are compliant and can be audited in line emerging standard in AI, such as the EU Artificial Intelligence Act, inspired by the work of the AI HLEG [1]. Every subsequent model released will be added to the chain, and at anytime the model itself can be used to generate a signature while shows the history.

The final set of tools included in the Practicum provide the overview of the history. The `lifecycle.get_history()` function returns the history in either the developers or the global repository of a given signature, providing the dates, basic information about the model as well as the important signature. The information for a default keras transformer training over 20 epoch is shown in figure 10

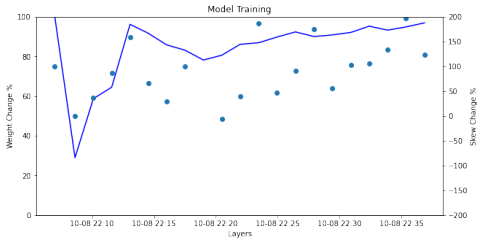

Fig. 10. Transformer Learning Process

The output of signatures is quite extensive when recording the history of training, although it has been successful in staying within the family branch. Additional functionality is provided by the `lifecycle.get_ancestor()`, which shows the local root node, or the ultimate parent in the global repository. Of course, this can be applied in two directions, and the `lifecycle.family_tree()` function records all children available in a nested group.

The true power of this function will be a full graphical GUI to explore the graph, and functions are avail to plot both the history (figure 11) as well as an interactive tool for comparing the entire tree. The performance of both is significantly faster than any other tools for comparing the layer structure of models in real time.

|

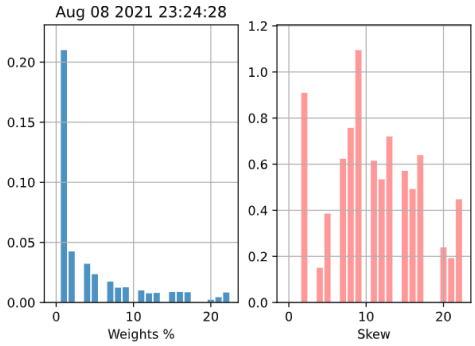

Fig. 11. Comparison of Before/After Training

V. Conclusions and Recommendations

The conclusion of the *Practicum* is there is currently no simple and efficient method of verifying the lifecycle of the development of the model. When models are released, either for production, competitions or uploaded to critical systems, they remain a blackbox. While developing a system for verifying the history of the model, it will allow us to both gain an insight into the development process, but also be able to check shared characteristics between machine learning deep learning systems.

We explored a number of novel elements to provide more information about a model that can be used to identify differences between models, and the difference between layers of models. The next phase of the research will be to develop a study based on real world use in a development environment. The practical application yielded more data than originally envisaged, particularly in relation to the transferability of the modules to all deep neural networks.

Visualisation applications were limited to graphs showcasing the difference between models. It was not necessary to explore 3D comparison primarily because the layer information was reduced to two variables. While they have the capacity to continuously update an visual family tree while the system is in training, a direct *tensorboard* style application may be more beneficial .

A. Recommendations

Our experimentation into the personality of a neural network model shows that there is additional scope for further research using these finding within the scope of interpretability and explainability. Comparisons between training models error rates and the difference in layers provide an indication into how training is successful, although there was an interesting observation where some layers were trained *more*, that is the similarity between layers changed out of sync with error rates.

There are several promising spillover activities that can follow this approach. The majority are related to explainability, although a more pertinent issue will be utilisation of the techniques in this paper to a practical application for real world auditing of models as they are being trained and released. In addition, the following areas could increase knowledge of uptake of explainable AI.

1) Increasing information stored. Creating the information blocks at a more granular level to generate the checksum with all the information currently stored can provide more utility. One the other hand, this would create a non-optimal storage if we included the full set of model weights. There may be a compromise to provide more visibility, while retaining unique characteristics of the model.
2) Introduction of blockchain verification. Currently using database solutions that can be accessed publicly. Integrating a distributed ledger to store the public models may provide more security in the models. Until, it is required, the one-way hashing algorithm provide security that the model used is identical, or closely related, to a signature.
3) Closer integration with Machine Learning providers. Including push functionality into the training and deployment process, as well as the inclusion of visualisation tools, could increase uptake and provide more sources to compare.
4) Testing in a model distribution environment. The solution can be used in an environment where pre-trained of default models can be verified prior to manipulation. This can be used to establish a metric of similarity between models (the functions have been included) once trained.

VI. Acknowledgement

I would like to sincerely thank Dr. Alessandra Mileo for the guidance, motivation and support in developing this practicum.

References

[1] High-Level Expert Group on AI, "Ethics guidelines for trustworthy AI." [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai

[2] M. Ryan, "In AI We Trust: Ethics, Artificial Intelligence, and Reliability," vol. 26, no. 5, pp. 2749–2767.

[3] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," vol. 6, pp. 52 138–52 160.

[4] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). [Online]. Available: http://arxiv.org/abs/1711.11279

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

[6] C. McLeod, "Trust," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University. [Online]. Available: https://plato.stanford.edu/archives/fall2020/entries/trust/

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." [Online]. Available: https://arxiv.org/abs/1512.03385v1

[8] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Chapter 15 - Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, Eds. Academic Press, pp. 293–312.

[9] R. Hecht-nielsen, "III.3 - Theory of the Backpropagation Neural Network**Based on "nonindent" by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE." in *Neural Networks for Perception*, H. Wechsler, Ed. Academic Press, pp. 65–93.

[10] H. M. Kim and M. Laskowski, "Toward an ontology-driven blockchain design for supply-chain provenance," vol. 25, no. 1, pp. 18–27.

[11] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets. [Online]. Available: http://arxiv.org/abs/1712.09913

[12] T. Mahoney, K. R. Varshney, and M. Hind, *AI Fairness: How to Measure and Reduce Unwanted Bias in Machine Learning*. [Online]. Available: http://proquest.safaribooksonline.com/?fpi=9781492077664

[13] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. [Online]. Available: http://arxiv.org/abs/1312.6034

[14] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network Dissection: Quantifying Interpretability of Deep Visual Representations. [Online]. Available: http://arxiv.org/abs/1704.05796

[15] G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. [Online]. Available: http://arxiv.org/abs/1610.01644

[16] G. Bebis, R. Boyle, B. Parvin, D. Koracin, I. Pavlidis, R. Feris, T. McGraw, M. Elendt, R. Kopper, E. Ragan, Z. Ye, and G. Weber, Eds., *Advances in Visual Computing: 11th International Symposium, ISVC 2015, Las Vegas, NV, USA, December 14-16, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science. Springer International Publishing, vol. 9474.

[17] J. Brownlee. How to Visualize Filters and Feature Maps in Convolutional Neural Networks. Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/

[18] N. I. o. S. and Technology, "Secure Hash Standard (SHS)."

[19] P. Golle, "Machine learning attacks against the Asirra CAPTCHA," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pp. 535–542.