Investigating the Effect of Latent Representations on Continual Learning Performance

By Henry Bourne, Supervised by Rihuan Ke

Abstract

In this report we demonstrate how neural networks trained on tasks sequentially forget how to perform previous tasks they've trained on when training on new tasks. We demonstrate forgetting on the split-MNIST using simple networks and then on split-CIFAR10 with networks composed of a pretrained encoder and a fully connected classifier on the end where the encoders were trained using different techniques. We found that using frozen latent representations or trainable latent representations had no effect on the forgetting rate of the network, however, trainable latent representations tended to be able to achieve a higher accuracy on the task currently being trained. We also found all our networks except the largest, trained on the largest pretraining dataset with a frozen latent representation had immediate complete forgetting of the previous task after one epoch of training on a new task. We also propose an extremely simple continual learning technique based on our findings.

1 Introduction

In this report we investigate the effect of latent representations on continual learning performance. Continual learning [3, 32] describes a challenge were we try and train models on a series of tasks that are presented to the network in a sequential fashion, we are specifically concerned with the subset of continual learning where neural network architectures are the models being trained. We will also be mainly concerned with the problem of continually learning an image classification task. Usually these tasks contain very different data such as distinct classes and this means the assumption that our data is independently identically distributed, which we need to perform minibatch stochastic gradient descent, is broken. This leads to the problem of catastrophic forgetting [37, 28, 6] where a neural network will forget to correctly classify data from a previous task whilst learning on a new task, which means after sequentially learning a series of tasks the network will only be able to correctly classify images (with good performance) that are similar to data presented in the last task.

We aimed through empirical analysis to investigate how using fixed latent representations might affect the degree to which a network forgets and also how we learnt the latent representation and what we learned it from might affect the degree of forgetting. We first demonstrated the problem of forgetting using simple networks, a fully connected network and a small convolutional neural network [23], on a simple continual learning dataset the MNIST handwritten digits dataset [5] partitioned into 5 tasks where each task contains all the data corresponding to two classes (two digits). We found that the networks immediately forgot all knowledge on the previous task as soon as they started training on a new task.

We then carried out experiments using Resnet18, VGG16 and autoencoder networks as encoders with some fully connected layers on the end for classification. These networks are all of varying sizes and had either no pretraining or were trained on CIFAR100 or Imagenet. For each network we also had two variations: one where the encoder was frozen and one where the encoder was trainable. We found that by the end of training on any task the previous task had been completely forgotten, ie. achieves an accuracy of zero or near zero on the previous task. Furthermore, we found that bar the network with frozen Resnet18 as the encoder every network immediately forgot all knowledge on the previous task (ie. achieving a training and testing accuracy of zero or very near zero) after just one epoch of training on the next task. This severe forgetting is somewhat surprising given the lesser extent of forgetting that has been shown in other papers such as in [36, 50], however, our increased number of tasks, smaller networks and not conducting interleaved training could explain our more severe results. Our results more closely resemble the kind of forgetting shown in [28], however, the networks and datasets used to obtain the results are extremely simple.

Our results from the network with frozen Resnet18 as the encoder showed some more gradual forgetting on some tasks. In [36] the authors found that larger networks and larger pre-training datasets led to reduced forgetting which our results compliment as Resnet18 is the largest network that we experimented with, used the largest pretraining dataset we used (Imagenet) and was the only network to show more gradual forgetting. However, this doesn't explain why the non-frozen version of this network didn't also demonstrate slightly more

gradual forgetting. This could suggest further avenues of exploration including conducting experiments on split-CIFAR10 with 5 tasks using larger networks with larger pre-training datasets and like we have done in this report comparing the amount of forgetting that occurs when we use a frozen encoder versus a non-frozen encoder. Our result with the frozen Resnet18 hints at the possibility that frozen representations might somewhat mitigate forgetting, which would run in opposition to what was found in [35] where they found that almost all forgetting happens in the deepest layers but may agree with the work in [38] and [26] where it appears that when training on tasks sequentially the encoder was mainly changed and not the classifier layers. An initial investigation into using larger networks showed even more gradual forgetting when the encoder was frozen (but not when the encoder was not trainable), suggesting there may be benefit to using fixed latent representations to hamper forgetting, further more rigorous investigation is of course required. However, we observed no changing in forgetting in any of the other networks when freezing the encoder which somewhat gives support to the findings in [35], that all or most forgetting happens in the deepest layers.

We also experimented with randomly initializing the fully connected, classifier, portions of the network at the end of each task and demonstrated that our results for the large part resembled the results without resetting the classifier. This is due to the fact that complete forgetting (accuracy on previously trained tasks is zero or near zero) is so immediate and that with a randomly initialized classifier the network is very quickly able to learn the new task. Motivated by these results we also introduced a novel CL technique were we solve the problem of CF by simply focusing on the deepest layers of the network. By simply training a new classifier head on each task we can achieve just as good accuracy on each task by the end of training and have zero forgetting as we don't train the classifier on any of the other tasks. This is a very simple and effective technique to implement, however, it requires us to store a separate classifier for each task and we must have knowledge as to what task we are currently training on.

All the code needed to produce the results in this report can be found on github at https://github.com/h-aze/compass_yr1/tree/master/Mini_Project. In the spirit of reproducibility all the results were obtained using a seeded random number generator (the seed used is the default seed). And all the experiments were run on one NVIDIA GeForce RTX 2080 Ti GPU.

2 Background

In this section we will outline what the problem of catastrophic forgetting is, and how it is related to the area of continual learning. We will also discuss the three main approaches to continual learning, and the different types of encoders and deep learning architectures that are used in this report to demonstrate and evaluate how forgetting occurs in neural networks.

2.1 Deep Learning

Deep learning is a subfield of machine learning that uses a combination of a neural network and backpropagation to learn a model from data [24, 40, 9, 43]. The "multi layer perceptron" or "feed forward neural network" is the classic version of a neural network in which there are layers made up of neurons (or perceptrons), each neuron in a given layer is connected to every neuron in the next layer (unless it's a neuron in the output layer) and to every neuron in the previous layer (unless it's the input layer). A network with more than one hidden layer is called a "deep neural network", hence the name Deep Learning (DL) and the model is inspired from the structure of the brain. A neuron in a hidden layer (a layer that is neither the input nor output layer) or output layer acts as a function, it computes a weighted sum of the outputs of all the neurons in the previous layer and adds a term called the bias, it then applies what is called an activation function to the weighted sum. An activation function is a differentiable non-linear function such as the Rectified Linear Unit (ReLU) function [29] which is the activation function we used in all of our neural architectures in our experiments. The ReLU function can be defined as follows:

$$\sigma_{ReLU}(x) = \max(0, x) \tag{1}$$

where max simply returns the larger of its two arguments. The neurons in the input layer each represent a feature of the input data. A neural network can then be written as a function f that takes in a vector of features x and outputs a vector of predictions y, ie. y = f(x), where if f is a network with one hidden layer then:

$$f(x) := \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1 x + \mathbf{b}_1) + \mathbf{b}_2) \tag{2}$$

where σ is the activation function, $\mathbf{W}_1$ and $\mathbf{W}_2$ are the weight matrices, and $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bias vectors. The weights and biases are learned by backpropagation, which is a method of training a neural network that uses gradient descent to minimise the loss function. The loss function is a function that measures how well the

network is performing, it is usually the mean squared error (if doing regression for example) or cross entropy (if doing classification). The loss function is then differentiated with respect to the weights and biases, and the weights and biases are updated in a direction that minimizes the loss function. This process is repeated until the loss function is minimized. The way backpropagation works is by propagating the error backwards through the network, we calculate the errors like so:

$$\delta_2 = (a_2 - y) * g'(z_2) \tag{3}$$
$$\delta_1 = (W_2)^T * \delta_2 * g'(z_1) \tag{4}$$
$$\tag{5}$$

where a_2 is the output of the network, y is the target output, g is the activation function, z_2 is the weighted sum of the output layer, z_1 is the weighted sum of the hidden layer, and δ_2 and δ_1 are the error terms for the output and hidden layers respectively. The error terms are then used to update the weights and biases as follows:

$$W_2 \leftarrow W_2 - \alpha * \delta_2 * (a_1)^T \tag{6}$$
$$b_2 \leftarrow b_2 - \alpha * \delta_2 \tag{7}$$
$$W_1 \leftarrow W_1 - \alpha * \delta_1 * x^T \tag{8}$$
$$b_1 \leftarrow b_1 - \alpha * \delta_1 \tag{9}$$

where α adjusts the rate at which we adapt our parameters (the learning rate). The key thing to note here is that the error terms which we use to calculate our parameter updates are calculated using the loss our network obtains on the current input[1]. Allegorically, we propagate the error backwards through the network to find which weights and biases are responsible for the error, and then we update those weights and biases accordingly to reduce the error. We will discuss why this is important in the context of catastrophic forgetting in 2.3.

This network resembles some of the first network architectures to appear in the literature, since then more complex neural architectures have arisen that illustrate better performance in many different scenarios. One such network is the Convolutional Neural Network (CNN) [23]. CNN's take inspiration from the visual cortex which has been demonstrated to have a hierarchical architecture [14]. A particular area it has shown impressive performance is in image classification which is the problem area we will focus on in this report. To put it simply it works by looking at a group of pixels, extracting information using a filter (a matrix of weights) and then applying a non-linear activation function to the result. The filter is then moved across the image (convolved), the process is then repeated on the output of this process. By repeating this process on the output of the previous layer we can extract more and more complex features from the image, typically we then feed the output of the convolutional layers to a small fully connected network to classify the image. We train the networks parameters using backpropagation in the exact same way as with a fully connected network.

2.2 Encoders

An encoder is a function we can use to give a useful, usually lower-dimensional, representation of data. In the context of machine learning an encoder entails algorithms such as Principal Component Analysis (PCA) [13] and t-SNE [51] which are techniques used to extract useful features from data. These features can then be used directly for analysis or as input to another model. What we will focus on are techniques that make use of neural architectures to encode data into useful latent representations.

One such technique involves training networks on an image classification task and then extracting and using a portion of the network as an encoder. CNN's for example are often used as encoders, the network is trained on an image classification task, we then take the convolutional part of the network (ie. get rid of the fully connected portion) and make this our encoder. In order to classify the image the network has to extract useful features from the image therefore we can use the portion of the network where it is extracting these features as an encoder. An example of a specific CNN architecture that is widely used due to its good performance is the VGG16 [47] network. The network has 16 layers, 13 convolutional layers and 3 fully connected layers. We used the convolutional portion of this network as one of our encoders in our experiments.

Other techniques involve more thought into how to train a network to produce robust and informative latent representations, two of the most famous techniques that achieve this are Auto Encoders (AEs) [12, 43, 9] and Variational Auto Encoders (VAEs) [18]. AEs are a type of neural network that are trained to reconstruct their input. They are constructed in a way that the dimensionality of the layers initially decreases and then

[1]Now, in reality we don't update the network after every input, we calculate the error and gradients for each input in a batch of inputs and then average the gradients and update the network once (called mini-batch gradient descent). This saves computation whilst also making our updates to the network more stable

increases again where the output is the same dimensionality as the input. The network is optimized to minimize the reconstruction loss which measures the difference between the input to the network and the output of the network, this trains the network to create a good representation of the input that can be modelled by the bottleneck layer[2]. The better the encoding at the bottleneck layer the better it will be at being able to accurately reconstruct the input when decoding.

The VAE differs from the AE in that it models the latent space as a probability distribution, specifically the Gaussian distribution, this allows us to sample from the latent space and generate new data (ie. it is a generative model). This has the added affect of making the latent space more robust and informative. It achieves this by having two heads on the bottleneck layer, one that outputs the mean of the latent space and one that outputs the variance of the latent space. The network is then optimized to minimize the reconstruction loss and the KL divergence between the latent space and a standard Gaussian distribution.

2.3 Catastrophic Forgetting

The problem of catastrophic forgetting first appeared at the end of the 1980s, where it was observed that multilayer perceptron models trained (using backpropagation) on tasks sequentially incurred decreased performance on past tasks that they had been trained on [37, 28, 6], this problem was named Catastrophic Forgetting (CF). Since, it has been observed in a whole variety of network architectures trained using backpropagation outside the classic multilayer perceptron such as in CNN's [1] and in LSTM's [42]. Precisely we can describe CF as follows:

Definition 2.1 ***Catastrophic Forgetting (CF):*** *The phenomenon where a neural network trained on a series of tasks sequentially incurs decreased performance on past tasks that it has been trained on. Catastrophic forgetting in the area of image classification will present itself as a decrease in the accuracy obtained on past tasks. We will use the term "complete forgetting" to refer to when the network achieves zero or close to zero percent accuracy.*

The problem of CF occurs due to the way we train these neural architectures, ie. due to backpropagation. As we mentioned in 2.1 a neural network is trained by propagating loss backwards through the network, finding which weights were responsible for the loss incurred and updating them accordingly so if we were to calculate the loss on the same data again it would be less. If we are working within the assumption that the samples of the data used to construct the mini-batches are all independently and identically distributed (iid.) then training this way causes no problems as on average we will be getting a good estimate of the loss on the data as a whole. However, if we are training on data that is not iid. then we will consistently get an unrepresentative value of the loss on the data as a whole and in turn change the network in such a way that it will not perform equally well on all the data. This is the case if we train on tasks sequentially, in this case when optimizing the network on the current task weights throughout the network will be changed to improve the performance on the current data being trained on, possibly at the detriment of the performance on the previous task, incurring forgetting.

Research in the area of understanding CF is sparse, especially in comparison to the amount of research that aims to mitigate it, and so deep understanding on the problem is somewhat limited.

In [35] the authors conducted a very comprehensive empirical study on where in the architecture forgetting most occurs and also analyzed how task semantics change the degree of forgetting. They investigated where in the network forgetting occurs by carrying out experiments on split-CIFAR10 that included freezing layers, resetting layers and analyzing hidden representation (network weights) similarity using representational similarity measures [34, 20]. Their conclusion was that forgetting occurs in deeper layers of the network and interestingly they were also able to show that CL techniques such as EWC [19] and experience replay [39] seemed to reduce forgetting by stabilizing deeper layers. They also showed that the degree of forgetting was dependent on the task semantics and through combination of empirical investigation and analytical model concluded that forgetting is most prevalent when the tasks have intermediate semantic similarity and that forgetting is reduced when the tasks are semantically either very similar or very different.

The findings above are confirmed by [49], where using their novel forgetting measure (DyRT) they were able to show that the bulk of forgetting occurs in the deepest layers (the classification layers) as opposed to the feature extraction portion of the network. This and the above is in contrast, however, to the work in [38] and [26] where it appears that when training on tasks sequentially the encoder was mainly changed and not the classifier layers.

[2]The bottleneck layer is the layer that is the smallest in dimensionality, it is the layer that the network uses to represent the input.

Additionally, in [49], they investigated if catastrophic forgetting occurs when performing continual reconstruction, ie. reconstruction tasks[3] in the continual learning setting. Empirical testing on 3D shape reconstruction datasets found that no CF was incurred when performing continual reconstruction versus in the batch scenario and in fact found better performance in some cases. They also confirmed this result in the 2D continual image reconstruction scenario on CIFAR100 and found similar results. These results are quite surprising as to our knowledge they are the only example of where CF in neural architectures has been shown to not occur without any sort of CL technique in place.

In [36] the authors investigate the effect of using pretrained models on catastrophic forgetting. In particular they investigate how model size and size of the pretraining dataset affects the amount of forgetting. They find that pretrained models forget less than models trained from scratch and that the bigger the model and pretraining dataset the bigger this effect is.

In [10] the authors investigate the effect of the activation function used on CF. There also exists literature which introduce methods aimed at visualizing CF, such as [8, 30].

2.4 Continual Learning

Continual learning (CL) [32] is a field in DL that focuses on developing techniques to allow networks to get good performance on problems that involve learning sequentially on tasks. This is a very different scenario to what DL models are normally trained to work in where you must have access to all the data you want the model to learn from at training time, known as the batch scenario. We now provide a formal definition of CL:

Definition 2.2 ***Continual Learning (CL):*** *A model that has the "ability to continually learn over time by accommodating new knowledge while retaining previously learned experiences"[32] is described as a continually learning model.*

One of the main problems encountered when learning sequentially is CF, however, there are also problems such as catastrophic remembering [45]. Continual learning is often described as the problem of solving the stability-plasticity dilema which means balancing the ability of the network to learn from the current task (its ability to be plastic) with its ability to not change too much as to not incur decreased performance on the previous task (its ability to be stable). The term task here is quite loose and can be defined in many ways, for example, it could be a different dataset, a different distribution on the same dataset, a different class in the same dataset, etc. For our purposes we will be looking at the case where each task contains a selection of classes from the same image classification dataset, ie. we partition the dataset into groups according to classes and label each of these groups as a different task, which is known as the problem of task incremental learning and is seen as one of the easier CL set-ups. Harder CL scenarios include class incremental learning where we learn sequentially each class and CL scenarios with no task or class boundaries (ie. we don't have knowledge as to what task or class we are training on). Something to note is we won't restrict ourselves to the data-stream task incremental learning set-up where we are only allowed to see each sample once, which is a harder subset of the task and class incremental scenarios.

The area of CL saw some initial work in the very early days when the problem of CF was identified [37, 6] and has since seen a lot of work in the last few years [3, 32]. Most of the work in this area comes under three main approaches to solving the issue: (1) Regularization techniques [19, 52] which aim to reduce forgetting by adding penalties for changing parameters in the network (2) Replay techniques [46, 39] which aim to reduce forgetting by either replaying old data to the network or learning to generate synthetic data and replay that to the network (3) Architectural approaches [27, 41] which aim to solve the problem by dynamically growing, pruning and freezing the network to mitigate forgetting. As a special mention, within the area of replay methods there exist methods that look at replaying latent representations of activations in the network to itself. This is known as latent replay, these methods in particular are somewhat relevant to our work as they make extensive use of foundational models. In [31] the authors carry out an extensive set of experiments on latent replay methods, including experiments pitting frozen encoders against models trained end to end. They found that using frozen encoders greatly reduced compute and that in some cases all they needed was a non-parametric model in conjunction with the encoder to mitigate forgetting by a substantial amount.

3 Related Work

The problem of CF crops up in many currently active research areas in DL including online learning a close cousin of CL, transfer learning, Multitask learning and problems where the data distribution changes over time.

[3]Reconstruction tasks involve optimizing for encoding data and then from the encoding reconstructing the original input, ie. what the AE and VAE do that we mentioned in 2.2.

The encoders we will be working with in this paper come from the field of representation/ foundational learning. We will now take a moment to briefly elaborate on these closely related areas.

3.1 Representation / Foundational Learning

The area of representation learning (also referred to as foundational learning) focuses on using machine learning algorithms to learn good representations and for computing representations. Work in this field includes "unsupervised feature learning and deep learning" and includes methods such as "probabilistic models, autoencoders, manifold learning, and deep networks" [2]. The AE and VAE we mentioned in 2.2 come from this area of research. Many representation learning methods become pervasive in DL systems for their ability to simplify a problem by generating informative features from data. This is also the case in CL where an encoder is often used as it "greatly simplifies the downstream task of classification" [44] and has empirically been shown to increase performance such as when using latent replay methods [31].

3.2 Other areas where CF appears

As well as in CL, CF has been shown to appear in many other areas of DL including multi-task and transfer learning [22] and learning where there is data distribution shift [50, 33]. CF occurs in these settings due to the violation of the iid. data assumption that we mentioned in 2.3. There is also the very closely related area of online learning [16] which aims to make networks capable of learning and adapting to better perform on current incoming data (which doesn't necessarily mean not forgetting previous tasks), which is similar to learning with data distribution shift, it's similar to CL as it also works in the scenario where data is presented to the network in a continual fashion. A subset of the online learning area that also gets much attention is data stream learning [7] where data flows into the network continuously and the network only sees each piece of data once. An important note is that certain CL scenarios qualify as also being an online learning scenario, this is often the case in robotics research where the agent has to continually learn from a data stream [25].

4 Methods

In this project we aim to understand in more depth the problem of CF. As mentioned in section 2.3 the literature on where and why CF presents and in what scenarios it is most pronounced is lacking, whereas the literature in the area of CL which aims to solve the problem of CF is much more abundant. This dichotomy is strange as it would make sense to have a better core understanding of CF, this would give CL researchers a better understanding as to what problem they are actually trying to solve. In this project we intended to find out the answers to some simple questions regarding CF:

- What's the effect of using a pre-trained encoder on catastrophic forgetting?
- What's the effect of using a pre-trained encoder on the ability to learn new tasks?
- How does the effectiveness of the above (or lack thereof) change with different encoders?
- How does the effectiveness of the above (or lack thereof) change if we freeze the encoder versus if we train the network end-to-end?

Our work is very close in spirit to [35], where the authors similarly investigated the degree of CF that occurs in a network where part of it is frozen. It differs in the fact that we use a CL scenario where there are 5 tasks as opposed to 2, we pretrain the encoders and we specifically investigate freezing the encoder portion of the network. In their experiments they found that forgetting occurred in deeper layers and that freezing shallow layers (where the encoder portion of the network is located) didn't result in decreased forgetting.

Somewhat similarly to us in [31] the authors investigated the effect of using frozen encoders on reducing CF, they found that it reduced compute and that it made the task of CL simpler and showed in some cases that using non-parametric models in conjunction with a frozen encoder resulted in minimized forgetting. However, in their paper they used latent replay techniques when training their models, whereas we will not be employing any CL techniques whatsoever in our experiments.

Finally, in [38] and [26] a subset of the experiments seem to suggest that when training on tasks sequentially the encoder was mainly changed and not the classifier layers. We will now introduce some datasets, metrics and networks we used in our experiments.

4.1 Datasets

In our experiments we used the MNIST [5], CIFAR10, CIFAR100 [21] and the Imagenet [4] datasets. Where CIFAR100 and Imagenet were mainly used for pretraining. We also used the split-MNIST [52] and split-CIFAR10 [52] benchmarks when working in the CL scenario, which is where we split the datasets into a number of tasks where each task contains all the data pertaining to some given classes in the dataset. In our experiments we split both datasets into 5 tasks where each class contained 2 classes from the dataset. These two benchmarks are commonly used in the area of CL.

4.2 Metrics

In our experiments we measured the accuracy and loss obtained by the network on training data and testing data. We measured the average loss and accuracy obtained over all the batches each epoch and we measured the loss and accuracy obtained on the whole test set at the end of each epoch. Note that the training set wasn't used at all in the training of the model.

When in the CL scenario we also tracked some additional metrics, mainly the loss and accuracy achieved on each individual task, the Task Accuracy (TA) and Task Loss (TL), during training (we calculate this in the same way as in the batch scenario). This is useful for observing how performance of the network changes over the course of training on each task and allows us to observe learning and forgetting on each individual task.

Using this we also calculated two additional metrics which are the Continual Task Accuracy (CTA) and Continual Task Loss (CTL). The CTA and CTL track the accuracy and loss achieved on the current task during training. The CTA and CTL are helpful as they give a good picture of how well the network is doing at adapting to the current task.

Another metric we tracked was the time taken for training as a proxy for the amount of compute required to train the network. We ran all the experiments on the same GPU, the NVIDIA GeForce RTX 2080 Ti. Note that for the networks with frozen encoders we encoded the data before training and then trained the network on the encoded data, the time for encoding the data isn't included in the compute times.

4.3 Networks

In our experiments we used a number of different networks, for our experiments with **MNIST** we used:

- **FCFFNN**: A simple Fully-Connected-Feed-Forward Neural Network composed of 2 hidden layers (both with 512 neurons).
- **CNN**: A simple Convolutional Neural Network with 2 convolutional layers (first with 16 filters and second with 32 filters, a kernel size of 3, stride 1 and padding 0) followed by max pooling layers (kernel size of 2) and with one fully connected layer on the end.

For both networks we used the Rectified Linear Unit (ReLU) activation function and the Stochastic Gradient Descent (SGD) optimizer (both in the CL and batch setting), we also used the ReLU activation function in all the network architectures we trained on CIFAR10. In our experiments with **CIFAR10** we used the following networks:

- **CNN**: A CNN based on the VGG16 architecture [47] with no pretraining, it has 3 fully connected layers at the end where both hidden layers have 512 neurons and the output layer has 10.
- **nfVGG-FCFFNN**: The same VGG16 network except with batch normalization [15] and dropout [48] pretrained on CIFAR100.
- **fVGG-FCFFNN**: The same as above except we freeze the convolutional layers after pretraining.
- **nfResnet18-FCFFNN**: The Resnet18 network [11] pretrained on Imagenet with a fully connected network on the end (exactly the same as that used with VGG).
- **fResnet18-FCFFNN**: The same as above except we freeze the Resnet18 layers after pretraining.
- **nfAE-FCFFNN**: Makes use of a convolutional autoencoder pretrained on CIFAR100 (tasked with image reconstruction). The encoder portion of the network is made up of 3 convolutional layers (first with 16 filters, second with 32 filters and third with 64 filters, a kernel size of 3, stride 1 and padding 1) with max pooling layers (kernel size of 2 and stride of 2) with one fully connected layer of size 256. The decoder portion of the network is made up of 3 convolutional layers (first with 64 filters, second with 32 filters and

third with 16 filters, a kernel size of 4, stride 2 and padding 1) with one fully connected layer of size 1024 located before the convolutional layers. We used a hyperbolic tangent activation function at the end of the decoder. After pretraining we only keep the encoder and add on the end a fully connected network (exactly the same as that used with VGG). Note that this is by far the smallest network out of this bunch.

- **fAE-FCFFNN**: The same as above except we freeze the encoder portion of the network after pretraining.

When working on the CIFAR10 and split-CIFAR10 benchmarks we made use of the Adam optimizer [17] and when in the CL scenario reset the optimizer at the end of each task to reset the momentum (during testing we found that this resulted in better performance). We also used the cross entropy loss function for all of our experiments across datasets. Note that we didn't use any form of data augmentation during pretraining or training, we didn't use any regularization techniques (such as weight decay) and any form of CL method. The thinking behind this was to keep the networks in their simplest forms as possible whilst still being capable of achieving good performance to minimize confounders. Also note that we use a wide range of networks above including networks pretrained on different datasets, using different architectures and different loss functions (the autoencoder aims to reduce reconstruction loss during training) this was done with the hope that the experiments would give a good picture of how well different networks architectures with varying richness in pretraining may differ in their level of forgetting.

4.4 Frozen versus Non-Frozen Encoders

In our experiment section we will look at the results of the networks mentioned above trained on split-CIFAR10 (with 5 tasks). By comparing the performance of networks with the encoder portion either frozen or unfrozen we aim to see the effect of using fixed features for CL versus using features that are changed over the course of learning. The results might indicate whether forgetting happens in early "encoding layers" of the network and could give answers to whether or not using frozen features does make the upstream task of CL easier [31], whether it reduces the amount of forgetting due to shallow layers no longer changing [38, 26] or whether it makes no difference at all if all forgetting occurs in the deepest layers [35].

4.5 Varying Type of Encoder

In our experiments we use encoders that were trained on an image classification task (Resnet18 and the VGG16 networks) and an encoder that was trained on an image reconstruction task (the AE). By comparing the performance of these networks we aim to see whether or not the type of encoder used has an effect on the amount of forgetting that occurs. Will the degree of forgetting of a network with an encoder trained on an image classification task be different to that of a network with an encoder trained on an image reconstruction task? Is an encoder trained to produce a low dimensional latent representation of an image better at retaining knowledge than an encoder trained to produce a representation useful for classifying an image? We aim to explore the answers to these questions in our experiments section.

4.6 Pretraining versus No Pretraining and Pretraining with Different Datasets

In our experiments we also compare the performance of networks that were pretrained on different datasets : None vs. CIFAR100 vs. Imagenet. By mixing the richness of the pretraining dataset we can also try and infer the effect of pretraining on the amount of forgetting that occurs. Will a network pretrained on a richer dataset forget less than a network pretrained on a less rich dataset? Based on [36] we might hypothesize that a richer pretraining dataset and a bigger network is less likely to forget. We also hope to infer if pretraining at all is beneficial for CL, if it is then we might expect the network with no pretraining to forget more than the networks with pretraining.

5 Experiment

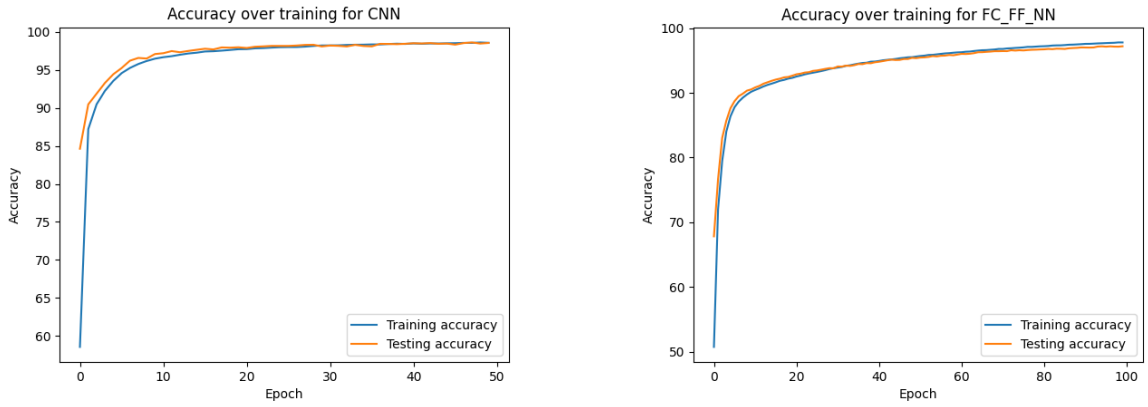
We will start by showing some results demonstrating forgetting in simple networks on a simple dataset, MNIST and split-MNIST, before moving onto our results on the CIFAR10 and split-CIFAR10 datasets using our more complex networks.

5.1 Demonstrating Forgetting

In this section we will demonstrate forgetting in simple networks, the FCFFNN and the CNN, on a simple dataset, MNIST. First we trained the networks in the **batch scenario on MNIST**, to gauge the kind of performance we should expect from these models on split-MNIST, the training and testing accuracy over training is shown in fig. 1. We clearly see an increase in accuracy over training, with the CNN achieving a final testing accuracy of 98.55% and the FCFFNN achieving a final testing accuracy of 97.19%, which is a good level of performance. Observing fig. 2 we also note that the loss is doing what we might expect; which is it decreases very quickly at first followed by a continual gradual decrease. Note that we had to run training on the FCFFNN for 50 more epochs than we trained the CNN to achieve the performance stated. The models that produced these results were selected after trying out a number of different values for the learning rates which ended up at 0.005 for the CNN and 0.007 for the FCFFNN. We also tried out different values for the batch size, but found that the default value of 64 worked best for both models.

We then trained these models in the **CL scenario on split-MNIST** (with 5 tasks) in the task-incremental setup. There was no modification to the networks, we used the same learning rate and batch size as in the batch setting. We sequentially trained the networks on each task and trained the networks on each task for 10 epochs. Figure 3 shows the CTAs obtained by the networks over training, we see that the CTA does decrease significantly when switching tasks (although this decrease is small in absolute amount), but that it increases to a very high accuracy by the end of training on the task. In fact, they achieve even better performance than in the batch scenario, although this is what we might expect due to the much simpler binary classification task that the network has to perform in the CL scenario. Figure 4 shows the loss obtained by the networks over training, this shows large spikes whenever training on new tasks begins which is what we might expect due to the new task containing different data, over the course of training on the task the loss then reduces. This indicates that both networks are successfully learning the new tasks when training on them and by the end of training have learnt the task proficiently.

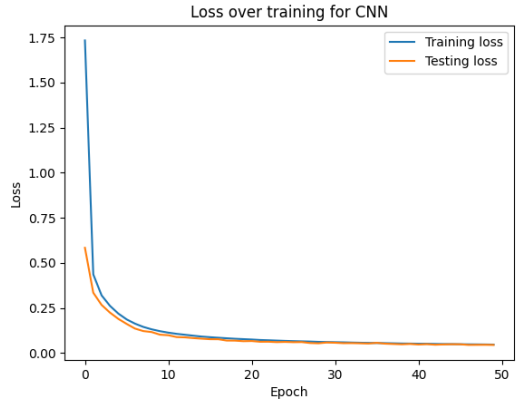
Figure 5 shows the training and testing accuracy each network achieved on each task over training, these plots confirm that the networks are learning the tasks well when training on them with the accuracy jumping very high. However, once the network moves onto learning a new task the accuracy on the previous task plummets very quickly. This decreased performance on the previous task is the network forgetting the previous task, we are therefore witnessing here the problem of CF. Note that the forgetting in these plots is very drastic with complete forgetting occurring after just one epoch of training on the next task with high frequency. Although we note that for some tasks, such as tasks 1 and 2 in experiments with the CNN, that complete forgetting doesn't occur until training begins on the next task. Figure 6 shows the loss each network achieved on each task over training, we see the loss of the current task being trained drop significantly during training, but after or before training on the task its loss is gradually increasing. This collection of plots illustrates very clearly the problem of CF in two different networks on one of the "easiest" CL benchmarks (split-MNIST in the task incremental scenario).



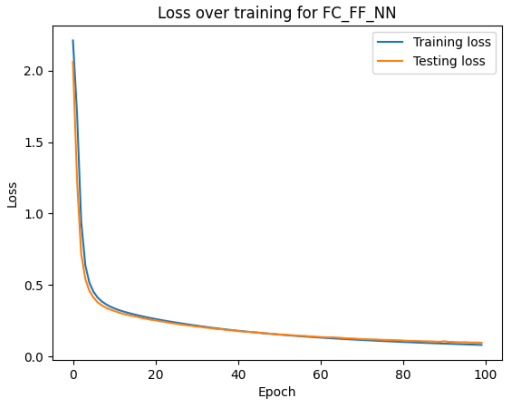
(a) Accuracy over training epochs for the CNN on MNIST in the batch scenario. The final testing accuracy it achieved was 98.55%.

(b) Accuracy over training epochs for the FCFFNN on MNIST in the batch scenario. The final testing accuracy it achieved was 97.19%.

Figure 1: Accuracy over training on MNIST

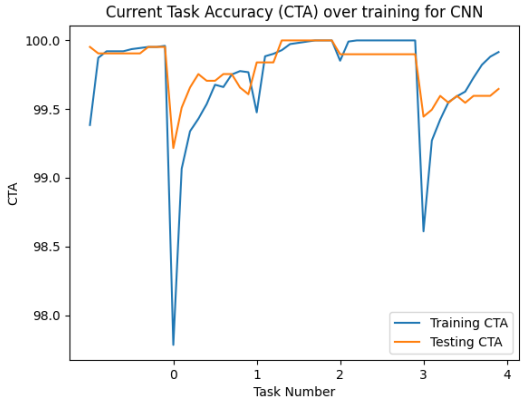


(a) Loss over training epochs for the CNN on MNIST in the batch scenario. The final testing loss achieved was 0.0450 (3 sf.).

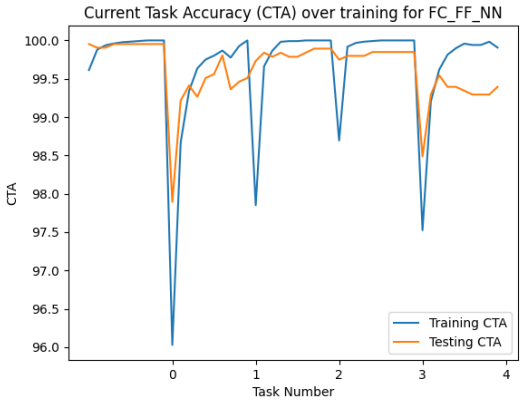


(b) Loss over training for the FCFFNN on MNIST in the batch scenario. The final testing loss achieved was 0.0956 (3 sf.).

Figure 2: Cross entropy loss over training on MNIST

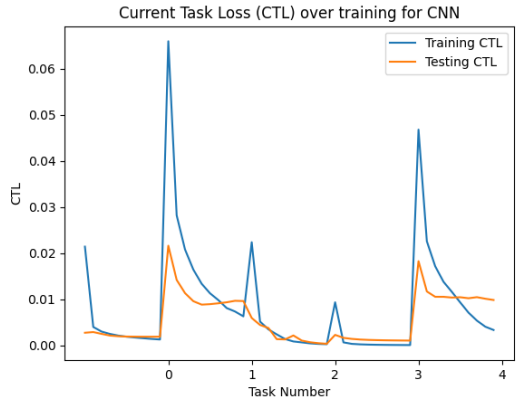


(a) Current task accuracy over training epochs for the CNN on split-MNIST. It achieved the following test accuracies (on the current task) at the end of training on each task: Task 0 - 99.95%, Task 1 - 99.61%, Task 2 - 100%, Task 3 - 99.9%, Task 4 - 99.65%.

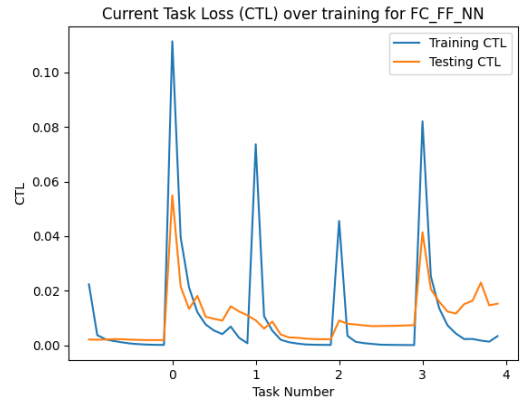


(b) Current task accuracy over training epochs for the FCFFNN on split-MNIST. It achieved the following test accuracies (on the current task) at the end of training on each task: Task 0 - 99.95%, Task 1 - 99.51%, Task 2 - 99.89%, Task 3 - 99.85%, Task 4 - 99.39%.

Figure 3: Accuracy over training on split-MNIST

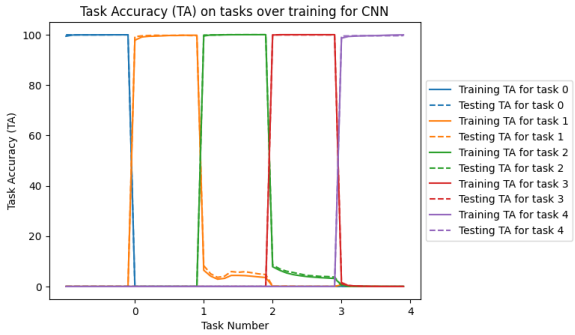


(a) Current task loss over training epochs for the CNN on split-MNIST.

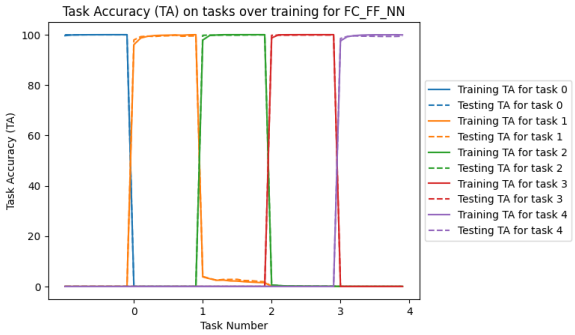


(b) Current task loss over training epochs for the FCFFNN on split-MNIST.

Figure 4: Accuracy over training on split-MNIST

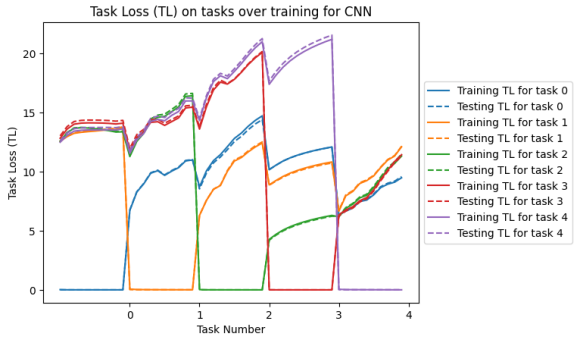


(a) Task accuracy on each task over training epochs for the CNN on split-MNIST. After training has finished on each task and training begins on the next the accuracy immediately falls to practically zero and stays there, except for task 1 and task 2 where the accuracy hovered around 4%.

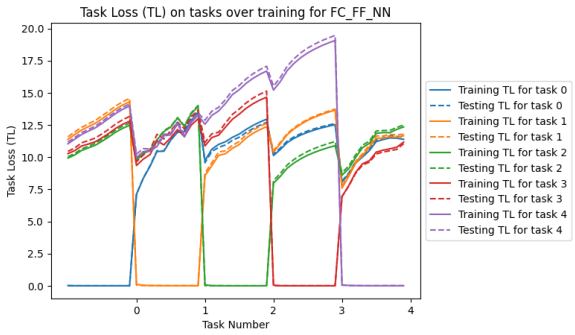


(b) Task accuracy on each task over training epochs for the FCFFNN on split-MNIST. After training has finished on each task and training begins on the next the accuracy immediately falls to practically zero and stays there, except for after training on task 1 where the accuracy hovers around 3%.

Figure 5: Accuracy over training on split-MNIST



(a) Task loss on each task over training epochs for the CNN on split-MNIST.



(b) Task loss on each task over training epochs for the FCFFNN on split-MNIST.

Figure 6: Cross entropy loss over training on split-MNIST

5.2 Investigating the effect of pre-training and freezing on CF

We ran each of the networks described in section 4.3 on split-CIFAR10 and kept tab on their loss and accuracy on testing and training data individually for each task aswell as the time needed for compute. To be able to use time as a proxy for compute we ran the experiments using the same computational resources.

We also ran all the networks on **CIFAR10 in the batch scenario** to assess the relative performance of these networks in a standard training set-up. Table 1 displays the final testing accuracies achieved by each of the networks along with the time it took for training to complete. nfResnet18-FCFFNN achieved the highest accuracy followed by nfVGG-FCFFNN and fResnet18-FCFFNN. It would make sense that the networks using Resnet18, the largest network also pre-trained on the largest dataset (Imagenet), would perform the best although also with the largest compute time. nfVGG-FCFFNN performed very well which isn't surprising given it is a very deep network, however, it is a surprise that its frozen counterpart (fVGG-FCFFNN) performed significantly more poorly, suggesting the encodings produced by the pre-trained encoder perhaps aren't general enough to enable good performance on a new dataset (even a very similar dataset in this case - CIFAR10 vs. CIFAR100) without fine-tuning the encoder. The performance of the networks using the encoders from an AE were poorer, with the frozen variation achieving the worst performance out of all the networks in addition to needing more epochs of training. This is likely due to the fact that the AE is trained to produce encodings that are good for reconstructing the input, not for classification, which could be hindering its performance. Finally, the CNN had the median performance, outperforming networks that were pretrained including: fVGG-FCFFNN, fAE-FCFFNN and nfAE-FCFFNN and needing the smallest amount of compute out of all the networks apart from fVGG-FCFFNN.

We then trained our networks in the **CL scenario on split-CIFAR10**. We trained all the networks for 15 epochs on each task, details on the hyperparameters used are discussed in section C. Figure 7 shows our results for TA's over training, the first thing to notice is that by the end of training on a task complete forgetting of the previous task had occurred. In fact, with every network bar fResnet18-FCFFNN this complete forgetting occurred instantly on all tasks. The networks had more differing performance in terms of CTA were the Resnet18 networks performed the best, interestingly it seems the frozen version performed slightly better. Also, interestingly, the CNN which had zero pretraining performed better in terms of CTA than the AE networks and fVGG-FCFFNN which all were pretrained on CIFAR100. For the most part the rankings of the networks in terms of lowest CTA by the end of training on any task seem congruent with the rankings of testing accuracy achieved in the batch scenario. Table 2 shows the time taken for training for each of these networks on split-CIFAR10, as expected frozen networks took the least amount of time to train. nfResnet18 took the longest amount of time to train by far, followed by the CNN. Another thing to note is the relatively short training time for nfAE-FCFFNN for a non-frozen network, this is likely due to its much smaller size compared to the other networks. Plots of the TL's over training are available in section A.

The main take-away from these experiments is that CF in all networks except nfResnet18-FCFFNN was absolute, which makes it impossible to compare the relative degree of forgetting of the networks and is somewhat a surprising result. A possible reason for the extent to which how quickly these networks forget could be due to the CL setup, forgetting in setups such as split-CIFAR10 with 2 tasks seems to be less severe going off of results such as in [35], we also didn't interleave training on tasks like in [50] where less severe forgetting was also observed. Our very catastrophic forgetting most mirrors results like in [28], although we note the networks and tasks used for these experiments were significantly more simple. Interestingly the network that showed some sort of resilience to forgetting was the largest network we worked with, resnet18, trained on the largest pre-training dataset, Imagenet. This could suggest that if we wanted less forgetting (or at least more gradual forgetting) perhaps we should be using larger networks with richer pretraining which would agree with the findings in [36], it might also explain the smaller amounts of forgetting found in [35] where they used larger networks than we have here. Another thing to notice is that only the frozen version was able to slow forgetting somewhat, giving motivation for further avenues of exploration.

Based on the hypothesis that bigger networks with big pretraining datasets could lead to less forgetting we ran some preliminary experiments where we used part of the Resnet50 network as our encoder and pre-trained it on Imagenet. This network is similar to the Resnet18 network we have been using except it is much deeper. Similarly to fResnet18-FCFFNN and nfResnet18-FCFFNN we created the **fResnet50-FCFFNN** and **nfResnet50-FCFFNN** networks which have the same construction as the networks using Resnet18 as the encoder except we swapped them out for the Resnet50 networks (with the last layer of the network discarded). Figure 8 shows the training and testing TA's of the network over training, again we observe more gradual forgetting for the frozen version of the network (fResnet50-FCFFNN), in fact this more gradual forgetting is even more pronounced than what the fResnet18-FCFFNN network achieved. Interestingly we again see the frozen network achieving more gradual forgetting meanwhile it's non-frozen counterpart is still forgetting completely after an epoch of training on the next task, eluding that using fixed latent representations could potentially be

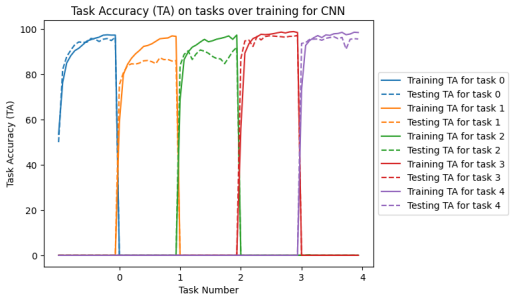
beneficial for mitigating CF. This gives some support to the idea that the quick and complete forgetting we have witnessed in our experiments could partly be down to using smaller models pretrained on smaller datasets. It also gives some support for further investigation into how the strength of CF may be affected by frozen versus non-frozen latent representations in larger networks with larger pretraining.

Model Name	**Final testing accuracy**	**Total compute time (seconds)**
CNN (100 epochs)	78.62%	2261.11
fVGG-FCFFNN (100 epochs)	58.95%	**2001.4**
nfVGG-FCFFNN (100 epochs)	86.19%	2413.16
fAE-FCFFNN (200 epochs)	50.46%	4036.73
nfAE-FCFFNN (200 epochs)	71.65%	2891.6
fResnet18-FCFFNN (100 epochs)	82.56%	3413.17
nfResnet18-FCFFNN (100 epochs)	**88.58%**	7296.88

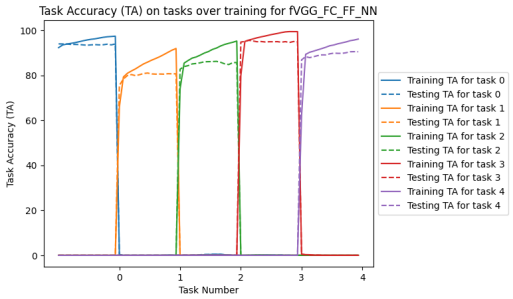
Table 1: Table of final testing accuracies and compute times for our models trained on CIFAR10 in the batch scenario. One NVIDIA GeForce RTX 2080 Ti was used to train each of these models.

Model Name	**Total compute time (seconds)**	**Total Trainable Parameters**
CNN	20218.98	15245130
fVGG-FCFFNN	6181.29	530442
nfVGG-FCFFNN	22893.2	15253578
fAE-FCFFNN	5319.08	**399370**
nfAE-FCFFNN	8315.19	685354
fResnet18-FCFFNN	**5502.08**	530442
nfResnet18-FCFFNN	31487.58	11706954

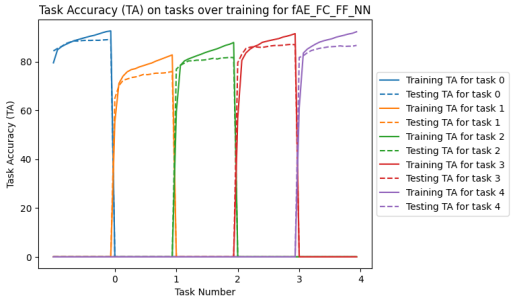
Table 2: Table of compute times and total number of trainable parameters of our models trained on split-CIFAR10, one NVIDIA GeForce RTX 2080 Ti was used to train each of these models.



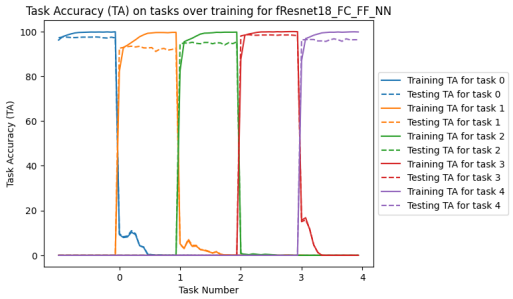
(a) All the TA's for CNN over training, it achieves at least 86% testing CTA by the end of training on each task. Note that learning on new tasks appears more gradual with this network. Complete forgetting happens instantly for all tasks.



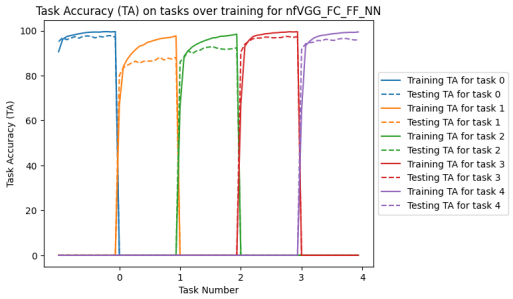
(b) All the TA's for the fVGG-FCFFNN over training, it achieves at least 80% testing CTA by the end of training on each task. Complete forgetting happens instantly for all tasks.



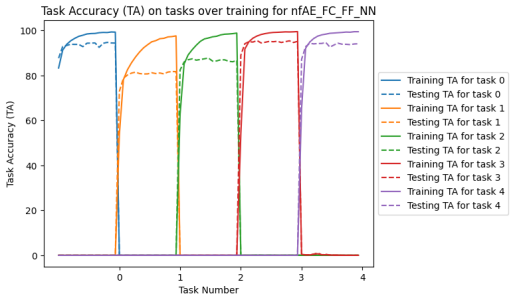
(c) All the TA's for the fAE-FCFFNN over training, it achieves at least 75% testing CTA by the end of training on each task, which is a fair bit less than what the other networks were able to achieve. Complete forgetting happens instantly for all tasks.



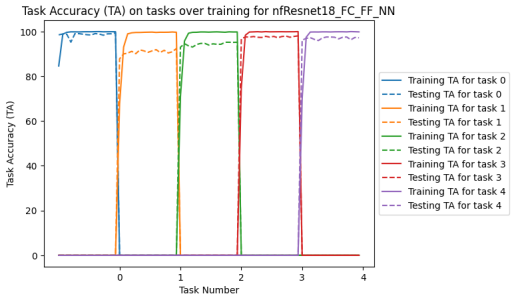
(d) All the TA's for fResnet18-FCFFNN over training, it achieves at least 94% testing CTA by the end of training on each task. Forgetting is again very evident, however, unlike other networks for some tasks it takes some time before the TA for the previous task drops to zero.



(e) All the TA's for nfVGG-FCFFNN over training, it achieves at least 88% testing CTA by the end of training on each task. Complete forgetting happens instantly for all tasks.

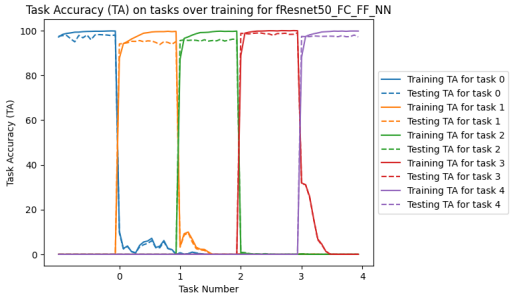


(f) All the TA's for nfAE-FCFFNN over training, it achieves at least 81% testing CTA by the end of training on each task. Again forgetting here is immediate and catastrophic.

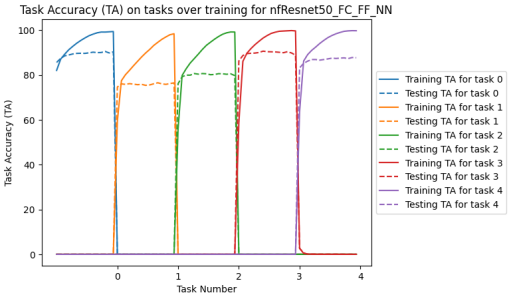


(g) All the TA's for nfResnet18-FCFFNN over training, it achieves at least 92% testing CTA by the end of training on each task, which is second only to it's frozen counterpart. However, unlike it's frozen counterpart, forgetting is immediate and catastrophic.

Figure 7: The testing and training TA's on each individual task over training epochs for all the networks we trained on split-CIFAR10. fResnet18-FCFFNN got the best performance in terms of CTA and was the only network to not forget all previous task knowledge straight after training on a new task.



(a) The TA's for fResnet50-FCFFNN over training. We see for every task (bar task 2) that forgetting is more gradual with it being especially pronounced for task 0 where complete forgetting only seems to occur right at the end of training on task 0.



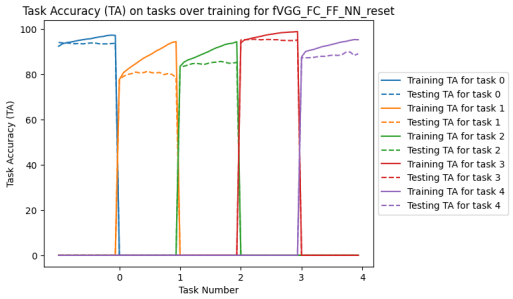
(b) The TA's for nfResnet50-FCFFNN over training. Here weo observe complete forgetting as soon as we begin training on the next task.

Figure 8: The testing and training TA's on each individual task over training epochs for networks with Resnet50 used for the encoder on split-CIFAR10. We observe some more gradual forgetting in the network that has the frozen encoder, however, the non-frozen network displays quick complete forgetting of the previous task.

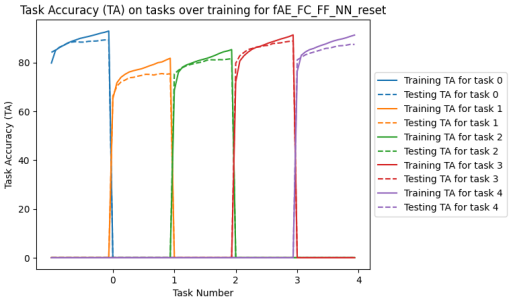
5.3 Investigating the effect of resetting on CF

We also carried out a series of experiments where we reset the fully connected classifier portion of the network, ie. randomly initialize the weights, at the end of each task. We ran the same networks as in section 5.2 on split-CIFAR10 and again kept tab on their loss and accuracy on testing and training data individually for each task aswell as the time needed for compute. We carried out these experiments to evaluate how the performance of training with a random classifier head compares to training with a classifier head that's been trained on previous tasks. Figure 9 shows the TA's over training, we observe extremely similar results to the networks without reset, except for with fResnet18-FCFFNN where the TA's of some tasks don't gradually reduce (which makes sense as we are randomly scrambling the weights).

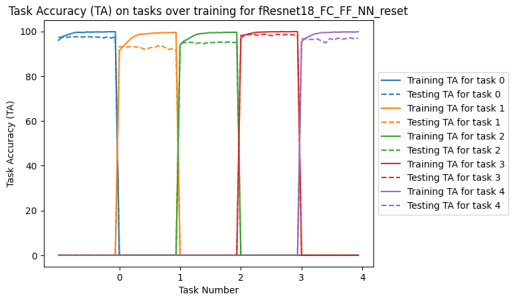
Using these results we suggest a novel CL technique, where we create a classifier head for each task. From our results it appears a method that uses a newly randomly initialized classifier for each task achieves performance very similar to networks where the classifier head has trained on other tasks and with the same amount of forgetting (in most cases) due to the severity of forgetting we have seen in our experiments. If we simply trained and kept a small classifier head for each task then we would be able to achieve the same level of CTA as without reset except we would have no forgetting as the classifier head only ever has trained on one task. This somewhat bypasses the problem of CF and seems like a potentially good naive method for CL. The obvious downsides of such a method being that we need to have knowledge of what task we are currently training on and we must create a new classifier head for each task. It's also not clear whether such a method might hold up in more complex problem settings. Plots showing the TL over training and compute times for training the networks with reset are available in section B.



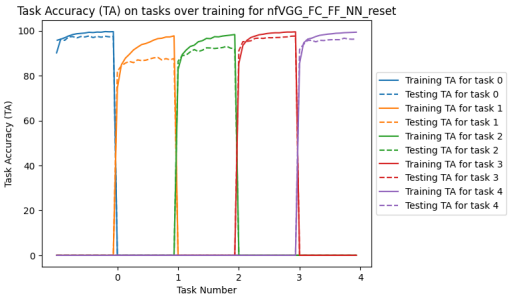
(a) All the TA's for the fVGG-FCFFNN with reset over training, it's results mirror extremely closely the results of the fVGG-FCFFNN without reset.



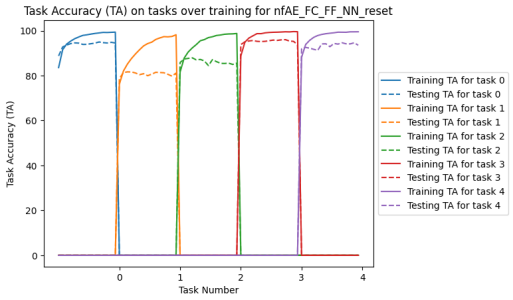
(b) All the TA's for the fAE-FCFFNN with reset over training, it's results mirror extremely closely the results of this network without reset.



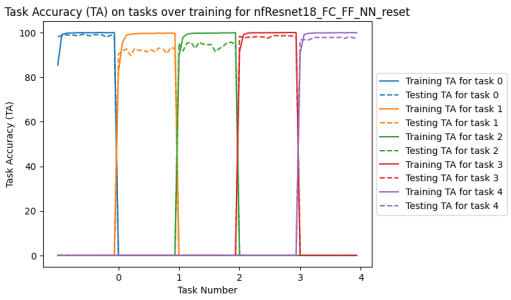
(c) All the TA's for fResnet18-FCFFNN with reset over training, it achieves very similar results to the fResnet18-FCFFNN without reset. Except with reset the more gradual forgetting isn't witnessed (as would be expected randomly initializing the head of the network).



(d) All the TA's for nfVGG-FCFFNN with reset over training, it's results closely resemble the results of this network without reset.



(e) All the TA's for nfAE-FCFFNN with reset over training, it's results also closely resemble the results without reset, except for a slightly more gradual learning curve when it's just started training on a new task.



(f) All the TA's for nfResnet18-FCFFNN with reset over training, it's results also closely resemble the results of this network without reset.

Figure 9: The testing and training TA's on each individual task over training epochs for all the networks we trained on split-CIFAR10. All the results here are extremely similar to the results in fig. 7 without reset, although the results for fResnet18-FCFFNN do differ slightly.

6 Conclusion

In this report we introduced and discussed the problem of CF and how it manifests itself when trying to carry out continual learning with neural networks trained on image classification tasks. We then demonstrated the problem of catastrophic forgetting using simple networks, a fully connected network and a small convolutional neural network [23], on a simple continual learning dataset the MNIST handwritten digits dataset [5] partitioned into 5 tasks where each task contains all the data corresponding to two classes (two digits). Where we found the networks to immediately get zero or near zero accuracy on testing and training data from the previous task as soon as they started training on a new task.

We then carried out experiments on the split-CIFAR10 dataset with 5 tasks where each task contained all the data belonging to two classes. We again found that all networks immediately forgot all knowledge on the previous task after just one epoch of training on the next task, except for the fResnet18-FCFFNN network which exhibited some more gradual forgetting. We found this very severe forgetting somewhat surprising given less severe forgetting has been shown to occur in the literature such as in [36, 50], however, our increased number of tasks, smaller networks and not conducting interleaved training could explain our more severe results. We also remarked that our forgetting more closely resembles the severity of forgetting shown in [28] which used extremely simple and small networks on very small and simple datasets. We hypothesized that the reason fResnet18-FCFFNN was able to show more gradual forgetting could be down to its larger size and pretraining dataset which would agree with the findings in [36], however, couldn't solace as to why the non-frozen version wasn't also able to show more gradual forgetting. On one hand it seems our results match up with what was found in [35] where the authors demonstrated that forgetting occurs almost exclusively in the deepest layers. However, on the other hand the ability of fResnet18-FCFFNN but not nfResnet18-FCFFNN to show more gradual forgetting could suggest that freezing shallow encoder layers might somewhat help to mitigate forgetting which may oppose the findings in [35]. Our very brief initial investigation into using larger networks for the encoder (Resnet50) provide some initial evidence that would support the latter statement, however, further more rigorous investigation is required.

We also ran some experiments where we reset the parameters of the fully connected layers, of the networks which made use of pretrained encoders, to random values at the end of each task. We showed that the results using this resetting technique were extremely similar (bar results with fResnet18-FCFFNN) to the results without resetting. Using these results we proposed a CL technique which could solve the CF problem in a naive way by having a separate classifier for each task. This could achieve similar accuracies to training a network sequentially whilst sidestepping the forgetting problem by only training each classifier on their assigned task. Although we acknowledged the drawbacks of such a technique, mainly that we must have a classifier for each task and we must have knowledge on what task we are currently training on. Further investigation of course is required to determine whether or not this technique is viable in practice on other datasets and using different networks.

In summary, we propose that further investigation is required in order to draw a conclusion as to whether or not using frozen latent representations might reduce the rate of forgetting in neural networks. We propose that experiments are carried out using much larger networks than we have here which have been pretrained on large rich datasets such as Imagenet. In these investigations we also propose that a mixture of encoders taken from image classification models and encoders taken from models trained on image reconstruction are used, to further explore the effect of the type of latent representation that has been trained on the degree of forgetting. In particular, we propose that the VAE [18] is experimented with due to the more robust latent space it's capable of learning and that these networks trained to optimize reconstruction loss are trained on much larger datasets (even larger than Imagenet) which are easier to find due to the fact that they don't require labels. Another avenue of investigation could be empirically testing the amount of forgetting that occurs based on the task number selected for split datasets such as split-MNIST, split-CIFAR10 and split-CIFAR100.

References

[1] Gaurav Arora, Afshin Rahimi, and Timothy Baldwin. Does an lstm forget more than a cnn? an empirical study of catastrophic forgetting in nlp. In *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*, pages 77–86, 2019.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[3] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[6] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[7] Joao Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1:45–55, 2012.

[8] Scott Gigante, Adam S Charles, Smita Krishnaswamy, and Gal Mishne. Visualizing the phate of neural networks. *Advances in neural information processing systems*, 32, 2019.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

[13] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[14] David Hunter Hubel and Torsten Nils Wiesel. Ferrier lecture-functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 198(1130):1–59, 1977.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[16] Lakhmi C Jain, Manjeevan Seera, Chee Peng Lim, and Pagavathigounder Balasubramaniam. A review of online learning in supervised neural networks. *Neural computing and applications*, 25:491–509, 2014.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[20] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[21] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[22] Sneha Reddy Kudugunta, Ankur Bapna, Isaac Caswell, Naveen Arivazhagan, and Orhan Firat. Investigating multilingual nmt representations at scale. *arXiv preprint arXiv:1909.02197*, 2019.

[23] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[25] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020.

[26] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[27] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[28] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[29] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[30] Giang Nguyen, Shuan Chen, Thao Do, Tae Joon Jun, Ho-Jin Choi, and Daeyoung Kim. Dissecting catastrophic forgetting in continual learning by deep visualization. *arXiv preprint arXiv:2001.01578*, 2020.

[31] Oleksiy Ostapenko, Timothee Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Continual learning with foundation models: An empirical study of latent replay. In *Conference on Lifelong Learning Agents*, pages 60–91. PMLR, 2022.

[32] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

[33] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems*, 32, 2019.

[34] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *Advances in neural information processing systems*, 30, 2017.

[35] Vinay V Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

[36] Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*, 2022.

[37] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[38] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[39] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[41] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[42] Monika Schak and Alexander Gepperth. A study on catastrophic forgetting in deep lstm networks. In *Artificial Neural Networks and Machine Learning–ICANN 2019: Deep Learning: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part II 28*, pages 714–728. Springer, 2019.

[43] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[44] Murray Shanahan, Christos Kaplanis, and Jovana Mitrović. Encoders and ensembles for task-free continual learning. *arXiv preprint arXiv:2105.13327*, 2021.

[45] Noel E Sharkey and Amanda JC Sharkey. An analysis of catastrophic interference. *Connection Science*, 1995.

[46] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[49] Anh Thai, Stefan Stojanov, Isaac Rehg, and James M Rehg. Does continual learning= catastrophic forgetting. *arXiv preprint arXiv:2101.07295*, 2021.

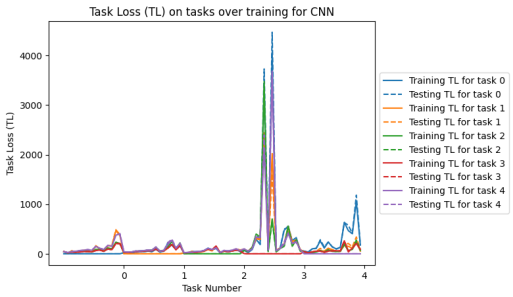
[50] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[51] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[52] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR, 2017.

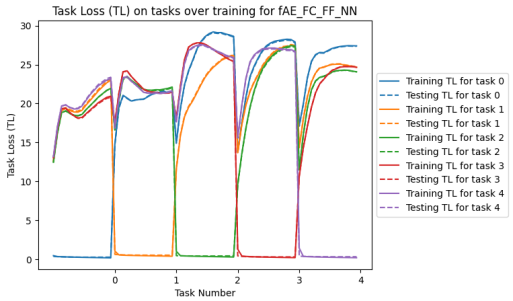
Appendices

A TL plots for experiments in section 5.2

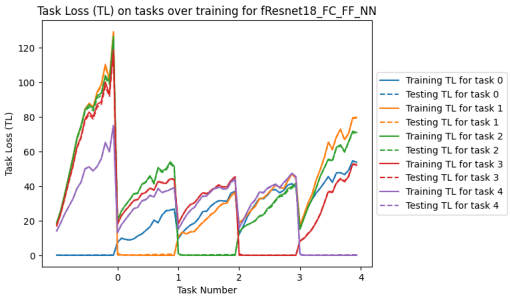


(a) TL's over training for the CNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases, we also see a big spike in the loss during training on task 2.

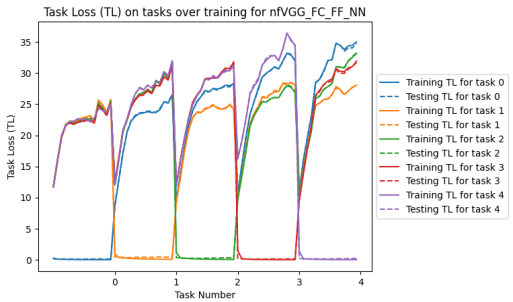
(b) TL's over training for the fVGG-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.



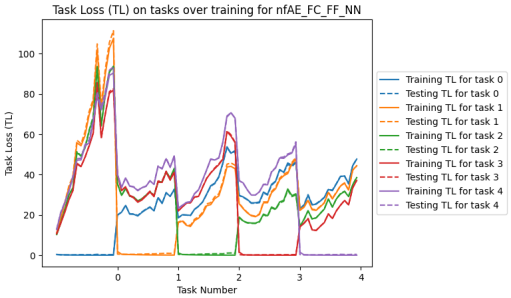
(c) TL's over training for the fAE-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.



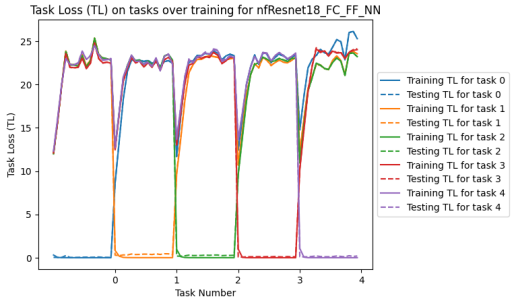
(d) TL's over training for the fResnet18-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.



(e) TL's over training for the nfVGG-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.



(f) TL's over training for the nfAE-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.



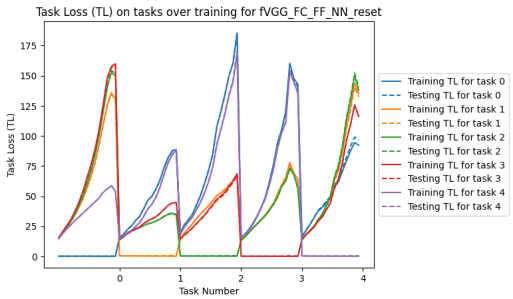
(g) TL's over training for the nfResnet18-FCFFNN on split-CIFAR10. We see that over the course of each task the loss obtained on tasks increases and that the loss of the current task being trained on decreases.

Figure 10: The testing and training TL's on each individual task over training epochs for all the networks we trained on split-CIFAR10.

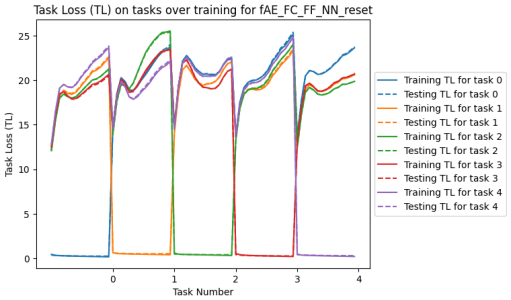
B Compute times and TL plots for experiments in section 5.3

Model Name	Total compute time (seconds)
fVGG-FCFFNN-reset	5559.18
nfVGG-FCFFNN-reset	22605.64
fAE-FCFFNN-reset	6435.23
nfAE-FCFFNN-reset	7740.14
fResnet18-FCFFNN-reset	**5425.43**
nfResnet18-FCFFNN-reset	31206.32

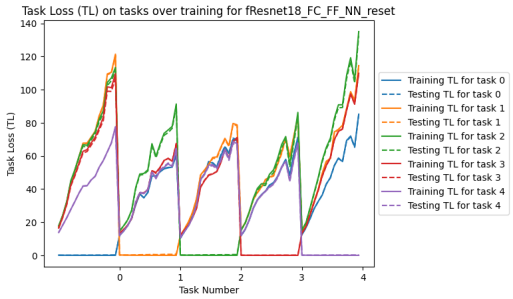
Table 3: Table of compute times and total number of trainable parameters of our models trained on split-CIFAR10 with reset, one NVIDIA GeForce RTX 2080 Ti was used to train each of these models.



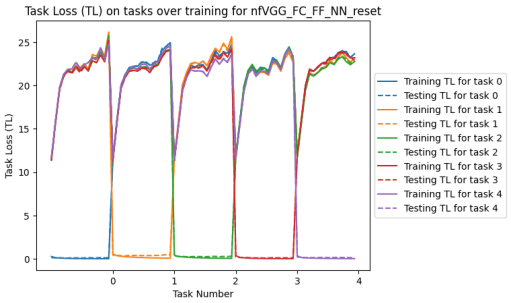
(a) All the TL's for the fVGG-FCFFNN with reset over training, it's results mirror extremely closely the results of the fVGG-FCFFNN without reset.



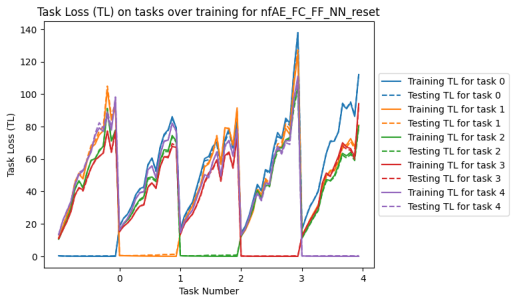
(b) All the TL's for the fAE-FCFFNN with reset over training, it's results mirror extremely closely the results of this network without reset.



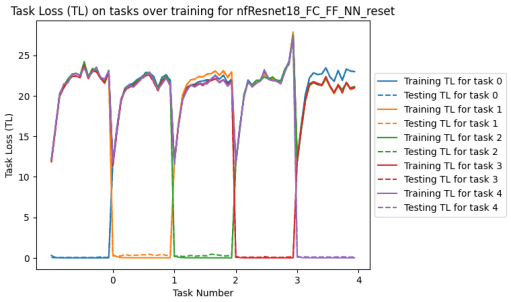
(c) All the TL's for fResnet18-FCFFNN with reset over training, it achieves very similar results to the fResnet18-FCFFNN without reset. Except with reset the more gradual forgetting isn't witnessed (as would be expected randomly initializing the head of the network).



(d) All the TL's for nfVGG-FCFFNN with reset over training, it's results closely resemble the results of this network without reset.



(e) All the TL's for nfAE-FCFFNN with reset over training, it's results also closely resemble the results without reset, except for a slightly more gradual learning curve when it's just started training on a new task.



(f) All the TL's for nfResnet18-FCFFNN with reset over training, it's results also closely resemble the results of this network without reset.

Figure 11: The testing and training TL's on each individual task over training epochs for all the networks we trained on split-CIFAR10.

C Hyperparameters used for all experiments

The networks as described in section 4.3 is how we kept them for all experiments, the only things that changed was in split-CIFAR10 we reset the Adam optimizer at the beginning of each task and we used different learning rates. We will now run through the learning rates we used for each experiment. With MNIST we used the same learning rates in the CL and batch scenarios and trained in both scenarios using SGD:

- **FCFFNN** - 0.007
- **CNN** - 0.005

With CIFAR10 we used the Adam optimizer with the following learning rates:

- **CNN** - 0.0001
- **nfVGG-FCFFNN** - 0.0008
- **fVGG-FCFFNN** - 0.0008
- **nfResnet18-FCFFNN** - 0.002
- **fResnet18-FCFFNN** - 0.002
- **nfAE-FCFFNN** - 0.001
- **fAE-FCFFNN** - 0.0025

With split-CIFAR10 we used the Adam optimizer and reset it at the beginning of each task with the following learning rates:

- **CNN** - 0.0001
- **nfVGG-FCFFNN** - 0.00001
- **fVGG-FCFFNN** - 0.00005
- **nfResnet18-FCFFNN** - 0.00001
- **fResnet18-FCFFNN** - 0.00005
- **nfAE-FCFFNN** - 0.00005
- **fAE-FCFFNN** - 0.00005

We used the same learning rates above in the reset experiments.