Deep Learning using Distributed Computing Systems for Big Data

Glenn Tarpey
Masters Student
CCT College Dublin
Dublin, Ireland

***Abstract* -** This paper addresses the challenges of big data and deep learning and how distributed systems can aid in their combined applications, it reviews current research on the topic, and demonstrates how this technology can be integrated with Apache spark for the efficient training of a Deep Learning model.

I. Research Question

How can deep learning models and distributed computing improve on the analysis of large datasets?

II. Introduction

Data has been increasing exponentially, up until 2003, 5 exabytes of data had been created. Now, in 2023, we are creating approximately 328 exabytes per day (Duarte, 2023). This enormous quantity of data provides significant challenges in order to store, analyse and extract meaningful insights. (Sagiroglu & Sinanc, 2013)

Technologies for distributing Big Data and the processing of it such as Apache Spark and Hadoop allow us to improve on our efficiency and effectiveness of comprehending these massive datasets. By adopting distributed computing architectures we provide greater opportunity for data analysis.

III. State of the art / Literature Review

A. Characteristics of Big Data

Big Data is generally summarized by 3 main characteristics: Variety, Volume and Velocity. Variety meaning the three main categories of data: structured (excel spreadsheets, relational databases), semi-structured (HTML docs, Log Files, Emails) and unstructured data (Articles, Images, Audio). Volume meaning the size of the data, and Velocity meaning the speed at which the data is generated and processed. (Sagiroglu & Sinanc, 2013)

This increasing demand for analytical insights to gain and maintain a competitive edge poses several challenges.

B. Challenges in Big Data Analytics

The ever growing amount of data and analytics required to provide organizations with a competitive edge towards improving our overall efficiency presents several challenges, including but not limited to Data Storage, Validity and Availability.

Managing data storage beyond its sheer volume necessitates systems that ensure data accessibility, manageability, and protection against corruption or loss. Validity guarantees the trustworthiness and authenticity of the data, also Availability seeks to provide on demand data when needed.

Distributed computing utilizes systems spread across an organizations network, this can be either locally, regionally or globally. The advantages include robust data storage, prevention of data loss, an immutable ledger of changes, and effective management of nodes and resources. This ultimately provides a way for organizations to harness their Big Data in a way that satisfies the needs of protected and secure data that is available when they want it. (Kshemkalyani & Singhal, 2011)

C. Deep Learning

Deep Learning, a subset of machine learning is widely employed to analyze unstructured data such as images, video and text. It uses neural networks, which take inspiration from how the human brain learns. These models can have multiple inputs and outputs with a huge amount of hidden layers in-between that utilize a series of weights and biases of each neuron to correctly make predictions. These hidden layers help to optimize the model and utilize complex data structures. Deep Learning finds applications in diverse fields, including image recognition, voice analytics, recommendation systems, and more.

Deep Learning networks however are computationally expensive and require a large amount of data for training. This combined with the ever increasing size of data available perfectly encapsulates why distributed computing and deep learning can give an organization the tools it needs to provide analytics within a reasonable time frame. (Lecun, et al., 2015)

D. Distributed File Systems

The Hadoop Distributed File System (HDFS), Apache Cassandra, and Apache Spark are three well-known distributed file systems commonly employed in various industries. Hadoop is primarily focused on batch processing of data using the Map Reduce model, which breaks up data into chunks (usually 128MB), maps out patterns in them and reduces the result for the user. (Chaudhari & Mulay, 2019)

Cassandra utilizes a NoSQL database that unlike Hadoop and Spark does not have a single coordinator, but instead has a peer to peer architecture where each node in the cluster has the same level of responsibility. This provides greater fault tolerance and scalability. (Chaudhari & Mulay, 2019)

Spark focuses on in memory data processing, providing it with much faster speeds than Hadoop and Cassandra which rely heavily on disk reads/writes. This does have a drawback of being more expensive to setup hardware, but for example in a test of a simple Logistic Regression on a data size of 29GB showed that on average Spark was 10 times faster than Hadoop, this was largely attributed to Sparks in

memory processing and memory persistence features. (Zaharia, et al., 2010)

E. Deep Learning Training

Techniques such as Gradient Descent are applied to minimize the value of a loss function in order to produce more accurate results, however this is extremely computationally expensive, as in order to find the perfect weight and bias values for each neuron we need to compute the opposite of the gradient to the loss function. (Kwiatkowski, 2021)

In general the steps to find the gradient are:

1. Randomly set your weights and biases to initialize the model
2. Feed forward a data sample through the network and obtain the output (involves computing the weighted sums and applying activation functions)
3. Compute the loss between the predicted output and the labelled output
4. Using back propagation calculate the gradient of the loss with respect to the weights and biases
5. Update each weight/bias by adjusting it by a given learning rate in the opposite direction of the gradient
6. Iterate through the rest of your data with steps 2 to 5
7. Once the loss function has been minimized to a satisfactory level you have trained your model

Although these steps are reasonably straightforward they are time consuming, as the size of your dataset increases so does the time to train your model. This issue combined with the scope of Big Data makes this an impractical exercise. As a result research has been done to find better methods of approximating loss function minimization more efficiently. Simpler training techniques like stochastic gradient descent, batching, and advanced optimizers like Adam have emerged as alternatives.

1) Stochastic Gradient Descent

Stochastic gradient descent is an extension on traditional gradient descent, it differs in that its gradient is evaluated on a random sample of the training data, and in addition the updates to the network are done iteratively for each sample, allowing for small steps towards the local minimum of the loss function very quickly. This method is far less greedy than gradient descent without sacrificing on efficiency however it can produce some noise as it doesn't take the most optimum route to the local minimum. (Ruder, 2016)

2) Mini-Batching

Another method is mini-batching, it compromises between traditional gradient descents and stochastic, in summary it divides the dataset into a number of mini batches, and then the models parameters are updated after the processing of each mini batch by averaging the gradients of each data sample in the batch.

3) Adaptive Moment Estimation (ADAM)

ADAM is an optimization algorithm that dynamically adjusts the learning rate of the model based on historical gradients by storing an exponentially decaying average of past squared gradients, this greatly accelerates convergence to a local minimum of a loss function. (Ruder, 2016)

F. Types of Neural Networks

Considering the variety of data available and wide range of applications for machine learning in enterprises, one must select and develop the best model for a given task. In the context of complex relationships within data Deep Learning models such as Neural Networks show their strength. The following types of neural networks are explored in this paper:

- Feed-forward Neural Networks (FNNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Auto encoders
- Transformers

A brief summary of each type of neural network is given below. (MyGreatLearning, 2022) (towardsai.net, 2022)

1) Feed-Forward Neural Networks

FNNs (otherwise known as Artificial Neural Networks) are a foundational type of model that uses a one way connected set of neurons where data only travels forward, this means there is no back propagation in the operation of the model (only in the training). They consist of 3 sections, the input layer, hidden layer and output layer. They are used mainly for pattern recognition, classification or regression functions. However they are not great with sequential data or spatial data meaning they are not the best choice for image or natural language processing or any application with intricate structure, this ultimately makes them unsuitable for deep learning tasks.

2) Convolutional Neural Networks

CNNs are designed to tackle grid like data such as images, they preserve the spatial relationship between pixel values by passing a mask/kernel over an image, reducing the spatial dimensions and reducing them to their key features. This can be thought of as edge detection, or in simple terms the silhouette of a figure, rather than the figure itself. This means CNNs can extract important features from images making them great at object detection, facial recognition and image classification.

3) Recurrent Neural Networks

RNNs are similar to FNNs in that they also feed forward data, however each of the neurons in the hidden layers

receives an input with a delay in time, this allows them to access previous information used in the network. This recurrent structure makes them great at applications like translation, sentiment analysis and text processing tasks involving sequential data. They are however susceptible to the vanishing and exploding gradient problem (as explained in the limitation below) when using long sequences of data.

4) Long Short-Term Memory Networks

LSTMs are an evolution on RNNs, they include a memory cell that maintains information for long periods of time, this LSTM unit is controlled by 3 gates that control the input, output and forget functions on the stored information. This allows the LSTM network an advantage over RNNs specific delay in time as the memory cell can remember data from a long time ago. This makes LSTMs excel at speech and writing recognition. LSTMs suffer less from the vanishing gradient problem than RNNs as a result of their long term memory but again reach a point of poor performance as the length of a sequence in data extends.

5) Transformers

Mainly used in Natural Language Processing, they use an attention mechanism that overcomes the problem with long sequences of data e.g. long sentences or documents. It does this by passing all information for all hidden states from the encoder to the decoder (in RNNs and LSTMS only the final hidden state is passed to the decoder). The decoder then applies a scoring system on these hidden states to amplify hidden states that are significant and diminishes ones that aren't. This allows the model to develop a context for the input data and does not need to do this in sequences like RNNs and LSTMs, it can run these encodings in parallel, speeding up training time significantly.

The encoder takes the input data sequence and generates encodings that establish which parts of a sequence are more important/relevant to each other and passes this to the next encoder layer.

The decoder takes these encodings and uses their derived context to generate the output sequence. It is a form of semi supervised learning, meaning they are pre trained using a large unlabelled data set and are fine-tuned using supervised learning to enhance performance. (Ankit, 2022) (Alammar, 2018)

G. Parallel and Distributed Deep Learning

Training a deep learning model can takes hours if not days on a single computer, also with the added challenge of the data being stored and accessed on a single machine. On a single machine the use of a Graphical Processing Unit (GPU) can help speed up the series of complex calculations needed to train the model, producing results up to 40 times faster in one study. (Hegde & Usmani, 2016)

As deep learning models increase in complexity along with an increased data size the need for parallel processing of data among a distributed file system is apparent. Two key methods are used to overcome this challenge.

1. Data Parallelism: which is when data is stored and accessed from a distributed file system, with the down side being the need to synchronize the models parameters across nodes, increasing communication overhead.

2. Model Parallelism: where a model is too big for a single machine and each layer of the model is stored on a separate machine, where forward and back propagation involves communication between multiple machines. This also adds the complexity of synchronization and communication overheads.

The approach of synchronizes nodes in a distributed system can be broken down into synchronous and asynchronous. In synchronous methods all loss gradients in a batch are computed using the same parameters, resulting in uniform synced updates across all nodes. This leads to steady progress but is limited by the speed of the slowest node in the cluster.

Asynchronous allows nodes to update independently which avoids any communication bottlenecks between machines but sacrifices on uniform convergence. One method of asynchronous training is Downpour SGD, it allows each node in a cluster of a distributed system calculate its gradient and send them back to the central machine. The central machine then aggregates the gradients from the other nodes once they have all been computed (which paradoxically is synchronistic). Once the user ensures minimum communication issues one paper saw with a 1.7 billion parameters model a speedup of more than 12 times by utilizing using 81 machines. (Dean, et al., 2012)

IV. EVALUATION

The key findings from the research conducted and literature review give an in depth overview of how the combination of Distributed Computing and Big Data facilitate the training and storage of Deep Learning models. The adoption of distributed file systems such as Hadoop, Cassandra and apache Spark show how the industry as a whole has improved on the efficiency of processing Big Data and the models on which they are trained. The combination of these technologies opens paths for more accurate insights and predictions on vast formats of data.

In summary, the key findings from the research and evaluation provide a comprehensive understanding of how Distributed Computing and Big Data intersect and enable more accurate insights and predictions on diverse data formats.

A. Limitations and Challenges

One of the primary challenges or limitations in these applications relates to the computational complexity and resource requirements for large-scale models. For example one paper found training a language model can take multiple weeks on at least 10 GPUs with an estimated cost of $1000 using Amazon Web Services. Although this can vary widely depending on your required time/resource requirements. (Li, et al., 2020)

On the other hand, running your own machines can be more cost-effective, but it involves higher initial costs and increased maintenance demands. One article found that the benefits of such as system can be up to 10 times cheaper over a 3 year period, with the drawback of the upfront investment in parts and hardware failure risk. (Chen, 2018)

These issues coupled with the simple cost of storing the data required to train a deep learning model can be staggering, with one article quoting “a petabyte Hadoop cluster will require between 125 and 250 nodes which costs ~$1 million”. (Savitz, 2012)

Another limitation is the communication bottleneck, essentially where a distributed system is limited by the slowest machine, if not managed properly this can degrade performance significantly. (Dean, et al., 2012)

Neural networks can suffer from the Vanishing and Exploding gradient problem, in simple terms when training networks gradients can become so small that they have minimal impact on minimizing the loss function (vanishing), or the opposing issue, gradients can become so large that they can hinder convergence to a local minimum (exploding). Techniques like gradient clipping (a threshold on the size of a gradient), batch normalization (shifts the mean and standard deviation of a batch to be 0 and 1 respectively to scale them into the same range) and alternative activation functions such as ReLU (Rectified Linear Unit that sets all negative values to 0).

B. Research Gaps

The research in deep learning using distributed computing has progressed, but there are still research gaps. One notable gap is the need for better data management and multi-user scheduling. While existing reports provides evidence of the effectiveness of combining these technologies, a deeper investigation into their applicability across industries and problem domains is warranted. (Mayer & Jacobsen, 2020)

Another gap is that the ethical considerations within deep learning and distributed computing remains a largely unexplored area. As these technologies are increasingly integrated into decision-making of large organizations the dangers of discrimination, false reporting and manipulation are apparent. Issues of privacy are often raised, with companies such as Facebook have amassed petabytes of user information. Understanding the ethical implications of automated analyses on vast datasets is crucial.

Policies such as the General Data Protection Regulation (GDPR) have made great contributions in protecting user’s rights to their data, however more regulation is needed on a global scale. (Helbing, 2019)

V. Technical Demonstration

A. Local Neural Network

The technical demonstration in this paper focused on the development of a CNN model for the classification of characters from the popular television series "The Simpsons" in images. The primary objective of this demonstration is to provide a comprehensive overview of the steps involved in training, testing, and evaluating a CNN for image classification, particularly tailored to a real-world application. A CNN was chosen due to its ability to capture intricate spatial features in images.

1) Data Preparation

The input images were resized to 64x64 pixels and into RGB format. Making the input shape of (64, 64, 3). Each character name was encoded using a map function turning them into a number to be classified.

A series of experiments with various configurations, including batch size, number of epochs, and learning rate schedules were completed. After testing 9 different versions of the model, the final selection was a batch size of 128, trained for 30 epochs, and employed with an Exponential Learning Rate Decay schedule. The decay steps parameter was calculated by dividing the number of samples by the batch size to get the iterations per epoch, and then multiplying this by the number of epochs. This ensured a level of predictability to the training process.

The data augmentation technique Image Data Generator was used to increase model robustness by extending our sample to include rotations, shifting, shearing, zooming, and horizontal flipping.

2) Training and Evaluating the Model

The final model included multiple convolutional and pooling layers, followed by batch normalization to stabilize training and reduce overfitting. Regularization techniques, such as L2 regularization and dropout, were applied to enhance model generalization. The model was compiled using the Adam optimizer with the learning rate schedule and categorical cross-entropy loss. Other optimizers such as SGD where applied in earlier models but when compared to the results of the Adam optimizer it was found it performed worse. This is consistent with the Adam optimizers increased speed towards convergence vs SGD’s slower approach. (Ruder, 2016)

The final model achieved an accuracy of 89% on a test dataset with a loss of 0.70, with the following metrics:

TABLE I. RESULTS

Metric	Value
Precision	0.90
Recall	0.89
F1-Score	0.89
Accuracy	0.89

Figure 1- Local CNN Results

The precision of 0.90 indicates that 90% of the model's positive predictions were correct, demonstrating its ability to minimize false positives. With a recall of 0.89, the model effectively identified 89% of the relevant instances within the dataset, striking a balance between false negatives and true positives.

B. Apache Spark – Distributed Computing Efficiency

An experiment was conducted to demonstrate Apache Sparks ability to utilize multiple worker nodes to speed up the processing of data. The objective was to have the master node sum numbers from 1 to 1000.

To ensure a consistent and reproducible environment, Amazon Web Services (AWS) was used, in particular, EC2 (Elastic Compute) instances. The master node utilized a t2.medium instance equipped with a 2-core CPU and 4GB

RAM, while the worker nodes were t2.small instances, each with a 1-core CPU and 2GB RAM.
Below is a table depicting the processing times:

TABLE II. RESULTS

Nodes	Time Taken (seconds)
1	6.89
2	6.49
3	5.76

Figure 2- Spark Test Results

The results clearly indicate that increasing the number of worker nodes led to faster task completion. In essence, this demonstrates how Apache Spark can aid organizations in reducing computational durations. With the added benefit of no maintenance due to AWS's robust capabilities.

C. *Apache Spark – Distributed Computing to support CNN Training*

Two experiments were conducted using Apache Spark to enhance the execution time of data processing required to train the CNN model above on the Simpsons dataset.
The setup for these experiments involved using EC2 and S3 services of AWS. A cluster was set up using a t2.2xlarge instance with 8 core CPU and 32GB RAM for the driver node and each worker node, this alteration to the instance type was needed as the RAM requirements for the CNN model training were substantially greater than the earlier example. An S3 bucket was used to store the training data for ease of access.

1) *Experiment 1*

Each worker node was given a partition of the dataset for a number of characters. For example worker one was given character 1 to 6, worker 2 was given 7 to 12 etc. Each Spark worker then trains their own model on its partition of the data. The weights of each of these models is returned to the master node and averaged, the distributed trained model is then evaluated on the test set. For comparison, the experiment also trains a CNN locally without the distributed system, allowing for a direct evaluation of the efficiency and accuracy of the distributed approach versus the local approach.
For ease of testing as well as allowing different data loads on the distributed/local models the data used was only a given percentage of the total. Below we can see the results for this experiment on a 3 worker cluster:

TABLE III. RESULTS

Percentage of Data	Distributed Training Duration (seconds)	Local Training Duration (seconds)
10%	101.69	114.91
20%	205.17	225.36
30%	300.64	370.85
40%	385.27	480.58

Figure 3 - Experiment 1 Results

As you can see from the results above as the percentage of data increased there was a non-linear increase in the time taken to train the local model. This shows how the use of a distributed training method can yield faster results than single machine training. It should be noted that for this experiment accuracy of the model was not recorded as this experiment solely focused on execution of training time. While model accuracy is vital, this specific experiment aimed to highlight the time differences in training, recognizing that optimal accuracy would require a different experiment focused on fine-tuning.
There were some limitations to this experiment, in hindsight, distributing the dataset by ensuring each worker node had samples from every character would have been more ideal. This approach ensures uniformity in the training data across nodes, potentially preventing any model biases arising from training only on specific character subsets. Seeing the efficiency gains in execution time, the insights from this experiment underscore the potential of distributed training in large-scale image classification tasks. Future work should focus on optimizing both computational speed and model accuracy, leveraging the lessons learned.

2) *Experiment 2*

This experiment focused on how distributed data loading and centralized training on the CNN model using different numbers of worker nodes in the Apache cluster is faster with more workers. Each worker received a partition of the characters and independently loaded each of the images for the characters, resized them and normalized them. Essentially handling all of the preprocessing needed prior to training. Once each worker had preprocessed the data the master node collected and combined the data and trained the model. The results of the experiment can be seen in the following table:

TABLE IV. RESULTS

Number of Workers	Duration (minutes)
1	53
2	49
3	49

Figure 4 - Experiment 2 Results

From the results above we can see that increasing the number of workers from 1 to 2 clearly showed a reduction in execution time, however we can see the increasing workers from 2 to 3 had no effect. This is potentially down to a few factors: A bottleneck of S3 access, overhead of managing an additional worker, or the constraints of the network bandwidth during data collection. In distributed systems, there's often a diminishing return on adding more workers, especially when external systems like S3 are involved. (Chou, 2022)
In hindsight, using a Hadoop Distributed File System (HDFS) to store the data might have been a more efficient choice compared to S3 as it provides better throughput and reduced latency. In future tests, we could focus on improving network settings, rethinking how we split the data, and refining the Spark setup to better use more workers.

VI. Further Development

VII. In future projects, there's an opportunity to explore more about managing servers, possibly using tools like Ansible. This can help in organizing tasks better and making sure everything works smoothly. There's also interest in learning more about creating worker images, which could make adding more workers easier and faster using AWS. Experimenting with technologies like Horovod or TensorFlow distributed is also of benefit, to show research more ways to use distributed data and model parallelism. Also exploring alternative data storage like the HDFS to overcome any potential bottlenecks of S3 storage. The paper touched upon the ethical implications of deep learning and distributed computing but could delve deeper into this area. These implications require more attention as these technologies become integral to decisions made in organizations.

VIII. Conclusion

This paper has shown the potential and the challenges of using distributed training methods, helping to understand how to make machine learning models work faster. The experiments and findings have given useful insights into how different setups can affect the time it takes to train models. There are still areas for further development but these first experiments provide a solid foundation for more research in distributed computing and machine learning.

IX. References

Alammar, J., 2018. *Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention).* [Online]
Available at: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
[Accessed 09 2023].

Ankit, U., 2022. *Transformer Neural Networks: A Step-by-Step Breakdown.* [Online]
Available at: https://builtin.com/artificial-intelligence/transformer-neural-network
[Accessed 09 2023].

Chaudhari, A. & Mulay, P., 2019. *SCSI: real-time data analysis with cassandra and spark. Big Data Processing Using Spark in Cloud.* s.l.:s.n.

Chen, J., 2018. *Why building your own Deep Learning Computer is 10x cheaper than AWS.* [Online]
Available at: https://medium.com/the-mission/why-building-your-own-deep-learning-computer-is-10x-cheaper-than-aws-b1c91b55ce8c
[Accessed 2023].

Chou, J., 2022. *Top 3 trends we've learned about the scaling of Apache Spark (EMR and Databricks).* [Online]
Available at: https://synccomputing.com/top-3-trends-weve-learned-about-the-scaling-of-apache-spark-emr-and-databricks/
[Accessed 09 2023].

Dean, J. et al., 2012. Large scale distributed deep networks. In: *Advances in neural information processing systems.* s.l.:s.n., p. 25.

Duarte, F., 2023. *Amount of Data Created Daily (2023).* [Online]
Available at: https://explodingtopics.com/blog/data-generated-per-day
[Accessed 08 2023].

Hegde, V. & Usmani, S., 2016. *Parallel and distributed deep learning.* s.l.:s.n.

Helbing, D., 2019. *Societal, economic, ethical and legal challenges of the digital revolution: from big data to deep learning, artificial intelligence, and manipulative technologies.* s.l.:Springer International Publishing.

Kshemkalyani, A. & Singhal, M., 2011. *Distributed computing: principles, algorithms, and systems.* s.l.:Cambridge University Press.

Kwiatkowski, R., 2021. *Gradient Descent Algorithm — a deep dive.* [Online]
Available at: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21
[Accessed 08 2023].

Lecun, Y., Bengio, Y. & Hinton, G., 2015. Deep learning. *nature,* Volume 521, p. 436–444.

Li, X. et al., 2020. A Community Platform for Research on Pricing and Distributed Machine Learning. In: *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS).* s.l.:IEEE, p. 1223–1226.

Mayer, R. & Jacobsen, H., 2020. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR),* Volume 53, p. 1–37.

MyGreatLeanring, 2022. *Types of Neural Networks and Definition of Neural Network.* [Online]
Available at:

https://www.mygreatlearning.com/blog/types-of-neural-networks/
[Accessed 09 2023].

Ruder, S., 2016. *An overview of gradient descent optimization algorithms,* s.l.: s.n.

Sagiroglu, S. & Sinanc, D., 2013. Big data: A review. In: *2013 international conference on collaboration technologies and systems (CTS).* s.l.:IEEE, p. 42–47.

Savitz, E., 2012. *The Big Cost Of Big Data.* [Online]
Available at:
https://www.forbes.com/sites/ciocentral/2012/04/16/the-big-cost-of-big-data/
[Accessed 2023].

towardsai.net, 2022. *Main Types of Neural Networks and its Applications — Tutorial.* [Online]
Available at: https://towardsai.net/p/machine-learning/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e
[Accessed 09 2023].

Zaharia, M. et al., 2010. Spark: Cluster computing with working sets. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing.* s.l.:s.n.

X. APPENDIX

Data Source:
https://www.kaggle.com/datasets/alexattia/the-simpsons-characters-dataset