Proceedings of the Informing Science + Information Technology Education Conference

An Official Publication of the Informing Science Institute
InformingScience.org

InformingScience.org/Publications

Online July 6-7, 2022

AN EXPLORATION OF A VIRTUAL CONNECTION FOR RESEARCHERS AND EDUCATORS BY EXPLORING STRATEGIES ENTERPRISE INFORMATION SYSTEMS SPECIALISTS NEED TO INTEGRATE NOVEL NEURAL NETWORK ALGORITHMS INTO AN IMAGING APPLICATION – A DESIGN SCIENCE STUDY

Emma Quindazzi	Infosys, Tampa, Florida, USA	emma@emmaquindazzi.com
Samuel Sambasivam*	Woodbury University, Burbank, CA, USA	Samuel.Sambasivam@Woodbury.edu

* Corresponding author

ABSTRACT

Aim/Purpose	The problem statement in the proposed study focuses on what strategies enterprise information systems specialists need to integrate novel algorithms into an imaging application that had not yet been identified. The aim is to demonstrate that a cross-convolutional neural network can be implemented within the home laboratory – an exploration of a virtual connection for research. An analysis of the works provides the basis for future extensibility of the software application for ImageJ2.
Background	The study was guided by the research question: What strategies do enterprise information systems specialists need to integrate novel algorithms into an imaging application? This study demonstrates how to bring a lab-tested application online within a home laboratory to further build upon those findings.
Methodology	A conceptual analysis was utilized for the artifact's creation within the umbrella of design science to aid in the data interpretation segment. A conceptual framework was developed to determine relevant subject matter, such as useful software applications and technological enhancements to an image application. A research sample was not used in this study.

Accepted by Editor Michael Jones │ Received: January 12, 2022 │ Revised: April 17, 2022 │ Accepted: April 20, 2022.

Cite as: Quindazzi, E., & Sambasivam, S. (2022). An exploration of a virtual connection for researchers and educators by exploring strategies enterprise information systems specialists need to integrate novel neural network algorithms into an imaging application – A design science study. In M. Jones (Ed.), *Proceedings of InSITE 2022: Informing Science and Information Technology Education Conference*, Article 24. Informing Science Institute. https://doi.org/10.28945/4953

Contribution This research contributed to the body of knowledge by using a cross-convolutional neural network to explore novel algorithms as an enterprise information system specialist and set up the imaging application called ImageJ2 for development. The study's setup is documented in a series of steps to demonstrate the how-to set up such a study.

Findings The findings were that it is possible to implement an existing work from within the home laboratory, steps of which are outlined for those to follow. Future work can be extended from the baseline workings. Furthermore, an analysis of the existing code was determined to see if the existing PyTorch code could be developed within Java to act as an extension to ImageJ2 later. By examining the programming code and the cross-convolutional functionality, a determination was made on the best Java mapping. A set of highlights of the processes used are included.

Recommendations for Practitioners The study intersects two differing realms: artificial intelligence and the enterprise information systems network. This study builds upon an AWS system and details the steps to implement an artificial intelligence system on the Amazon Web Service (AWS) platform. The researcher investigated existing software imaging Java applications and probed the areas of potential extensibility for implementing the artificial intelligence novel algorithms.

Recommendations for Researchers The concept of being able to bring online a turnkey set of computer hardware off-the-shelf within the home laboratory may be an unexplored avenue to some researchers. The recommendation is to encourage researchers to see past the constraints that a lack of hardware may bring about for computer science research. The researcher no longer has to be within an office laboratory on campus to access power computers. The doors are open to exploring a whole new world. It explores how to move through the various linkage issues with old libraries and new systems. Other avenues this research can enhance are extending the ImageJ libraries as a Java plugin with the novel algorithms such as the cross-convolutional network implemented on the AWS platform.

Impact on Society Expanding the accessibility to the researcher and practitioner field could be profound. No longer is the researcher or practitioner constrained to the office laboratory. This work shows how to move through the issues to invoke the software to produce and investigate existing software. The findings and guidance of this study are profound.

Future Research Future research should focus on implementing the mappings table as determined in previous research. A future design could be pursued from the analysis and implementation of a Java neural network algorithm as a basic implementation. Java can chain elements to begin replicating the algorithm by Xue et al. (2019). The cross-convolutional element should have further analysis to ensure the full replication within Java.

A cross-convolutional neural network (CCNN) Java algorithm could be implemented and trial run within the code; this would allow for the leverage of the heavy lifting of the program code to predict image motion. There seemed to be few approaches regarding predicting motion frames of a future image, and such predictions could be a giant leap for the health care world regarding microscopic and x-ray images.

Keywords artificial intelligence, artificial intelligence in medicine, butterfly recognition, CIBR, computer vision, content-based image retrieval systems and Java, convolutional neural network, cross-convolutional neural network, deep learning, digital imaging, future image prediction, image recognition, ImageJ, ImageJ2, recurrent neural network, video motion

INTRODUCTION

To date, analysis is ongoing in image recognition with predicting future frames captured from video. Since its launch by Wayne Rasband and, later on, by Schneider et al. (2012), ImageJ has been a Java-based imaging software application. ImageJ was a world-renowned open-source software product, and ImageJ2 was a later launch of the original ImageJ software. The core part of the software library from ImageJ was included in ImageJ2. ImageJ2 (Rueden et al., 2017) was an open-source software application to manipulate static images.

The research problem addressed in the proposed study was to explore the strategies enterprise information systems specialists needed to integrate novel algorithms into an imaging application (Rueden et al., 2017; Schneider et al., 2012). Various algorithms were analyzed, primarily the cross-convolutional neural network by Wu (2020). Furthermore, later analysis involved the prediction of the 2D image movement of a butterfly. The sample evaluated the existing Lua-based cross-convolutional algorithm developed by Wu (2020) and Xue et al. (2019).

Image prediction from video analytics was a field of increasing interest. Having computers interpret images as a 2-year-old would visualize them was an ongoing challenge (Szeliski, 2010). Several studies have analyzed video content that produced frames for prediction. The work of Liu (2009), Wu (2020), and Xue et al. (2019) determined image prediction by a series of steps using encoders and decoders. These encoders and decoders manipulated the image to be captured for motion prediction through a series of frames, synthesizing out several frames from one central frame (Liu, 2009; Wu, 2020; Xue et al., 2019).

Within this study, an exploration took place via steps to bring online the demonstration run by Xue et al. (2019) from within a home office. The demonstration used motion encoders, image encoders, feature maps, and kernel decoders. Measurements of the movement were captured as predicted, and ground truth element comparisons were given between software platforms. The study explored the software ImageJ2 (Rueden et al., 2017), widely used within the medical field and renowned for its handling of images. Suggestions for future work were covered within this study with ImageJ2 and the prediction of an image from motion or a video sequence.

As a solution to the issue of synthesizing frames, the researcher investigated plugins for ImageJ2 that could predict unwanted clinical outcomes, such as Gomez-Perez et al. (2016) mentioned. This study partially focused on medical imaging tools and research demonstration software using cross-convolutional neural networks. The medical field has contributed significantly to the progression of image analysis, and there was a lack of efficient automated deep convolutional neural network analysis tools for medical imaging (Suleymanova et al., 2018). The cross-convolutional technique was a novel algorithm (Wu, 2020). After extensive research, the researcher found a lack of plugins and the ImageJ2 core library source code lacking in the cross-convolutional algorithmic technique.

The researcher had determined that the solution was to explore the possibility of executing the demonstration run by Xue et al. (2019) from within the home office lab. The researcher presented images on the invocation of an initial plugin. This study aimed to analyze the demonstration by Xue et al. (2019). The future integrations of the plugin and core code will be suggested in future work.

PROBLEM STATEMENT

The problem to be addressed in the proposed study was what strategies enterprise information systems specialists need to integrate novel algorithms into an imaging application that had not yet been identified (Rueden et al., 2017; Schneider et al., 2012). Wu (2020) provided a cross-convolutional neural network novel algorithm that could predict 2D image movement from video motion. Studies by Rueden et al. (2017) offered the ImageJ2 next generation of the image analysis tool. Furthermore, Wolfgang et al. (2020) compared various algorithms for image analysis. The research emerged, covering motion with video using ImageJ2 as a testing platform (Popek & Iskander, 2020).

Previous studies incorporated the slanted edge algorithm as a well-known function to determine an image's quality (Roland, 2015). Furthermore, bioimaging has produced other open-source tools, namely BioImageXD, a well-known and widely used application (Kankaanpää et al., 2012). However, ImageJ2 and BioImageXD were vastly different. ImageJ2 was a platform for manipulating singular images, and BioImageXD was an autonomous platform that used multiple algorithms for 3D visualization. Consequently, the core libraries for ImageJ2 included many algorithms but lacked novel algorithms (Arena et al., 2016; Rueden et al., 2017).

The baseline of this study was to determine if it was possible to implement the demonstration code produced by Xue et al. (2019). By analyzing the baseline, the researcher would understand how the algorithms operate. Additionally, the researcher had to gain a foundational knowledge of existing research. The study analyzed work elsewhere established with static images and convolutional neural networks in a software application, one of which was called Glucotrol XL (Wolfgang et al., 2020). Optical coherence tomography (OCT) was a well-known imaging technology within medicine that used 2D and 3D images. Furthermore, biological research has included cell classification using microscopic images using convolutional neural networks (Oei et al., 2019).

ImageJ2 is used for static image analysis, and any image enhancement is implemented with Python Imaging Library Pillow. The ImageNet (ILSVRC2012) dataset provided a wealth of images for the study by Oei et al. (2019). Despite these studies, there seemed to be a lack of research that implemented motion prediction using a cross-convolutional algorithm in the main codebase of ImageJ2. Research has emerged as an emerging area for motion prediction as a frame, with video prediction remaining an unsolved issue (Wu, 2020).

Increasingly, artificial intelligence has shaped enterprise networks worldwide (Gilbert, 2018). With the advent of digital transformation and lowering operational costs, companies looked for artificial intelligence to drive the business forward. This study was from an enterprise information specialist's perspective exploring artificial intelligence functionality regarding a novel algorithm integrating into a Java-based software imaging application. The Java programming language significantly contributed to enterprise applications' success (Antoniadis et al., 2020). This study included invoking a Java imaging application as a suggestion for integrating the state-of-the-art neural network technique.

PURPOSE

The purpose of the proposed design science study was to explore what strategies enterprise information systems specialists need to integrate novel algorithms into an imaging application. The research focused on existing and novel algorithms, such as the cross-convolutional neural network algorithm. The study did not involve human subjects. The research followed the design science approach (Dresch et al., 2014; Wieringa, 2014). The use of the design science methodology allowed for the exploration of a software artifact.

SIGNIFICANCE OF THE STUDY

This study is significant as it could offer guidance on implementing a novel algorithm, such as the cross-convolutional neural network (Wu, 2020), within a home laboratory. Cross-convolutional neural networks have been used to predict image movement from a motion video. Work has been performed within Java, ImageJ2, an open-source software application using plugins and other standalone forms of software for image retrieval and image matching. This study included work from previous researchers, such as Liu (2009), Wu (2020), and Xue et al. (2019). For this reason, the study focused on the demonstration code for the recent studies obtained in the public domain, obtaining a baseline for this study.

The existing demonstration code by Wu (2020) synthesized future frames using deep learning. A comparison of the motion of image shapes was made, and the KL divergence was calculated. Movements were predicted from a single snapshot. For example, a person exercising would move their leg up and down; furthermore, the work of Xue et al. (2019) was used as a foundation for the study by Wu (2020).

Experiments were performed by Wu (2020) with Atari games of basketball players, with the quantitative results measured as ground truth segments for 30 images. A comparison was drawn between two baselines, and the test was named the parts, structure, and dynamics (PSD) test. The focus of this study was on the set of shapes provided by previous research.

The results found that the human exercise (Liu, 2009; Wu, 2020; Xue et al., 2019) dataset tested as an intersection over union vastly outperformed the two baseline models. This study used another dataset, the toy shapes dataset (Liu, 2009; Wu, 2020; Xue et al., 2019), and did not use any human subjects in the research. The convolutional neural network algorithm had to be implemented initially. The work of this research study was groundbreaking; the researcher had not found any existence of such a study that has focused on setting up and attempting to run a demonstration by Xue et al. (2019).

LITERATURE REVIEW

THE HISTORY OF BUTTERFLY EVOLUTION AND IDENTIFICATION

The butterfly image can be used in future work for this research and serves as a challenging area for predicting butterfly type. To begin, the researcher implemented a set of images and neural networks provided by Liu (2009), Wu (2020), and Xue et al. (2019). An exploration can incorporate the video motion of a butterfly and the prediction of image scenes; consequently, obtaining the pattern from the butterfly and predicting the type of butterfly.

Furthermore, to understand how images were retrieved using butterflies as an example, one must examine the existing research on how butterfly colors, patterns, and wing shapes were formed. Several researchers studied wing pigment synthesis to understand how the different color patterns (Futahashi et al., 2012; Nijhout, 1980) were formulated on a butterfly. Since Darwin's theory, mimicry has been in existence (Mallet, 2001). Mimicry is when a butterfly develops patterns that look like or mimic another butterfly species to avoid predation.

The groundplan (Nijhout, 2001) covers the various pattern variants on a butterfly wing. The basal, central, border ocelli, marginal, and submarginal bands all make up the ground plan elements. Consequently, eyespots on butterflies could be externally manipulated to reproduce the pattern – studies of gene-expression patterns determined how the butterfly obtained its eyespots (Beldade & Brakefield, 2002). The groundplan may contribute to image retrieval and specifics for feature vector maps.

THE BENEFITS OF THE JAVA PROGRAMMING LANGUAGE

Java is an open-source programming language now owned by the Oracle Corporation. Java is mainly for business use and is an object-oriented programming language. Matlab (n.d.) is a commercial software package often used within the scientific community for artificial intelligence. Commercial software's disadvantages were that they tend to be costly to use and requires a site-wide license (Sage &

Unser, 2003). Furthermore, Java offered the enterprise functionality with its Java enterprise platform (JEE) and had an open-use policy.

Additionally, existing software applications lacked a way to communicate with one another. The Java programming language was used in many situations to create a bridge between two applications. Indeed, Inés et al. (2019) highlighted a lack of a software bridge between bio-tools, ImageJ, and DeepLearning4j. According to Kainz et al. (2015), commercial image retrieval software did not have the extensibility advantage of open-source software, and licensing issues for commercial software were one of the biggest reasons for lack of use.

The Java programming language has an extensive library that can be implemented for neural networks. Java is a prevalent language within the artificial intelligence field and has been used for so many research applications ranging from sentiment analysis for writing (Zahidi et al., 2021) to cancer type prediction with recurrent type networks (Karim, 2018).

Artificial Intelligence, Deep Learning, and the Role of Mullerian Mimicry

The research by W. Zhang et al. (2018) mentioned a lack of a general definition for the term deep learning. An investigation of existing definitions of deep learning was performed. The conclusions were that deep learning was defined as the knowledge that resides over two or more variables or instances that establish a relationship. An in-depth learning methodology was proposed with a series of steps and remarks. Bishop's (2006) early work outlined that searching for patterns in data has long been an issue.

Research continued to evolve with deep learning to transform challenging datasets. Barbastathis' (2020) research stated that machine learning is highly efficient with the hardware system it was run on, and it delivered excellent results for computational imaging. Additionally, these architectures mainly used deep learning networks.

Neural networks are nonlinear regression techniques (Kuhn & Johnson, 2013) and operate on the theoretical concept of how the brain works. Many computer vision algorithms and techniques have been used, and more efficient algorithms have been trending (Szeliski, 2010). Research has found that deep learning approaches have been highly successful (Karim et al., 2020).

Deep learning-based approaches that are state-of-the-art tend to outperform classical clustering analysis (Karim et al., 2020). Deep learning has grown and has been highly successful in image recognition. A total of 80 million different connections may be obtained from layer number one. Convolutional neural networks can automatically implement a more vital descriptive ability (Tan et al., 2020). The automation of descriptives would prove highly advantageous over manually sifting through data.

Lepidoptera, also known as the butterfly insect species, ranges from 15,000 to 21,000, according to Almryad and Kutucu (2020). Some of the issues of butterfly classification consisted of slow retrieval and low accuracy ratings. The study concluded that an 80% successful match rate could be achieved using deep learning and a convolutional neural network. Fine-grained categorization of butterfly specimens was a current study area such to Lin et al. (2019). Consequently, there was a need for an automated system for fast image classification that did not need human intervention throughout the entire process.

Convolutional Neural Networks (CNN) for Image Recognition Software

Directly applicable to the realm of neural networks were CNNs. CNNs were a set of connections defined as a convolutional layer (Karim, 2018). A CNN recollects its previous knowledge, a form of memory. The lower layer focuses on small areas of an image, whereas the higher layer overlaps the lower-level features making them much more extensive. The neighborhood filtering, also known as convolution, is detailed in the research by Szeliski (2010). Figure 3 shows the input image along with the various convolutions.

The new image output was a convolutional map, where the maps are made into flattened vectors named characteristics, also known as the CNN code. To conclude, CNN was highly advantageous over supervised learning due to the vast amounts of data the machine can learn and accurately deliver. Karim (2018) states that CNN has been widely used in computer vision.

State-of-the-art neural network algorithms

Artificial neural networks (ANNs) consist of many state-of-the-art algorithms under the guise of deep learning. Deep learning resided under the umbrella of machine learning within artificial intelligence (Karim, 2018). Deep neural networks were considered state-of-the-art in deep learning. One of the most successful models was the multilayer perceptron using the feed-forward function (Bishop, 2006). An investigation into cross CNNs with image recognition follows.

Cross-Convolutional Neural Networks (CCNNs) for image recognition

Recently, research expanded to include CCNNs. Y. Zhang et al. (2019) introduced the CCNNs for intrusion detection. The You Only Look Once (YOLO) algorithm aimed to reduce the neural network volume and the time taken to train the model. Indeed, speed improvements were significant benefits to the nascent model. A timed comparison of a 3x3 convolutional model using four other models found that as the network depth increases, so did the optimization of the parameters. Previously, work by Du (2018) evaluated the use of the YOLO algorithm.

Furthermore, this algorithm also surpassed the R-CNN algorithm. Work has been ongoing by Wu (2020) regarding cross-convolutional networks with 2D shapes and animated sprites in games. The analysis had also involved video footage and analysis of image pairing, and a relationship was determined without any supervision. Convolutional kernels were models with motion prior to and future captured (Burger & Burge, 2016). Additionally, the nascent recurrent neural networks were currently in development.

Recurrent Neural Networks (RNNs) for image recognition

RNNs can map their sequential input data to an output, which gives the network a form of memorization technique (Wang et al., 2020). There were two improvements to existing models: the long-term memory and gated recurrent units (Karim, 2018). The nodes in each RNN layer had a relationship with the neighboring node; however, the issue with RNN was that there was a gradient problem with the data. Memorization of the data was vital and allowed for efficiency.

RNNs allowed for data memorization due to the nodes' cross-functionality (Wang et al., 2020). Studies on pothole detection, single rain streak removal, and multi-deep ranking for supervised hashing with image retrieval were ongoing. Frameworks were proposed, and classification work continued to gain ground in the field. Furthermore, work continued in the healthcare sector, such as Karim (2018), a cancer patient project using an RNN using DeepLearning4j. Also, there were many existing automatic identification and retrieval software systems, some of which are described next.

Existing automatic identification and retrieval software applications

There was a wealth of automatic identification software; however, these systems had accuracy issues (Pavithra & Sharmila, 2019). Many studies have attempted to retrieve images by recognizing objects. A study involving medical image analysis by Litjens et al. (2017) could have contributed to the progression of the butterfly image dataset.

RESEARCH METHOD AND DESIGN

This study's research method is design science and consists of an object of research, namely several artifacts, one of which is the demonstration by Xue et al. (2019). The second is ImageJ2. The study would have a direct approach, investigating the artifact (Wieringa, 2014). The study does not involve the use of human subjects. This research aimed to produce an artifact implementation set that could be used for open-source software novices to implement.

Design science was a method that could be used to solve a problem in the creative science realm. Design science is instrumental when a clear path to a solution was unknown (Wieringa, 2014). For example, within this context, the design of a fast calculation for artificial intelligence using an enterprise programming language such as Java. In this study, the research question was to move toward implementing the artificial intelligence model from a home laboratory with the model's novel algorithms' implementations. The study uses the 12-step method for design science research as Dresch et al. (2014) proposed.

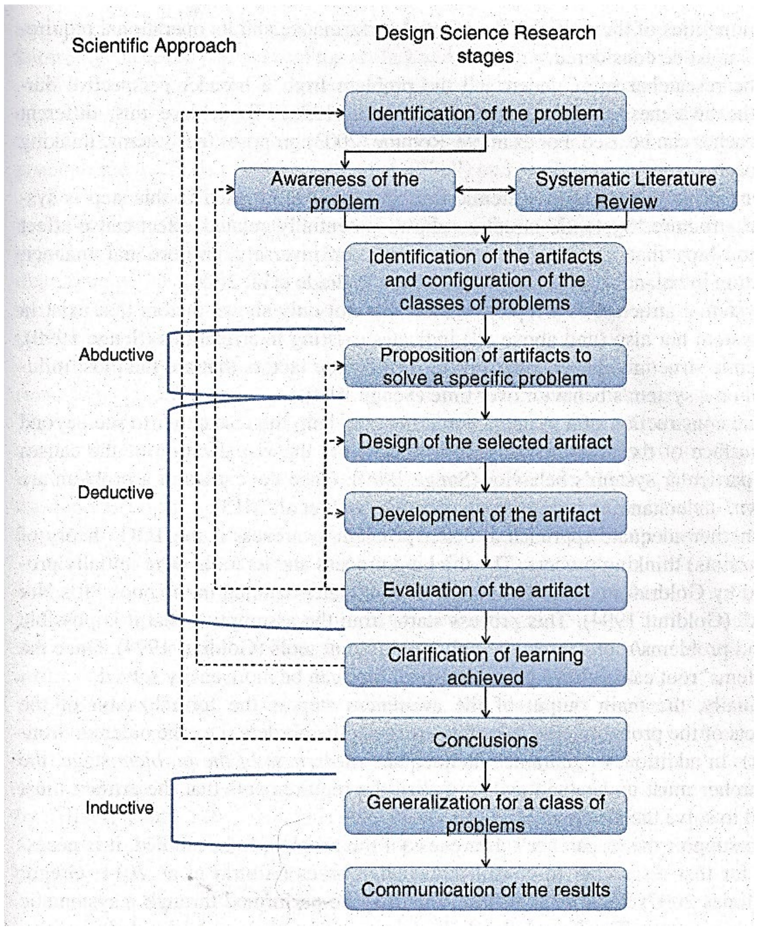

Scientific Approach
Design Science Research stages
Identification of the problem
Awareness of the problem
Systematic Literature Review
Identification of the artifacts and configuration of the classes of problems
Abductive
Proposition of artifacts to solve a specific problem
Design of the selected artifact
Deductive
Development of the artifact
Evaluation of the artifact
Clarification of learning achieved
Conclusions
Inductive
Generalization for a class of problems
Communication of the results

Figure 1. The proposed method for design science research

A hypothesis was not imposed on the study's data; therefore, this study does not include a hypothesis. Quantitative approaches could be used, but the meaning of why those techniques are used is vital (Glaser, as cited in Holton & Walsh, 2016). This design science study would involve (1) conducting background research, (2) simulating the experiments, (3) implementing the plugin for ImageJ2, and (4) analyzing the artifact. An approach taken by Xue et al. (2019) and Wu (2020) has been used. In design science, the artifact would be the demonstration, the implementation of the ImageJ software, and the analysis of the source code from the demonstration. Further, a detailed analysis of a Java to PyTorch mapping for future work.

Participants

There were no participants in the study directly. Images used from previous works by Xue et al. (2019) and Wu (2020) were used.

Instruments

The researcher used a single-case mechanism type of experiment - a form of program test via software implementation for a cross-convolutional neural network.

Procedures

The procedures used were diary notes and images of screenshots of the steps taken and processes used to resolve issues with the software implementations.

Design

The study design followed the 12-step process mentioned in Figure 1 for design science (Dresch et al., 2014).

Data analysis

Data analysis would involve using the installation and execution of the existing code base for ImageJ and ImageJ2. Along with installing the demo software for the work with CNNs (Wu, 2020; Xue et al., 2019).

RESEARCH QUESTION

The research question for this design science study was:

RQ1: What strategies do enterprise information systems specialists need to integrate novel algorithms into an imaging application?

DATA COLLECTION

The study implemented a conceptual analysis for the approach (Wieringa, 2014) while implementing a conceptual framework within design science. The conceptual analysis incorporated the documents such as articles in periodicals, conference proceedings, and other peer-reviewed, highly cited, and relevant literature, books, and software artifacts gathered in part from the discovery of the conceptual framework. The traditional research approach is limited to artifact design. The primary goal of the design science methodology is to provide a better fit regarding the inception, construction, and creation of an artifact into a methodology. The traditional sciences such as biology, chemistry, physics, and sociology focus more on the "how", such as how can one chemical compound affect another compound (Simon, as cited in Dresch et al., 2014).

The data collection instrument was a Mac Pro with 16 GB with 1TB of data. The Mac Pro was running version 10.15.7, a Mac Operating system Catalina version. The data collection would consist of investigating the existing software artifact as a downloadable demonstration from MIT by Xue et al. (2019). The Amazon Web Service (AWS, 2021) (was implemented via a paid subscription basis). The Amazon Web Service allowed for the functionality of the CUDA GPU, for which the researcher's existing machine did not have the capacity. The Eclipse IDE (The Eclipse Foundation, 2021) was downloadable as freeware, and the ImageJ plugin implementation tutorial libraries were also a freely available download (Rueden & Schindelin, n.d.).

Data Analysis

The data analysis was classified as requirements gathering within a design science research methodology (Wieringa, 2014). Data analysis involved using the installation and execution of the existing code base for ImageJ and ImageJ2, along with installing the demo software for the work with CNNs (Wu, 2020; Xue et al., 2019). The requirements were analyzed via a source code line-by-line set of comments. The existing API for PyTorch was referenced, and a table mapping was implemented for the bridge between Java and PyTorch.

The study ran into some issues with access to the online demonstration code by Wu (2020) and Xue et al. (2019). The exercise demonstration was the only one accessible, and the researcher reached out via email, and access was granted. The researcher did not have to use alternative datasets such as the Middlebury optic flow evaluation website (Baker, as cited in Szeliski, 2010). Another is the TV-L1 Optical Flow on the GPU (Zach, as cited in Szeliski, 2010).

Additionally, there were alternatives to the cross-convolutional algorithm. A direct convolution that used video motion or extended short-term memory networks could have been an option. Alternatively, the study of DeepClass4Bio API by Inés et al. (2019) could have bridged the gap between the ImageJ2 and PyTorch. The algorithm by Wu (2020) could have been added via the bridge to ImageJ2.

Findings

The data collection study focused on the work by Wu (2020) and Xue et al. (2019). The researcher set up the Cuda GPU and used the Ubuntu 18.04 operating system for the demonstration for compilation and execution. The report following outlined the steps taken with figurative screenshots. The Eclipse environment was set up locally on the Mac OS within the home office laboratory, and ImageJ was set up locally. Eclipse, Maven, and ImageJ software were installed.

Table 1 defines a mapping between Java using the DeepLearning4j and PyTorch libraries, a first pass of the existing code. The demo.py code produced by Xue et al. (2019) and adapated with source code comments can be found in the Appendix. Previously, the various encoders and decoders were inherited via the nn.Module deep learning class. The researcher used this approach with Java, like a series of chain commands to configure the various manipulations on the tensor. The various layers could be built and instantiated individually, such as ConvolutionLayer layer0 = new ConvolutionLayer.Builder(5,5). Within Java, this layer could call methods on the convolution, one of which can be a stride: the number of pixels the window moves over once the operation is complete (Amidi & Amidi, 2021). Other methods were dropout(0.4) which decreases the overfitting, and nOut(10), determined as the number of feature maps (Karim, 2018).

Table 1. Exploratory first pass view of mapping of demo.py classes to Java classes

Mapping of PyTorch	PyTorch Library Name	Mapping of Java	Java Library Name
ImageEncoder MotionEncoder KernelDecoder ImageDecoder	Super class nn.Module	Image Encoder/Decoder Motion Encoder/Decoder	org.deeplearning4j.nn.conf.NeuralNetConfiguration
Conv2DLayer	Lasagne	Convolution2D	org.deeplearning4j.nn.conf.layers.Convolution2D
LinearLayer	nn.Linear	MultiLayerConfiguration	org.deeplearning4j.nn.conf.MultiLayerConfiguration

UNet	UNet	UNet	org.deeplearning4j.zoo.model.UNet

The build method created the layers as a chained command at the end of the method calls. The example in Figure 1 had a 5 x 5 kernel with one channel, a stride of 3 x 3, and a feature map of 15. The activation type was rectified in linear units. There are three types of ReLU; these are ReLu, LeakyReLU, and ELU (Amidi & Amidi, 2021). This function aimed to bring nonlinear elements into the network. The LeakyReLU is used in the demonstration by Xue et al. (2019), which aimed at eliminating the dying ReLu negative values issue.

```
ConvolutionLayer layer0 = new ConvolutionLayer.Builder(5,5)
                                        .nIn(nNoOfChannels)
                                        .stride(3,3)
                                        .nOut(15)
                                        .dropOut(0.4)
                                        .activation(Activation.RELU)
                                        .build()
```

Figure 2. Chained Java command for a convolutional layer

Table 2 details the function mapping from PyTorch for the convolutional 2D unsqueeze function. There was no direct mapping for this functionality within Java to the researcher's knowledge. In this instance, a potential option was to use a bridge with a C++ frontend and a PyTorch backend (Audet, 2021). JavaCPP allowed for the use of the C++ API via the Java programming language. The C++ API provided an equivalent unsqueeze method that could potentially be implemented that changed the tensor dimensions, i.e., from a 2D to a 3D tensor (Iacob, 2021), thus making the implementation accessible via the Java language.

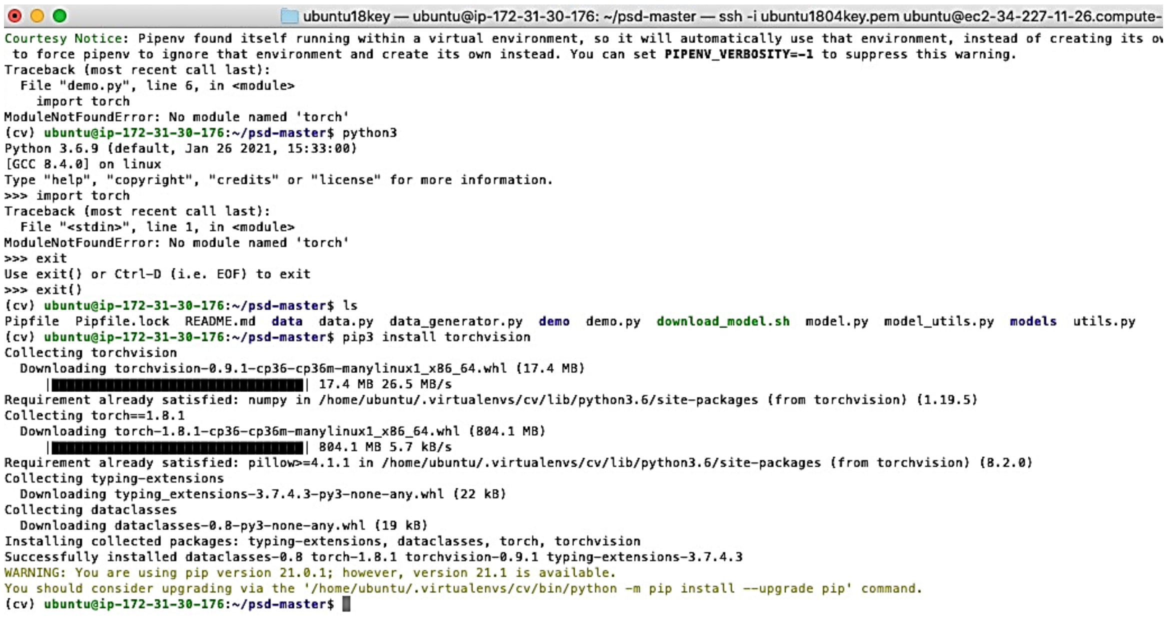
Table 2. Exploratory first pass view of mapping of demo.py method to Java

Mapping of PyTorch	PyTorch Method Name	Mapping of Alternative Language	
conv_cross2d	Conv2d unsqueeze, appended, and concatenated to an output array	C++ API provides an equivalent unsqueeze method, which can be accessed via JavaCPP. A C++ frontend is used in conjunction with PyTorch	org.bytedeco.PyTorch.presets

According to Nguyen et al. (2019), Torch7 was no longer under development, and the most recent version at the time of research was Torch7. The PyTorch library was mainly used; a Python interface was used with the Torch7 same underlying C libraries. Therefore, the researcher concluded that it was viable to use the PyTorch API library as a reference to analyze the code functions.

Furthermore, it has become apparent that t7 models were no longer compatible with PyTorch. The serialization libraries used in Torch7 were also not available for PyTorch. According to Amir (2019), the t7 models cannot be imported from Lua Torch to PyTorch. Xue et al. (2019) also created a second demonstration code project with a person exercising developed in Lua Torch7. The Lua Torch7 model could be converted using pre-existing libraries such as Pip in a Conda environment. A pre-existing repository was created to do this Clcarwin (2017) as outlined by Arul (2021, April 19). The t7 model could be converted from Torch7 into PyTorch. However, this study did not use this research, and the focus of this study was on the shapes dataset.

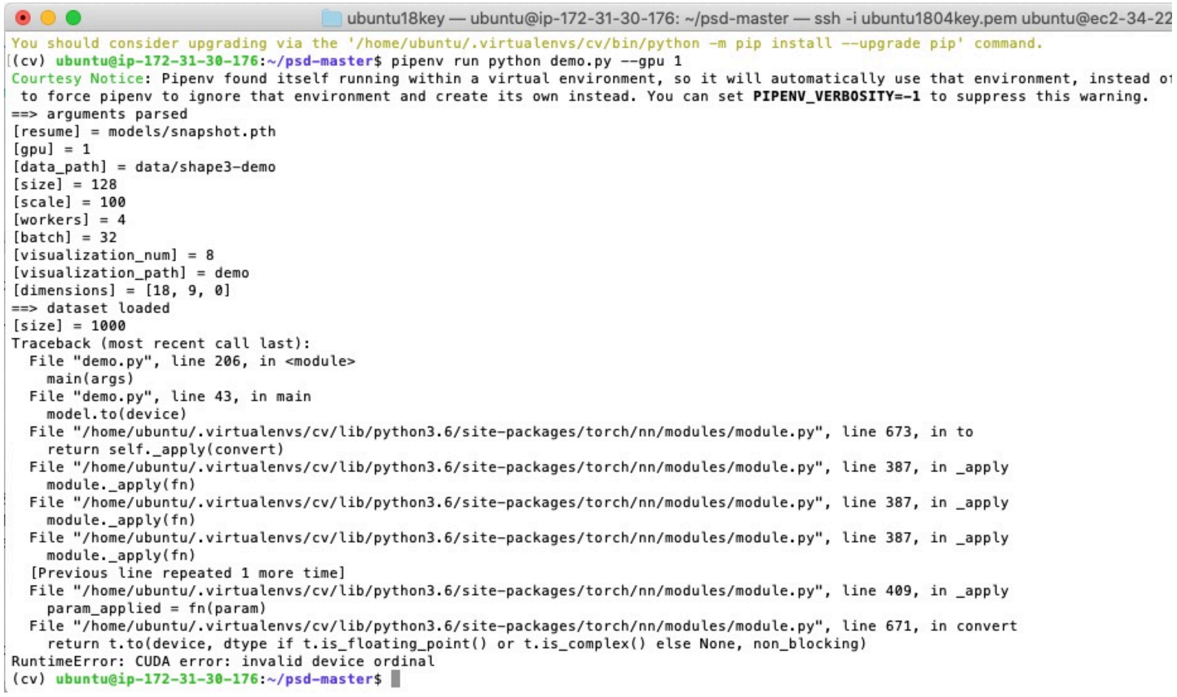
This study is further detailed in Quindazzi (2021), demonstrating the preliminary steps to arrive at Figure 3. As shown in Figure 3, the error was rectified by importing the torchvision libraries via the pip3 install torchvision command. The classes were downloaded and installed for Torch. Next, the researcher reverted to running the original command for the shapes demo software. pipenv run python demo.py --GPU 2 ran.



```
ubuntu18key — ubuntu@ip-172-31-30-176: ~/psd-master — ssh -i ubuntu1804key.pem ubuntu@ec2-34-227-11-26.compute-
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its ov
 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
Traceback (most recent call last):
  File "demo.py", line 6, in <module>
    import torch
ModuleNotFoundError: No module named 'torch'
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ python3
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'torch'
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ ls
Pipfile  Pipfile.lock  README.md  data  data.py  data_generator.py  demo  demo.py  download_model.sh  model.py  model_utils.py  models  utils.py
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ pip3 install torchvision
Collecting torchvision
  Downloading torchvision-0.9.1-cp36-cp36m-manylinux1_x86_64.whl (17.4 MB)
     |████████████████████████████████| 17.4 MB 26.5 MB/s
Requirement already satisfied: numpy in /home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages (from torchvision) (1.19.5)
Collecting torch==1.8.1
  Downloading torch-1.8.1-cp36-cp36m-manylinux1_x86_64.whl (804.1 MB)
     |████████████████████████████████| 804.1 MB 5.7 kB/s
Requirement already satisfied: pillow>=4.1.1 in /home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages (from torchvision) (8.2.0)
Collecting typing-extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting dataclasses
  Downloading dataclasses-0.8-py3-none-any.whl (19 kB)
Installing collected packages: typing-extensions, dataclasses, torch, torchvision
Successfully installed dataclasses-0.8 torch-1.8.1 torchvision-0.9.1 typing-extensions-3.7.4.3
WARNING: You are using pip version 21.0.1; however, version 21.1 is available.
You should consider upgrading via the '/home/ubuntu/.virtualenvs/cv/bin/python -m pip install --upgrade pip' command.
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ █
```

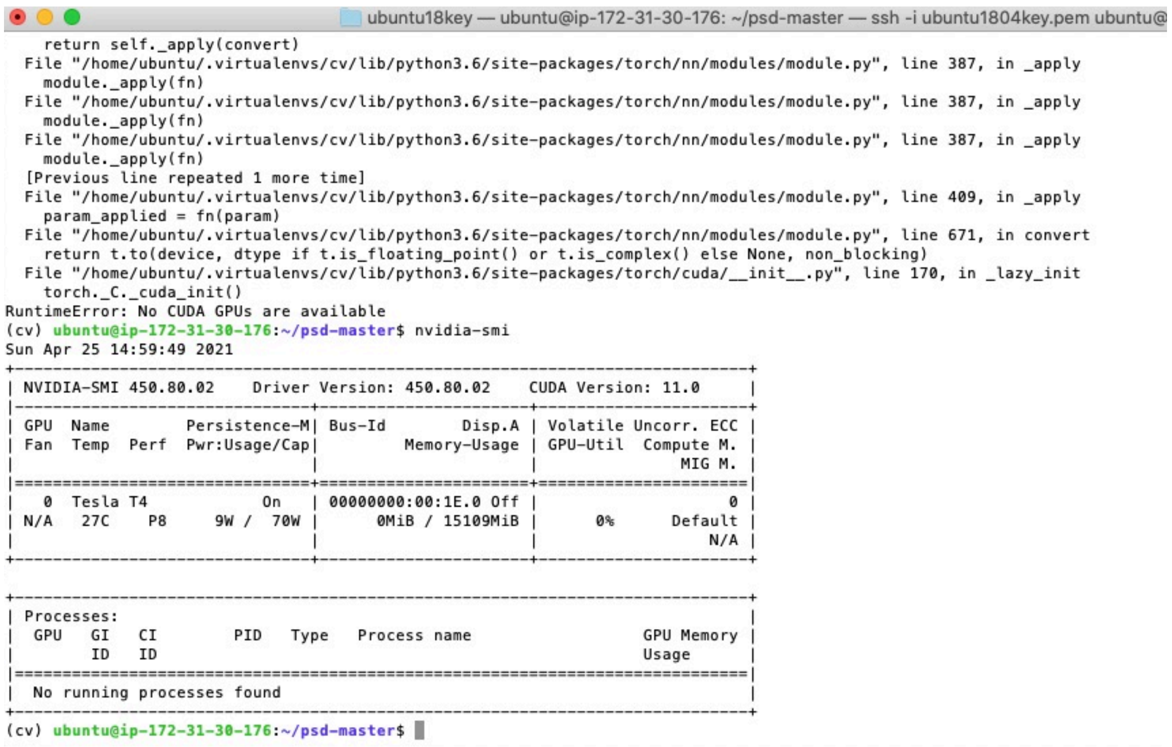
Figure 3. Torch module not found error

The researcher has almost completed the execution of the demonstration shapes file. The bottom of the screen shows in Figure 4 a runtime error for CUDA with an invalid device ordinal. The following steps investigated the CUDA GPU and what was available on this instance via AWS EC2 Tesla T4. As can be seen in Figure 5, the hardware was available. The researcher started to investigate why there was an issue regarding invoking the GPU. Strangely it seemed no CUDA runtime was available. NVIDIA recommended the CUDA GPU makers to run the following commands in this situation: sudo make and ./devicequery in the relevant directories (Franklin, 2021). The Tesla T4 device query was executed without any issue. If the GPU had not been available, this terminal output would not have been executed. Figure 6 shows that there is 1 CUDA-capable device.


ubuntu18key — ubuntu@ip-172-31-30-176: ~/psd-master — ssh -i ubuntu1804key.pem ubuntu@ec2-34-22

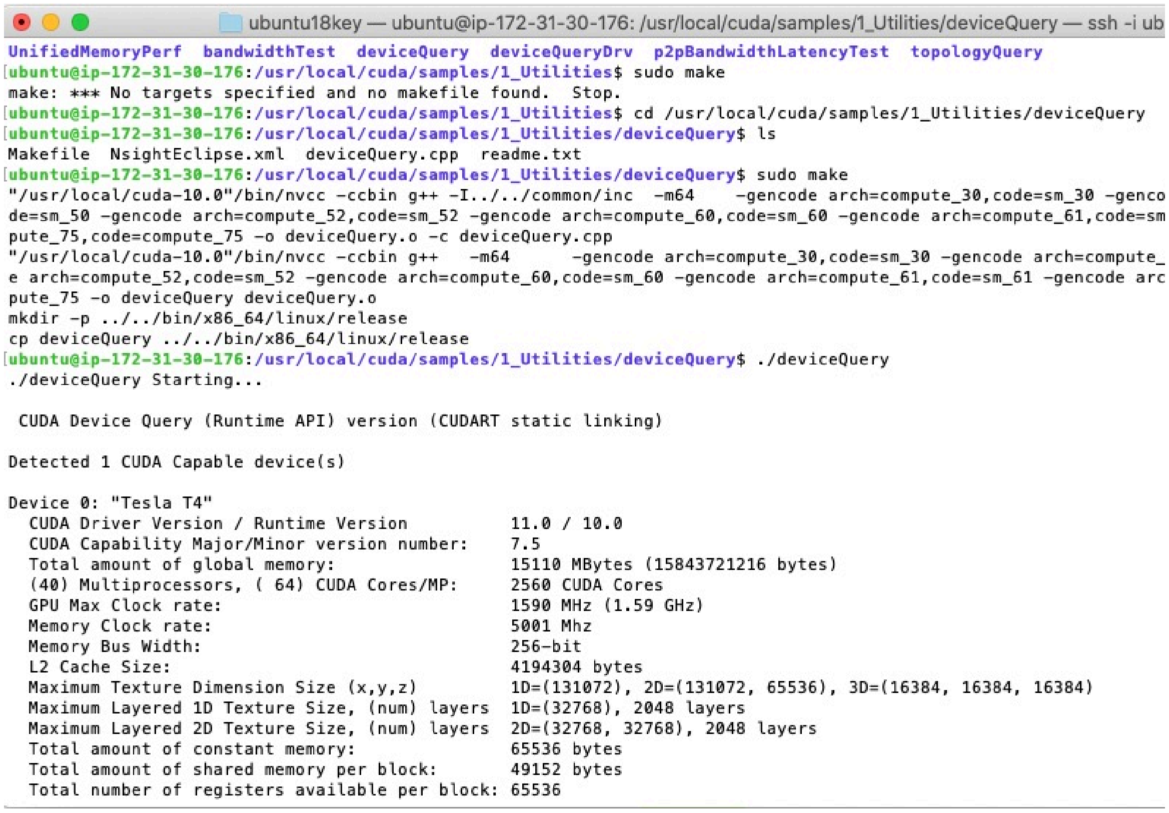
```
You should consider upgrading via the '/home/ubuntu/.virtualenvs/cv/bin/python -m pip install --upgrade pip' command.
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ pipenv run python demo.py --gpu 1
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of
 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
==> arguments parsed
[resume] = models/snapshot.pth
[gpu] = 1
[data_path] = data/shape3-demo
[size] = 128
[scale] = 100
[workers] = 4
[batch] = 32
[visualization_num] = 8
[visualization_path] = demo
[dimensions] = [18, 9, 0]
==> dataset loaded
[size] = 1000
Traceback (most recent call last):
  File "demo.py", line 206, in <module>
    main(args)
  File "demo.py", line 43, in main
    model.to(device)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 673, in to
    return self._apply(convert)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  [Previous line repeated 1 more time]
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 409, in _apply
    param_applied = fn(param)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 671, in convert
    return t.to(device, dtype if t.is_floating_point() or t.is_complex() else None, non_blocking)
RuntimeError: CUDA error: invalid device ordinal
(cv) ubuntu@ip-172-31-30-176:~/psd-master$
```

Figure 4. Demo.py (Xue et al., 2019) beginning to run


ubuntu18key — ubuntu@ip-172-31-30-176: ~/psd-master — ssh -i ubuntu1804key.pem ubuntu@

```
    return self._apply(convert)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  [Previous line repeated 1 more time]
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 409, in _apply
    param_applied = fn(param)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 671, in convert
    return t.to(device, dtype if t.is_floating_point() or t.is_complex() else None, non_blocking)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/cuda/__init__.py", line 170, in _lazy_init
    torch._C._cuda_init()
RuntimeError: No CUDA GPUs are available
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ nvidia-smi
Sun Apr 25 14:59:49 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0 |
| N/A   27C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
(cv) ubuntu@ip-172-31-30-176:~/psd-master$
```

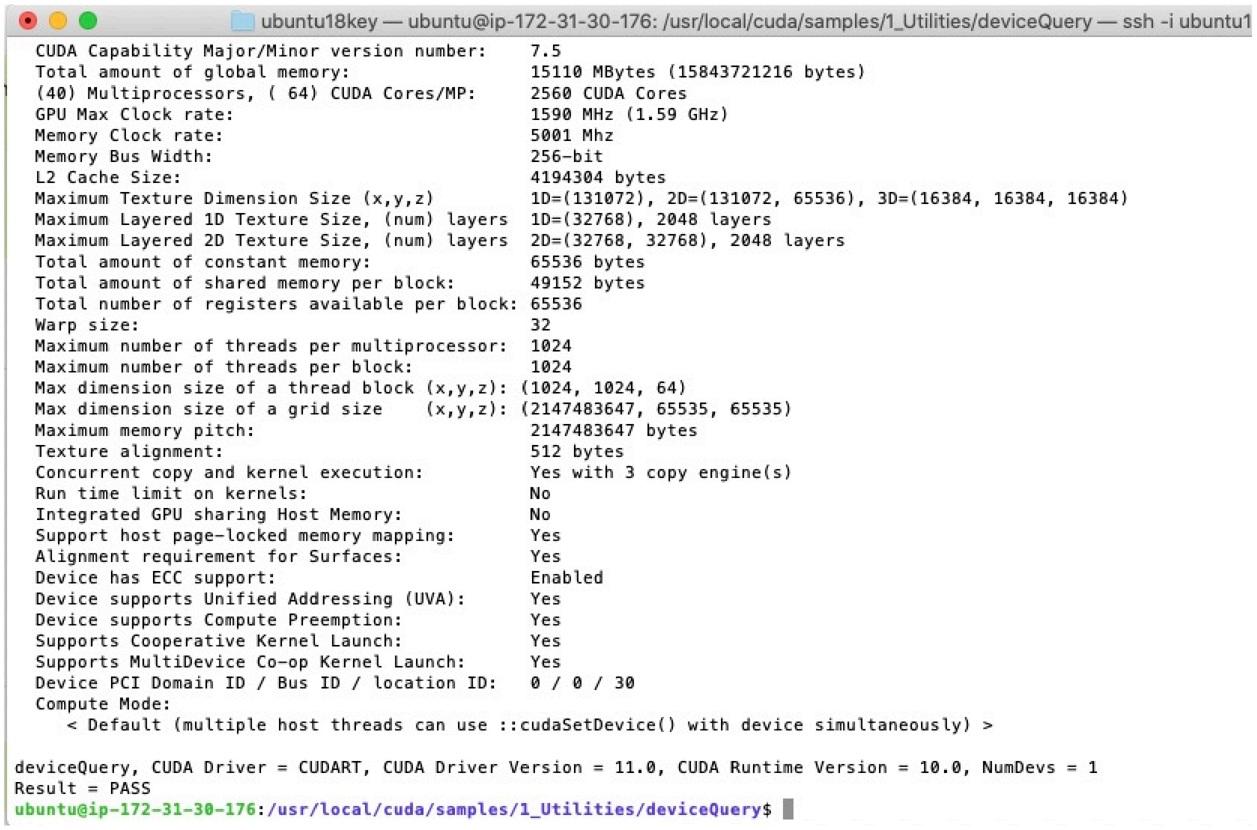
Figure 5. AWS EC2 G4dn.large TESLA T4 instance



```
ubuntu18key — ubuntu@ip-172-31-30-176: /usr/local/cuda/samples/1_Utilities/deviceQuery — ssh -i ub
UnifiedMemoryPerf  bandwidthTest  deviceQuery  deviceQueryDrv  p2pBandwidthLatencyTest  topologyQuery
[ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities$ sudo make
make: *** No targets specified and no makefile found.  Stop.
[ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities$ cd /usr/local/cuda/samples/1_Utilities/deviceQuery
[ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities/deviceQuery$ ls
Makefile  NsightEclipse.xml  deviceQuery.cpp  readme.txt
[ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities/deviceQuery$ sudo make
"/usr/local/cuda-10.0"/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_30,code=sm_30 -genco
de=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm
pute_75,code=compute_75 -o deviceQuery.o -c deviceQuery.cpp
"/usr/local/cuda-10.0"/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_30,code=sm_30 -gencode arch=compute_
e arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arc
pute_75 -o deviceQuery deviceQuery.o
mkdir -p ../../bin/x86_64/linux/release
cp deviceQuery ../../bin/x86_64/linux/release
[ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities/deviceQuery$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          11.0 / 10.0
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 15110 MBytes (15843721216 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
```

Figure 6. CUDA available

Figure 7 showed the result as a PASS for the CUDA GPU, the device was operating correctly, and the GPU had been detected in the container. It would have received a FAIL result had the GPU not been detected. To follow was an analysis of the results of the terminal output of the final execution of the recommended command: pipenv run python demo.py --GPU 1.



ubuntu18key — ubuntu@ip-172-31-30-176: /usr/local/cuda/samples/1_Utilities/deviceQuery — ssh -i ubuntu1

```
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 15110 MBytes (15843721216 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 30
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.0, CUDA Runtime Version = 10.0, NumDevs = 1
Result = PASS
ubuntu@ip-172-31-30-176:/usr/local/cuda/samples/1_Utilities/deviceQuery$
```

Figure 7. The number of devices listed and the pass for the CUDA GPU

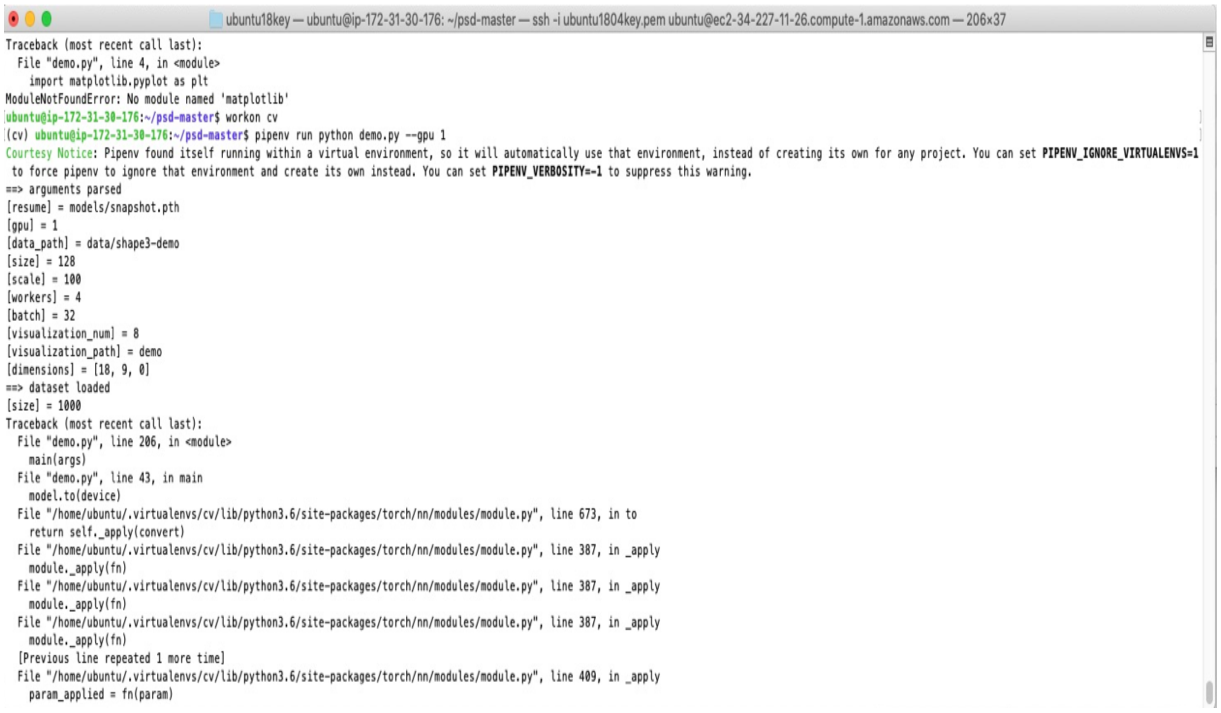
On further debugging the GPU issue and changing the GPU ID for the actual GPU number of devices on the amazon instance, the command was changed to pipenv run python demo.py --GPU 1; however, the same previous runtime error on the CUDA GPU still occurred. Figure 4 previously showed the lines of code where the issue occurred with the Runtime error: CUDA error: invalid device ordinal. The demo.py file was investigated further, and an analysis of the following lines of code that were potentially causing the issue in the demo.py (Xue et al., 2019):

```
def main(args):
  # set device (CPU / GPU)
  if args.GPU == '-1':
    device = torch.device('CPU)
  else:
    device = torch.device('cuda:{}'.format(args.GPU))
```

The execution failed to invoke the GPU NVIDIA CUDA device with an argument of 1.

Figure 8 details the output and shows the GPU device set to 1. Figure 8 shows the demo.py file error at line 43 with the model.to(device) code. The following is the set of code that was executed at line 41-43 in the other demo by Xue et al. (2019):

```
model = PSD(dimensions=args.dimensions, size=args.size)
model.load_state_dict(torch.load(args.resume, map_location='cpu'))
 model.to(device)
```



```
ubuntu18key — ubuntu@ip-172-31-30-176: ~/psd-master — ssh -i ubuntu1804key.pem ubuntu@ec2-34-227-11-26.compute-1.amazonaws.com — 206×37
Traceback (most recent call last):
  File "demo.py", line 4, in <module>
    import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'
ubuntu@ip-172-31-30-176:~/psd-master$ workon cv
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ pipenv run python demo.py --gpu 1
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIRTUALENVS=1
 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
==> arguments parsed
[resume] = models/snapshot.pth
[gpu] = 1
[data_path] = data/shape3-demo
[size] = 128
[scale] = 100
[workers] = 4
[batch] = 32
[visualization_num] = 8
[visualization_path] = demo
[dimensions] = [18, 9, 0]
==> dataset loaded
[size] = 1000
Traceback (most recent call last):
  File "demo.py", line 206, in <module>
    main(args)
  File "demo.py", line 43, in main
    model.to(device)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 673, in to
    return self._apply(convert)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  [Previous line repeated 1 more time]
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 409, in _apply
    param_applied = fn(param)
```

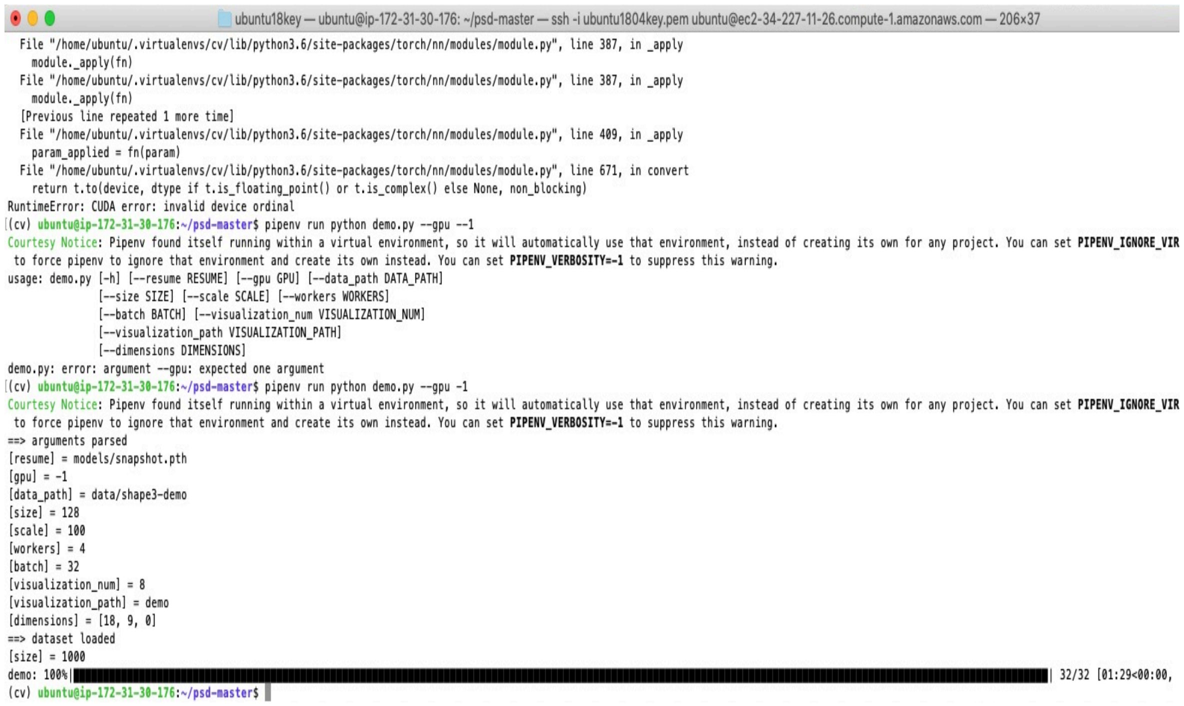
Figure 8. GPU was set to 1 device, yet there was a traceback error

There is no if condition within this code block on lines 41-43. The location of the load via torch points directly to a CPU and not a GPU. Furthermore, this line of code is exactly where the code failed for the GPU loading. Therefore, the researcher wanted to experiment with a -1 GPU switch, setting the device to a CPU as it appeared the code would run with this switch. With the switch set as -1 as pipenv run python demo.py --GPU -1 set as a CPU, the code executed perfectly, and the output was given as in Figure 9, a 100% successful execution.

The researcher's knowledge extends within this exploration; an assumption was made that the code for the demo.py shapes project was tailored to a CPU and not a GPU. As the instance used was a potent AWS Tesla T4, the GPU is by default recognized as a CPU, meaning that the GPU is the primary device run. That, by default, could be the -1 flag invoking that GPU. One way to test this would be to run a GPU monitor while the data is invoked to check that the GPU is fired.

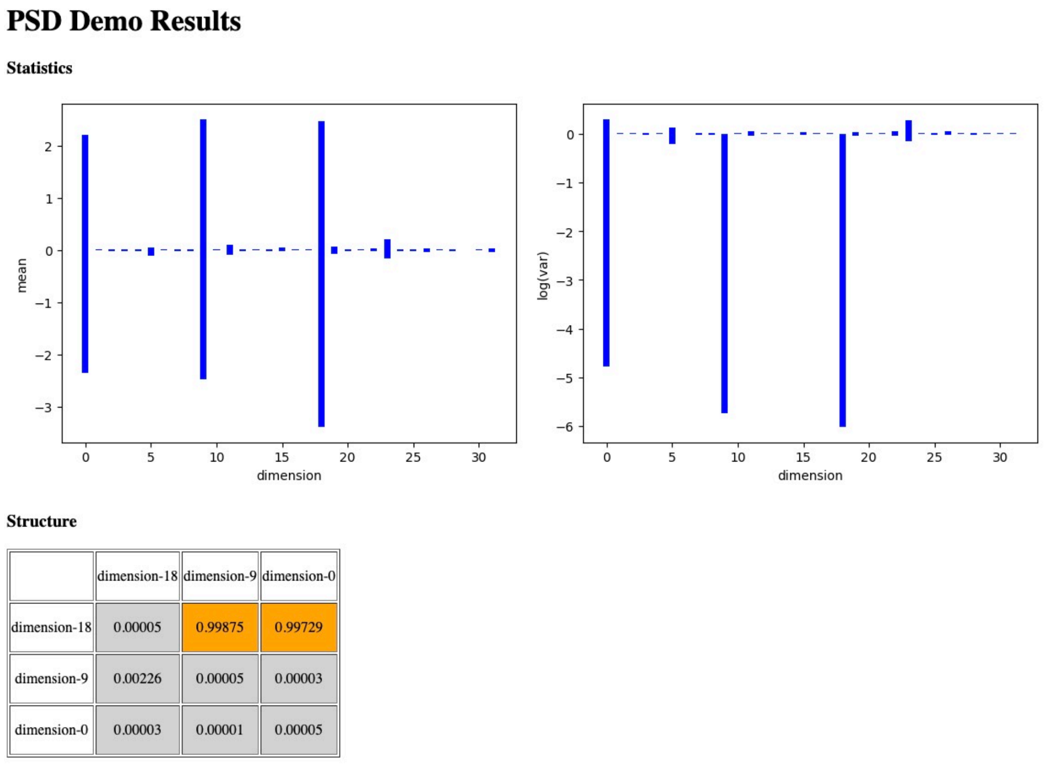
Figure 9 shows the 100% complete demonstration progress bar for running the demo.py file. The various arguments parsed can be seen. The toy shapes dataset was loaded, and the output is seen in Figures 9 and 10.

Figure 10 shows the working HTML file with the sourced images and the output. To date, the researcher has two other artifact constructs that are used. The previous analysis included annotating the existing code base for the shapes demonstration by Xue et al. (2019). The second is the study of the mappings between PyTorch and Java as a mapping table.


ubuntu18key — ubuntu@ip-172-31-30-176: ~/psd-master — ssh -i ubuntu1804key.pem ubuntu@ec2-34-227-11-26.compute-1.amazonaws.com — 206×37

```
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 387, in _apply
    module._apply(fn)
  [Previous line repeated 1 more time]
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 409, in _apply
    param_applied = fn(param)
  File "/home/ubuntu/.virtualenvs/cv/lib/python3.6/site-packages/torch/nn/modules/module.py", line 671, in convert
    return t.to(device, dtype if t.is_floating_point() or t.is_complex() else None, non_blocking)
RuntimeError: CUDA error: invalid device ordinal
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ pipenv run python demo.py --gpu --1
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIR
 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
usage: demo.py [-h] [--resume RESUME] [--gpu GPU] [--data_path DATA_PATH]
               [--size SIZE] [--scale SCALE] [--workers WORKERS]
               [--batch BATCH] [--visualization_num VISUALIZATION_NUM]
               [--visualization_path VISUALIZATION_PATH]
               [--dimensions DIMENSIONS]
demo.py: error: argument --gpu: expected one argument
(cv) ubuntu@ip-172-31-30-176:~/psd-master$ pipenv run python demo.py --gpu -1
Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIR
 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
==> arguments parsed
[resume] = models/snapshot.pth
[gpu] = -1
[data_path] = data/shape3-demo
[size] = 128
[scale] = 100
[workers] = 4
[batch] = 32
[visualization_num] = 8
[visualization_path] = demo
[dimensions] = [18, 9, 0]
==> dataset loaded
[size] = 1000
demo: 100%|████████████████████████████████████████████████████████████████████████████████| 32/32 [01:29<00:00,
(cv) ubuntu@ip-172-31-30-176:~/psd-master$
```

Figure 9. Code attached to the CPU implementation of shapes demo.py (Xue et al., 2019)



PSD Demo Results

Statistics

Structure

	dimension-18	dimension-9	dimension-0
dimension-18	0.00005	0.99875	0.99729
dimension-9	0.00226	0.00005	0.00003
dimension-0	0.00003	0.00001	0.00005

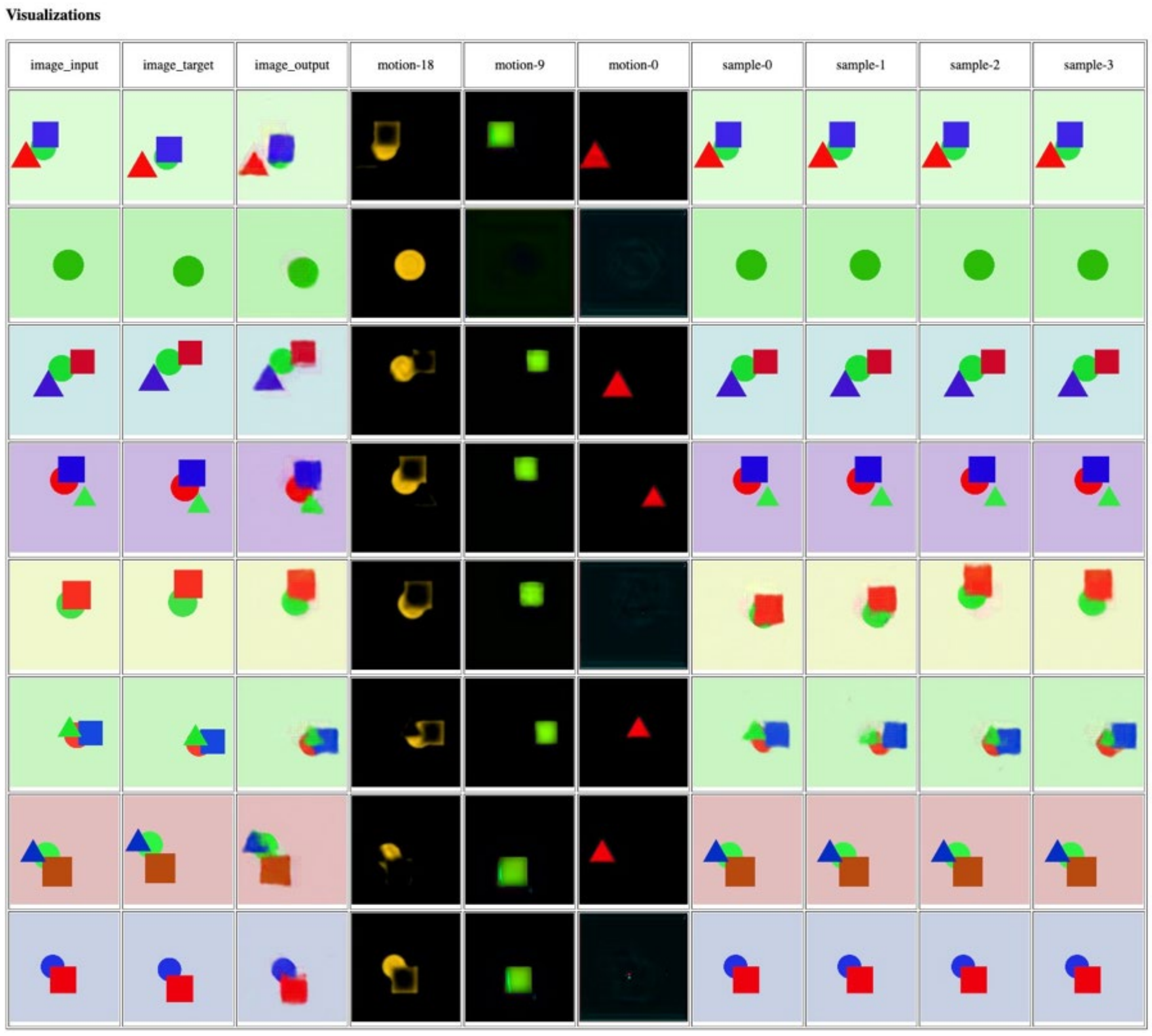


Figure 10. The shapes demonstration webpage output

Data were collected to get the demonstration programming code up and running. The shapes dataset used a different operating system version than the other exercise demonstration data set (Wu, 2020; Xue et al., 2019). The researcher wanted to replicate the exact version of the Ubuntu operating system to ensure the shapes demonstration would attempt to mirror the original implementation and hold the least risk of failure. By mirroring the implementation, the chances of success increased. An extensive number of libraries were required to be linked and imported via the Ubuntu system. There were many dependencies involved in implementing the work by Xue et al. (2019).

The shapes demonstration programming code was developed in PyTorch 3.17. A successful demonstration was invoked with the detailed steps. A code analysis was also performed, and a mapping table was provided between PyTorch and Java. The shapes dataset was examined line by line. To follow, to the best of the enterprise information systems researcher's knowledge, the dataset appeared to have been executed correctly. The output was generated along with the relevant directories. The processes used in the findings are included in Recommendations for Practitioners.

DISCUSSION

This design science study followed an iterative research cycle following the 12 main steps to conducting design science research (Dresch et al., 2014). The initial steps involve identifying the problem, systematic literature review, and identification of the artifacts. The design science approach enables a continuous feedback loop between a number of the research stages. The research involved a first pass look into the existing environments for artificial intelligence, such as Amazon Web Services.

The researcher overcame several hurdles through a series of steps. The configuration of the shapes dataset demonstration by Xue et al. (2019) was documented via a series of figures and commentary. The research showed a step-by-step strategy for implementing source code and executing that code within a legacy operating system via an Amazon Web Services image. A second pass was invoked of the design science paradigm. The source code was analyzed, and a mapping table was implemented, drawing from the PyTorch libraries and the existing Java libraries.

The proposed design science study explores the strategies enterprise information systems specialists need to integrate novel algorithms into an imaging application. The study problem was what strategies do enterprise information systems specialists need to integrate novel algorithms into an imaging application that were not identified (Rueden et al., 2017; Schneider et al., 2012). The conceptual framework details the audience and the research question problem and method. The conceptual framework encompassed the problem: a lack of novel algorithms within image processing software. The audiences were biologists and zoologists, and the method was the technological enhancements to an image application. The question within the conceptual framework was, "What are the strategies enterprise information systems specialists need to integrate novel algorithms into an imaging application?"

PRACTICAL IMPLICATIONS OF FINDINGS

The study findings were that executing an artificial intelligence CCNN does take a substantial amount of time. Also, a significant amount of knowledge is required to understand how that operating system would integrate with the dependency libraries. Often libraries were out of date and, once the dependency was downloaded, a more recent version would be retrieved from GitHub rather than the version needed. Appropriate, but the version would be out of sync for the project. The other issue was that the Ubuntu version used for the demonstration was mentioned within the source code. However, that version was out of date. How would the researcher implement such a demonstration with no working knowledge, having never performed such a study before?

The Amazon Web instance proved to be a considerable choice. The researcher could select from many different environments. The choice of the Tesla T4 instance was the preferred option due to its use of the NVIDIA GPU Tensor Core. The AWS instance took approximately four days to invoke, and an application form had to be submitted and approved by the provider. Once granted, the researcher waited another day for further provisioning and then was able to move forward and start the instance and action development.

Furthermore, once the AWS instance type had been selected, the linkages to the libraries were a significant issue. The time-consuming nature of getting all the libraries to run together with an out-of-date operating system was challenging. At times, there were few solutions to problems, and there were many solutions at other times. It was challenging to find the most appropriate solution that would best fit the situation.

Surprisingly, the practical forums online proved to offer several solutions. Meaningful solutions that had been tried and tested and that, in some cases, had also worked. Some of the data collection books purchased for the study were not used.

This study offers a roadmap – a set of processes as highlights and a guide to anyone wishing to set up the demonstration for the shapes dataset by Xue et al. (2019). The research offers the steps followed for setting up to re-create the environment. Furthermore, it offers the foundation and a fast track for future researchers who are novices to the field and to advance, building on the work of Xue et al. (2019).

RECOMMENDATIONS FOR PRACTITIONERS

The study problem addressed the strategies enterprise information system specialists needed to integrate novel algorithms into an imaging application that had not been identified (Rueden et al., 2017; Schneider et al., 2012). The study did indeed explore the aspects an enterprise information specialist would need to implement the work by Xue et al. (2019). The design science findings from the systemic literature review were that many software applications were available for image analysis, and the field that contributed the most significant amount of knowledge within image analysis was that of the healthcare sector. From analyzing the various software applications within Java for image analysis, a choice was made to focus on the ImageJ2 software implementation.

The ImageJ2 application offers the practitioner an extensive wealth of flexibility. The study findings imply that the practitioner has been provided with guidelines that can be implemented should issues arise. It is possible to implement the demonstration code from an advanced neural network algorithm by implementing an image about the Ubuntu version 18.04.5 release via an Amazon Web Services instance.

The academic knowledge presented resources regarding image analysis and pattern recognition. There were many avenues and approaches to analyzing an image. Some of the most significant focal points were the medical field and wildlife and biology – image analysis regarding the butterfly patterns and other implementations. The study initially investigated the history of butterfly evolution and identification and focused on research such as the Nymphalid Groundplan (Nijhout, 2001). A determination was made to use the Java open-source programming language for future programming work. The language of choice was also chosen as the enterprise information systems specialist was experienced in this language. It also was widely used within the field of image analysis.

An investigation into the literature incorporated the various state-of-the-art algorithms such as CCNNs and RNNs. The review also considered Content Image Based Recognition (CIBR) systems and the functionality offered by those systems. According to Litjens et al. (2017), there was a lack of pooling layers within the convolutional networking realm within CIBR systems. ImageJ and ImageJ2 were some of the most widely known open-source Java software applications. The study explored the various plugins available for the software.

The study also investigated the work of frameworks within Java that would offer a deep learning infrastructure functionality, such as that of DeepLearning4j (Murthy et al., 2020). Future research can also use this investigation and build upon the existing DeepClass4Bio API that can map to various bio-tools such as ImagePy and Icy as a bridge to PyTorch. ImageJ plugins used TensorFlow as their foundation for neural networks. Furthermore, various image analysis systems for butterflies were also investigated.

The healthcare sector DICOM standard allowed for consistency of images with a standard implemented by the NIH, and standardized metadata was implemented with compliant DICOM devices. There was a lack of consistency between other software imaging applications due to standardization. The standardization implemented in the health care industry could offer many avenues for the forward progression of butterfly image recognition. ImageJ2 was identified as an area that required further development. Suggestions were given for algorithmic development within ImageJ2; it was determined that often existing software was out of date with those novel algorithms (Schindelin et al., 2015).

The work of Wu (2020) and Xue et al. (2019) offered the potentiality of a layman using a state-of-the-art novel algorithm, the cross-convolutional neural network, to predict image motion. This study implies that the information contained can act as a guide for implementation to the novice user within the realm of deep learning in artificial intelligence. It aims to understand that it is possible to invoke complex algorithms from within the home office laboratory through the various servers available via the internet.

To a novice practitioner and academic, the inferences reached within this study were to demonstrate to those with a home office laboratory that an intense study such as this can be replicated. The issues were worked through one by one as provided in the guidelines in chapter 4 of Quindazzi (2021). It is possible to implement complex artificial intelligence applications as an enterprise information systems specialist with little knowledge of neural network systems as per the results section of chapter 4. Recommendation 1 details the highlights of the processes used in the research study.

RECOMMENDATION 1

Highlights of the Processes Used

1) The researcher used the internet to search for those keywords when occurrences happened with linkage and dependency library errors. The work by Byake (2019) with the pipenv command not found. The discovered web page gave a solution, and the study was able to move forward.
2) When the researcher found gaps in their knowledge, such as the Amazon Web Service instances, the researcher went to the website and read the user guide of the services.
3) Torch7 assumptions were drawn between the old code and the new. PyTorch was based on the inactive Torch7 libraries and was extensively documented. Parallels were drawn between the old and new languages available.
4) A detailed design science paradigm was used iteratively for the steps involved (Dresch et al., 2014).
5) Previous experience from the researcher's background was implemented with the mapping table from PyTorch to Java. A previous project in the industry that the researcher had implemented was an API bridge between two languages, and the ideas were taken from this and implemented in this project.
6) Fact-finding and understanding of a new language were implemented with a line-by-line analysis of the source code from the shapes dataset. A dissection of the details for the core implementations in PyTorch. Comments were annotated to the right of the source code and included in Appendices B and C in the study by Quindazzi (2021).
7) An analysis of the two demonstration forms of software, the exercise data set and the shapes data set, was invoked when issues arose with the GPU. A comparison of what source code worked on one demonstration to what did not work on the other demonstration was compared, and an inferred conclusion was drawn.

CONCLUSION

This design science exploratory study aimed to explore the strategies enterprise information systems specialists needed to integrate novel algorithms into an imaging application. The study problem addressed the strategies enterprise information system specialists needed to integrate novel algorithms into an imaging application that had not been identified. The study used an AWS instance to run the demonstration code by Xue et al. (2019). The research determined a roadmap for implementing the CCNN for motion prediction.

The findings consisted of approaches 1 through to 7 detailed in the study by Quindazzi (2021). The steps advised on the strategies taken when the researcher encountered an issue. The steps ranged from dependency issues with libraries and linkages to the approach and design science paradigm that the study took. Other findings were gaps in the researcher's knowledge, such as a lack of understanding of Amazon Web Services and adopting a practitioner, tried and tested experiential approach to resolving gaps from one language to another, PyTorch to Java.

The findings were documented via figures showing the issues with linkages and libraries. The study results indicated that it was possible to recreate the environment for an advanced neural network system with little knowledge of artificial intelligence systems. The study was explored from an enterprise information system architect's perspective. The results also showed that academic research could meet practice. The practice forums were used to obtain solutions to run the demonstration code and work through issues.

The key takeaways were that the hardware was very accessible to date. Should advanced research need to occur without such a sophisticated lab, the hardware was no longer a limiting factor. With the advent of the cloud AWS, Lambda, and many other server images, the opportunities to have such robust systems as pay to play are right at the researcher's fingertips. Many exciting discoveries are just on the horizon, with more and more areas of research broadening.

This study implies that programmers can use this study and work through it from start to finish to get the demonstration program up online, to invoke the ImageJ software with the plugin features for the Hello World software as also demonstrated. The hope is that the existing exploration will further the field of scientific research to contribute to the body of knowledge. The main objective is that the fellow researcher is brought up to speed quickly with this document, will further research in artificial intelligence for enterprise information systems specialists, and contribute further to the body of knowledge.

FUTURE RESEARCH

Based on the findings of the implementation of the work by Xue et al. (2019), the following recommendations are made for future research.

FUTURE RESEARCH RECOMMENDATION 1

In this study, the works by Xue et al. (2019) and Wu (2020) were analyzed. A future researcher could use this study as a basis to get the previous work up and running, and this can be used as a basis to build upon the research further.

FUTURE RESEARCH RECOMMENDATION 2

The other novel algorithms could be implemented using new concepts for processing images in this study.

FUTURE RESEARCH RECOMMENDATION 3

In this study, further mappings from PyTorch to Java could be implemented (Quindazzi, 2021). These mappings could be developed in Java and run within the Java system, moving away from PyTorch.

FUTURE RESEARCH RECOMMENDATION 4

In this study, a future researcher could implement ImageJ2 and add it with different flavors of CCNN algorithms and different algorithms within Java for image processing.

FUTURE RESEARCH RECOMMENDATION 5

This study presented concepts with butterflies and the nymphalid groudplan (Nijhout, 2001). This discussion could provide further image prediction work based on butterfly wing patterns.

Future research should focus on implementing the mappings table (Quindazzi, 2021). A future design could be pursued from the analysis and implementation of a Java neural network algorithm as a basic implementation. As suggested, Java can chain elements to begin replicating the algorithm by Xue et al. (2019). The cross-convolutional element should have further analysis to ensure the full replication within Java.

The future study should focus on ImageJ and the core code within the software; the preliminary steps were provided in the work by Quindazzi (2021). The researcher could build upon this study using the libraries such as the Hello World trial code in Eclipse and deciphering where a CCNN Java algorithm would best reside. The algorithm could then be implemented and trial run within the code; this would allow for the leverage of the heavy lifting of the program code to predict image motion. There seemed to be few approaches regarding predicting motion frames of a future image. Such predictions could be a giant leap for the health care world regarding microscopic and x-ray images.

References

Almryad, A. S., & Kutucu, H. (2020). Automatic identification for field butterflies by convolutional neural networks. *Engineering Science and Technology, an International Journal*, *23*(1), 189-195. https://doi.org/10.1016/j.jestch.2020.01.006

Amidi, A., & Amidi, S. (2021). *Convolutional neural networks cheat sheet.* https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks#filter

Amir. (2019, January 18). *Loading Torch7 trained models (.t7) in PyTorch.* https://stackoverflow.com/questions/41861354/loading-torch7-trained-models-t7-in-pytorch

Antoniadis, A., Filippakis, N., Krishnan, P., Ramesh, R., Allen, N., & Smaragdakis, Y. (2020). Static analysis of Java enterprise applications: Frameworks and caches, the elephants in the room. *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 794-807. https://doi.org/10.1145/3385412.3386026

Arena, E. T., Rueden, C. T., Hiner, M. C., Wang, S., Yuan, M., & Eliceiri, K. W. (2016). Quantitating the cell: Turning images into numbers with ImageJ. *Wiley Interdisciplinary Reviews: Developmental Biology*, *6*(2), e260. https://doi.org/10.1002/wdev.260

Arul. (2021, April 19). *Loading Torch7 trained models (.t7) in PyTorch.* https://stackoverflow.com/questions/41861354/loading-torch7-trained-models-t7-in-pytorch

Audet, S. (2021, April 15). *How can I increase the dimensions of tensor in Java?* https://stackoverflow.com/questions/63877136/how-can-i-increase-the-dimensions-of-tensor-in-java

AWS. (2021). *AWS Deep Learning Base AMI (Ubuntu 18.04).* https://aws.amazon.com/marketplace/pp/B07Y3VDBNS

Barbastathis, G. (2020). On the use of deep learning for computational imaging. *Optical Trapping and Optical Micromanipulation XVII.* https://doi.org/10.1117/12.2571322

Beldade, P., & Brakefield, P. M. (2002). The genetics and evo-devo of butterfly wing patterns. *Nature Reviews Genetics*, *3*(6), 442-452. https://doi.org/10.1038/nrg818

Bishop, C. M. (2006). *Pattern recognition and machine learning.* Springer Verlag.

Burger, W., & Burge, M. J. (2016). *Digital image processing: An algorithmic introduction using Java.* Springer. https://doi.org/10.1007/978-1-4471-6684-9

Byake. (2019, May 3). *How to properly install Pipenv on WSL Ubuntu 18.04?* https://superuser.com/questions/1432768/how-to-properly-install-pipenv-on-wsl-ubuntu-18-04

Clcarwin. (2017, November 13). *Convert torch to pytorch.* https://github.com/clcarwin/convert_torch_to_pytorch

Dresch, A., Lacerda, D. P., & Antunes, J. A. V., Jr. (2014). *Design science research: A method for science and technology advancement.* Springer. https://doi.org/10.1007/978-3-319-07374-3

Du, J. (2018). Understanding of object detection based on CNN family and YOLO. *Journal of Physics: Conference Series*, *1004*, 012029. https://doi.org/10.1088/1742-6596/1004/1/012029

The Eclipse Foundation. (2021). *IDE.* [Computer software]. https://www.eclipse.org/ide/

Franklin, D. (2021, March 5). *No CUDA runtime is found, using CUDA_HOME='/usr/local/cuda'.* https://forums.developer.nvidia.com/t/no-cuda-runtime-is-found-using-cuda-home-usr-local-cuda/170143

Futahashi, R., Shirataki, H., Narita, T., Mita, K., & Fujiwara, H. (2012). Comprehensive microarray-based analysis for stage-specific larval camouflage pattern-associated genes in the swallowtail butterfly, *Papilio xuthus*. *BMC Biology*, *10*(1), 46. https://doi.org/10.1186/1741-7007-10-46

Gilbert, M. (2018). *Artificial intelligence for autonomous networks*. CRC Press. https://doi.org/10.1201/9781351130165

Gomez-Perez, S. L., Haus, J. M., Sheean, P., Patel, B., Mar, W., Chaudhry, V., McKeever, L., & Braunschweig, C. (2016). Measuring abdominal circumference and skeletal muscle from a single cross-sectional computed tomography image. *Journal of Parenteral and Enteral Nutrition*, *40*(3), 308-318. https://doi.org/10.1177/0148607115604149

Holton, J. A., & Walsh, I. (2016). *Classic grounded theory: Applications with qualitative and quantitative data*. SAGE Publications. https://doi.org/10.4135/9781071802762

Iacob. (2021, January 21). *What does "unsqueeze" do in Pytorch?* https://stackoverflow.com/questions/57237352/what-does-unsqueeze-do-in-pytorch

Inés, A., Domínguez, C., Heras, J., Mata, E., & Pascual, V. (2019). DeepClas4Bio: Connecting bioimaging tools with deep learning frameworks for image classification. *Computers in Biology and Medicine*, *108*, 49-56. https://doi.org/10.1016/j.compbiomed.2019.03.026

Kainz, P., Mayrhofer-Reinhartshuber, M., & Ahammer, H. (2015). IQM: An extensible and portable open source application for image and signal analysis in Java. *PLOS ONE*, *10*(1), e0116329. https://doi.org/10.1371/journal.pone.0116329

Kankaanpää, P., Paavolainen, L., Tiitta, S., Karjalainen, M., Päivärinne, J., Nieminen, J., Marjomäki, V., Heino, J., & White, D. J. (2012). BioImageXD: An open, general-purpose and high-throughput image-processing platform. *Nature Methods*, *9*(7), 683-689. https://doi.org/10.1038/nmeth.2047

Karim, M. R. (2018). *Java deep learning projects: Implement ten real-world deep learning applications using Deeplearning4j and open source APIs*. Packt Publishing.

Karim, M. R., Beyan, O., Zappa, A., Costa, I. G., Rebholz-Schuhmann, D., Cochez, M., & Decker, S. (2020). Deep learning-based clustering approaches for bioinformatics. *Briefings in Bioinformatics*, *22*(1), 393-415. https://doi.org/10.1093/bib/bbz170

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. https://doi.org/10.1007/978-1-4614-6849-3

Lin, Z., Jia, J., Gao, W., & Huang, F. (2019). Increasingly specialized perception network for fine-grained visual categorization of butterfly specimens. *IEEE Access*, *7*, 123367-123392. https://doi.org/10.1109/ACCESS.2019.2938537

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A., Ciompi, F., Ghafoorian, M., Van der Laak, J. A., Van Ginneken, B., & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, *42*, 60-88. https://doi.org/10.1016/j.media.2017.07.005

Liu, C. (2009). *Beyond pixels: exploring new representations and applications for motion analysis* (Doctoral dissertation, Massachusetts Institute of Technology).

Mallet, J. (2001). Mimicry: An interface between psychology and evolution. *Proceedings of the National Academy of Sciences*, *98*(16), 8928-8930. https://doi.org/10.1073/pnas.171326298

Matlab. (n.d.). *MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink*. https://www.mathworks.com/products/matlab.html?s_tid=hp_ff_p_matlab

Murthy, C. B., Hashmi, M. F., Bokde, N. D., & Geem, Z. W. (2020). Investigations of object detection in images/Videos using various deep learning techniques and embedded platforms – A comprehensive review. *Applied Sciences*, *10*(9), 3280. https://doi.org/10.3390/app10093280

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., López García, Á., Heredia, I., Malík, P., & Hluchý, L. (2019). Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey. *The Artificial Intelligence Review*, *52*(1), 77-124. https://doi.org/10.1007/s10462-018-09679-z

Nijhout, H. F. (1980). Pattern formation on lepidopteran wings: Determination of an eyespot. *Developmental Biology*, *80*(2), 267-274. https://doi.org/10.1016/0012-1606(80)90403-0

Nijhout, H. F. (2001). Elements of butterfly wing patterns. *Journal of Experimental Zoology*, *291*(3), 213-225. https://doi.org/10.1002/jez.1099

Oei, R. W., Hou, G., Liu, F., Zhong, J., Zhang, J., An, Z., Xu, L., & Yang, Y. (2019). Convolutional neural network for cell classification using microscope images of intracellular actin networks. *PLOS ONE*, *14*(3), e0213626. https://doi.org/10.1371/journal.pone.0213626

Pavithra, L., & Sharmila, T. S. (2019). Optimized feature integration and minimized search space in content-based image retrieval. *Procedia Computer Science*, *165*, 691-700. https://doi.org/10.1016/j.procs.2020.01.065

Popek, M. P., & Iskander, D. R. (2020). A new approach to the phase-based video motion magnification for measuring Microdisplacements. *IEEE Transactions on Instrumentation and Measurement*, *69*(2), 354-361. https://doi.org/10.1109/TIM.2019.2904074

Quindazzi, E. (2021). *Exploring strategies, enterprise information systems specialists need to integrate novel algorithms into an imaging application* [Doctoral dissertation, Colorado Technical University]. https://www.emma-quindazzi.com/doctoral/Exploring_Strategies_Enterpris.pdf

Roland, J. K. (2015). A study of slanted-edge MTF stability and repeatability. *Image Quality and System Performance XII*. https://doi.org/10.1117/12.2077755

Rueden, C. T., & Schindelin, J. (n.d.). *ImageJ tutorials repository for the commands, simple branch* [Computer software]. https://imagej.net/develop/plugins

Rueden, C. T., Schindelin, J., Hiner, M. C., DeZonia, B. E., Walter, A. E., Arena, E. T., & Eliceiri, K. W. (2017). ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics*, *18*(1). https://doi.org/10.1186/s12859-017-1934-z

Sage, D., & Unser, M. (2003). Teaching image-processing programming in Java. *IEEE Signal Processing Magazine*, *20*(6), 43-52. https://doi.org/10.1109/msp.2003.1253553

Schindelin, J., Rueden, C. T., Hiner, M. C., & Eliceiri, K. W. (2015). The ImageJ ecosystem: An open platform for biomedical image analysis. *Molecular Reproduction and Development*, *82*(7-8), 518-529. https://doi.org/10.1002/mrd.22489

Schneider, C. A., Rasband, W. S., & Eliceiri, K. W. (2012). NIH image to ImageJ: 25 years of image analysis. *Nature Methods*, *9*(7), 671-675. https://doi.org/10.1038/nmeth.2089

Suleymanova, I., Balassa, T., Tripathi, S., Molnar, C., Saarma, M., Sidorova, Y., & Horvath, P. (2018). A deep convolutional neural network approach for astrocyte detection. *Scientific Reports*, *8*(1), 1-7. https://doi.org/10.1038/s41598-018-31284-x

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer Science & Business Media.

Tan, E. J., Wilts, B. D., Tan, B. T., & Monteiro, A. (2020). What's in a band? The function of the color and banding pattern of the banded swallowtail. *Ecology and Evolution*, *10*(4), 2021-2029. https://doi.org/10.1002/ece3.6034

Wang, X., Zhao, Y., & Pourpanah, F. (2020). Recent advances in deep learning. *International Journal of Machine Learning and Cybernetics*, *11*(4), 747-750. https://doi.org/10.1007/s13042-020-01096-5

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer. https://doi.org/10.1007/978-3-662-43839-8

Wolfgang, M., Weißensteiner, M., Clarke, P., Hsiao, W., & Khinast, J. G. (2020). Deep convolutional neural networks: Outperforming established algorithms in the evaluation of industrial optical coherence tomography (OCT) images of pharmaceutical coatings. *International Journal of Pharmaceutics: X*, *2*, 100058. https://doi.org/10.1016/j.ijpx.2020.100058

Wu, J. (2020). *Learning to see the physical world* (Doctoral dissertation, Massachusetts Institute of Technology).

Xue, T., Wu, J., Bouman, K. L., & Freeman, W. T. (2019). Visual dynamics: Stochastic future generation via layered cross convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(9), 2236-2250. https://arxiv.org/pdf/1807.09245v3.pdf

Zahidi, Y., Younoussi, Y. E., & Al-Amrani, Y. (2021). A powerful comparison of deep learning frameworks for Arabic sentiment analysis. *International Journal of Electrical and Computer Engineering, 11*(1), 745-752. https://doi.org/10.11591/ijece.v11i1.pp745-752

Zhang, W., Yang, G., Lin, Y., Ji, C., & Gupta, M. M. (2018, June). On definition of deep learning. *2018 World Automation Congress (WAC), Stevenson, Washington*. https://doi.org/10.23919/WAC.2018.8430387

Zhang, Y., Chen, X., Guo, D., Song, M., Teng, Y., & Wang, X. (2019). PCCN: Parallel cross convolutional neural network for abnormal network traffic flows detection in multi-class imbalanced network traffic flows. *IEEE Access*, *7*, 119904-119916. https://doi.org/10.1109/ACCESS.2019.2933165

APPENDIX

The Initialization Component Analysis of demo.py Analysis, adapted from Xue et al. (2019).

```
import argparse
import os
import matplotlib.pyplot as plt
import numpy as np
import torch from torch.utils.data
import DataLoader from tqdm
import tqdm from model
import PSD from data
import MotionDataset from utils
import mkdir, flow2im, imwrite
import dominate from dominate.tags
import *
def main(args):
  # set device (cpu / gpu)
  if args.gpu == '-1': //take the command line args import no gpu but a cpu
    device = torch.device('cpu')
  else: //else take the argument on the command line and put into a cuda array
    device = torch.device('cuda:{}'.format(args.gpu)) //call and load it as a torch device
  data, loaders = {}, {} //set up the data and loaders array
  data['demo'] = MotionDataset(
    data_path = args.data_path,
    split = 'demo',
    size = args.size,
    scale = args.scale
  ) //call the MotionDataset class, load in the path in for the demo into the data['demo'] location
  loaders['demo'] = DataLoader(
        dataset = data['demo'],
        batch_size = args.batch,
        shuffle = True,
        num_workers = args.workers
    ) //call the dataloader class
    print('==> dataset loaded') //print to the screen = => dataset loaded
    print('[size] = {0}'.format(len(data['demo']))) //print the length of data['demo']
    model = PSD(dimensions=args.dimensions, size=args.size) //call the PSD class model.py holds
this class – details are in the code to follow
    model.load_state_dict(torch.load(args.resume, map_location='cpu')) //a stat_dict python dic-
tionary object where the layers mapped to its parameter tensor such as a convolutional layer
    model.to(device)
    structure = torch.sigmoid(model.structural_descriptor.structure).data.cpu().numpy()//the sig-
moid function takes the structure as a tensor input. Structure is the new returned tensor with sig-
moid. Cpu copies the tensor to the cpu but if it is already on the cpu nothing will change. The
numpy function call creates a numpy type array taken from the tensor. The tensor and numpy ar-
ray share memory.
     visualization_path = args.visualization_path
    mkdir(visualization_path, clean=True) //make a directory of virtualization path set clean to
true
    figure_path = os.path.join(visualization_path, 'figures') //join the virtualization path to the
word figures
    mkdir(figure_path, clean=True) //make a directory figure_path
    with torch.no_grad():
        means, log_vars = [], []
        for batch in tqdm(loaders['demo'], desc = 'demo'):
            batch = {k: v.to(device) for k, v in batch.items()}
            batch_size = batch['image_inputs'].size(0)
            total_size = len(data['demo'])
            outputs= model.forward(
                image_inputs=batch['image_inputs'],
                flow_inputs=batch['flow_inputs'],
                returns = ['mean', 'log_var']
            ) //calls model.forward inputs image, flow_inputs and log_var. Calculate output tensors
from input tensors
            means.extend(outputs['mean'].data.cpu().numpy())//may append the array parameter to
the end of the target array
            log_vars.extend(outputs['log_var'].data.cpu().numpy())
    means = np.asarray(means)
    log_vars = np.asarray(log_vars)
    x, ym, yv = [], [], []//set the arrays
    for k in range(means.shape[1]):
        x.extend([k, k])
        ym.extend([np.min(means[:, k]), np.max(means[:, k])]) //extend from min to max
        yv.extend([np.min(log_vars[:, k]), np.max(log_vars[:, k])])
     plt.switch_backend('agg') //all matplotlib pyplot functions -> the output for the html page
for the demo
    plt.figure()
    plt.bar(x, ym, .5, color = 'b')
    plt.xlabel('dimension')
    plt.ylabel('mean')
    plt.savefig(os.path.join(figure_path, 'means.png'), bbox_inches = 'tight')
    plt.figure()
    plt.bar(x, yv, .5, color = 'b')
    plt.xlabel('dimension')
    plt.ylabel('log(var)')
    plt.savefig(os.path.join(figure_path, 'vars.png'), bbox_inches = 'tight')
    batch = iter(loaders['demo']).next()
    batch = {k: v[:args.visualization_num].to(device) for k, v in batch.items()}
    batch_size = batch['image_inputs'].size(0)
    outputs = outputs= model.forward(
        image_inputs=batch['image_inputs'],
        flow_inputs=batch['flow_inputs']
    )
    samples = []
    for k in range(4):
        indices = np.random.choice(len(data['demo']), batch_size)
        sample = model.forward(
            image_inputs=batch['image_inputs'],
            mean = torch.tensor(means[indices], device=device), //feed torch tensor
            log_var = torch.tensor(log_vars[indices], device=device),
        )
        samples.append(sample['image_outputs'].cpu().numpy())
        vis_image_inputs = batch['image_inputs'].cpu().numpy()
        vis_image_targets = batch['image_targets'].cpu().numpy()
        vis_image_outputs = outputs['image_outputs'].cpu().numpy()
        vis_motions = outputs['motion_outputs'].cpu().numpy()
        for i in range(args.visualization_num):
            imwrite(os.path.join(figure_path, '{}_image_input.png'.format(i)), vis_image_inputs[i])
            imwrite(os.path.join(figure_path, '{}_image_target.png'.format(i)), vis_image_targets[i])
            imwrite(os.path.join(figure_path, '{}_image_output.png'.format(i)), np.clip(vis_image_out-
puts[i], 0, 1))
            for dim in args.dimensions:
                imwrite(os.path.join(figure_path, '{}_motion_{}.png'.format(i, dim)), flow2im(vis_mo-
tions[i, :, dim, ...]))
            for k in range(4):
                imwrite(os.path.join(figure_path, '{}_sample_{}.gif'.format(i, k)), [vis_image_inputs[i],
np.clip(samples[k][i], 0, 1)])
    with dominate.document(title='PSD') as web: //use dominate to setup the html page, headers,
images etc
        h1('PSD Demo Results')
        h3('Statistics')
        img(src=os.path.join('figures', 'means.png'))
        img(src=os.path.join('figures', 'vars.png'))
        h3('Structure')
        with table(border=1, style='table-layout: fixed;'): //setup a table with content
            with tr():
                with td(style='word-wrap: break-word;', halign='center', align='center',):
                    p('')
                for y in args.dimensions:
                    with td(style='word-wrap: break-word;', halign='center', align='center',):
                        p('dimension-{}'.format(y))
            for x in args.dimensions:
                with tr():
                    with td(style='word-wrap: break-word;', halign='center', align='center',):
                        p('dimension-{}'.format(x))
                    for y in args.dimensions:
                        value = structure[x][y]
                        bgcolor = 'Orange' if value > 0.5 else 'LightGray'
                    with td(style='word-wrap: break-word;', halign='center', align='center',
bgcolor=bgcolor):
                        p('%.5f' % value)
        h3('Visualizations')
        cols = ['image_input', 'image_target', 'image_output']
        with table(border=1, style='table-layout: fixed;'):
            with tr():
                for col in cols:
                    with td(style='word-wrap: break-word;', halign='center', align='center',):
                        p(col)
                for dim in args.dimensions:
                    with td(style='word-wrap: break-word;', halign='center', align='center',):
                        p('motion-{}'.format(dim))
                for k in range(4):
                    with td(style='word-wrap: break-word;', halign='center', align='center',):
                        p('sample-{}'.format(k))
            for id in range(args.visualization_num):
                with tr():
                    for col in cols:
                        with td(style='word-wrap: break-word;', halign='center', align='top'):
                            img(style='width:128px', src=os.path.join('figures', '{}_{}.png'.format(id, col)))
                    for dim in args.dimensions:
                        with td(style='word-wrap: break-word;', halign='center', align='center',):
                            img(style='width:128px', src=os.path.join('figures', '{}_motion_{}.png'.for-
mat(id, dim)))
                    for k in range(4):
                        with td(style='word-wrap: break-word;', halign='center', align='center',):
                            img(style='width:128px', src=os.path.join('figures', '{}_sample_{}.gif'.format(id,
k)))

    with open(os.path.join(visualization_path, 'index.html'), 'w') as fp:
        fp.write(web.render())
if __name__=='__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--resume', default = 'models/snapshot.pth')
    parser.add_argument('--gpu', default = '0') //gpu defaults to zero
parser.add_argument('--data_path', default = 'data/shape3-demo')
parser.add_argument('--size', default = 128, type = float)
parser.add_argument('--scale', default = 100, type = float)
parser.add_argument('--workers', default = 4, type = int)
parser.add_argument('--batch', default = 32, type = int)
parser.add_argument('--visualization_num', default = 8, type = int)
parser.add_argument('--visualization_path', default = 'demo')
parser.add_argument('--dimensions', default = '18,9,0')
args = parser.parse_args()
if args.dimensions is not None:
    args.dimensions = [int(x) for x in args.dimensions.split(',')]
print('==> arguments parsed')
for key in vars(args):
    print('[{0}] = {1}'.format(key, getattr(args, key)))

main(args)
```

AUTHORS

Dr. Emma Quindazzi works as a lead technical consultant at Infosys and works on complex Java development client projects for some of the world's leading enterprises. Emma has taught at Johns Hopkins Center for Talented Youth for several years. She has over 20 years of experience in software development within varying sectors such as banking, telecommunications, and Big Four accounting. Emma attained her Doctor of Computer Science from Colorado Technical University; U.S.A. Emma studied for her MSc Software Engineering at the University of the West of England in Bristol, England. Her first degree was a BSc (Hons) in Computing and Informatics at the University of Plymouth in Devon, England. Computer science is her passion.

Dr. Samuel Sambasivam is Chair and Professor of Computer Science Data Analytics at Woodbury University, Burbank, CA. He is Chair Emeritus and Professor Emeritus of Computer Science at Azusa Pacific University. He served as a Distinguished Visiting Professor of Computer Science at the United States Air Force Academy in Colorado Springs, Colorado for two years. In addition, he has concurrently served 13 years of progressive doctoral teaching roles at Colorado Technical University (CTU) including chair of doctoral programs, lead computer science doctoral faculty instructing core/concentration computer science courses, and as dissertation chair/committee member. His research interests include Cybersecurity, Big Data Analytics, Optimization Methods, Expert Systems, Client/Server Applications, Database Systems, and Genetic Algorithms. He has conducted extensive research, written for publications, and delivered presentations in Computer Science, data structures, and Mathematics. Dr. Samuel Sambasivam earned his Ph.D. in Mathematics/Computer Science from Moscow State University, a Master of Science in Computer Science with Honors from Western Michigan University, Pre-PhD in Mathematics/Computer Science from Indian Institute of Technology (IIT) Delhi, a Master of Science Education in Mathematics with Honors from Mysore University (NCERT-Delhi), and a Bachelor of Science in Mathematics/Physics/Chemistry with Honors from the University Madras (Chennai). He is a voting senior member of the ACM.