An Experiment on KGA Unlearning for Convolutional Neural Networks

Mathieu van luijken
Technische Universiteit Eindhoven
Eindhoven, Netherlands
M.f.a.c.v.Luijken@student.tue.nl

Francesco Ranx Peeters
Technische Universiteit Eindhoven
Eindhoven, Netherlands
F.R.Peeters@student.tue.nl

ABSTRACT

CNN models are widely used in image recognition. One of their main purposes in this domain consists of recognizing natural data subjects. In many legislations natural data objects have the right to remove themselves from the data bases on which models are trained. Machine Unlearning aims to remove the person not only from the data set but also the influence of the subject on the parameters in the model. Previous work suggests a novel unlearning framework called Knowledge Gap Alignment as a method of unlearning neural networks. KGA computes gradients for unlearning based on the knowledge gap between two different models on the same data set. Previous work examines the performance of this technique on large scale NLP bench marking data sets, models and tasks. Our approach introduces the KGA unlearning framework to the domain of CNN image recognition. Experiments on often used image recognition data sets using proven model architectures show that while KGA yields a provable result in unlearning it also results in catastrophic forgetting and reduces model performance in almost all of our test cases.

ACM Reference Format:
Mathieu van luijken and Francesco Ranx Peeters. 2023. An Experiment on KGA Unlearning for Convolutional Neural Networks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Image recognition algorithms have long been used as effective tools for computer vision [1]. Being able to analyze and recognize imaging data has been proven useful for a number of different use cases. In this domain Deep Learning (DL) has shown to be most effective and in specific Convolutional Neural Networks (CNN) and their various DL architectures, such as VGGNet and Resnet [2, 3], have become the state of the art when it comes to Image recognition. One of the main strengths of CNNs is that they make use something called Translational Invariance meaning the subject of the data in the image can be anywhere in the image and the CNN can possibly recognize it [4]. This allows these types of models to train on very few samples and yield good, reliable results which is called few-shot learning. CNNs being few-shot learners makes them great algorithms for tasks such as facial recognition and image segmentation and as such are often used for these tasks[5, 6].

The use of natural persons as a data subject however brings with it a great many considerations, privacy being maybe the most important and prevalent. The GDPR, which is a set of guidelines and laws on data protection, states that any natural person must be removed from data collection and storage if they have retracted their consent for such practices. While removing data subjects from databases itself is a simple task, often times these databases are also used to train DL networks. DL networks are black-box models often trained on large data sets and as such it can be complex and challenging to remove the influence of a small set of data subjects from the model. Since training these large models can be expensive we can not simply retrain the entire model on the original data set and train the entire algorithm again.

Machine Unlearning (MU) is a method which aims to facilitate these models to forget data from data subjects which are to be removed from the initial training data [7]. Methods of MU aim to update weights and parameters of the model in such a way that the original influence of the data subjects that are to be removed is no longer active in the model while maintaining performance for all other data. There has already been existing research on machine unlearning for CNNs [8]. However a newly advanced method of unlearning which proposes a general framework for unlearning on various DL architectures called Knowledge Gap Alignment (KGA) has not yet been performed on CNNs[9]. The KGA method which was originally inspired by [10] makes use of the so called knowledge gap between models, which is defined as the difference of output probabilities between two structurally identical models. The framework holds no strong assumptions on model output, is capable of unlearning small sets of to be forgotten data and can efficiently handle a large number of remove requests.

Our approach takes the KGA framework, which has so far only been used for unlearning in a NLP setting using Transformer architectures, and hopes to expand on the existing research by applying this framework to an image recognition setting using CNN models. One additional question of interest we aim to answer is how a fine-tuning method called layer freezing can be used to enhance the performance and efficiency of the KGA framework. Layer freezing is a fine tuning method for DL networks where the parameters of certain layers in the network are not updated when training. In our case this means that the parameters of layers in the original network are not updated when we are unlearning the network. This method of fine tuning has already been proven to be more efficient and to be able to enhance the performance of CNNs [11].

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Since it is known that the first layers of a CNN model abstract more general features from the data and the objective of unlearning is to remove specific data subjects from the model it makes sense to unlearn only the final layers of the model[12]. Layer freezing may also prove to be more efficient, as unlearning a model can also be seen as retraining it having less parameters to retrain may prove to be less computationally expensive than retraining the entire model. Our results show that at the very least for the models and data sets we used in this paper the KGA unlearning method does not apply well to CNNs. We do see a difference in model output probabilities for the models of interest indicating a success in unlearning. However in all of our test cases the model performance of the unlearned model is worse by a factor of at least 2. The results do indicate however ever so slightly that the freeze fine tuning method can be efficient as we do still observe a good unlearning result whilst model performance is increasing the more we freeze the model. Important to note here is that there were time constraints making it so that the original models were not trained to their fullest capabilities which may have impacted our results.

2 RELATED WORK

We discuss earlier research in the field of unlearning and a multitude of different approaches on different research topics within the field of unlearning. We also introduce two different categories of unlearning and cutting edge techniques used for unlearning.

Unlearning has a multitude of different approaches that can be used in various settings. Some efforts, focussing on additions to the architecture to achieve either faster retraining or effective unlearning in few-shot models [13]. Efforts for developing more general approaches that can be used on an unrelated already trained model have been also been formed inducing a framework for unlearning[8, 9]. These frameworks have the added benefit of being applicable to all models that require adherence to the GDPR and other legislatures.

For our approach the framework is used as is described in [9]. This framework at its centre compares structurally identical models that are trained on different data and computes a knowledge gap. This so-called knowledge gap between the models serves as a heuristic to unlearned performance between models. The authors both denote its efficiency in removing larger unlearning sets, whilst retaining accuracy as well as its ability to not rely on larger unlearning data sets. In convolutional neural networks specifically, there is often the distinction made regarding the unlearning of a set class or a set of samples which is also an interesting avenue of research that has been done [14]. Here the unlearning technique can not focus on the unique aspects of every single forget example but must remove more general features that exist within a class without decreasing performance for other classes.

The process of unlearning is also one of computational cost versus achieved result. one paper proposed a neural tangent kernel (NTK) based method to approximate the training process introducing a scrubber that that removed information from network weights, imposing the condition of SGD based optimization during training [8]. The constraints and conditions within these approximate methods where however plagued by the exponential amount of data that was necessary to compute the Hessian function, and shortcut approaches were a necessity. However other methods were later used that circumvented the need for computing this at all. To increase the generalizability of these

These approximate methods can be described as methods aimed to make the model behave very similar to the exact model, or rather the model where the influence of the forget set is completely removed from the model. As mentioned earlier approximate methods need to compute the hessian of the training data end the gradient of the removal data which is time consuming [15, 16]. The counterpart of these approximate methods are called exact methods which instead of approximately removing the influence of the data from the model can ensure that the effects are removed completely. Earlier research explored exact unlearning for simpler algorithms such as Naive Bayes classifiers and K-means clustering but showcase that for the method of exact unlearning the computational requirements are even higher than the earlier mentioned approximate methods[17, 18]. Because of these computational requirements these exact models did not scale to neural network unlearning. A recent paper however proposed a general method SISA where the different models are trained on non-overlapping data shards and then used in different aggregating methods to obtain an unlearned model [19]. It has been shown however that these methods do not scale well if the set of forget examples becomes larger.

Other sophisticated methods of unlearning involve a teacher network have been developed since, for which examples are One shot unlearning and bad teaching[15]. These methods strive to reduce the computational cost, restrictions during training, and improve generalizability. The efforts have exploiting competent and incompetent teachers into the network to induce forgetfulness. This is done by selectively transferring knowledge from both of these teachers to a student to obtain a model that does not contain any have any influence of the forget set.

3 PRELIMINARIES

We define the unlearning task on different known bench marking data sets suitable for Convolutional Neural Networks and review the unlearning framework as well as the Resnet18 and VGG11 architectures.

3.1 Notation

We denote the sample space for the CNN models given a data set as Z containing all samples in the data set. Given this there can be Z^2 combinations of training and test sets. We denote the data sets as D and the CNN models we train as A where A is a function mapping our input data D to a prediction space H, $A : D \rightarrow H$. The unlearning objective takes as input the following: $D \subset Z$ which is the full training set, $Df \subset D$ the data to be forgotten and $D(A)$ the original model trained on all training examples.

These inputs are then further divided as necessary to perform the unlearning. In addition to Df and D we create Dr (the retain set) which can be denoted as $Dr = D/Df$ and Dn, which is a set of unseen data in our case taken from the test set. We denote Dt as $Dt = Dtest/Dn$. These data sets are then used to train three additional models $A(Df)$, $A(Dr)$ and $A(Dn)$ each trained on their respective data sets. Important to note here is that each of the models uses the exact same archetype and number of parameters but can output completely differing output probability distributions as they are trained on different data sets.

3.2 KGA Framework

The KGA framework is a framework first devised in [9]. The framework inspired on the earlier work [10] aims to be a general unlearning framework for a variety of neural network archetypes and different data types. The underlying idea of the framework is to make use of different supportive pre-trained neural networks to assist in the unlearning of the original model.

The framework has two main goals as described in the paper:
Goal 1: Make model $A*$ output behavior on the forget set Df comparative to its output behaviour on the unseen data in the test set Dn.
Goal 2: Maintain performance of $A*$ on Dr
We propose a slight alteration of goal 2 and the addition of a third goal as follows:
Goal 2: Maintain performance of $A*$ on Dt in comparison to the performance of Ad on Dt.
Goal 3: Make model $A*$ output behavior on Dt comparable to that of Ar on Dt.
We agree on the first goal as one of the main objectives of unlearning is to have the model act as if Df is part of Dt. However we do not fully agree with the second goal. While we do strongly believe the performance is of importance, we also feel that testing on a set of unseen data not including the forget set better reflects the performance of a retrained model rather than a set of data it has already been trained on.
The addition of the third goal is an extension of the first goal. The output behavior of $A*$ on Df is a strong criteria however the model Ar reflects the model as if there never were any of the instance of Df in D and can thus be regarded as the target model. Comparing the output behavior of $A*$ and Ar on unseen data Dt can be considered an additional criteria for the unlearning of model $A(D)$.

The knowledge gap between two models, as used in this paper, is the distance between two different models on the same data set. Both models have the same architecture but have differing training data sets.
As we do not alter the first goal of the paper we assume the same approach as described in the paper. We achieve this goal by targeting comparable output distributions of $A*$ on Df as $A*$ on Dn. As Dn can contain labels not yet seen we cannot directly infer from this data set and thus resort to using the knowledge gap principle to achieve this:

$$A^* = \arg\min A|dis_{(D_n)}(A_D, A_D) - dis_{(D_f)}(A, A_f)| \quad (1)$$

Here the distance can be any measure of difference between the two output distributions of the two models described. However in our paper we will use the Kullback-Leibler divergence as the measure of difference [20] between two different probability distributions. The main reasoning for this is that it is always computable and that it can later be used in to compute the Jensen Shannon divergence.
The second and third goal can be obtained as well in much the same way that is done in the paper however as we added and altered the goals in the original paper we also will slightly alter the approach that we use. The papers' training makes use of model An for which it uses both Dn and Dr. We will disregard the data of Dr as we think that training on unseen data only will strengthen our approach to the second goal since we use An for updating the model which should now be more robust for the Dt. We also believe that by not including any data of Dr we can better make conclusions on the probability distribution distance between the outputs of $A*$ and Ar on Dt by not using Dr as much during the unlearning of $A*$.

3.3 KGA implementation

As we do not alter the objectives themselves we can maintain the implementation that was used and as such device the knowledge gap alignment as such:

$$L_a = \sum_{(y,z)\subset(Df,Dn)} |KL[(Pr_{(A*)}(y)||Pr_{(Af)}(y)] \\ -KL[Pr_{(Ad)}(z)||Pr_{(An)(z)}]| \quad (2)$$

Here we use the aforementioned KL divergence to denote the difference between the probability distribution of two of our models on the same data set. Here (y,z) are random pairs of instances from Df and Dn to alleviate overfitting. Note that since we use only the unseen set here for z the second part of the equation is also equipped for optimizing goals 2 and 3.
This objective however good it might be for forgetting the data that we want to forget can also lead to catastrophic forgetting [21]. To counteract these effects we optimize $A*$ on Dr as:

$$L_r = \sum_{x\subset Dr} KL[Pr_{(A*)}(x)||Pr_{Ad}(x)] \quad (3)$$

which brings two advantages where one is not mentioned in the paper. The first being that training on Dr optimizes towards the initial data distribution meaning we keep performance in comparison to Ad. The second advantage not mentioned in the paper is that we optimize towards model Ar which as earlier mentioned can be seen as the target model. Both of these objectives are jointly optimized and we create the final objective:

$$L = L_a + \alpha * L_r \quad (4)$$

The model can be summarized using the pseudo code as mentioned in Alg 1

3.4 layer freezing:

Each of the models that can be used in the framework contains layers which can be trained or chosen to be not trained. When fine-tuning the models, layers that are not trained are denoted as being frozen. In the case of CNNs however it is known that specifically the convolutional part of the model operates as blocks of layers rather than individual layers. These blocks of layers consist of sequential layers used in the CNN architecture and are predefined in the literature, an example if this can be seen in Fig 2 where the green layers

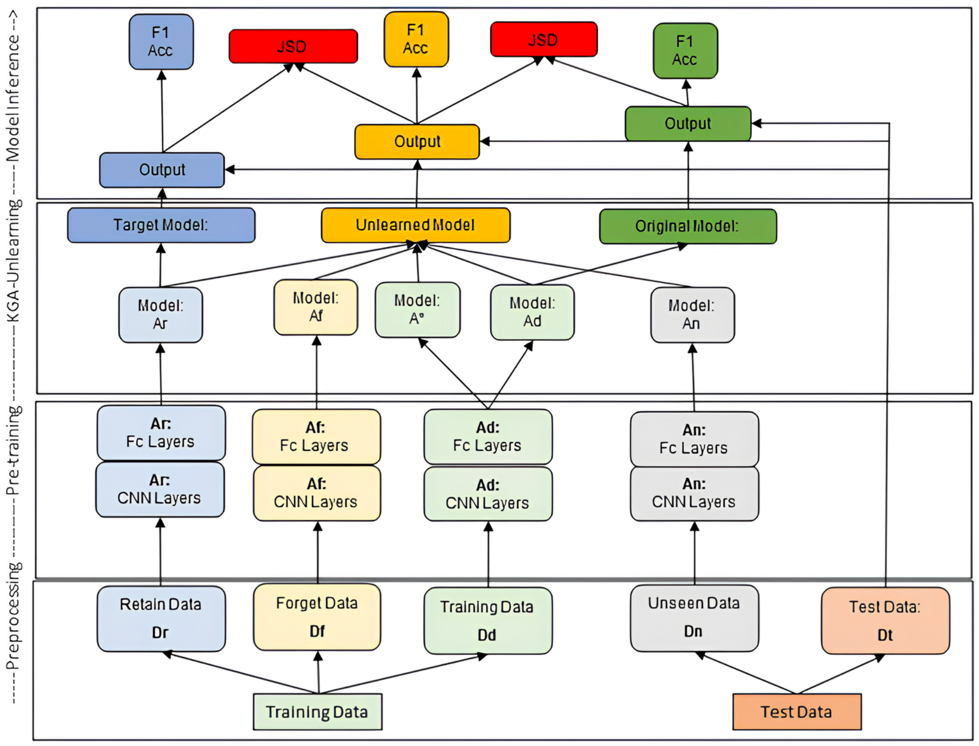

Figure 1: The unlearning framework

Algorithm 1 Knowledge Gap Alignment Unlearning

```
Require: data $D$, $Df$, $Dn$, trained model $Ad$
Ensure: unlearned model $A*$
  Train model $Af$ based on $Df$, model $An$ based on $Dn$
  Initialize $A*$ with $Ad$
  for step in 1 to MAX_STEPS do
      curr_step = 0
      for batch in $Dr$ do
          Compute $L_r$ as in Eq (3)
          Scale $L_r * (1 - LOSS\_RATIO)/INNER\_STEPS$
          if curr_step mod INNER_STEPS = 0 then
              Randomly sample (x, y) from $(Df, Dn)$
              Compute $L_a$ as in Eq (2)
              Optimize $A*$ using $L$ as in Eq (4)
          end if
          $curr\_step+ = 1$
      end for
  end for
```

represent frozen layers. Since either model has a differing amount of layers there is a different amount of combinations possible for each of the models. The Resnet18 model consists of 4 different convolutional blocks and as such has 16 distinct possible fine-tuning combinations, whereas the VGG11 model has 8 convolutional blocks and as such has 256 possible fine-tuning combinations.

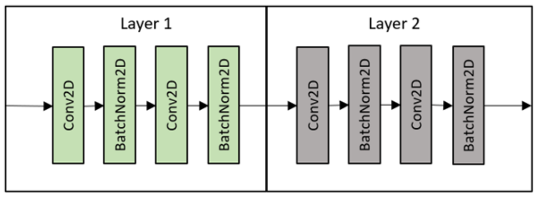

Figure 2: Example of layer freezing

Denoting the frozen layers will be the specific frozen layers in superscript as such: $Resnet18^{(1,2,3)}$. This notation denotes a model for which the first, second and third layers are frozen. For the sake of clarity a model which is completely unfrozen will be denoted with the number 0 as frozen layers E.G $Resnet18^{(0)}$. Unfortunately due to time constraints it is not possible to compute every model fine-tuning combination and it is opted to examine four combinations for each of the models. Literature shows that specifically for CNN models freezing the earlier layers and unfreezing the later layers is most effective both for compute efficiency [11] and model performance [12]. Therefore we opt to freeze the first layers of the model in ascending order as can be seen in table 1.

Fine-tuning using layer freezing aligns with the object of unlearning. In order to have stable results even when unlearning one objective can be to maintain as much of the graph of the network as possible whilst unlearning specific subjects.

model	combination
Resnet18:	$h^{(0)}$
Resnet18:	$h^{(1)}$
Resnet18:	$h^{(1,2)}$
Resnet18:	$h^{(1,2,3)}$

Table 1: Fine-tuning Combinations

4 EVALUATION

The setup of our experiment is described and the obtained results are displayed and then discussed.

4.1 Exerperimental Setup

We evaluate the performance of the models and fine-tuning on two commonly used image data sets Cifar10 and Cifar100. We pre train the models in the same manner as we trained the original model on each of the individual data sets as can be seen in Fig 1. The data sets themselves differ in the complexity of the data itself, both sets contain the exact same number of training images and text images and each image contains the same number of pixels. The only difference between the two sets is the number of classes, where the Cifar-10 set contains ten classes the Cifar-100 set contains 100 classes all categorized in 20 superclasses each containing 5 subclasses. This difference in complexity between both data sets makes it possible to see how the unlearning method works on data sets with differing complexity. This is interesting as often times unlearning is performed on few-shot or one-shot models.

The performance of the unlearning technique is evaluated on three different models, namely Ad, $A*$ and Ar using predefined Dt. We use three different measures:
Model accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

Model F1 score:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN} \quad (6)$$

Jenssen-Shannon Divergence:

$$JSD(P(x), Q(x)) = \frac{KL[P(x)||M(x)] + KL[Q(x)||M(x)]}{2} \quad (7)$$

Where

$$M(x) = \frac{P(x) + Q(x)}{2}$$

We split the data into training set of 50000 examples and a test set of 10000 examples. These sets are further split into Df and Dr which are both halve of the training set. For the splitting of the test set we use 10% of the data for Dn and the other 90% for Dt. All of the pre-trained models are trained for 20 epochs using the same hyper parameter settings to obtain egality between the results. The main unlearning was done in 15 epochs with a number of inner steps between each model update of 15 and a batch size of 256. Note that we are calling the inner loop, which updates the model, after a constant number of batch updates. This means that the more we split the data set into smaller batches the more batch updates we have and thus the more often the model is updated. However counteracted by the fact that we scale L_r by the number of inner steps, we must still scale the number of inner steps based on our batch size if we change this. Using this experimental setup we hope to measure the difference in performance between the original model, our unlearned model and the theoretical target model using the accuracy score and F1 score. Then to test our goals of comparable output probabilities between the models we use the Jensen-Shannon divergence between models $(Ad, A*)$ and $(A*, Ar)$.

4.2 Results

We report the results of the experiments for the Cifar-10 data set for both Dt as well as Df in Tables 2 and 3. The results for the Cifar-100 set again for both Dt and Df are displayed in Tables 4 and 5. We report the results of the JSD algorithm as well as the F1 and accuracy score for all three of the models of interest.We we aim to see the lowest JSD score for both of the model combinations, whereas for F1 score and accuracy we aim for the highest possible scores. The tables compare the score for each of the measures for each of the fine-tuning combinations. We report the results on two different seeds.

Model	JSD		F1 Score			Accuracy		
	(Ad,A*)	(A*,Ar)	Ad	$A*$	Ar	Ad	$A*$	Ar
Resnet18$^{(0)}$	0.030 ±0.004	0.008 ±0.000	0.61 ±0.01	0.04 ±0.02	0.53 ±0.01	0.62 ±0.01	0.10 ±0.00	0.54 ±0.01
Resnet18$^{(1)}$	0.028 ±0.002	0.008 ±0.01	0.61 ±0.01	0.16 ±0.14	0.53 ±0.01	0.62 ±0.01	0.22 ±0.12	0.53 ±0.01
Resnet18$^{(1,2)}$	0.029 ±0.002	0.007 ±0.00	0.61 ±0.01	0.12 ±0.07	0.53 ±0.01	0.62 ±0.01	0.17 ±0.05	0.54 ±0.01
Resnet18$^{(1,2,3)}$	0.021 ±0.006	0.006 ±0.04	0.61 ±0.01	0.30 ±0.11	0.53 ±0.01	0.62 ±0.01	0.32 ±0.22	0.54 ±0.01
VGG$^{(0)}$	0.031 ±0.04	0.004 ±0.001	0.72 ±0.01	0.02 ±0.00	0.44 ±0.01	0.72 ±0.02	0.10 ±0.00	0.47 ±0.01
VGG$^{(1)}$	0.032 ±0.04	0.004 ±0.001	0.72 ±0.01	0.02 ±0.00	0.44 ±0.01	0.72 ±0.02	0.10 ±0.00	0.47 ±0.01
VGG$^{(1,2)}$	0.032 ±0.04	0.005 ±0.02	0.72 ±0.02	0.02 ±0.00	0.44 ±0.01	0.72 ±0.01	0.10 ±0.00	0.47 ±0.01
VGG$^{(1,2,3)}$	0.032 ±0.05	0.005 ±0.02	0.72 ±0.02	0.02 ±0.00	0.44 ±0.01	0.72 ±0.01	0.10 ±0.00	0.47 ±0.01

Table 2: Results for the Cifar10 Data set

Tables 2 and 3 compare the results for the different models over the unseen test set data for the two different seeds, here the main results are the initial seed and the resulting difference between itself and the second seed. For the ResNet18 model we observe a consistenly lowering JSD if we freeze the model for both measures of JSD. We also notice a general increase in performance as we freeze more of the model. This holds generally for both the forget data and the test data. We also notice that the forget data generally performs very close to the forget data in terms of performance on the unlearned model and the retain model. As for the VGG11Net model we observe a more resilient model where the JSD does not increase or decrease when we unfreeze more layers this also holds for the F1 score and the accuracy score. Again we observe that the forget data and the unseen test data perform similarly for the

Model	JSD		F1 Score			Accuracy		
	(Ad,A*)	(A*,Ar)	*Ad*	*A∗*	*Ar*	*Ad*	*A∗*	*Ar*
Resnet18$^{(0)}$	0.064 ±0.010	0.019 ±0.011	0.99 ±0.00	0.03 ±0.01	0.51 ±0.01	0.99 ±0.00	0.10 ±0.00	0.53 ±0.00
Resnet18$^{(1)}$	0.058 ±0.009	0.017 ±0.005	0.99 ±0.00	0.15 ±0.13	0.51 ±0.01	0.99 ±0.00	0.22 ±0.12	0.53 ±0.00
Resnet18$^{(1,2)}$	0.054 ±0.04	0.013 ±0.01	0.99 ±0.00	0.12 ±0.06	0.51 ±0.01	0.99 ±0.00	0.17 ±0.05	0.53 ±0.00
Resnet18$^{(1,2,3)}$	0.048 ±0.015	0.015 ±0.003	0.99 ±0.00	0.29 ±0.10	0.51 ±0.01	0.99 ±0.00	0.31 ±0.21	0.53 ±0.01
VGG$^{(0)}$	0.075 ±0.002	0.011 ±0.003	0.95 ±0.01	0.02 ±0.00	0.42 ±0.02	0.95 ±0.02	0.10 ±0.00	0.45 ±0.00
VGG$^{(1)}$	0.075 ±0.001	0.011 ±0.003	0.95 ±0.01	0.07 ±0.05	0.42 ±0.02	0.95 ±0.02	0.10 ±0.00	0.45 ±0.00
VGG$^{(1,2)}$	0.075 ±0.02	0.010 ±0.002	0.95 ±0.02	0.02 ±0.00	0.42 ±0.02	0.95 ±0.02	0.10 ±0.00	0.45 ±0.00
VGG$^{(1,2,3)}$	0.075 ±0.02	0.010 ±0.001	0.95 ±0.01	0.02 ±0.00	0.42 ±0.01	0.95 ±0.02	0.10 ±0.00	0.45 ±0.00

Table 3: Results for the Cifar10 Forget Data set

Model	JSD		F1 Score			Accuracy		
	(Ad,A*)	(A*,Ar)	*Ad*	*A∗*	*Ar*	*Ad*	*A∗*	*Ar*
Resnet18$^{(0)}$	0.049 ±0.008	0.023 ±0.09	0.99 ±0.01	0.00 ±0.00	0.15 ±0.00	0.99 ±0.00	0.01 ±0.00	0.21 ±0.00
Resnet18$^{(1)}$	0.039 ±0.010	0.017 ±0.014	0.99 ±0.01	0.00 ±0.00	0.15 ±0.00	0.99 ±0.00	0.02 ±0.01	0.21 ±0.00
Resnet18$^{(1,2)}$	0.030 ±0.10	0.010 ±0.006	0.99 ±0.01	0.02 ±0.02	0.15 ±0.00	0.99 ±0.00	0.05 ±0.05	0.21 ±0.00
Resnet18$^{(1,2,3)}$	0.026 ±0.030	0.010 ±0.018	0.99 ±0.01	0.06 ±0.06	0.15 ±0.00	0.99 ±0.00	0.10 ±0.09	0.21 ±0.00
VGG$^{(0)}$	0.026 ±0.004	0.001 ±0.001	0.30 ±0.04	0.00 ±0.030	0.02 ±0.01	0.40 ±0.04	0.00 ±0.01	0.03 ±0.01
VGG$^{(1)}$	0.026 ±0.09	0.001 ±0.002	0.30 ±0.04	0.00 ±0.00	0.02 ±0.01	0.40 ±0.04	0.00 ±0.01	0.03 ±0.01
VGG$^{(1,2)}$	0.026 ±0.012	0.001 ±0.009	0.30 ±0.04	0.00 ±0.00	0.02 ±0.01	0.40 ±0.04	0.00 ±0.01	0.03 ±0.01
VGG$^{(1,2,3)}$	0.026 ±0.012	0.001 ±0.010	0.30 ±0.04	0.00 ±0.00	0.02 ±0.02	0.40 ±0.04	0.00 ±0.01	0.03 ±0.01

Table 5: Results for the Cifar100 Forget Data set

Model	JSD		F1 Score			Accuracy		
	(Ad,A*)	(A*,Ar)	*Ad*	*A∗*	*Ar*	*Ad*	*A∗*	*Ar*
Resnet18$^{(0)}$	0.049 ±0.001	0.008 ±0.00	0.24 ±0.01	0.00 ±0.00	0.16 ±0.00	0.31 ±0.00	0.01 ±0.00	0.22 ±0.00
Resnet18$^{(1)}$	0.024 ±0.001	0.008 ±0.000	0.24 ±0.01	0.00 ±0.00	0.16 ±0.00	0.31 ±0.00	0.02 ±0.00	0.22 ±0.00
Resnet18$^{(1,2)}$	0.025 ±0.002	0.006 ±0.001	0.24 ±0.00	0.02 ±0.00	0.16 ±0.00	0.31 ±0.00	0.05 ±0.04	0.22 ±0.00
Resnet18$^{(1,2,3)}$	0.017 ±0.006	0.004 ±0.002	0.24 ±0.00	0.06 ±0.06	0.16 ±0.00	0.31 ±0.00	0.1 ±0.1	0.22 ±0.00
VGG$^{(0)}$	0.013 ±0.006	0.001 ±0.000	0.27 ±0.00	0.00 ±0.00	0.17 ±0.08	0.36 ±0.00	0.01 ±0.00	0.04 ±0.02
VGG$^{(1)}$	0.013 ±0.010	0.001 ±0.003	0.27 ±0.00	0.00 ±0.01	0.17 ±0.08	0.36 ±0.00	0.01 ±0.00	0.04 ±0.02
VGG$^{(1,2)}$	0.013 ±0.010	0.001 ±0.003	0.27 ±0.00	0.00 ±0.00	0.17 ±0.08	0.36 ±0.00	0.01 ±0.00	0.04 ±0.02
VGG$^{(1,2,3)}$	0.013 ±0.010	0.001 ±0.05	0.27 ±0.00	0.01 ±0.00	0.17 ±0.07	0.36 ±0.00	0.01 ±0.00	0.04 ±0.02

Table 4: Results for the Cifar100 Data set

Model	Cifar 10	Cifar 100
Resnet18$^{(0)}$	27 min 0 sec	27 min 19 sec
Resnet18$^{(1)}$	26 min 58 sec	26 min 20 sec
Resnet18$^{(1,2)}$	26 min 57 sec	26 min 39 sec
Resnet18$^{(1,2,3)}$	29 min 43 sec	26 min 17 sec
VGG11$^{(0)}$	3 min 25 sec	3 min 20 sec
VGG11$^{(1)}$	3 min 25 sec	3 min 25 sec
VGG11$^{(1,2)}$	3 min 22 sec	3 min 21 sec
VGG11$^{(1,2,3)}$	3 min 22 sec	3 min 22 sec

Table 6: Training time in seconds

unlearned model and the retain data model. In all cases we observe a stark decrease in performance between the original model and the unlearned model as well as a marginal decrease in performance for the retain model. Tables 2 and 3 again compare the results for the different models over the unseen test set data for the two different seeds, here the main results are the initial seed and the resulting difference between itself and the second seed. The results indicate a similar result as for the Cifar10 Dataset. The JSD for the Resnet18 model decreases as more layers get frozen in almost all cases. The VGG11 model now shows no differences in JSD for any model configuration and is even more stable than its Cifar10 results. The general performance for the unlearned Resnet18 model starkly decreases but increases a slightly as we freeze more layers which is in contrast to the VGG11 model which has the unlearned model performing at a nearly consistent 0% accuracy. The results between the unseen test set and the forget data between models does not differ much if any at all as well .

We were also interested in how the freezing of layers impacted the efficiency of the unlearning and report the time it took for each model to unlearn in Table 6. Important to note here is that the results for the Resnet18 model were run on the NVIDIA Quadro P1000 GPU and the VGG11 model was run on the NVIDIA RTX 4090 GPU, since we were interested in results for the fine tuning for each model and not between model architectures this does not influence the conclusions drawn in any way. The table shows an almost identical training time between both Cifar10 and Cifar100 for all model configurations. The time difference between each fine-tuning configuration is non-varying as well.

4.3 Discussion

The results show that unlearning using the KGA framework is successful in the sense that the model performs the same on unseen data as on the forget set. However in spite of this the performance of the unlearned model strongly decreases in comparison to the original model. This result is consistent for all fine-tuning combinations and we find no difference in compute time for these as well.

The JSD for the Resnet18 models for both data sets is in line with the expectation of successful unlearning, the JSD for the original model and our unlearned model is higher than that of the unlearned model and the target retain data model in almost all cases. We also see a decrease in JSD resulting from our freezing, this is logical as we are keeping more of the model parameters the same as the original model. The performance scores for the Resnet18 models on the unseen test set and the forget set show us a stark decrease in performance in general, this is likely due to the unlearning having catastrophic forgetting as a result. The increasing performance when the model becomes more frozen then also is in line with expectations, the more we keep the model the same as the higher performing original model the more we expect the performance to be on par. Interestingly the performance of the original model and the performance of the retain model on the forget set is radically different. We observe a near perfect score for the original model and a much lower score on the retain model. Since both models were identically trained the difference must lie in a differing quality of the data in the forget set and the retain set. As for the differences in results for the unlearned model and the retain model on both the unseen test set and the forget set there are almost none, both data sets obtain almost equal results for both of the models indicating that the unlearning of the forget data was successful.

Then for the VGG11 model we observe a different result regarding the JSD, namely the JSD does not decrease nor increase between different fine-tuning combinations. This is most likely due to the nature of VGG11, which has many more layers than Resnet and freezing the first 3 layers might not be sufficient to obtain different results. We do however see that the JSD between the original model and the unlearn model is much higher than that of the unlearn model and the retain model which is again in line with expectations much in the same way as for the Resnet model. These results again also stay consistent for the differing data sets and the different seeds that were tested on. As for the performance metrics we see that the original model performs better than its Resnet counterpart on the unseen test data but under performs when used for the forget data. This may indicate that the forget set contains data subjects that are more fit for the Resnet architecture. We also again see a decrease in performance between the original model and the retain model strenghtening our hypothesis that the forget examples are more easily classified. Again we see a rather large difference between the original model and the retain model and a decrease to almost 0% accuracy for the unlearned model, this decrease is much more significant when compared tot the more frozen Resnet results indicating that the complexity of the VGG11 model undermines the effectivity of the layer freezing method for few layers. We do however again see that the scores for the unseen test set and the forget set on the unlearned model are nigh indistinguishable indicating again that the forgetting of the samples in the forget set was sufficient in spite of the loss in performance.

5 CONCLUSION

We investigate the effectiveness of the KGA unlearning framework on CNNs and the respective impact of layer freeze fine tuning methods on the efficiency and performance of this framework. Our experiments on two often used image recognition data sets and several layer freeze scenarios with the Resnet18 and VGG11 model architectures show a strong decrease in model performance for all of the mentioned experimental scenarios. In spite of this the unlearning itself can be considered successful as we show a decrease in overlap on output probability distributions between our unlearned model and the original model and an increase regarding the unlearned model and the target model as well as near identical results for the unlearned model when used on unseen test data and the forget set. In spite of the decrease in genearal performance, the layer freezing part of our research could be considered succesfull as it increases the performance of the Resnet model without negatively impacting the forgetting and has no influence on the VGG11 set.
An explanation of these results lie in the Translational Invariance that comes with CNNs. Where we do see that this property makes the model efficient in its parameters it might make it more susceptible for Catastrophic Forgetting [21] when the model is unlearned by retraining. Though to be conclusive on any of our hypothesis more research is necessary.

In future work, we need to address the clear limitations of our study such as investigating the effects of hyper parameter tuning the unlearning and testing if the results are consistent across multiple random training/testing splits. More research is required also to investigate the impact of which layers to fine-tune and how this choice interacts with aforementioned hyper parameters especially on deeper networks such as the VGG11 model. Also testing the approach on the main intended purpose of few-shot CNNs trained on large scale image recognition data for natural data subjects would be an interesting approach. Testing for different data sets and models and interpreting these conclusions might strengthen the conclusions that can be made as well. Finally it would be interesting to systematically study how the freezing and unfreezing of layers interacts with the information captured in each of the layers of our model.

REFERENCES

[1] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[4] V. Biscione and J. Bowers, "Learning translation invariance in cnns," *arXiv preprint arXiv:2011.11757*, 2020.
[5] G. S. Dhillon, P. Chaudhari, A. Ravichandran, and S. Soatto, "A baseline for few-shot image classification," *arXiv preprint arXiv:1909.02729*, 2019.
[6] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," *IEEE Transactions on Neural Networks*, vol. 21, no. 7, pp. 1048–1059, 2010.

[7] E. Romero, I. Barrio, and L. Belanche, "Incremental and decremental learning for linear support vector machines," in *International Conference on Artificial Neural Networks*, pp. 209–218, Springer, 2007.

[8] A. Golatkar, A. Achille, and S. Soatto, "Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pp. 383–398, Springer, 2020.

[9] L. Wang, T. Chen, W. Yuan, X. Zeng, K.-F. Wong, and H. Yin, "Kga: A general machine unlearning framework based on knowledge gap alignment," *arXiv preprint arXiv:2305.06535*, 2023.

[10] M. E. E. Khan and S. Swaroop, "Knowledge-adaptation priors," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19757–19770, 2021.

[11] X. Xiao, T. B. Mudiyanselage, C. Ji, J. Hu, and Y. Pan, "Fast deep learning training through intelligently freezing layers," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1225–1232, IEEE, 2019.

[12] K. Goutam, S. Balasubramanian, D. Gera, and R. R. Sarma, "Layerout: Freezing layers in deep neural networks," *SN Computer Science*, vol. 1, no. 5, p. 295, 2020.

[13] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli, "Zero-shot machine unlearning," *IEEE Transactions on Information Forensics and Security*, 2023.

[14] A. K. Tarun, V. S. Chundawat, M. Mandal, and M. Kankanhalli, "Fast yet effective machine unlearning," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[15] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli, "Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 7210–7217, 2023.

[16] R. Mehta, S. Pal, V. Singh, and S. N. Ravi, "Deep unlearning via randomized conditionally independent hessians," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10422–10431, 2022.

[17] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE symposium on security and privacy*, pp. 463–480, IEEE, 2015.

[18] A. Ginart, M. Guan, G. Valiant, and J. Y. Zou, "Making ai forget you: Data deletion in machine learning," *Advances in neural information processing systems*, vol. 32, 2019.

[19] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 141–159, IEEE, 2021.

[20] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[21] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

5.1 Details of Experimental Setup

Pre-trainig the models:
Both the Resnet18 and the VGG11 network are loaded in using the original architecture from pytorch with respective outputs for the Cifaf10 and Cifar100 data sets. The train and test sets are predetermined by the pytorch documentation. All of the models are trained with a set seed and a dataloader for which the batch size is set to 128 and shuffle true. The models are optimized using the Stochastic Gradient Descent using a learning rate of 0.001 using the Cross Entropy Loss for a set 20 epochs.
Unlearning the models:
The KGA unlearning made use of the same data loader objects and data as the pre-trained set meaning the forget set and retain set did not change and the models were also unlearned with batch size 128 and shuffle set to true. The unlearning was optimized by the Adam optimizer for which the learning rate was 0.05 and adam epsilon was 1e-8 and the custom loss functions' retain loss ratio was set to 0.1. The models were all unlearned for 10 epochs and the number of inner steps before the outer loop looped was set to 15.