Project Requirements

Aim

The aim of this project is to train and test deep learning models in order to determine

which is more suitable to classify the morphology of galaxies from an astronomy dataset.

The project will compare the performance of AlexNet, ResNet models and DenseNet models using accuracy, loss and F1 score as performance heuristics.

Objectives

1. Collect a dataset of galaxy images, coupled with their morphological type. The image set acts as our input features X, having m samples and n features. The morphological labels represent our target variables y, also with m samples.
2. Load the categorisation dataset and keep the features that consist of image file names and classification. Irrelevant classifications should be dropped along with their associated file name so that the models are not trained on them. The strings for the categories are then encoded so they can be used to train the model.
3. Load the images and implement relevant preprocessing. Deep learning image models typically have a fixed size input layer of n features $w \times h \times c$ (width, height and colour channels) so it is important to know if the dataset images vary in size. The images will be scaled to $224 \times 224 \times 3$ and then noise reduction will be applied to improve model performance.
4. Implement deep learning models (AlexNet, ResNet models and DenseNet models) for predicting the categorisation of images of galaxies. The input layer is the same size as the size the images were scaled to, and the output layer is the same size as the number of encoded categories in the target variables.
5. Deploy on GPU accelerated hardware and train the models with the training dataset. Models will be tested and performance heuristics will determine which model is better at categorising galaxy morphology.
6. Analise, using multiple model heuristics, the performance of each model and conclude which works the best for classification.

Dataset

Technologies

In order to achieve the objectives for this project, several technologies are used together. For this section each technology's utility is described. The descriptions are arranged in topics.

Language and Software Engineering Tools

Language: Python Python is a high level, multi paradigm, dynamically typed, interpreted program language often used in data analysis and machine learning projects because of its large library ecosystem and code simplicity. While python trades off performance for ease of use, Python supports a degree of interoperability with libraries written in low level languages such as C and C++ making it ideal for processing data in machine learning projects. Because of its widespread adoption, learning the language and debugging are easier than other languages due to online support, which makes development time faster.

Language Server Protocol (LSP) Server: Jedi and Pyright A language server protocol is an agreed protocol between a programming language and development software that allows features such as static analysis and code completion. For the project, being able to find errors in code before they arise, using static analysis tools, reduces development time because less debugging needs to be done. Code completion also helps reduce development time because it can save time on typing. Jedi is specifically good because it doesn't require any configuration outside of integrating with the code editor.

Pyright is also a LSP that is specifically for static type checking. Static type checking is a kind of static analysis that makes sure functions and values take and return appropriate types, which may cause errors in dynamically typed languages when methods or protocols are not supported for a given type.

Editor: Helix Helix comes from the VI family of editors which are known for modal editing, meaning that different key commands and combinations are used instead of shortcuts like other editors. Commands that are issued together can achieve very complex and specific needs so that an advanced user can perform complex and specific editing tasks quickly. Unlike the main VI family of editors (vi, vim, neovim), helix has inbuilt LSP support so configuring a setup for project work takes little time.

Data Loading and Preprocessing

Dataset Loading: Pyspark Pyspark is an API library for Apache Spark, a java framework used for parallel batch processing. CSV files from galaxy zoo's data release are loaded and preprocessed using spark, since it contains hundreds thousands of samples. Spark chunks a full dataset into worker nodes (also called

clusters), parallelised on central processing unit (CPU) cores which contain an executor. Each worker node, controlled by a cluster manager, contains an executor which performs operations on the subset of a dataframe in the spark node, known as a resilient distributed dataset (RDD). Spark is 'fault tolerant' meaning if an error is encountered during execution on a work node the cluster manager uses backup data to clone the erroneous node and then resume execution. In this way, Pyspark makes it possible to load, transform, filter and query large data sets within python programs much faster than in serial.

Dataset storage: Pandas Pandas is a python library that allows the storing and operation on tabular data similar to Pyspark but in serial. While much faster than Pandas due to its parallelisation capabilities, Pyspark has a large memory footprint since it stores copies of data in case a process fails, so to make sure memory is not bottle-necking training performance, pandas is the preferred choice for storage. Pyspark provides a dataframe method to convert to a Pandas dataframe so transferring the data between the two object types is a non-issue.

Train-Validation Splitting: Sci-Kit Learn While a smaller component of the project, the importance of splitting a dataset into testing and validation sets is fundamental to evaluating performance of any machine learning model. The machine learning framework Sci-Kit Learn (SKLearn) provides the helpful function `test_train_split` which shuffles and divides the dataset.

Image preprocessing: OpenCV Python Open Computer Vision (OpenCV) is an open-source C++ library written to efficiently process image and video data for the field of computer vision. As with Pyspark, since the library is not native python, the OpenCV Python (cv2) library is used for compatibility. In the project, the library handles image preprocessing i.e. denoising to train models most efficiently. OpenCV is deployable on graphical processing unit (GPU) hardware meaning that it will not bottleneck performance of training. This is a departure from other python image processing libraries (such as PIL) which tend to be CPU or single thread based.

Deep learning

Deep Learning backend: TensorFlow Tensorflow is an open-source machine and deep learning framework which allows machine learning engineers to build custom machine learning models. TensorFlow offers a wide variety of tools for architectural design, training and deployment of models, particularly for ML models that are based on neural networks like CNNs. The core library (Tensorflow Core) contains many modules for handling input-output (IO) operations e.g. `tf.image`, `tf.io` etc., deployment configuration i.e. tf.compat, training e.g. `tf.training`, `tf.testing`, and neural network functions i.e. `tf.nn`. Having extended functionality integrated into the core of a library makes it easier to

develop with. In software engineering this is informally referred to as "batteries included".

The main advantage of using TensorFlow for deep learning projects is that the highly process-intensive operation of model training can be deployed on the GPU. Both forward and particularly back propagation take a long time to process when applied to neural network parameters, stored as matrices and vectors, a.k.a tensors. If these processes were performed on a single thread such as on the CPU, each tensor item would be evaluated individually which on a small scale may not be an issue since a similar specification CPU core is much more powerful than a GPU core. However, as a machine learning model such as an artificial neural network (ANN) grows larger in dataset size, layer number and layer size, training times greatly increase making training and testing impractical. The GPU is specifically designed to parallelise similar processes by dividing a single process into many similar sub-processes running in separate GPU cores. Machine learning model training and use can be deployed on the GPU by TensorFlow, abstracting the complexities of programming GPU parallelisation while greatly reducing runtime.

Keras Another advantage of TensorFlow is that it has a large ecosystem, meaning additional compatible packages that can be used to extend its functionality.