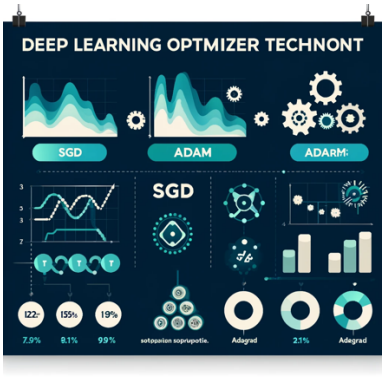
EXPLORING THE IMPACT OF GRADIENT CENTRALIZATION ON MODERN OPTIMIZATION TECHNIQUES



Aniket Chougule A20552758
Sahil Bhaware A20552865
Vandan Vaishya A20552905

ILLINOIS INSTITUTE OF TECHNOLOGY

Computer Science
CS-584 Machine Learning
Prof. Steve Avsec

INDEX

1. INTRODUCTION

The impact of deep learning is already reaching many different areas. Visual recognition has become more powerful and natural language is now easier for computers to learn. There are three principal reasons for this: smarter neural network architectures, very large datasets, and — most important — better loss functions (that is, optimization methods).

The bottleneck in training DNNs is the iteration of updating the DNN parameters to minimize the loss function. The loss function, as defined in the third section, measures the discrepancy between the outputs predicted by DNNs and the actual outputs.

The performance of Deep learning models can be very sensitive to the details of the optimization techniques used, affecting both how quickly and cheaply a model can be trained, but – perhaps most importantly – whether it can accurately generalize to make predictions on things it has never seen before.

The Recent improvements allow machines to learn more effectively; Stochastic Gradient Descent (SGD), Adam, and the related mutant RMSprop are some of the algorithms that currently allow for the training of neural nets. Most of these techniques have been directed towards optimizing the neural networks by innovating the variants of the gradient descent method such as stochastic gradient descent (SGD), gradient descent with moment (SGDM), and more adaptive algorithmic method such as Adaptive Gradient Algorithm (Adagrad), Root Mean Square Propagation (RMSProp), and Adaptive Moment Estimation (Adam), which are geared

towards solving issues of vanishing gradient and being trapped in a bad local minimum. A few techniques such as batch normalization and weight normalization which stabilize the learning process by normalizing the input layer or the weights have also been used. Batch normalization normalizes the activations across a mini-batch and so speeds up training, allows higher learning rates, and reduces sensitivity to network initialization. As before, the weight standardization (WN) scale-invariant procedure rescales the weights matrices so that they have zero mean and unit variance, again removing the sharp pikes from the optimization landscape and speeding up the learning process.

2. NORMALIZATION TECHNIQUE

1. Batch Normalization

Batch Normalization increases the efficiency of training deep neural networks. Introduced by Sergey Ioffe and Christian Szegedy in 2015, it mainly addresses issues with the speed and stability of neural network training. Which solved by standardizing the inputs of each layer. For each mini batch of input layer, it normalizes the data to have a mean of 0 and a variance of 1. After this re-scaling and re-shifting was done with the two new parameters per layer learned during training.

Batch Normalization stabilizes learning, reduces internal covariate shift, enables higher learning rates, and decreases sensitivity to initialization. Typically, batch normalization is applied after computing the linear transformations and before applying the activation function.

2. Weight Normalization

Weight normalization is an innovation focusing to enhance the efficiency of stochastic gradient descent optimization, designed by Tim Salimans and Diederik P. Kingma in 2016. It was invented from batch normalization by focusing on normalizing the weights rather than inputs of neural network.

The primary characteristics of weight normalization include:

1. This method speeds up the learning process by treating the size and weight adjustments differently. Due to this, model finds the best settings quickly.

2. The method simplifies the whole process because it reduces the complexity that comes when different parts of the model depend on each other too much. This makes the optimizer's job simpler.
3. It's faster than other methods like batch normalization, which required more calculations based on a group of data examples.

Weight normalization transforms the model's parameterization. In a neural layer with weight matrix W, it restructures W in terms of a parameter vector v and a scalar g . Thus W is determined by($g \times \frac{v}{||v||}$). This approach is integrated into both the forward and backward phases of the training process.

3. PROPOSED TECHNIQUE

1. Introduction to Gradient Centralization

Gradient Centralization stands out as a distinctive optimization technique, diverging from traditional methods by targeting the gradients rather than the activations or weights. It zeroes in on centralizing the gradients of the loss function in relation to the weights, significantly refining the training dynamics. Remarkably straightforward to apply, GC can be incorporated into any gradient-based optimizer with minimal adjustments, often just an additional line of code. This ease of integration, coupled with its theoretical and empirical benefits, makes GC an intriguing area of exploration.

2. Theoretical and Practical Advantages of Gradient Centralization

Gradient Centralization revamps the optimization landscape by centralizing gradients, which in turn regularizes both the weight and feature spaces of the network. Theoretically, this adjustment creates an optimization problem where the loss function exhibits greater Lipschitz continuity. Practically, it yields a more consistent and efficient training regimen.

Moreover, GC enhances the model's generalization capability, tweaking its ability to generalize from training data to unseen data—a vital feature in deep learning where overfitting and performance on novel data are perennial challenges.

3. Objective and Scope of the Current Study

This project aims to carry out an in-depth and comparative study of Deep Neural Networks trained along with without Gradient Centralization. The focus of this study will be on the general image classification. With perspective to performance metrics from these applications, this project will try to quantify the influence of GC on DNNs training efficiency and generalization. Finally, this report would provide a complete analysis of the models evaluated based on the experiments performed on DNNs trying to identify the disparities based on training dynamics and outcome metrics with and without GC. Essentially, with the above-required experiments and examination through statistical techniques, the study would provide valuable discourse on the impact of gradient centralization as a mathematical optimization towards training deep learning architectures.

Comparison with Normalization

Attribute	Gradient Centralization	Batch Normalization	Weight Normalization
Primary Focus	Directly modifies the gradients	Normalizes the activations of each layer	Normalizes the weights by decoupling the magnitude and direction
Objective	Stabilize training by centering gradients to zero	Mitigate internal covariate shift and stabilize training	Accelerate convergence by improving the conditioning of the optimization problem
Impact on Learning	Potentially speeds up convergence and stabilizes training	Helps in faster convergence, allows for higher learning rates, less sensitive to initialization	Often results in faster convergence than standard parameterizations
Computational Complexity	Low, involves basic arithmetic operations on gradients	Medium, involves computing mean and variance across mini-batches plus additional scale and shift parameters	Low, involves normalizing weight vectors
Usage Scenario	Useful in any neural network training using gradient descent	Effective in deep networks with deep stacks of layers where data distribution shifts significantly	Useful in networks where rapid convergence is necessary, such as recurrent networks

4. OPTIMIZATION TECHNIQUES

1. Stochastic Gradient Descent(SGD)

It is a simple and powerful tool for fine-tuning neural networks. It makes small adjustments to the model's settings based on a few training examples at a time called mini batches. This method is quick and uses less memory, even when dealing with lots of data. which makes this better choice than trying to learn from all the data at once. The disadvantage of is because it uses only a small chunk of data for each update, it can sometimes be unpredictable and less smooth in finding the best settings for the model.

2. Gradient Descent with Momentum (GD with momentum)

Gradient Descent with Momentum is an upgraded version of SGD that helps the model to learn faster by giving it a push in the right direction. This method remembers a bit from the previous steps and uses that to make the learning process smoother and quicker. It's work better when the path to the best solution is twisty or when the data is a bit messy.

3. Adaptive Gradient Algorithm (Adagrad)

Adaptive Gradient Algorithm (Adagrad) is a sophisticated optimizer that adapts the learning rate to the parameters for gradient-based optimization. It applies smaller updates for parameters related to frequently occurring features and larger updates for those tied to rare features. Adagrad is particularly advantageous for sparse data. Nonetheless, a drawback is that

the learning rate can diminish significantly during training, which may prematurely halt the learning process.

4. Root Mean Square Propagation (RMSProp)

RMSProp is an optimization technique that continues from the Adagrad technique, it also changes the learning rate of the parameters, but uses a moving average of the squared gradients to compute the learning rate, and the gradient decreases with the gradient magnitude. The reason for this implementation of an algorithm is that it helps RMSProp to escape diminishing learning rates as on Adagrad. This optimizer adapts the learning rate according to the parameters. It is helpful because any real-world data is not like a normal distribution with zero mean. RMSProp optimizer is an adaptive learning-rate optimization algorithm designed. Due to the adaptation, RMSProp is stable in practice and does well in a range of more complex models and data. Adaptation technology makes it easy to apply.

5. Adaptive Moment Estimation (Adam)

Adam optimizer is a combination of RMSProp and momentum. Instead of using a sum of gradients and squared gradients for scaling the learning rates, Adam creates a new way of optimization. By storing a short functional one that relates to the gradient, and the other one square averages to the post, the way is updated. The reason Adam is an optimizer because it works well on a wide range of problems, and it is very quick to get good results. Adam also uses adaptive learning rates, which can be great for networks or a dataset.

5. DATASET

1. Summary

The CIFAR datasets, crafted by Vinod Nair, Geoffrey Hinton, and Alex Krizhevsky, encompass CIFAR-100, which is known for its complexity. With a greater number of classes and fewer examples within each, compared to CIFAR-10, CIFAR-100 presents a more challenging benchmark for machine learning models.

2. Content of the Dataset

I. **Number of Classes:** 100

II. **Number of Images:** 60,000

III. **Images per Class:** 600

IV. **Training Set:** 50,000 images (500 images per class)

V. **Test Set:** 10,000 images (100 images per class)

3. Details of the Image

I. **Pixel Size of Image:** 32 × 32

II. **RGB color channels:** 3

III. **Picture Type:** As the images are of 32 by 32 pixels and have a low resolution. Convolutional neural networks may be trained on this manageable dataset.

4. Hierarchy of Classes

CIFAR-100 stands out from many other picture datasets due to its hierarchical class structure. 20 super classes are created from the 100 classes. Two labels are included with every image:

I. **"Fine" label:** the category it falls inside (0-99)

II. **"Coarse" label:** the superclass to which it belongs (0-19)**.**

5. Application

The CIFAR-100 dataset is often used by the research community to create and evaluate machine learning algorithms in the following categories:

1. **Image Recognition**: Construct models that can perfectly distinguish objects and classify it to 100 different categories.
2. **Transfer Learning**: Modification of previously learned models on CIFAR-100 to new tasks, given that they are focused on similar kinds of tasks or datasets.
3. **Neural Network Architecture Research**: The CIFAR-100 allows researchers to think about and compare various architectural innovations for deep learning models.

6. IMPLEMENTATION

1. Data Handling and Initialization

For CIFAR-100 data to be used, the processing must involve reading it from its stored format and normalizing pixel values. It would also involve encoding the class labels into the proper format for any classification-oriented task. Preprocessed data is used during neural network training to ensure that the labels are in the right format for error calculation and that the inputs are on the same scale.

This script also initializes the weights and biases in various layers by setting the initial values. This process is very important because , it gives the optimization algorithms' starting point. To maintain a suitable variance in activations between layers, the network follows specific initialization procedures, which also help in speeding up the training process.

2. Forward and Backward Propagation

Every step at training, the code calculates what the network thinks about answers for a given set of data. It then measures how far off this answer are from the actual results using function called cross-entropy loss function. This helps to understand how much error the network is making. To deal with complex patterns, the model uses functions that help it to learn not just straight-line relationships but more complex ones too, that's called forward propagation.

Then, during backward propagation, the code figures out how it needs to adjust the dials and levers of the model (the weights and biases) to get closer to the right answers. It's like the

network is learning from its mistakes, tweaking how it thinks based on the errors it made. This learning involves math concept called the chain rule, which helps to spread out the error across all the parts of the network.

3. Gradient Centralization

Gradient Centralization (GC) stands out in its application during the gradient computation phase of the training process. By deducting the average value from the gradient of each layer, GC notably stabilizes the learning by directly reducing the internal covariate shift in the gradient space. This stabilization not only accelerates the convergence but also support the generalization capability of deep learning models. This ensures that the updates to the model parameters are more uniform and focused by centralizing the gradients, thereby streamlining the training efficiency.

4. Optimization and Training

Our code uses different methods to tweak the model during training. These methods change the model's settings(parameters and hyperparameters) by using information from the gradients. Information is a mathematical calculation that show how much and in what direction the settings should change to improve the model. Some of the methods used in our code is SGD, Adam, and RMSProp, can adjust speed up training and make the model learn better and faster.

5. Output and Utility Operations

The model handles the data in small groups during training phase. which save memory and reduce the amount of computation. All this done with changing the accuracy of gradients . It also uses these small batches to add a bit of randomness to the training process. This randomness helps to prevent the training from getting stuck at suboptimal points in the process ,called local minima, of finding the best.

5. Testing and Accuracy Calculation

a. **Forward Propagation for Testing**: During the testing phase, our code employs a forward propagation function same as used in training, utilizing the same network architecture to process test data without backpropagation. Inputs traverse the network layers, where we use ReLU for the hidden layers and SoftMax for the output layer.

b. **Prediction:** Our model makes predictions according to the SoftMax probabilities. It identifies the class with the highest likelihood for each input.

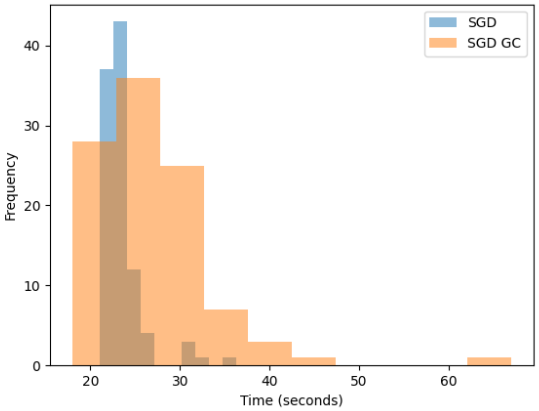
c. **Accuracy Measures:** It is calculated by the percentage of correctly predicted labels against all predictions. It is simple and straightforward metric for the model's precision across the test set.

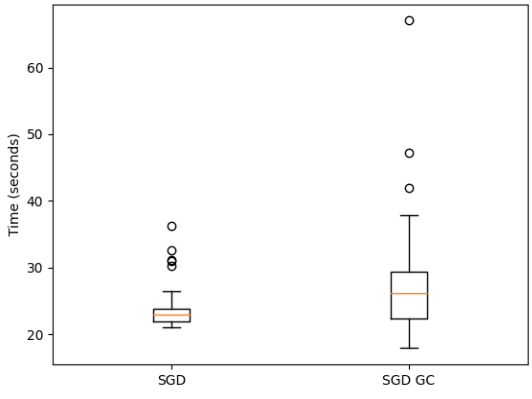
7. RESULTS

1. SGD

Overall, SGD with Gradient Centralization revealed small increments in accuracy in comparison with the standard SGD. In addition, the slight addition of the variability in training time indicates that although GC tends to stabilize the computations, it may also add elements of complexity to the training. In conclusion, although the increments in accuracy are minimal, SGD with GC can be useful for tasks for which even minor performance boosts are essential.

Metric	Without GC	With GC
Accuracy	0.984794	0.985569
Mean Time (s)	23.46077182	26.74265938
Standard Deviation	2.409982786	6.798599911

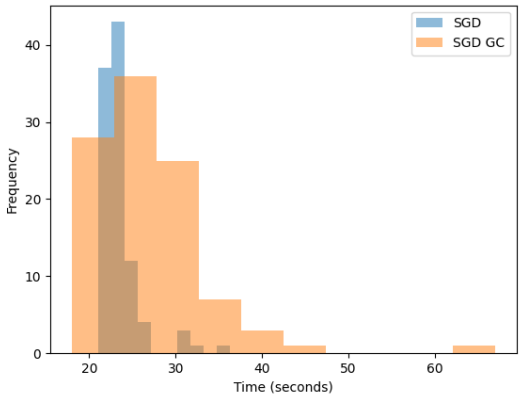


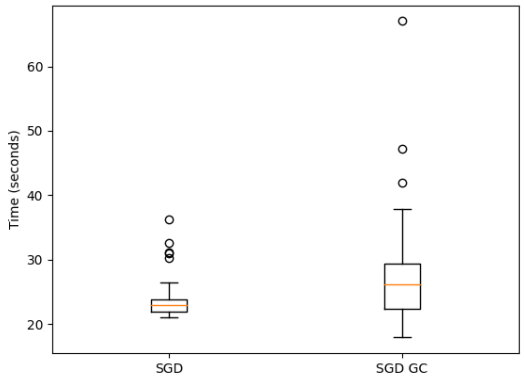


2. SGDM

SGDM uses a technique called momentum that considers the average of previous gradients to speed up SGD towards the best solution. When Gradient Centralization (GC) is added to SGDM, it not only improves accuracy but also leads to longer training times. Although GC helps SGDM converge faster, it requires careful adjustments, like fine-tuning training settings, to manage the increased computational demand effectively.

Metric	Without GC	With GC
Accuracy	0.986122	0.98826
Mean Time (s)	170.4330603	220.591015
Standard Deviation	14.23996121	35.66662485

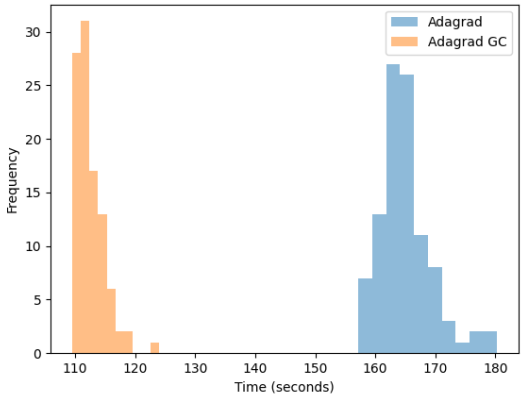


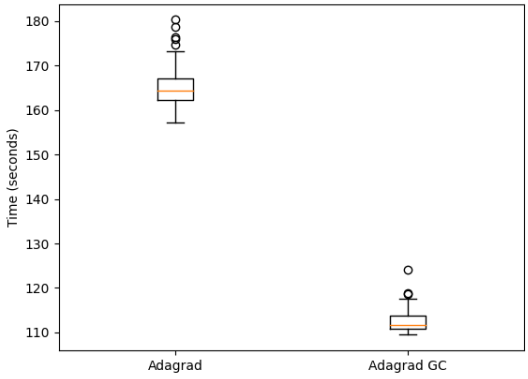


3. Adagrad

Adagrad is a well-preferred algorithm known to personalize the learning rate to parameters while making larger updates for unusual parameters. As a result of the introduction of GC, there was an improvement in accuracy, and there was a substantial reduction in mean training time. Furthermore, the lower standard deviations in training times with GC provide evidence that GC may scale more efficiently. This explains why GC might be the preferred choice to train deep models more rapidly.

Metric	Without GC	With GC
Accuracy	0.989916	0.98992
Mean Time (s)	164.9530506	112.4821352
Standard Deviation	4.371660734	2.372575783

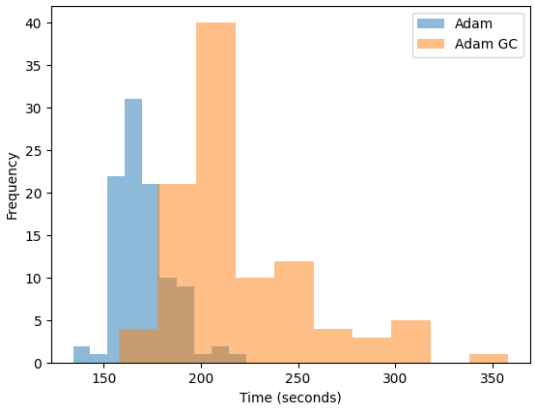


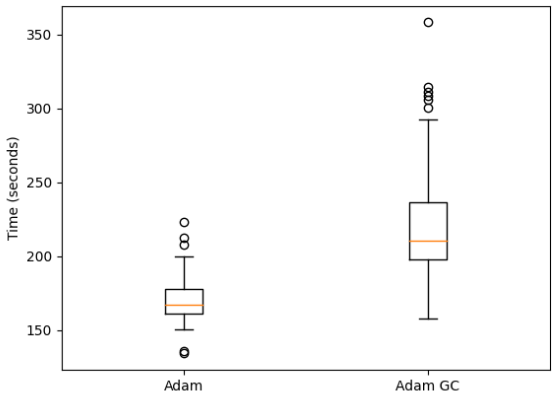


4. Adam

Adam uses the strengths of AdaGrad and RMSProp, showcasing robust performance across a variety of tasks. Integrating GC into Adam doesn't drop accuracy but does lead to extend training periods with greater variability. This suggests that while GC doesn't directly improve Adam's performance, it does not decrease it either. Instead, GC influence the speed of convergence and the uniformity of training due to the added complexity in gradient processing.

Metric	Without GC	With GC
Accuracy	0.99	0.99
Mean Time (s)	170.4330603	220.591015
Standard Deviation	14.23996121	35.66662485

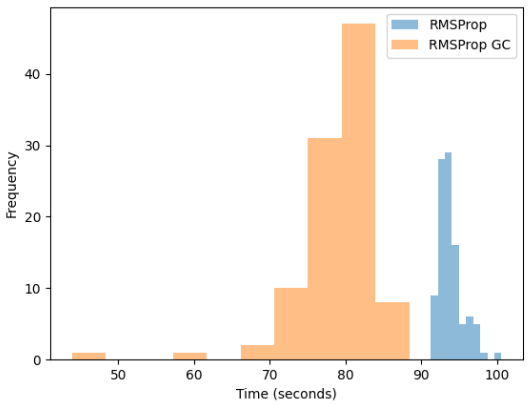


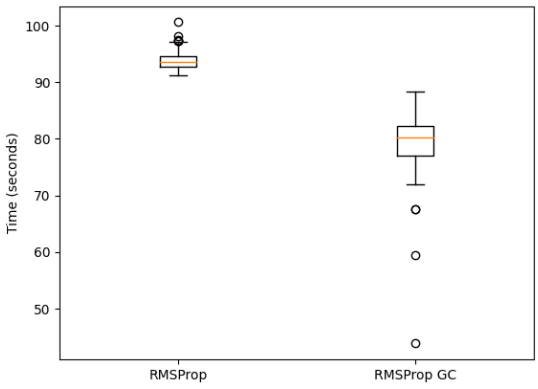


5. RMSProp

RMSProp, an optimizer designed to address the diminishing learning rates of Adagrad, maintained perfect accuracy while showing a reduction in mean training times with a slight increase in the variability of those times when applying GC. Thus, GC help to fine-tune the update steps more accurately in RMSProp, potentially enabling faster convergence in certain scenarios.

Metric	Without GC	With GC
Accuracy	0.990001	0.99
Mean Time (s)	93.87637178	79.10892237
Standard Deviation	1.628721159	5.591199711





8. CONCLUSION

We tested a new technique called Gradient Centralization and found that it helpful for training neural networks. It was tested on common methods like SGD, Adam, and RMSProp on tough tasks like recognizing images and objects using a set of images known for being difficult.

Gradient Centralization make the training steadier by adjusting the gradients. Which means learning process become faster and more reliable, creating better and more consistent model with better performance. Interestingly, this new technique also made the models slightly more accurate in some cases, which shows that it could help the models to apply what they've learned to the new, unseen data.

When we use this new trick to teach our model, it can take a bit more time and effort, especially with technique like Adam and SGDM. But often, this extra time pays off. because the model gets better at its job. So, even though it's a bit more work, it's usually worth it. This method is especially good when we need our model to be good at a some of different tasks and to be able to handle new challenges well.

Looking ahead, it would be good to try out Gradient Centralization with different types of very new neural network designs and problems to see where it works best. Combining it with other techniques and applying it to situations where they use a lot of data might also show more about how it works. More research could help understand exactly how Gradient Centralization helps during the training process.

In short, result show that Gradient Centralization is a promising addition to the technique for training neural networks, making them learn faster, be more stable, and do better on new challenges. It opens many opportunities for more exploration, application, and use of this technique in the future.

9. REFERENCE

1. https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123460613.pdf
2. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43442.pdf
3. https://proceedings.neurips.cc/paper/2016/file/ed265bc903a5a097f61d3ec064d96d2e-Paper.pdf
4. https://arxiv.org/pdf/1609.04747
5. https://en.wikipedia.org/wiki/Stochastic_gradient_descent
6. https://optimization.cbe.cornell.edu/index.php?title=Momentum
7. https://optimization.cbe.cornell.edu/index.php?title=AdaGrad
8. https://optimization.cbe.cornell.edu/index.php?title=RMSProp
9. https://en.wikipedia.org/wiki/Adam
10. https://www.cs.toronto.edu/~kriz/cifar.html
11. https://www.kaggle.com/datasets/fedesoriano/cifar100
12. https://keras.io/api/
13. https://scikit-learn.org/stable/modules/neural_networks_supervised.html#