ISSN: 0974-5823 Vol. 7 No. 2 February, 2022

*International Journal of Mechanical Engineering*

# DEEP LEARNING ON CANADIAN INSTITUTE FOR ADVANCED RESEARCH -10 USING SERVERLESS COMPUTING

[1]**Gulab Sha Shaik**
**Designation** - M. tech
**Department** - Computer Science Engineering
**University** - Koneru Lakshmaiah Education Foundation

[2]**Dr. K. Thirupathi Rao**
**Designation** - Professor
**Department** - Computer Science Engineering
**University** - Koneru Lakshmaiah Education Foundation

## 1. Abstract

These days, deep learning is getting more attention from the experts and researchers in the fields of computer science, machine learning, and artificial intelligence. Deep learning is found to be more efficient and advanced than machine learning (ML) in training the neural network systems in different environments for solving various highly intricate problems. New models, techniques and environments are being developed regularly now. Cloud computing, on the other hand, is an indispensable part of deep learning and all allied technologies. It provides many opportunities for training and development of deep learning processes. Again, the development of Server less computing has been found to be advantageous for various deep learning models. This paper has aimed to evaluate two of the major constraints in deep learning, viz. memory and time for CIFAR-10 dataset training. We have executed AWS lambda, Dynamo DB, and Elastic File System (EFS) with the help of VGG-19 for deep learning framework. Our experiment reveals that the memory problem can be overcome to a great extent when a system has been created combining Lambda, EFS, and S3 Bucket. Again, with the combined application of distributed and serial workers we got an impressive result in overcoming the time-relation issues.

**Keywords:** Deep learning,Server less, AWS Lambda, cloud computing, CIFAR-10, Neural Network

## 2. Introduction

Deep learning is an advanced from of machine learning (ML) and artificial intelligence (AI) that can resemble the human thinking pattern more efficiently than ML or AI. Deep learning can be efficiently applied to large datasets where multi-layer neural networks functioning in diversified ways can learn from a large pool of data. Here, the systems are trained to interpret the constant flow of data based on the past outcomes. In deep learning, the main advantage is that the decisions can be developed without distinctly coding for decision-making process. The decisions are made on the basis of machine learning from past and present training sessions. It can also be noticed that the training process is complex and time consuming. It contains several threads or processes that require high-range CPUs, GPUs, and RAMs. Getting these components separately can be expensive.

Serverless computing [1] is a reliable component of cloud computing system that has can be applied effectively to solve difficult problems. This technology is executable and result-oriented having the capacity to coding and running software systems and various need-based services without adopti8ng external services. With the help of cloud computing serverless resources can be hired only when it needs that support to overcome storage and time constraints in deep learning. As the demand for new models are increasing training on the cloud has become every effective. Serverless computing is possible in a closed environment as well. This can be called serverless environment that allows the developers and researchers to focus on creating appropriate solutions than solving casual problems related to servers. Moreover, multiple hypotheses can be tested simultaneously that saves time and energy as well. Serverless solutions on cloud can be scaled up or down as per the situations. These are need-based, agile, and stateless. Naturally, these are cost-effective at the end. Developers and researchers finds it very helpful as they no need to focus on the external servers and can code easily and learnt to maintain their software's on the cloud.

At this point, it is important to understand the pros and cons of deep learning models in the current state. Deeper understanding of the advantages and disadvantages of an application like deep learning help to ease the development of serverless runtimes significantly. It also help to add new and important features in the backend that ultimately help to improve the applications even further.

**Our key contributions in this paper includes the following:**

- Proposing some key architectures for serverless computing that would effectively train large neural networks by offering data parallelism.
- Proposing options for reducing training delay by optimizing parallel computing architectures.
- Proposing effective serverless environment for significantly increasing the storage required for cloud computing and training bulky models.

- Optimizing the cost with using only free account in which some resources are available with no cost and some used as we go with pay as you proceed.

## 3. Literature Review

A significant part of literature available in this domain can be categorized into two heads. First, benefits on cost performance over other runtimes like distributed computing or virtual machine. Second, expanding the application of serverless computing during the runtime. Examples of the first category includes tasks that are event-driven and having short runtime. Most of the applications in the second category do not consider ML and their remains no target to develop serverless systems for neural network trainings to increase storage and decrease runtime.

In one of the previous research projects works [1] the usefulness of serverless computing for developing large neural network models has been focused. Here, the researchers have used AWS Lambda environment in the Mx Net deep learning framework.

Literature available in this discipline includes statistically analyzed [3] 25 open-source projects where serverless architectures have been used that develop useful metrics that applies diverse features of software system preservation. It is revealed that migrating a software application on the Lambda positioning architecture reduces the hosting cost by more than 70%. In this architecture high throughput not the low latency is important [2].

## 4. Methodology

Deep learning efficiency is primarily based on the efficiency of the training processes and sessions. A successful training over serverless computing environment will help to overcome the time and memory constraints. This is the main challenge in serverless computing. Apart from this, computational capacity is another challenge.

In this project, out focus will remain on solving these different challenges with the use of AWS Lambda [10]. This is a popular event driven service developed by Amazon Web Services (AWS). For memory configuration, the much-needed computational capacity of AWS Lambda increases proportionately. So, having the system with higher memory serverless environment might provide higher computational capacity.

### A. Storage

AWS Lambda has short lifetime memory of 512 MB only in the "/tmp" directory during the runtime. This can create a severe memory issue as any data stored during the runtime could not be traced later. This makes loading libraries like PyTorch a big problem [11]. Even dynamically loading other types of libraries like TensorFlow [12] can be equally problematic.

To overcome this issue researchers and developers have proposed different solutions. Caballer et al. [12] formed a "Docker image" then uploaded it in the Lambda function. A dynamic environment is developed that allows regular update of the uploaded library and consequent dataset in the image space. This makes data locally available saving time and reducing latency to a great extent. However, it should be kept in mind that whi8le using this method, we have to add the necessary development kit from the AWS itself not otherwise and the library that would be used for the development purpose. This method may also require an AWS repository for storing the container images [19]. This is called Electric Container Registry (ECR). As far as AWS is concerned, an ECR acts like a Docker Hub [20]. Another productive approach is adding a "coating" to Lambda. A coating is nothing but compressed files. All the compressed files and libraries from the lambda function are stored in "/opt" directory. The user can add any library to his file and makes possible that all smaller changes anytime without uploading huge quantities of data for a simple change that can take large space/memory otherwise.

Application of "simple storage service" of AWS can also be fruitful. This is known as S3 bucket [19]. In this way, buckets can be filled with objects and later accessed through SDK. In all types of AWS Lambda runtime, AWS SDK is immediately available. Here S3 bucket can be used to store dataset and make available whenever required. This helps to find more space for loading dataset and relevant training. S3 is dynamic in the sense that it can support byte streaming that can be used anytime even during runtime for storing datasets and developing models. Create some folders to all workers and add more workers to assign these folders. However,these can add some overheads to the scheduled tasks. In S3 bucket object locking system is not available. Thus, one task can apprise the whole model. Thus, the same object cannot be used for all workers that is become an advantage for the system users. This makes blocking unauthorized users easy and immediate. Using Amazon Virtual Private Cloud (VPC) to connect to S3 resources and on-premises resources will be beneficial and secure. The storage challenge can be effectively tackled with the various useful services of AWS.

Amazon offers Elastic File System (EFS). It is a simple elastic file system (EFS) that you can configure and forget. The flexibility and utility of Amazon EFS is enormous. There is no minimum fee or configuration charge. One has to pay as per usage. Williams et al. [17] mentions that storage capacity can be enhanced to a great extent with Amazon EFS. Amazon EFS can be mounted on Amazon Lambda [18]. When EFS is mounted, it creates the system to be fitted on the lambda, as it gets more space for training the model. Amazon EFS also has the facility of file locking that offers both shared locking and individual locking systems. This helps several workers to go for the required updated from their parts without worrying about the uniformity and reliability of the sile system.

### B. Time

Time constraints in the implementation of codes is another major hurdle in serverless computing. A user does not manage the server; hence, the user does not have the right or capacity to configure the code execution time. If any suspicious or error code is running, then cloud can stop the process automatically as it provides only 15 mins . AWS, the cloud service provider, allows the users to configure the maximum duration for which the respective codes will run but they have limited the execution time to 15 minutes maximum.

### C. Sequential Execution

In Serverless environment, we can call lambda instance with another lambda instance.the parameters are passed from the callee lambda to called lambda and it is viceversa until the training process going on.Thus supporting sequential process and it ends when the desired output is received.Thus we need the constant contingent on the calling instances incrementserially.

### D. Training Process

Today's serverless computing is a challenge to traditional training process as it gives faster results. In the traditional process, it is mandatory to takewhole datasets and essential parameters instantly accessible. In serverless environment training process is a bit different. Here, the parameters need to be simulated during the computing transition process has become complex to process uninterrupted. This is discussed elaborately in point C. Ant serverless computing instance is basically stateless. This means the information of the present execution is not carried forward to the next level of execution. This is advantageous in many instances as we can emphasis the distributed training.

### E. Distributed Training

Distributed training encompasses recapitulating the datasets and appraising the weights of the model. Specific algorithms are used to update the weights of a model. The frequency of updating the weights depend on the algorithm type. Gradient descent is a popular choice in this case. It contains three variants, viz.

Batch Gradient Descent (BGD) [21]. If the dataset is large, it would naturally contain several data points. In that case pretty long time is needed for one iteration. The same process is repeated as long as any one data point is present. This process, as predictable, is time consuming. In many cases it may go beyond the permissible code execution time.

Stochastic Gradient Descent (SGD) [22] is comparatively faster and takes lesser time than the previously discussed one. Here, one data set is considered as a whole dataset and the weights are updated accordingly. The developer gets instant feedback. It may seem an advantageous process but not always. Since it assumes all data points as a whole dataset, it cannot separate outliers. Outliers are data points that shows a different trend. Thus, outliers can distort the weights of a model. In this matter, BGD is more reliable than SGD. In a distributed environment, SGD seems not suitable.

The third one is Minibatch Gradient Descent (mini BGD) [23]. It is the miniature version of BGD.

## 5. Proposed System

### A. Architecture

As section 4 shows, some services are appropriate for certain scenario while in other scenarios they are irrelevant. We need to understand which services are suitable for which scenarios. Wd can mix the services for the most dynamic approach for getting the desired results. Lambda, EFS, and S3 can be combined to overcome the storage or memory issues. S3 helps to add extra space to manage multiple workers working simultaneously in a system. S3 follows the mechanism of "Write Once Read many" where data can be written once [30]. For making any change to the file, the whole file must be written once again. We will use EFS to store libraries, dataset, and model. For solving the storage issue, we will customize lambda and add storage with the help of EFS.

For solving the time constraints, we will again customize the system. We will use a combination of workers at any point of time. At first, we will choose a set of workers who will work in a distributed manner in a parallel system. Each worker will work on distinct tasks. As the training sessions completes, the workers store the result in pre-defined EFS directory. Later, lambda will use those result gradients to update the weight. The workers are invoked again, this continues. How many workers to be taken during training process depends on the parametric values.

We will apply mini BGD for customizing a distinct training approach. We will segregate the dataset in several parts and distribute them among several worker branches as we will finds suitable. Then, each worker branch will apprise the weight of the used model. This methodology will reduce the networking issues. It will further help to retrieve the parameters.

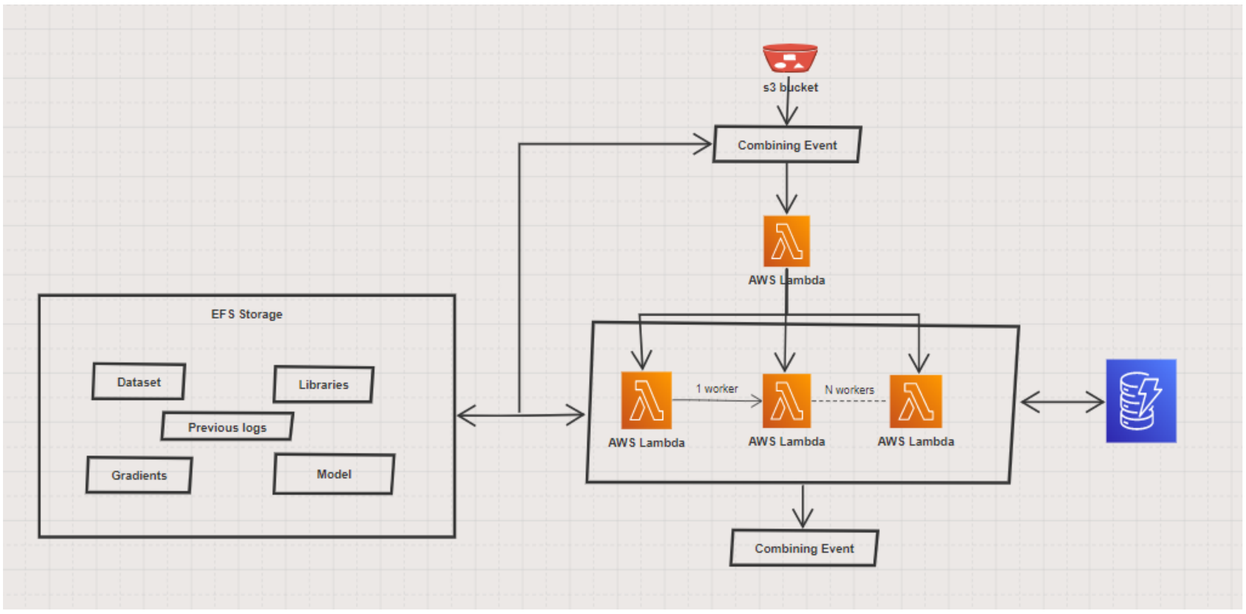

*Fig 1. the architecture for adding phase and progress the training process*

Figure 1 shown above is depicting an architecture for adding phase and progressing through the training process. This architecture is based on AWS EFS,S3,DBDynamo and AWS Lambda. AWS Lambda here is stimulated with S3 bucket and DynamoDB. AWS EFS keeps provision for deep learning by maintaining the libraries. It also maintains the dataset that is an inherent part of training, the gradient that is obtained from training, previous logs from the past model and the model that structures the training approach. Only single lambda function is used to write the whole code. The lambda function also has two other purposes. First, it is used for parameter server that helps to calculated the gradient. Second, it is also used to combine the gradients obtained from the worker branches. Moreover, the lambda function is useful in updating DynamoDB table that provides all updates to the operators. Just by inserting data in the DynamoDB, an user/operator can tr5ack the status of the training process.

## B. **Training**

At the beginning of the training process, we need to invoke the S3 Bucket, which is having IAM role key . The key allows only specified users to modify the system. This triggers the lambda function which process training. S3 writes for all workers the activation of the lambda function as it bears the raise in the storage. Then the Lambda in JSON file gets started with event," start" and also assigns some workers for the training process to reduce the time.

{ "start" : true, "workers" : 6}

As soon as lambda gets triggered, it assigns with different unique IDs to each worker and sends them in training(0-(N-1)) where N is no. of workers. To work in lambda function.

{ "workerno": 3,"workers": 6,  "epochsdone" :7}

For Worker invocation event.

{ "combine": true, "workers":6, "epochsdone": 7}

**For Worker invocation event:**

At first the lambda related to a worker gets the detail of the event and collects the data stored in the EFS. At first, it loads the weights assigned and available for the pre-trained models. Then, the system starts the training process on the dataset. The gradients so formed for each step are saved on EFS. As the execution timing comes to an end, the lambda function merge all the available gradients. As the training phase completes, only one lambda function could be traced that initiates the combining phase. Here, each worker in the training process checks how many gradient files are available on EFS. This process to decrease the weight on the lambda, while the gradients is equals to workers then an adding event takes place.

**For adding phase invocation event:**

We can find that the event phase carries the key named "combine" with it with an aim to inform the lambda function that the phase is a combine phase. During the phase, lambda function sums up all the available gradients that are generated by the worker lambdas. After that it updated the weight. For updating the weight, the formula proposed by Li et al. [24] is used. This is the phase when a decision is made regarding the future of the training, i.e., whether it needs further intervention or it can be stopped right now.

## C. Setup

In this project, the whole architecture is based on CloudFormation. This allows machine learning (ML) with the help of serverless computing 20 and develops pertinent resources in the AWS environment using various templates. Templets have multiple purposes in this domain. It creates essential resources, store the essential libraries, and processes the datasets as required.

We realized that some more resources are required apart from AWS Lambda and EFS to continue with our experiments. We collected these resources from AWS. While developing the resources, we paid maximum attention to the security of the system. Also managed Identity and Access Management (IAM) so the system and data accessibility remains within our control.

Figure 2 depicts the complete architecture that involves all AWS services used in the project and how they are linked with one another.

**Our experiment requires the following resources:**

- Virtual Private Cloud (VPC)

VPC is essential in this experiment as this creates a logical boundary. Such logical boundary is essential for the isolation of resources required from the pool of AWS services. VPC helps to interact freely yet with utmost security.

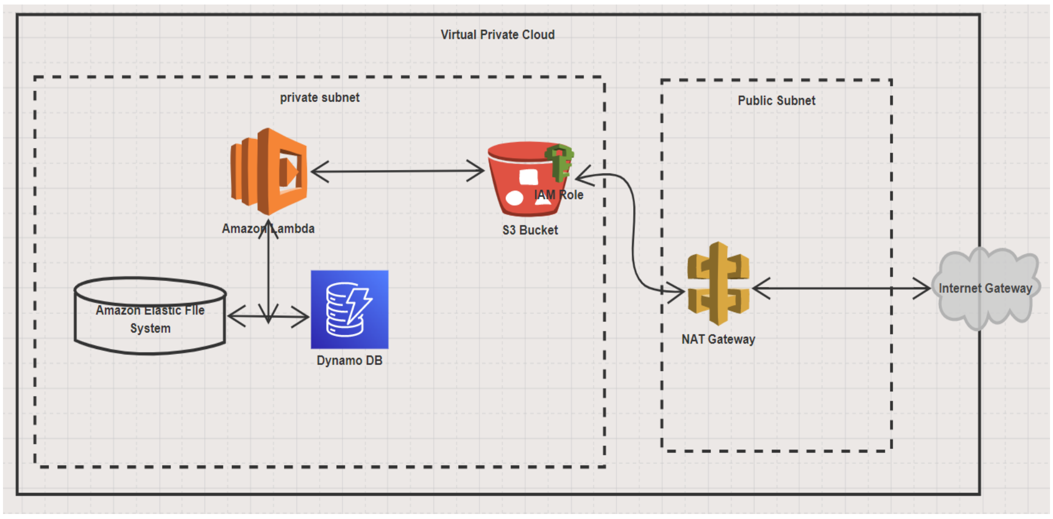

*Fig 2. Full architecture diagram*

- **Target for Mount**

Several components are imporrtant for our expereiment in this domain. Take a look:

- **Subnet**

These are “virtual subnetworks” in the defined VPC. These can create small but logical partituions within the VPC to maintain the features of the resources. These also help to interact flebibly.

In our project, we accessed two types of subnets. These are as follows:

- The public subnet

As the name suggests, the resources avaialable in this subset are available in the internet for mass use.

- The Private subnet

As the name suggests, the resources of this subnet are not available on the intertnet neither these resourcesz are meant for public use. Initially, we will deply Lambda and EFS for maintaining the provacy of the subnets to be used uin our project.

- **Net gateway**

Net gateway also needs specification to avoid any later negative consequences. Private subnet is given the access to the internet through “Network Address trasnslation”. This is an advanced level gateway that masks the resources while letting them accessible to the private network.

- **The internet gateway**

The internet gateway is meant for accessing the VPC of the system. It permits predefined public subnets and the NAT gateway that can invariably connect to the internet without compromising the security.

- **Elastic File System (EFS)**

The storage facility that can store libraries, datasets, and the model. It can also save and preserve the gradients while a training continues.

- **Mount target and related security**

The “logical mounting points” are called mount targets. These are traced in the AWS EFS and supports the other filesystems that are required while a training in continuing in the main system. Related securirty is important to maintain the quality of the filesystems. The security groups are created that would act like a firewall. The securiuty group will permi only the whitelisted sources.

- **DynamoDB**

This forms a table with the status of the training in progress. The status of progress helps to find the standard of training later.

- **Lambda**

This is applied to train the model as per the prefixed system. Any lambda can request for the easy accessibility of the file system but they need to pass through the systematic permission process.

- **IAM Role**

Lambda’s filesystem accessibility depends on IAM. To maintain the security of the architecture at the optimum level, we allowed only Lambda.

- **S3 Bucket**

It offers updated bucket management, access to objects, and related auditing on a regular basis. By default, we employ Amazon S3 buckets and objects are in private mode.

## D. Libraries

While mapping the resources for their optimized applications, we also used libraries in the AWS EFS. These libraries will be accessible to lambdas. We have decided to install PyTorch[11] and Tensorflow[12]- both of which are popular ML libraries. These libraries have many advantages. These can be shared among several developers simultaneously. These libraries are also appropriate for deep learning. Some libraries are preinstalled in the project.

## E. Dataset

In this project, CIFAR10 dataset [26] is used. It contains 60,000 images and 10 classes. Each class contains 6000 images. Out of these 60000 images, 50000 images are kept for training and 10000 of testing.

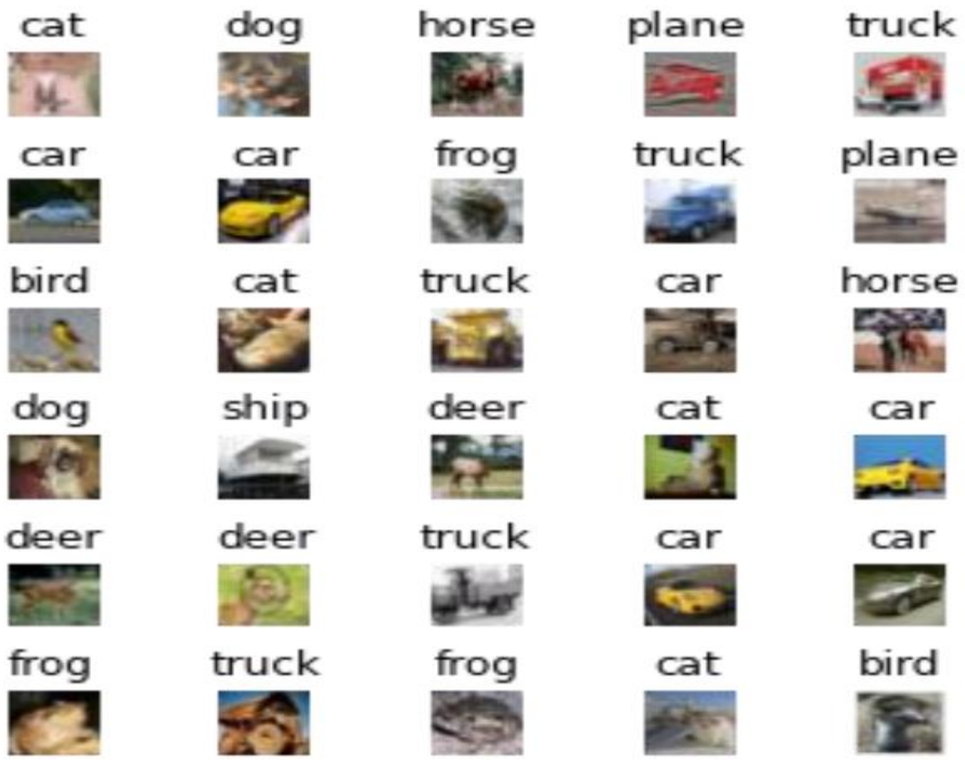

*Fig. 3Sample dataset.*

The dataset is at first downloaded. It is then extracted in the ZIP format and stored in the pickle format to form 6 different files. In which, 5 files of them is used for training purpose and last file is used for the system validation purpose. The pickle library is activated for reading the datasets. With an aim to reduced the timing of each lambda implementation, we went through all datasets before the setup process. This is called prepossessing of data. With preprocessing, we resized the images to fit them well with the model. The storage efficiency is enhanced with the use of NumPy arrays. We converted the 6 files to it. Then, the datasets are stored in four different filesystems. These four different filesystems include training data, test data, training labels, and test labels. This saves much time since Lambda needs to read fewer files during the training now.

## 6. Experiments

### A. Model

VGG19 [26] is used in our project for examining our proposed system. This has a model length of 549MB [27]. We know AWS Lambda has 512MB limit. This means, VGG19 goes beyond that length. Thus, we resolve to have a transfer skilling for the proposed model. But that can take more than 15 minutes. This is more than the maximum runtime of AWS Lambda.

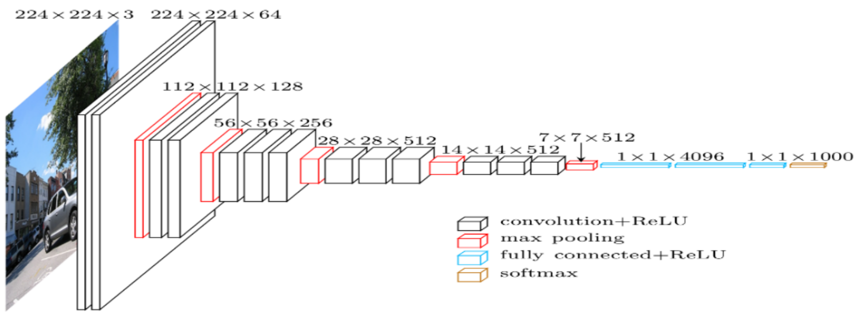

*Fig 4. VGG-19 Model*

Figure 4 above depicts a typical VGG19 model. It has a single SoftMax layer in the tail. The dimension of input image is 224x224x3. First two parameters depict the width and heught while the last one shows the number of colors per pixle. Here the number is 3 that means there are three colort values per pixel. As we use CIFAR20 for the project, we decided to reduce the dataset image from 32x32 to 224x224. The images used in this project already have 3 color pixel with range of each pixel fixed at 255. After reducing the image volumes, we stored them as libraries in the AWS EFS directory. After preprocessing, the convolution layers with maxium-pool layer taking the 3x3 to 2x2 filter. It can generate 112x112x64 output image from 224x224x64 input image. The, this output image processes to 2-3x3 convolutions layers with 56x56x128 maximum-pool layer and 128 filters. This repeates for convolution layers with dimension 4-3x3 with m,aximum-pool layer and 256 filters. Then repeates again 2 times with 512 filters with other parameters remaining the same to form the final output. It ultimately forms a convolution neural network with 7x7x512 dimensions. After this, the output is transformed to one dimension having intensely connected layer having 4096 rectified linear activation units. The output again passes through SoftMax classifier rthat helps to categorize the input images.

## 7. Results

### A. Lambda Configuration – Time

The runtime period is a serious concern in lambda function. We need to adjust it first. We have made it in between 1 second to 15 minutes. There can be a memory clash. So, we maintain it at 10GB. To obtain the most suitable runtime configuration, we experimented with the system every 1 minute with 18 steps per epoch.

It depicts that we can achieve 70 steps maximum per execution during the training period. But, every additional minute of execution provide 4-5 additional steps.

To count the epochs accurately, we decided to maintain 3 epochs per worker. Meanwhile, any additional time is rejected when a batch is found to take longer than average time.

### B. Lambda Configuration – Memory

The memory is another serious concern for the experiment. For the lambda function we need to configure it properly. A lambda function memory can remain in between 128MB to 10GB. We targeted to experiment with 1 GB setup memory increment and apply the AWS memory used log to find the maximum memory used during the runtime. We have fixed the runtime to 15 minutes that is the highest label for lambda function. After that we registered the highest number of epochs. It shows that more than 5GB memory can provide 3 epochs of training within 15 minutes. If we can complete 3 epochs, it is possible to train a model with 5GB memory. However, we found that with 8GB memory, we can keep 3 minutes buffer too.

### C. Number of workers

Ideal number of workers is another serious concern in our expereiment. We are targeting fasteszt possible training time which require optimized level of workers. We deploy several workers in parallel. As multiple workers are working in parallel, our focus is be on the runtime. We started with one worker in linear technique. Then we steadily increased the number of workers one by one till 33 workers are acting parallely.

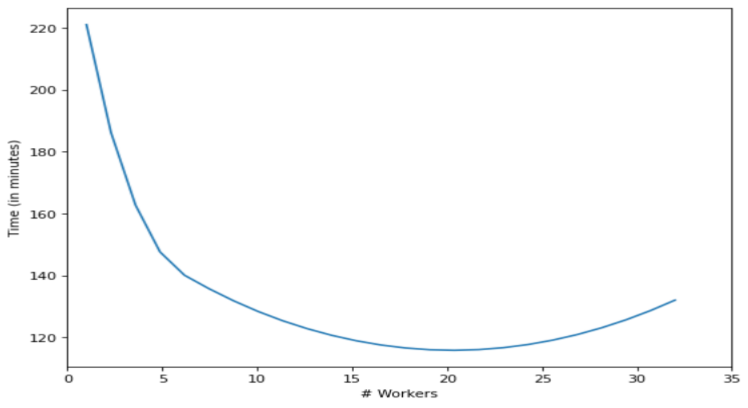

*Fig 5. Time vs Workers*

We could see that with the increase in number of workers the epoch time decreases steadily. But, beyond 17 workers, the training time again started increasing. With 33 workers active parallelly, the time taken is almost similar to 8 workers being active in the same environment. This means additional workers beyond certain level doesn't work. We find that any number of workers between 16-20 is the ideal number of workers for the optimized output. The cause of delayed training even when more than essential number of workers are working lies in the complexity at the mixing stage. In that stage all the resources from the workers are added from the EFS directory and Calculated the sum.

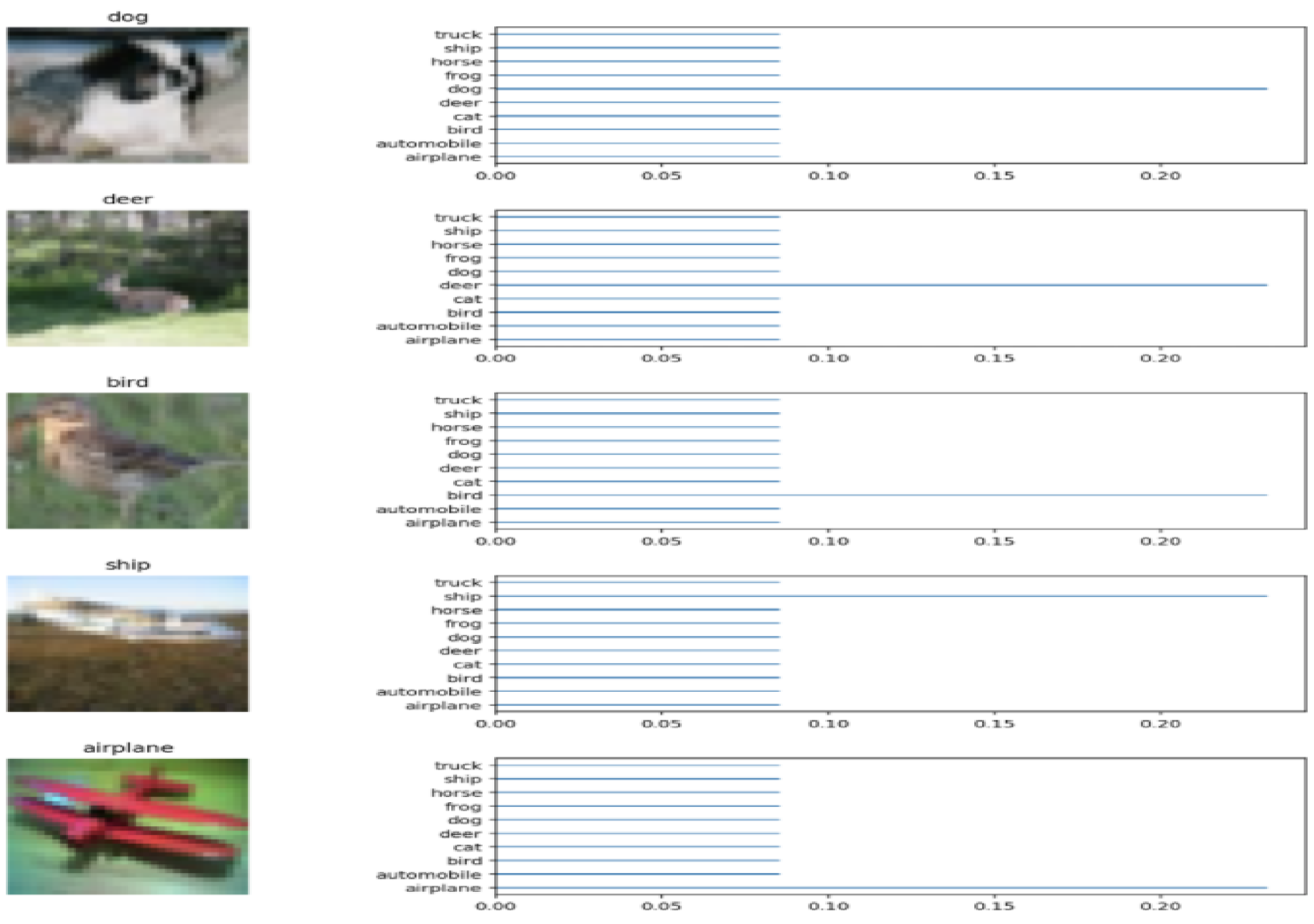

*Fig 6. Results of the trained model*

Figure 6 shows a model with 17 workers being active. It shows that with 17 workers the model learns comprehensively and fast. At this level of training, the model point out 28 images accurately out of total 30 images. As, we have taken random images prediction. Here, the overall percentage of accuracy 91% that is quite impressive.

## 8. Conclusion

The popularity and need of deep learning technique is in high demand today. It maintains a remarkable identity almost every day, and continues to train newer models.People demand of more advanced and comprehensive training techniques is increasing simultaneously. Our result indicates that model accuracy is within acceptable range while it overcomes the time and add an overhead to the provide storage. Technology today is providing different updated services for cloud computing theories, as this may give new dimension. Serverless computing has a high impact in this field. The time and storage constraints has been providing novel approaches to distributed training. Our research will show newer avenues in deep learning and its applicability in diverse fields. Deep learning is the future of technology. Training different types of deep learning models with the support of serverless computing systems is always challenging but it offers several opportunities as well.

## 9. Limitations and Future Studies

Serverless computing has multiple benefits and its demand has been surging with time. It leaves a long-lasting effect on the present workloads. In our research we have observed that serverless computing is quite effective in handling diverse deep learning workloads. This framework is an opportunity for the future. Deep learning gains maximum benefits from GPUs while ECS Fargate [29]. In ECS Fragate there is no time barrier that also quite advantageous. In future we may get to see the advancement of cloud computing in terms of medicine, cybersecurity, Transportation, Aerospace etc. However, our research offers some exciting insight in handling AWS environment that will help the future researchers to use AWS environment more productively.

## References (APA 6th edition)

[1]. VatcheIshakian, Vinod Muthusamy, Aleksander Slominski, “Serving deep learning models in a serverless platform”, ”2018, IEEE International Conference on Cloud Engineering “

[2]. Gojko Adzic and Robert Chatley. 2018,” Serverless Computing: Economic and Architectural Impact”. In Proceedings of 2018, 11th Joint Meeting of the European Software Engineering Conference ,Germany

[3]. Louis Racicot , Nicolas Cloutier , Julien Abt and Fabio Petrillo”, “Quality Aspects of Serverless Architecture: An Exploratory Study on Maintainability”,” 14th International Conference on Software Technologies (ICSOFT 2019)”, “IEEE”.

[4]. N. Amjady and M. Hemmati, “Energy price forecasting - problems and proposals for such predictions,” IEEE Power and Energy Magazine, vol. 4, no. 2, pp. 20–29, Mar. 2006.

[5]. Y. Khourdifi and M. Bahaj, “Applying Best Machine Learning Algorithms for Breast Cancer Prediction and Classification,” 2018 International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS), Dec. 2018.

[6]. N. Savage, “Going serverless,” Communications of the ACM, vol. 61, no. 2, pp. 15–16, Jan. 2018.

[7]. B. Hassan, Saman A. Barakatand Qusay I. Sarhan (2021),” Survey on serverless computing”,” Journal of Cloud Computing: Advances, Systems and Applications”, “SpringerOpen”.

[8]. L. Feng, P. Kudva, D. Da Silva, and J. Hu, “Exploring Serverless Computing for Neural Network Training,” 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), Jul. 2018.

[9]. “Convolutional Neural Network (CNN): TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn. [Accessed: 29-Mar-2021].

[10]. R. W. Hendrix, “Lambda,” Amazon. [Online]. Available: https://aws.amazon.com/lambda/ [Accessed: 29-Mar-2021].

[11]. A. Paszke et al., "PyTorch: An imperative style high-performance deep learning library", Proc. Adv. Neural Inf. Process. Syst., pp. 8024-8035, 2019.

[12]. M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous systems", OSDI, 2016.

[13]. A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, “Serverless computing for container-based architectures,” Future Generation Computer Systems, vol. 83, pp. 50–59, Jun. 2018.

[14]. “ECR,”. [Online]. Available: https://aws.amazon.com/ecr/. [Accessed: 29-Mar2021].

[15]. P. Garcia Lopez, M. Sanchez-Artigas, G. Paris, D. Barcelona Pons, A. Ruiz Ollobarren, and D. Arroyo Pinto, “Comparison of FaaS Orchestration Systems,” 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Dec. 2018.

[16]. “Step functions,” Amazon, 1975. [Online]. Available: https://aws.amazon.com/step-functions/. [Accessed: 29-Mar-2021].

[17]. M. Sindi and J. R. Williams, “Using Container Migration for HPC Workloads Resilience,” 2019 IEEE High Performance Extreme Computing Conference (HPEC), Sep. 2019.

[18]. “EFS,” Amazon, 2000. [Online]. Available: https://aws.amazon.com/efs/. [Accessed: 29-Mar-2021].

[19]. “S3,” Amazon, 2002. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 29-Mar-2021].

[20]. “Docker Hub.” [Online]. Available: https://hub.docker.com/. [Accessed: 29-Mar-2021].

[21]. “Gradient Descent Algorithms,” An Introduction to Neural Networks, 1995.

[22]. H. Robbins and S. Monro, “A Stochastic Approximation Method,” The Annals of Mathematical Statistics, vol. 22, no. 3, pp. 400–407, Sep. 1951.

[23]. S. Khirirat, H. R. Feyzmahdavian, and M. Johansson, “Mini-batch gradient descent: Faster convergence under data sparsity,” 2017 IEEE 56th Annual Conference on Decision and Control (CDC), Dec. 2017.

[24]. M. Li et al., "Scaling distributed machine learning with the parameter server", Proc. OSDI, vol. 14, pp. 583-598, 2014.

[25]. F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lockfree approach to parallelizing stochastic gradient descent," in NIPS, 2011.

[26]. K. Simonyan and A. Zisserman, Very deep convolutional networks for largescale image recognition, 2014. [27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images", 2009.
[27]. "Keras Applications," Team Keras, 2020. [Online]. Available: https://keras.io/api/applications/. [Accessed: 29-Mar-2021]
[28]. J. Deng, W. Dong, R. Socher, L. -J. Li, K. Li and L. Fei-Fei, "Imagenet: A largescale hierarchical image database", Computer Vision and Pattern Recognition 2009. CVPR 2009. IEEE Conference on., pp. 248-255, 2009.
[29]. https://aws.amazon.com/fargate/.
[30]. Naik, Vidish, "Machine Learning Using Serverless Computing" (2021). *Master's Projects*. 993.