A Comparative Study of Machine Learning Software Packages for the Morphological Classification of Radio Galaxies

Madhia Shabih
Computer Science Department
Stellenbosch University
Stellenbosch, South Africa
24397644@sun.ac.za

Supervisor: Trienko Grobler
Computer Science Department
Stellenbosch University
Stellenbosch, South Africa
tlgrobler@sun.ac.za

ABSTRACT

This study presents a comparative analysis of three prominent machine learning software packages—PyTorch, TensorFlow, and JAX (Just After eXecution)—focusing on their application to morphological classification of radio galaxies. Their performance was evaluated across multiple metrics: model accuracy, F1-score, training and inference times, memory utilization, and API design and usability. For radio galaxy classification, four state-of-the-art architectures were employed (ConVxpress, First-class, MCRGNet, and Toothless) and tested on the MiraBest and FR-DEEP datasets. Additionally, a simple convolutional neural network architecture was employed on the MNIST dataset as a standardized benchmark. Due to implementation constraints, JAX evaluations were limited to the MNIST experiments. Results reveal distinct advantages for each software package: TensorFlow demonstrates superior training efficiency, PyTorch excels in memory management and developer experience, while JAX achieves the fastest inference times, particularly beneficial for real-time classification tasks. This systematic comparison provides empirically-grounded insights into framework-specific trade-offs, offering practical guidance for researchers and practitioners in selecting appropriate tools for radio galaxy classification and similar computer vision tasks.

1 INTRODUCTION

Large datasets are being generated by existing and upcoming radio telescopes. These datasets are a source of astrophysical phenomena that may lead to new discoveries (Ndung'u et al. 2023). Radio telescopes detect and record radio waves. The Square Kilometre Array (SKA) is expected to generate data at Exabyte rates. Other similar projects include MeerKAT, which generates raw data at 2.2 TB/s (Booth and Jonas 2012) and the Murchison Widefield Array (MWA), which generates data rates of 300 GB/s (Lonsdale et al. 2009). Additionally, the LOw-Frequency ARray (LOFAR) generates raw data at 13 TB/s (Haarlem et al. 2013). Furthermore, it is estimated that the ongoing Evolutionary Map of the Universe (EMU) will eventually detect up to 70 million radio sources (R. Norris et al. 2011) and SKA is expected to discover more than 500 million radio sources (R. Norris et al. 2011).

The morphological classification of radio sources is critical in radio astronomy. It aids scientists in understanding the physical properties and characteristics of celestial objects based on their form and structure (Ndung'u et al. 2023). The aforementioned dataset sizes and rates, however, imply that manual classification is not a viable option anymore and the exploitation of machine learning techniques is unavoidable.

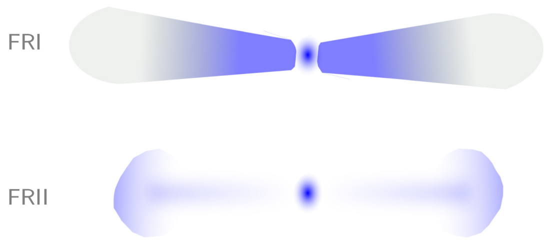

Figure 1: A typical Fanaroff Riley I & II classification of radio galaxies from Ndung'u et al. 2023.

Radio galaxies are galaxies that are only observable in the radio spectrum and have extended lobes and jets. They are visible due to radio emissions that are generated via active supermassive black holes at their centres. Fanaroff and Riley (1974) proposed to classify these galaxies into two major families.

- Fanaroff Riley I (FR-I)
  These are centre-brightened galaxies with one or two lobes.
- Fanaroff Riley II (FR-II)
  These are composed of edged-brightened lobes separated by a core at the centre.

The Fanaroff and Riley class dichotomy is visually depicted in Figure 1.

Deep learning is a type of machine learning that is commonly used in applications like image recognition and language processing. A deep learning neural network that shows great promise in performing radio galaxy morphological classification is a Convolutional Neural Network (CNN) (Krizhevsky, Sutskever, and Hinton 2017). This is a type of neural network that extracts its own features to perform classification. It has three main types of layers (Géron 2019):

- **Convolutional Layer:** This layer allows the network to concentrate on small low-level features (such as lines or simple textures) in the first layer, then assemble them into larger, higher-level features (such as shapes) in subsequent layers.
- **Pooling Layer:** This layer reduces the size of the input image in order to reduce the computational load, the memory usage and the number of parameters. This limits the risk of over-fitting.
- **Fully Connected Layer:** The fully connected layer integrates various features extracted in the convolutional and pooling layers and maps them to a specific outcome or class.

Figure 2 provides an abstraction of CNN functionality.

Three prominent software packages that can be utilized to implement a CNN are:

- **Tensorflow**[1]**:** A comprehensive machine learning platform developed by Google, known for its production-ready deployment capabilities and wide ecosystem support.
- **PyTorch**[2]**:** A dynamic, intuitive deep learning software package created by Facebook, favored for its ease of use in research and experimentation.
- **Just After eXecution (JAX**[3]**):** A high-performance numerical computing library that combines NumPy's familiar application programming interface (API) with graphics processing unit (GPU) and tensor processing unit (TPU) acceleration and automatic differentiation for machine learning research.

This work aims to compare commonly used CNN architectures realized using Tensorflow, PyTorch and JAX when used for the morphological classification of radio galaxies. JAX, in particular, is a relatively new software package that has the potential to become very popular. In a bid to gain insight on the trade-offs of each, this work compares the performance of the respective packages. The rest of the paper is structured as follows: Section 2 reviews the state-of-the-art of machine/deep learning algorithms in morphological classification. Section 3 presents the experimental set-up. Section 4 outlines the methodology and metrics of the experiments. Section 5 presents the experimental results. Section 6 discusses said results. Finally, Section 7 summarizes the paper, highlighting major insights.

2 RELATED WORK

2.1 Scope and Methodology

The literature review focuses on recent advancements in computer intelligence for the morphological classification of radio sources. Steven Ndung'u et al.'s (2023) review provides a concise overview of the use of machine learning techniques for the morphological classification of radio galaxies. It will be utilized in this work to summarize the state-of-the-art of these techniques. The authors utilized the Web of Science (WoS) and NASA's Astrophysics Data System (ADS) to gather relevant literature. They applied search queries to retrieve papers from 2017 to 2023, filtering out irrelevant content. Additionally, the authors incorporated seven recently published papers recommended by peers and excluded duplicate and irrelevant papers, resulting in a final set of 32 articles for the literature review. The surveyed papers encompass a variety of machine and deep learning-based methods applied in radio astronomy, with a predominant focus on shallow and deep CNNs.

2.2 Relevant Findings

2.2.1 Morphological Classification. The developed FR classification approaches involve deep learning methodologies, such as unsupervised, semi-supervised or supervised machine learning. Deep learning methodologies are currently the most popular approaches within the literature, as illustrated in Figure 3. However, conventional machine learning techniques have also been used for the classification of FR galaxies.

2.2.2 Model-centric approaches. Deep Learning has witnessed the development of novel CNN architectures for classification problems. Some of these architectures have also been applied in radio astronomy, such as AlexNet (Krizhevsky, Sutskever, and Hinton 2017), VGG-16 (Liu and Deng 2015) and DenseNet (Huang et al. 2017) (Ameya Samudre et al. 2022). Overall, different model architectures have improved and increased the generalization of deep learning algorithms in classification problems.

Ndung'u et al also mentions that in general the literature seems to indicate that deeper networks tend to achieve better recognition performance results when compared to their shallower counterparts (Tang et al. 2022).

Over-fitting, however, remains a big problem in the radio galaxy classification literature. A notable contributor to this is the availability of only a few, small labelled astronomical datasets for building models. One of the ways that this issue can be circumvented is to apply dropout. This involves the random dropping out of weakly connected neurons during training (Tang et al. 2022).

2.2.3 Data-centric approaches. The quality and robustness of machine and deep learning models is highly dependent on the quality of the data utilized (Ndung'u et al. 2023). Radio images must be free from Radio Frequency Interface[4] (RFI) and artifacts before they can be processed further (Ndung'u et al. 2023). Each sample must also belong to a definite radio galaxy class and there must be enough data diversity in the training dataset.

Data augmentation involves increasing the size and diversity of the training set. Ideally, the augmentations will inject additional important information into an otherwise insufficient dataset. Strategies utilized in data augmentation include positional augmentation (scaling, flipping, rotating, affine transformation), colour augmentation (brightness, contrast and saturation), oversampling minority classes and using generative adversarial networks (GANs) to generate new instances of under-representative classes (Rustige et al. 2023). This approach has been espoused to mitigate over-fitting (Aniyan and Thorat 2017) (Alhassan, Taylor, and Vaccari 2018) (Lukic et al. 2018), improve performance of machine and deep learning models (Lukic et al. 2018) (Maslej-Krešňáková, El Bouchefry, and Butka 2021) (Rustige et al. 2023), address rotational invariance (Becker et al. 2021), increase the size and diversity of training data (Aniyan and Thorat 2017) (Alhassan, Taylor, and Vaccari 2018) (Becker et al. 2021) (Z. Ma et al. 2019a) (Hossain et al. 2023) and address class imbalance, especially among minority classes (Lukic et al. 2018).

The augmentation strategy employed has a direct impact on a model's recognition performance and its ability to generalize. For the use case in question, an increase in brightness improves a model's performance while decreasing the brightness degrades it (Maslej-Krešňáková, El Bouchefry, and Butka 2021). A downside to data augmentation is that an inherent bias or data error will be inherited by the augmented data (Ndung'u et al. 2023).

[1]TensorFlow - machine learning software package
[2]PyTorch - machine learning software package
[3]JAX - machine learning software package

[4]Radio frequency interference is the conduction or radiation of radio frequency energy that causes an electronic or electrical device to produce noise that typically interferes with the function of an adjacent device (Techopedia n.d.).

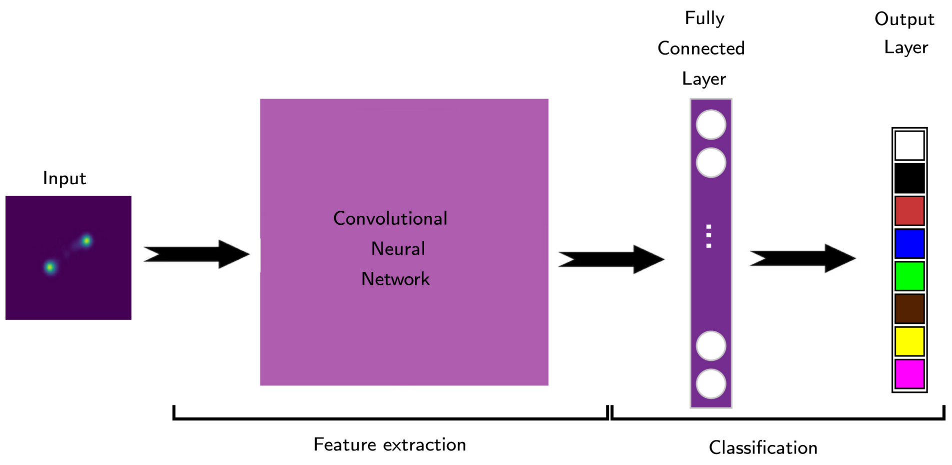

Figure 2: The fundamental building blocks of a standard Convolutional Neural Network from Ndung'u et al. 2023.

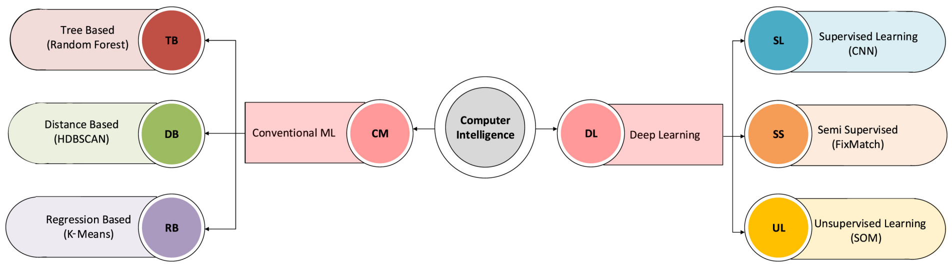
Tree Based (Random Forest)
TB
Distance Based (HDBSCAN)
DB
Regression Based (K-Means)
RB
Conventional ML
CM
Computer Intelligence
DL
Deep Learning
SL
Supervised Learning (CNN)
SS
Semi Supervised (FixMatch)
UL
Unsupervised Learning (SOM)

Figure 3: Computer intelligence methodologies applied in the classification of radio galaxies from Ndung'u et al. 2023.

Models don't always classify a test image to the correct class when the image presented is in a different orientation. This can be circumvented by applying Group Convolutional Neural Networks (G-CNNs) (A. M. Scaife and F. Porter 2021), adding rotations to the training set (Becker et al. 2021), or including a pre-processing step to standardize rotation of all radio sources.

Feature engineering is targeted at providing *precise physical properties* of the image data and aimed at improving classification accuracy. This provides machine learning algorithms with features of high importance resulting in high recognition performance (Sadeghi, Javaherian, and Miraghaei 2021). The downside to this is that it requires domain expertise to design descriptions of a feature.

2.2.4 Weak supervision approaches. The cost of labelling large radio astronomical datasets is very high. Hence, it is paramount to explore algorithms and strategies with the capacity of utilizing massive unlabeled public catalogs. There are three weakly supervised models discussed that can be utilized: transfer learning, semi-supervised learning and N-shot learning.

Transfer learning reuses knowledge gained from an already trained model that trained on a massive dataset to fine tune it on other tasks. It is effective if the training set is small (Ameya Samudre et al. 2022). The pre-trained model's weights and biases assist the model in identifying low-level features of the object. Complex features are learned by fine tuning the last layers of the model using the available small dataset.

Semi-supervised learning (SSL) involves self-supervised learning on an unlabelled dataset followed by supervised fine-tuning on a labelled dataset. This approach has been investigated and has achieved high performances (Z. Ma et al. 2019a) (Zhenzhen Ma et al. 2019b) (Slijepcevic et al. 2022).

N-shot learning algorithms leverage the limited labeled dataset available. These algorithms have shown excellent performance on standard benchmark datasets, but poor performance compared to state-of-the-art supervised machine learning approaches applied to real datasets (A. Samudre et al. 2021).

2.3 Contributions

There has been no comparison study to the author's knowledge for commonly used CNN architectures realized using TensorFlow, PyTorch and JAX, when utilized to perform the morphological classification of galaxies. In consequence, this work addresses a gap in the existing literature, and has yet to explore this aspect.

3 PRELIMINARIES

3.1 Standardized Benchmark Experiment

3.1.1 Implementation. The standardized benchmark experiment is implemented in the Python 3[5] language. Three machine learning software packages are utilized: Tensorflow(v2.16.1), PyTorch(v2.2.2) and JAX(v0.4.33). The package versions chosen are recent stable release. The experiment uses a 12th Gen Intel® Core™ i7-1255U CPU.

3.1.2 Dataset. The project utilizes the following dataset:

MNIST: The Modified National Institute of Standards and Technology (MNIST) database[6] (LeCun, Cortes, and Burges 2010) of hand-written digits consists of 10 classes in which we can classify numbers from 0 to 9. The dataset consists of 80 000 images in total, split into 60 000 training images and 10 000 test images. Table 1 presents each split.

Table 1: Number of images in each split of the MNIST dataset

Split	Number of Images
Training	60,000
Testing	10,000

3.1.3 Architecture. A simple convolutional neural network is utilized. The architecture is present in Table 8 in Section 8.

3.2 Radio Galaxy Classification Experiment

3.2.1 Implementation. The radio galaxy classification experiment is implemented in the Python 3 language. Two machine language software packages are utilized: Tensorflow(v2.16.1) and PyTorch(v2.2.2). The experiment uses a 12th Gen Intel® Core™ i7-1255U CPU

3.2.2 Dataset. The project utilizes the following datasets:

MiraBest: The MiraBest Batched dataset[7] is a collection of labelled images of radio galaxies, specifically focusing on the Fanaroff-Riley galaxies. The dataset is taken from the catalogue selected by **Mira**ghaei and **Best** in 2017 (Davies et al. 2017). The training dataset consists of 1069 images, 517 of which are FR-I and 552 are FR-II. The testing dataset consists of 153 images, 74 of which are FR-I and 79 are FR-II. Table 2 presents the class breakdown. The training dataset is split into a new training set and validation set using a sampling rate of 20% for the validation set.

Table 2: Class breakdown of the MiraBest dataset.

FR Classes	No. of Images (Split)
FR-I	591 (517 training, 74 testing)
FR-II	631 (552 training, 79 testing)

FR-DEEP:. The FR-DEEP Batched dataset[8] (H. Tang, A. M. M. Scaife, and Leahy 2019) is a collection of labelled images of radio galaxies, specifically focusing on the Fanaroff-Riley galaxies. The labels for the samples are compiled from the Faint Images of the Radio Sky at Twenty-centimeters (FIRST) catalog of FRI radio galaxies (FRICAT) (Capetti, Massaro, and Baldi 2016). It also uses the National Radio Astronomy Observatory VLA Sky Survey (NVSS) –FIRST Galaxies (CoNFIG) sample (Gendre and Wall 2008). The FR-DEEP Batched Dataset is comprised of two separate sub-datasets: FR-DEEP-N and FR-DEEP-F. The two subsets provide images of the same objects taken from NVSS (NRAO 1998) and the VLA-FIRST (NRAO 1995) Survey. The FR-DEEP-F sub-dataset is utilized in this study. The training dataset consists of 550 images, 242 of which are FR-I and 308 of which are FR-II. The test dataset consists of 50 test images, 22 of which are FR-I and 28 are FR-II. Table 3 presents the class breakdown. The training dataset is split into a new training set and validation set using a sampling rate of 20% for the validation set.

Table 3: Class breakdown of the FR-DEEP dataset.

FR Classes	No. of Images (Class)
FR-I	264 (242 training, 22 testing)
FR-II	336 (308 training, 28 testing)

3.2.3 Architecture. This work utilizes the following architectures:

ConvXpress: The architecture of ConvXpress (Becker et al. 2021) is presented in Table 9 in Section 8.

First-class: The architecture of First-class (Alhassan, Taylor, and Vaccari 2018) is presented in Table 10 in Section 8.

MCRGNet: The architecture of MCRGNet (Morphological Classification of Radio Galaxy Network) (Z. Ma et al. 2019a) is presented in Table 11 in Section 8.

Toothless: The architecture of Toothless (Aniyan and Thorat 2017) is presented in Table 12 in Section 8.

4 METHODOLOGY

4.1 Standardized Benchmark Experiment

4.1.1 Independent Variables.

- machine-learning software package utilized (TensorFlow, PyTorch or JAX).

4.1.2 Dependent Variables.

- accuracy score
- F1 score
- training time in seconds
- inference time in seconds
- memory usage in megabytes

See section 4.3 for details on the metrics used.

[5]Python 3
[6]MNIST Database
[7]MiraBest Batched Dataset
[8]FR-DEEP Batched dataset

4.1.3 *Controlled Variables.*

- The same hardware is utilized.
- Experiments are compared over the same dataset (MNIST).
- Experiments are compared over equivalent architectures of the simple convolutional neural network across the different software packages.
- The experiment is run six times.
- The experiment utilizes five epochs for each run.

4.2 Radio Galaxy Classification Experiment

4.2.1 *Independent Variables.*

- machine-learning software package utilized (TensorFlow and PyTorch).

4.2.2 *Dependent Variables.*

- accuracy score
- F1 score
- training time in seconds
- inference time in seconds
- memory usage in megabytes

See section 4.3 for details on the metrics used.

4.2.3 *Controlled Variables.*

- The same hardware is utilized.
- Experiments are compared over the same dataset (MiraBest or FR-DEEP-F).
- The training dataset is split so that 80% is the training set and 20% is the validation set (see Figure 4).
- The training dataset is split for each run by sampling the dataset randomly, without replacement, to construct the training set and having the remaining images used for the validation set. Hence, the training set and validation set have different samples for each run.
- Experiments are compared over equivalent architectures (ConvXpress, First-class, MCRGNet or Toothless).
- The experiment is run 3 times with differing initial seed values.
- The experiment utilizes the following number of epochs[9]:
  - 100 epochs for ConvXpress
  - 100 epochs for First-class
  - 60 epochs for MCRGNet
  - 30 epochs for Toothless

4.3 Metrics

The metrics used to evaluate the different software packages are accuracy, F1 score, training time, inference time, memory usage and the API-design and Usability Survey.

4.3.1 Accuracy. This is the ratio of the number of correct predictions and the total number of predictions.

The accuracy can be calculated using the following formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

[9]The number of epochs selected is based on the point at which overfitting begins to occur in each architecture.

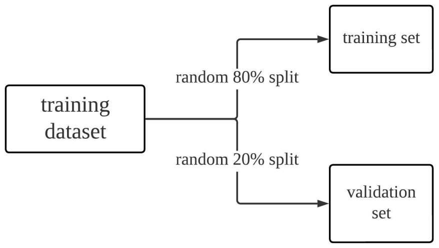

Figure 4: Training test split for run N

As such, accuracy offers a more intuitive measure of classification performance.

4.3.2 F1 Score. The accuracy metric is only reliable if a dataset is class balanced. However, real world datasets are heavily class imbalanced.

In relation to a binary class dataset, where there is a class labelled *positive* and *negative*, precision, for instance, would indicate how many *positive* predictions were correct. Recall would, conversely, indicate how many of the *positive* class samples present were correctly identified.

F1 score combines precision and recall using their harmonic mean. Maximizing the F1 score implies maximising both the precision and recall of a model.

The F1 score is calculated as:

$$\text{F1 score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where precision and recall are defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

4.3.3 Training Time. Training time is the duration it takes to train a machine learning model. It depends on factors such as:

- complexity of model
- size of the dataset
- computational resources available

A long training time indicates that a model is computationally intensive. However it may also be indicative of better performance. The training time is reported in seconds in this study.

4.3.4 Inference Time. Inference time is the amount of time it takes for a machine learning model to process new data and make a prediction. Fast inference times are important for real-time applications, such as image recognition. The inference time is reported in seconds in this study.

4.3.5 Memory Usage. This indicates the amount of memory in RAM used by the model during training and inference. Less memory usage is important when deploying models on devices with limited resources, such as smartphones. The memory usage is reported in megabytes in this study.

4.4 API-design and Usability Survey

For details on the API-design and Usability Survey, see Section 8.1.

4.5 Lines of Code

As of this report's writing, this experiment comprises 1881 lines of code. This does not include scripts that were used to run experiments or code that was used to generate plots.

5 RESULTS

Table 4: Performance Metrics Comparison between Machine Learning Software Packages for the Standardized Benchmark Experiment

Package	Accuracy	F1 Score
TensorFlow	98.35 ± 0.002	98.35 ± 0.002
PyTorch	98.78 ± 0.002	98.78 ± 0.002
JAX	98.66 ± 0.000	98.66 ± 0.000

Table 5: Performance Metrics Comparison between TensorFlow and PyTorch for the Radio Galaxy Classification Experiment on MiraBest

	TensorFlow		PyTorch	
Package	Accuracy	F1 Score	Accuracy	F1 Score
ConvXpress	70.37 ± 0.82	70.07 ± 0.71	80.39 ± 1.41	80.15 ± 1.67
First-class	78.00 ± 0.31	77.98 ± 0.29	83.88 ± 1.34	83.77 ± 1.43
MCRGNet	58.61 ± 0.81	56.70 ± 0.98	67.98 ± 0.92	67.49 ± 1.01
Toothless	73.42 ± 3.93	72.69 ± 4.91	51.63 ± 0.00	35.16 ± 0.00
Average	70.10 ± 1.97	69.36 ± 1.72	70.97 ± 0.92	66.14 ± 1.03

Table 6: Performance Metrics Comparison between TensorFlow and PyTorch for the Radio Galaxy Classification Experiment on FR-DEEP-F

	TensorFlow		PyTorch	
Package	Accuracy	F1 Score	Accuracy	F1 Score
ConvXpress	88.67 ± 1.89	88.69 ± 1.88	93.33 ± 0.94	93.35 ± 0.93
First-class	90.67 ± 0.94	90.65 ± 0.92	92.00 ± 1.63	91.97 ± 1.64
MCRGNet	70.00 ± 2.83	69.66 ± 2.87	86.67 ± 1.89	86.44 ± 1.92
Toothless	88.00 ± 5.89	87.99 ± 5.90	84.67 ± 1.89	84.50 ± 2.07
Average	83.83 ± 2.14	83.75 ± 2.14	86.17 ± 2.11	86.32 ± 2.14

6 DISCUSSION

6.1 Standardized Benchmark Experiment

Interpretation. All three software packages demonstrated exceptional performance, achieving accuracy and F1 scores above 98% (see Table 4). The inter-package variations were minimal (<0.5%). This aligns with theoretical expectations, as these metrics primarily depend on model architecture, dataset characteristics, and training protocol rather than the implemented software package. The identical accuracy and F1 scores across all packages indicate balanced performance across classes, with JAX notably exhibiting zero variation (±0.000), suggesting exceptional stability.

Performance Analysis. Training time measurements revealed substantial differences among the software packages. JAX required significantly longer training periods (528.63s) compared to its alternatives (see Figure 5). TensorFlow emerged as the most efficient (46.03s), while PyTorch demonstrated intermediate performance (74.59s), trending closer to TensorFlow's efficiency. Notably, TensorFlow achieved approximately 11.5x faster training times compared to JAX. For inference tasks, JAX demonstrated superior performance (0.68s), potentially attributed to its Accelerated Linear Algebra (XLA[10]) compilation capabilities. TensorFlow (1.01s) and PyTorch (1.28s) followed respectively. While these differences are less pronounced than training disparities, with all software packages operating within a 0.6s range, it is noteworthy that TensorFlow and PyTorch's optimal inference times exceeded JAX's worst-case performance. Memory utilization varied significantly across software packages. JAX exhibited the highest memory footprint (1604.66 MB), while PyTorch demonstrated good memory efficiency (665.11 MB). TensorFlow occupied an intermediate position (903.67 MB), with PyTorch notably consuming less than half the memory resources of JAX.

Implications. These findings suggest framework-specific optimization scenarios: TensorFlow appears optimal for training-intensive applications, PyTorch for memory-constrained environments, and JAX for inference-prioritized deployments. The optimal software package selection should align with specific performance priorities regarding speed, memory efficiency, or inference timing.

Limitations. A significant limitation of this study is its exclusive focus on CPU performance. JAX's architecture is specifically optimized for hardware accelerators such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs). Consequently, these findings primarily generalize to CPU-based implementations.

Future Work. Further research should extend this comparative analysis to GPU and TPU environments to provide a more comprehensive understanding of software package performance across different hardware accelerators.

[10]XLA: A compiler that optimizes models for high-performance execution

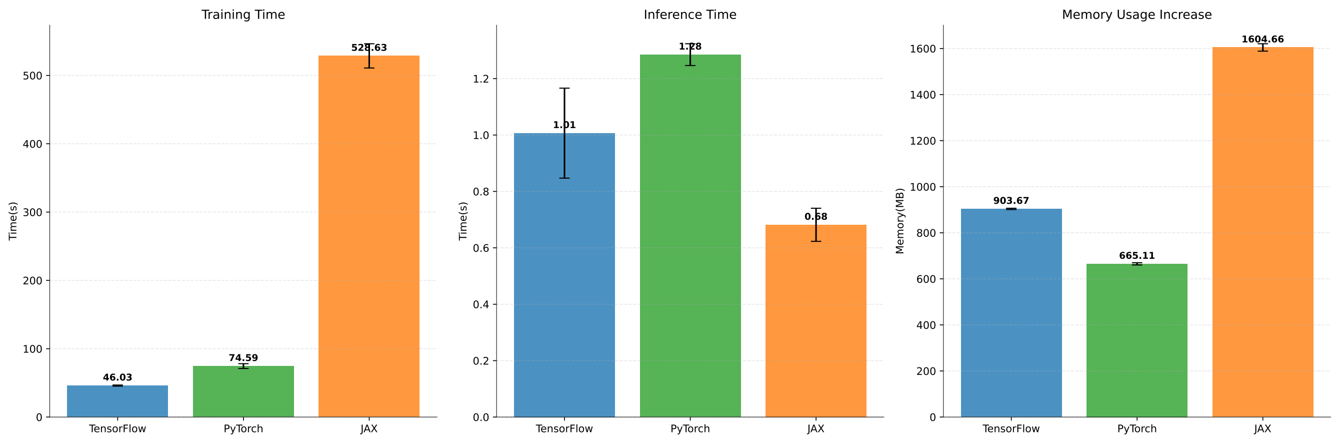

Figure 5: Computational Performance Comparison of Deep Learning Software Packages for the Standardized Benchmark Experiment

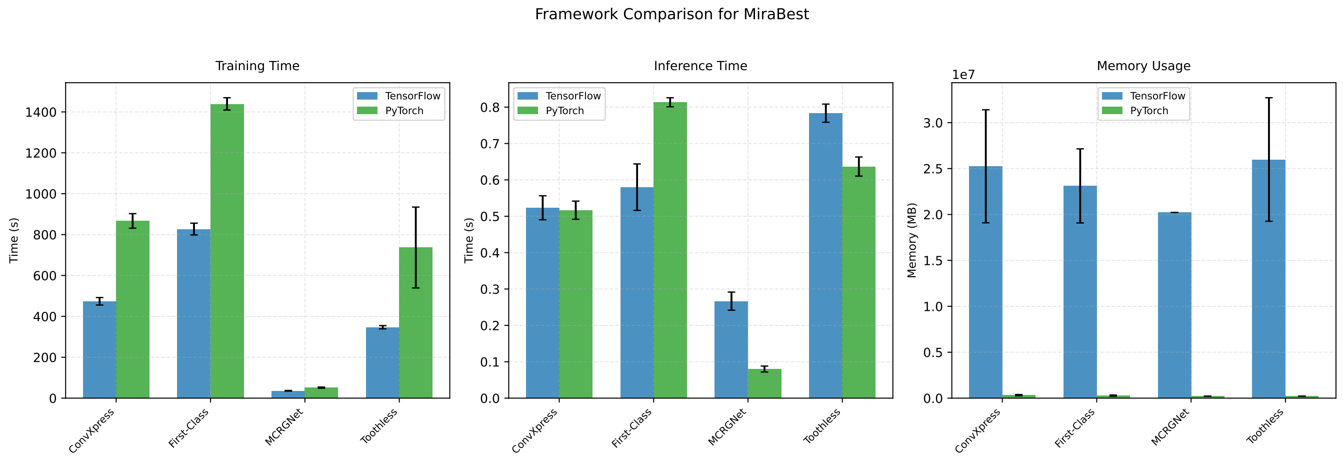

Figure 6: Computational Performance Comparison between TensorFlow and PyTorch for the Radio Galaxy Experiment on MiraBest

Table 7: Difference in Average Accuracy and F1 Score between TensorFlow and PyTorch

Dataset	TensorFlow Average		PyTorch Average		Difference
	Accuracy	F1 Score	Accuracy	F1 Score	
MiraBest	70.10 ± 1.97	69.36 ± 1.72	70.97 ± 0.92	66.14 ± 1.03	**Accuracy: +0.87, F1: -3.22**
FR-DEEP-F	83.83 ± 2.14	83.75 ± 2.14	86.17 ± 2.11	86.32 ± 2.14	**Accuracy: +2.34, F1: +2.57**

6.2 Radio Galaxy Classification Experiment

Performance Analysis. Analysis of classification metrics across both datasets reveals distinct performance patterns between PyTorch and TensorFlow implementations (see Tables 5 and 6). PyTorch demonstrates superior performance metrics for most models, with Toothless being a notable exception. The FR-DEEP-F dataset consistently yields higher performance scores compared to MiraBest across both software packages. Toothless exhibits particularly interesting behavior, showing strong performance on FR-DEEP-F but degraded metrics on MiraBest when implemented in PyTorch. TensorFlow implementations generally display lower standard deviations, indicating enhanced stability, while PyTorch shows

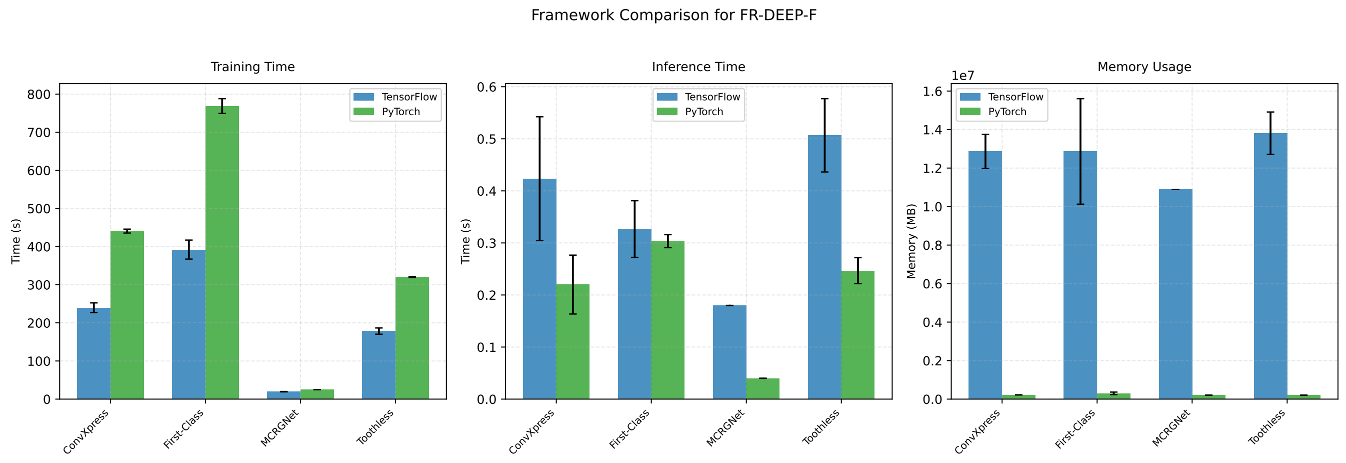

Figure 7: Computational Performance Comparison between TensorFlow and PyTorch for the Radio Galaxy Experiment on FR-DEEP-F

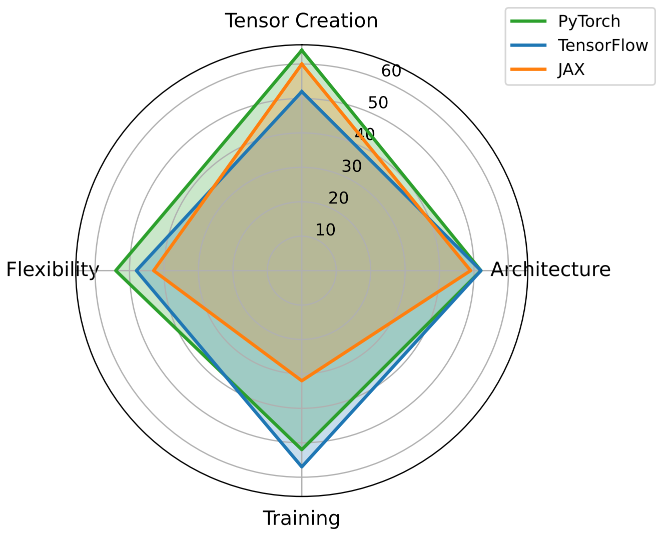

Figure 8: API-design and Usability Survey Results of Machine Learning Software Packages

marginally higher variations despite often achieving better absolute performance. The inter-framework differences are more pronounced compared to the standardized benchmark experiment variation (<0.5%), with average accuracy differences of 0.87% and 2.34% on MiraBest and FR-DEEP-F respectively, and F1 score differences of -3.22% and 2.57% (see Table 7).

Computational Efficiency. TensorFlow consistently achieves faster training times across all models on both datasets (see Figures 6 and 7). Inference time differences between software packages are minimal (<0.25s), with PyTorch showing marginally faster inference across most models, except for First-class on MiraBest. Notably, PyTorch demonstrates significantly lower memory utilization compared to TensorFlow. The performance profile of each software package shows distinct characteristics:

- PyTorch: Superior or comparable inference times, optimal memory efficiency, but longer training periods
- TensorFlow: Faster training execution, higher memory requirements, and competitive inference performance

Implications. These findings align with the standardized benchmark results, suggesting consistent software package characteristics across different applications. The results indicate that software package selection should be guided by specific application requirements:

- Training-intensive applications may benefit from TensorFlow's superior training efficiency
- Resource-constrained environments might favor PyTorch's memory optimization
- Inference time differences appear negligible for practical applications

Limitations. Several factors constrain the generalisability of these results:

- Limited experimental repetitions (three runs per configuration), though standard deviations remain notably low (see Figures 6 and 7)
- Exclusive focus on CPU performance
- Potential architectural discrepancies between software packages due to default parameter initialization differences

Future Work. Future research directions should address current limitations and expand the scope of analysis:

- Inclusion of JAX in the comparative software package analysis
- Increased experimental iterations to enhance statistical reliability
- Implementation of rigorous architectural equivalence between software packages

- Extension of performance analysis to GPU and TPU environments

6.3 API-design and Usability Survey

Architecture Design. An interesting equilibrium emerges in architecture design capabilities. The identical scores of PyTorch and TensorFlow (~ 74%) with JAX close behind (70%) suggest that all software packages have reached a mature state in supporting neural network architecture design and that basic architecture design has become standardized across software packages (see Figure 8). This may indicate that developers can expect similar experiences when building common network architectures.

Training Capabilities. Training shows the widest disparity among all the categories. TensorFlow leads with ~ 81%. This is likely due to it's concise built-in training API. PyTorch follows with a ~ 74%, while JAX's score is significantly lower (~ 46%). This suggests that JAX's functional approach to training loops may present a steeper learning curve.

Software Package Flexibility. PyTorch's high flexibility score (~ 77%) aligns with it's reputation for research-friendly design. TensorFlow's score of (~ 67%) reflects it's enterprise focused approach. JAX's lower flexibility score (~ 61%) might be offset by its unique advantages in functional programming and compilation.

Implications. PyTorch emerges as a strong choice, with high scores in tensor operations and flexibility. Its intuitive design makes it particularly suitable for learning and experimentation. TensorFlow's balanced performance across categories makes it a reliable choice. It is particularly strong in training capabilities, suggesting good production readiness. JAX shows promise for specialized cases requiring functional programming benefits.

Limitations. Several limitations should be considered when interpreting the results:

- The survey does not cover aspects of user-experience such as:
  - Installation and setup
  - Documentation and learning resources
  - Community and support

  This limits the universality of findings in terms of overall user-experience.
- The small sample size ($N = 7$) limits the generalisability of the findings.
- Participants' prior experience with specific software packages may have influenced their ratings.
- The evaluation was conducted at a single point in time and may not reflect rapid developments in these software packages.

7 CONCLUSION

This study demonstrates that software package performance characteristics observed in standardized benchmarks effectively generalize to radio galaxy classification tasks. The comparative analysis reveals advantages for each software package, enabling informed selection based on specific application requirements. TensorFlow emerges as the preferred software package for training efficiency, consistently demonstrating faster training times across both standardized and domain-specific tasks. PyTorch distinguishes itself through optimal memory utilization and excels in developer experience, making it particularly suitable for resource-constrained environments. While JAX demonstrated favourable inference performance in standardized benchmarks, its capabilities in radio galaxy classification remain unexplored, as it was not implemented for the domain-specific architectures in this study. Several limitations warrant consideration:

- The study's findings primarily generalize to CPU-based implementations
- Absence of JAX implementation for radio galaxy classification tasks

Future research directions should address these limitations through:

- Comprehensive evaluation of JAX performance in radio galaxy classification
- Extended performance analysis across different hardware accelerators (GPU, TPU)
- Increased experimental iterations to enhance statistical reliability
- Systematic assessment of software package usability through structured user experience studies

These findings contribute to the growing body of knowledge regarding machine learning software package selection for optimization of radio galaxy classification tasks.

REFERENCES

Alhassan, W, A Taylor, and M Vaccari (2018). "The FIRST Classifier: compact and extended radio galaxy classification using deep Convolutional Neural Networks". In: *Monthly Notices of the Royal Astronomical Society* 480.2, pp. 2085–2093. URL: https://academic.oup.com/mnras/article/480/2/2085/5060783.

Aniyan, A and K Thorat (2017). "Classifying radio galaxies with the convolutional neural network". In: *Astrophys. J. Suppl.* 230, p. 20. DOI: 10.3847/1538-4365/aa7333. URL: https://iopscience.iop.org/article/10.3847/1538-4365/aa7333.

Becker, B. et al. (2021). "CNN architecture comparison for Radio Galaxy Classification". In: *Monthly Notices of the Royal Astronomical Society* 503.2, pp. 1828–1846. DOI: 10.1093/mnras/stab325. URL: https://doi.org/10.1093/mnras/stab325.

Booth, R and J Jonas (2012). "An overview of the Meerkat project". In: *Afr. Skies* 16, p. 101.

Capetti, A., F. Massaro, and R. D. Baldi (2016). "FRICAT: A FIRST catalog of FRI radio galaxies". In: *arXiv preprint arXiv:1610.09376.* [Submitted on 28 Oct 2016]. URL: https://arxiv.org/abs/1610.09376.

Davies, Luke J. M. et al. (2017). "GAMA/G10-COSMOS: The 1.8 deg^2 G10 region of the GAMA survey – I. Panchromatic data from ultraviolet to far-infrared". In: *Monthly Notices of the Royal Astronomical Society* 466.4, pp. 4346–4363. DOI: 10.1093/mnras/stw3303. URL: https://academic.oup.com/mnras/article/466/4/4346/2843096?login=true.

Encyclopædia Britannica (2024). *Radio Source.* https://www.britannica.com/science/radio-source. [Accessed on April 26, 2024].

Fanaroff, B and J Riley (1974). "The morphology of extragalactic radio sources of high and low luminosity". In: *Monthly Notices of the Royal Astronomical Society* 167, 31P–36P.

Gendre, M. A. and J. V. Wall (2008). "The Combined NVSS–FIRST Galaxies (CoNFIG) sample – I. Sample definition, classification and evolution". In: *Monthly Notices of the Royal Astronomical Society* 390.2. [Published: 07 October 2008], pp. 819–828. DOI: 10.1111/j.1365-2966.2008.13792.x. URL: https://academic.oup.com/mnras/article/390/2/819/1032320?login=true.

Géron, Aurélien (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

Haarlem, M P van et al. (2013). "LOFAR: The LOw-frequency array". In: *Astron. Astrophys.* 556. http://dx.doi.org/10.1051/0004-6361/201220873, A2.

Hossain, M. S. et al. (2023). "Morphological classification of radio galaxies using semi-supervised group equivariant CNNs". In: pp. 601–612.

Huang, G. et al. (2017). “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 2261–2269. doi: 10.1109/CVPR.2017.243. url: http://dx.doi.org/10.1109/CVPR.2017.243.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2017). “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* 60.6, pp. 84–90.

LeCun, Yann, Corinna Cortes, and CJ Burges (2010). “MNIST handwritten digit database”. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Liu, S. and W. Deng (2015). “Very Deep Convolutional Neural Network Based Image Classification Using Small Training Sample Size”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. IEEE, pp. 730–734. doi: 10.1109/ACPR.2015.7486599. url: http://dx.doi.org/10.1109/ACPR.2015.7486599.

Lonsdale, C J et al. (2009). “The Murchison Widefield Array: Design overview”. In: *Proc. IEEE* 97, pp. 1497–1506.

Lukic, V. et al. (2018). “Radio Galaxy Zoo: Compact and Extended Radio Source Classification with deep learning”. In: *Monthly Notices of the Royal Astronomical Society* 476.1, pp. 246–260. doi: 10.1093/mnras/sty163. url: https://doi.org/10.1093/mnras/sty163.

Ma, Z. et al. (2019a). “A machine learning based morphological classification of 14,245 radio AGNs selected from the best–Heckman sample”. In: *Astrophysical Journal Supplement Series* 240, p. 34.

Ma, Zhenzhen et al. (2019b). “Classification of Radio Galaxy Images with Semi-Supervised Learning”. In: *International Conference on Data Mining and Big Data*. Vol. 11618. Lecture Notes in Computer Science. Springer, pp. 191–200.

Ma, Zhixian (2018). In: url: https://github.com/myinxd/MCRGNet/blob/master/mcrgnet/ConvAE.py.

Maslej-Krešňáková, V., K. El Bouchefry, and P. Butka (2021). “Morphological classification of compact and extended radio galaxies using convolutional neural networks and data augmentation techniques”. In: *Monthly Notices of the Royal Astronomical Society* 505.1, pp. 1464–1475. doi: 10.1093/mnras/stab1400. url: https://doi.org/10.1093/mnras/stab1400.

Ndung’u, Steven et al. (2023). “Advances on the morphological classification of radio galaxies: A review”. In: *New Astronomy Reviews* 97. doi: 10.1016/j.sigpro.2013.03.009. url: https://www.sciencedirect.com/science/article/pii/S1387647323000131.

Norris, R.P. et al. (2011). “EMU: evolutionary map of the universe”. In: *Publ. Astron. Soc. Aust.* 28, pp. 215–248.

NRAO (1995). *Faint Images of the Radio Sky at Twenty-Centimeters (FIRST)*. [Accessed: October 16, 2024]. url: https://www.cv.nrao.edu/first/.

— (1998). *NRAO VLA Sky Survey (NVSS)*. [Accessed: October 16, 2024]. url: https://www.cv.nrao.edu/nvss/.

Porter, Fiona Alice May (2023). *CRUMB: the Collected Radiogalaxies Using MiraBest dataset*. Zenodo. doi: 10.5281/zenodo.7746094. url: https://zenodo.org/record/7746094.

Rustige, L. et al. (2023). “Morphological classification of radio galaxies with Wasserstein Generative Adversarial Network-supported augmentation”. In: *RAS Techniques and Instruments* 2.1, pp. 264–277. doi: 10.1093/rasti/rzad016. url: https://doi.org/10.1093/rasti/rzad016.

Sadeghi, Mahnaz, Mohsen Javaherian, and Hanieh Miraghaei (2021). “Morphological-based Classifications of Radio Galaxies Using Supervised Machine-learning Methods Associated with Image Moments”. In: *The Astronomical Journal* 161, p. 94.

Samudre, A. et al. (2021). “Data-efficient classification of Radio Galaxies”. In: *Monthly Notices of the Royal Astronomical Society*. doi: 10.1093/mnras/stab3144. url: https://doi.org/10.1093/mnras/stab3144.

Samudre, Ameya et al. (2022). “Data-efficient Classification of Radio Galaxies”. In: *Monthly Notices of the Royal Astronomical Society* 509, pp. 2269–2280.

Scaife, A. M. and F. Porter (2021). “Fanaroff–Riley classification of radio galaxies using group-equivariant convolutional Neural Networks”. In: *Monthly Notices of the Royal Astronomical Society* 503.2, pp. 2369–2379. doi: 10.1093/mnras/stab530. url: https://doi.org/10.1093/mnras/stab530.

Slijepcevic, Ivana V. et al. (2022). “Radio Galaxy Zoo: Using Semi-Supervised Learning to Leverage Large Unlabelled Data Sets for Radio Galaxy Classification Under Data Set Shift”. In: *Monthly Notices of the Royal Astronomical Society* 514, pp. 2599–2613.

Smith, John (2020). *Understanding Confusion Matrix*. Accessed on April 26, 2024. url: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

Tang, H et al. (2022). “Radio galaxy zoo: giant radio galaxy classification using multidomain deep learning”. In: *Monthly Notices of the Royal Astronomical Society* 510, pp. 4504–4524. doi: 10.1093/mnras/stab2774. url: https://academic.oup.com/mnras/article/510/3/4504/6459220.

Tang, H., A. M. M. Scaife, and J. P. Leahy (2019). “Transfer learning for radio galaxy classification”. In: *arXiv preprint arXiv:1903.11921*. url: https://arxiv.org/abs/1903.11921.

Techopedia (n.d.). *Radio Frequency Interference (RFI)*. Accessed: 2024-05-16. url: https://www.techopedia.com/definition/9074/radio-frequency-interference-rfi.

Vidhya, Analytics (2020). *Understanding Confusion Matrix in Machine Learning*. Accessed on April 26, 2024. url: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/.

Wu, Chen et al. (2018). “Radio Galaxy Zoo: Claran – A Deep Learning Classifier for Radio Morphologies”. In: *Monthly Notices of the Royal Astronomical Society* 482, pp. 1211–1230. doi: 10.1093/mnras/sty2646.

8 APPENDIX

8.1 API-design and Usability Survey

8.1.1 Study Design. To evaluate the user experience and API design of each software package, a structured survey was conducted using a custom questionnaire. The questionnaire was implemented on Google Forms and distributed to postgraduate students whom had either taken a machine learning module or had a machine learning related thesis.

8.1.2 Participants. The study included 7 participants ($N = 7$), all of whom were computer science postgraduate students with previous experience in machine learning. Participation was voluntary, and all respondents had practical experience with at least one of the software packages under evaluation.

8.1.3 Questionnaire Design. The questionnaire was structured into three main categories:

(1) **Tensor Creation and Manipulation:** Evaluated the intuitiveness and consistency of basic tensor operations.
(2) **Model Definition:** Assessed the clarity and flexibility of defining neural network architectures, including custom layers and models.
(3) **Training Loop Implementation:** Examined the ease of implementing training procedures and customizing the training process.

Each category contained multiple questions rated on a 5-point Likert scale, where 1 indicated poor usability and 5 indicated excellent usability. Each section was accompanied by code examples with equivalent implementations across the three packages.

8.1.4 Data Collection. The questionnaire was distributed electronically via Google Forms over a two-week period.

Layer Type	Filters/Units	Additional Info
Conv2D + AvgPool	6	ReLU, Kernel(5,5), Same padding
Conv2D + AvgPool	16	ReLU, Kernel(5,5)
Dense	120	ReLU
Dense	84	ReLU
Dense	10	Linear

Table 8: Simple Neural Network Architecture

Layer Type	Filters/Units	Additional Info
3× Conv2D	32	ReLU, He uniform
MaxPool + Dropout	-	Pool(2,2), Drop(0.25)
3× Conv2D	64	ReLU, He uniform
MaxPool + Dropout	-	Pool(2,2), Drop(0.25)
3× Conv2D	128	ReLU, He uniform
MaxPool + Dropout	-	Pool(2,2), Drop(0.25)
2× Conv2D	256	ReLU, He uniform
MaxPool + Dropout	-	Pool(2,2), Drop(0.25)
Dense	500	Linear, L2(0.01)
Dense	2	Softmax

Table 9: Simplified ConvXpress Architecture

Layer Type	Filters/Units	Additional Info
Conv2D + MaxPool	32	ReLU, He normal
Conv2D + MaxPool	64	ReLU, He normal
Conv2D + MaxPool	194	ReLU, He normal
Dense	194	ReLU, He normal
Dense	2	Softmax

Table 10: Simplified First-Class Architecture

Layer Type	Filters/Units	Additional Info
2× Conv2D + Drop	8	ReLU, Strides(2)
2× Conv2D + Drop	16	ReLU, Strides(2)
Conv2D + Drop	32	ReLU, Strides(2)
Dense	64	ReLU
Dense	2	Softmax

Table 11: Simplified MCRGNet Architecture

Layer Type	Filters/Units	Additional Info
Conv2D + BN + Pool	96	ReLU, Kernel(11,11)
Conv2D + BN + Pool	256	ReLU, Kernel(5,5)
3× Conv2D + BN	384, 384, 256	ReLU, Kernel(3,3)
MaxPool	-	Pool(3,3)
2× Dense + Drop	4096	ReLU, Drop(0.5)
Dense	2	Softmax

Table 12: Simplified Toothless Architecture