BTP-I Report
Few Shot Class Incremental Learning

Parth Shettiwar 170070021
Guide: Professor Biplab banerjee
Co-Guide: Professor Shubhasis Chaudhari

July-December 2020

Contents

1 Abstract

For the development of real world systems, incremental learning plays an important role. The main aim of this project was to perform a classification of objects belonging to different classes in a Few shot Class incremental setup. As the name suggests, this problem has dual complexities which directly affects the accuracies:

- Learning from Few samples without overfitting
- Catastrophic forgetting(Owing to Continual Learning setup)

After doing a comprehensive literature survey, a novel architecture using GAN: *Pseudo Prototype Generator* to produce pseudo-prototypes was proposed. The results have been reported on Imagenette dataset as of now. The results have been promising and can be extended to larger datasets. Before this final setup, we have implemented a nearest mean classifier based network to see its feasibility. This was followed by an ablation with different types of loses and shown its results. Finally we propose a Revised Pseudo Prototype Generator which would, we predict, give a high accuracy on any dataset.

2 Motivation

Almost all strong ideas in Machine learning get inspiration from the behaviour of human brain. Such parallel has intrigued me from time to time and is an active area of research in the current world. In this work, I explored a less researched but highly relatable to humans, a specific problem of Lifelong learning. A human can easily understand and remember the features of a new object even when it sees it only once or twice. At the same time, we humans continuously get exposed to new and new types of objects on a daily basis, still we don't forget the characteristics of older objects which we had observed. However in machine learning domain, this is quite a challenge to train networks in a continual learning setup and efforts have been made in past to mimic the human brain memory.

Utilizing this opportunity, I decided to try and solve this problem as a Final year BTP Project under the guidance of Prof. Biplab Banerjee. A comprehensive literature survey was made to analyse all the previous methods implemented to solve this problem or the related problems. We analysed the various challenges in this setup and came up with an end to end trainable network in an attempt to solve this problem. I would also like to thank Professor Vinay P Namboodiri from IIT Kanpur, for his useful guidance.

3 Introduction

The ability to incrementally learn new classes is crucial to the development of real-world artificial intelligence systems. For practical use, we train CNN models on large scale image datasets and then deploy them on smart agents. The model on agent is usually pre-trained on the training set of the pre-defined classes, and is required to adapt to the new user-defined classes by learning from new samples. From the users' perspective, they are only willing to annotate very few image examples for the new class, as the labeling process consumes manpower. Therefore, it is crucial for CNNs to be capable of incrementally learning new classes from very few training examples. This is few-shot class-incremental learning (FSCIL).

At the same time, the memory footprint or the model size shouldn't increase as the new classes keep coming. A plausible approach would be to simply train the new class samples on the base network,and update its weights. However, as mentioned in the survey [1], the models in continual learning setup are prone to *catastrophic forgetting or catastrophic interference*, i.e. training a model with new information interferes with previously learned knowledge. However, this can be avoided, if we were to retrain the network on whole dataset observed till now. This methodology would be very inefficient and hinders the learning of novel data in real time. Furthermore, as the time increases, training of network would take more and more time due to increasing data observed.
The second major problem in this setup is "*Overfitting on new samples*" where the model is prone to overfit to new classes. This is mainly because we have very less data available for incremental classes.

A quite similar problem is Incremental Few shot Learning(IFSL) as solved by [2] . The main difference between these 2 setups is that in the FSCIL setting, we can have any number of incremental learning sessions, however in IFSL, we only have 1 incremental session over a base class and testing is done on joint test set consisting of base class and the incremental class samples. Effectively making IFSL setting as Few-shot learning problem.

Few shot learning problem and Class Incremental Learning have received considerable attention from the research community individually in the past, but not in cohesion. At the same time, problem of FSCIL has also not been adequately explored. The specified FSCIL setup, has only been performed by a recent work [3] till now. In this project, I performed a comprehensive literature survey to explore all the available methods done in past in this domain as well as the related domains. Further more, we come up with a Generative modelling setup where we use a GAN to generate pseudo prototypes for classes observed till now. Using a prototypical network, mitigates the problem of Overfitting as approached by [4]. At the same time, to solve the problem of Catastrophic forgetting, we apply necessary distillation loses in our network inspired from the work of [5] . We finally report the results on Imagenette dataset (subset of 10 classes of Imagenet), which look promising and can be further extended to larger datasets like CUB200 and CIFAR100. In short, the main contributions from this work are:

- A comprehensive literature survey of the current Few shot and Class incremental setups. We explore the merits and demerits of the past approaches and how we approached the relatively newer FSCIL problem.
- We first implemented a nearest mean classifier based network in Few shot fashion to see the feasibility of Prototypical Networks. We show results on omniglot dataset for the same.
- We devised a novel GAN based pseudo prototype generator which when combined with appropriate loses and performed in a right setting solves the problems of both catastrophic forgetting and Few shot overfitting. We report our results on Imagenette dataset, which is subset of 10 classes of Imagenet.
- We finally explore a revised version of the above pseudo prototype generator to overcome its shortcomings.

4 Related Work

As mentioned before, not much work has been done in the domain of FSCIL. Hence in this section, we explore the related work done in domains of Few shot learning and Class incremental learning separately.

4.1 Few Shot Learning Approaches

Few-shot learning (FSL) aims to adapt the model to recognize unseen novel classes using very few training samples. Usually, one considers the N-way-K-shot classification in which train set contains I = KN examples from N classes each with K examples There are mainly 2 types of approaches used here: Matching Networks, and Prototypical Networks:

1. **Matching Nets**: Matching Nets meta-learns different embedding functions (f and g) for the training sample $x_i$and test sample x_{test}. The residual LSTM (resLSTM) proposes better designs for f and g. The final label is generally computed as the linear combination of the available labels weighted with the attention as formulated by [6]. The Relation Nets,as proposed by [8] also work on the same principle.

2. **Prototypical Networks (ProtoNet)** [4, 10]: Instead of comparing $f(x_{test})$ with each fx_i where $x_i \in$ train, Prototypical Networks only compare $f(x_{test})$ with the class prototypes in train set. For class n, its prototype is simply

$$c_n = \frac{1}{K} \sum_{i=1}^{i=K} f(x_i) \tag{1}$$

where there are K samples from class n. ’s are from class n. Empirically, this leads to more stable results and reduces the computation cost. The final classification is done simply by any distance metric like nearest prototype by euclidean distance or cosine metric.

3. **Other methods**[8, 9]: Lately Graphical Neural Networks have also been used where they leverage information from local neighborhoods. The basic idea is that each image can be presented as a node in a graph and the information (node representation) can be propagated between them according to how similar they are. Usually the classification is done implicitly according to the distance between nodes representation.

The survey [11] enumerates the all FSL approaches in depth done till 2020.

4.2 Class Incremental Learning Approaches

Many type of approaches have been formulated in past to either solve the problem of CIL (Class Incremental Learning) or Multi-Task Incremental learning, with solving the problem of catastrophic forgetting common in both cases:

1. **Regularization-based methods**. A first family of techniques is based on regularization. They estimate the relevance of each network parameter and penalize those parameters which show significant change when switching from one task/class to another. The difference between these methods lies on how the penalization is computed. For instance, the EWC approach in [12, 14], weights network parameters using an approximation of the diagonal of the Fisher Information Matrix (FIM). In [13], the importance weights are computed online. They keep track of how much the loss changes due to a change in a specific parameter and accumulate this information during training. A similar approach is followed in [15], but here, instead of considering the changes in the loss, they focus on the changes on activations. This way, parameter relevance is learned in an unsupervised manner.

2. **Architecture-based methods**: A second family of methods to prevent catastrophic forgetting produce modifications in a network's morphology by growing a sub-network for each task, either logically or physically . This is done by methods such as network pruning, dynamic expansion, and parameter masking. [16, 17] progressively increase the networks dimension. Networks like Piggyback [18] and Packnet [19], prunes the network to create free parameters for the new task, while HAT [20] learns the attention masks for old tasks and use them to constrain the parameters when learning the new task.

3. **Rehearsal-based methods** The third family of methods to prevent catastrophic forgetting are rehearsal based. Existing approaches use two strategies: either store a small number of training samples from previous tasks or use a generative mechanism to sample synthetic data from previously learned distributions.

   - In the first category, iCaRL [21] stores a subset of real data (called exemplars). For a given memory budget, the number of exemplars stored should decrease when the number of classes increases, which inevitably leads to a decrease in performance. A similar approach is pursued in [22], but the gradients of previous tasks are preserved. In [24] the authors propose two losses called the less-forget constraint and inter-class separation to prevent forgetting. The less-forget constraint minimizes the cosine distance be- tween the features extracted by the original and new models. The inter-class separation separates the old classes from the new ones with the stored exemplars used as anchors. An interesting work, [23], a bias correction layer to correct the output of the original fully-connected layer is introduced to address the data imbalance between the old and new categories.

   - The second category of networks are the one which use GAN to replay samples of previous classes. For example, DGR(deep generative replay) [26] learns a generative adversarial network to produce observed samples for the task solver .The recognition performance is affected by the quality of the generated samples. Similarly an improved version of DGR was introduced in [25, 27], where they use a class-conditional GAN to generate synthetic data.

4. **Distillation loss:** In some of the generative rehearsal mechanisms and some recent networks like [5, 28, 23, 24], distillation loss is incorporated to maintain the classification performance on old

classes. The distillation loss in all these networks is usually computed between the immediately previous session and the current session embeddings or feature vector for previous class samples. The distillation loss has been proved very effective to mitigate catastrophic forgetting.

In our FSCIL setting, we specifically have used a combination of Prototypical Networks for Few shot problem and a combination of Regularization-based method (EWC), distillation loss and Generative rehearsal mechanism for CIL problem. Mainly the advantage of using prototypical networks is that we need very less computation cost as compared to other Meta-Learning approaches. Further, intuitively, a prototype helps to avoid overfitting as it is the mean of all samples, leading to producing stable results as mentioned before. The generative mechanism has the advantage that we don't need any storage for saving exemplars(which can grow as the classes increase), similarly architecture based methods face the issue of increasing network as network increases or limited pruning possible given a fixed network.

The only relevant work with our setup is the latest CVPR 2020 paper [3]. In future, we would be comparing our evaluations with the results from this paper.

5 Problem Statement

The definition of Few shot Class Incremental Learning is as follows:
Suppose we have a stream of labelled training sets D(1), D(2) ,. . . . , where D(t) = $(x_j^{(t)}, y_j^{(t)})_{j=1}^{|D^t|}$ where "t" is the training session. $P^{(t)}$ is the set of classes of the t -th training set, where $\forall i, j, P^{(i)} \cap P^{(j)} = \phi$. $D^{(1)}$ is the large-scale training set of base classes, and $D^{(t)}, t > 1$ is the few-shot training set of new classes. The model Θ is incrementally trained on $D^{(1)}, D^{(2)}, \cdots$ with a unified classification layer, while only $D^{(t)}$ is available at the t -th training session. After training on $D^{(t)}$, Θ is tested to recognize all encountered classes in $P^{(1)}, \cdots, P^{(t)}$. For $D^{(t)}, t > 1$, we denote the setting with N classes and K training samples per class as the N -way K -shot FSCIL.

6 Approach

In this section, we explore the approaches which we thought of during the project. We first started with a basic implementation of prototypical network to know its feasibilty for our final FSCIL model and got some results on omniglot dataset. The setup for this was just FSL(Few shot learning). This was followed by a Generative modelling setup, where we generated Past classes pseudo prototypes to solve the problem of forgetting. In this we followed the desired FSCIL setup as was described above. Finally, a revised version of Pseudo Prototype Generator is discussed, whose implementation is currently underway.

6.1 Prototypical Network with Nearest Mean classifier

Before describing the model, we will just formally define the FSL (Few shot learning) setup.
We are given a small support set of N labeled examples $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_I, y_I)\}$ where each $\mathbf{x}_i \in R^D$ is the D -dimensional feature vector of an example and $y_i \in \{1, \ldots, N\}$ is the corresponding label. S_n denotes the set of examples labeled with class n. Here we consider the N-way-K-shot classification in which train set contains I = KN examples, from N classes each with K examples.

In the training, we generate such episodes of N-way-K-shot and update the loss. In each episode we produce prototypes for each class as follows:

$$\mathbf{c}_n = \frac{1}{|S_n|} \sum_{(\mathbf{x}_i, y_i) \in S_n} f_\phi(\mathbf{x}_i) \tag{2}$$

where f is the embedding function, such that, embedding function $f_\phi : R^D \to R^M$ with learnable parameters ϕ. This results in R^M, dimension prototypes c_n. We finally formulate the loss L as follows (assume sample x belongs class n):

$$L = CE(p_\phi(x, n)) + d(f_\phi(x), c_n) \tag{3}$$

Here $d(f_\phi(x), c_n)$ is the euclidean distance between $f_\phi(x)$(embedding of x) and c_n (prototype of class n). Also, CE is the cross entropy loss defined on probability distribution p_ϕ which is defined as follows:

$$p_\phi(y = n \mid \mathbf{x}) = \frac{\exp\left(-d\left(f_\phi(\mathbf{x}), \mathbf{c}_n\right)\right)}{\sum_{n'} \exp\left(-d\left(f_\phi(\mathbf{x}), \mathbf{c}_{n'}\right)\right)}$$

. Intuitively, p_ϕ is the probability distribution which depends on distance of embedding of sample from all the prototypes generated in a episode.

While training, in each episode, we sample Support and Query set from the available data for some sampled set of classes from total number of classes N. The algorithm which is followed for training our prototypical network(adopted from [4]) is as follows:

Algorithm 1: $N_C \leq$ N is the number of classes per episode, N_S is the number of support examples per class, N_Q is the number of query examples per class. RANDOMSAMPLE(S, N) denotes a set of N elements chosen uniformly at random from set S, without replacement.

Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_I, y_I)\}$, where each $y_i \in \{1, \ldots, N\}$.$\mathcal{D}_n$ denotes the subset of $\mathcal{D}$ containing all elements $(\mathbf{x}_i, y_i)$ such that $y_i = n$. Output: The loss J for a randomly generated training episode.

```
$V \leftarrow$ RANDOMSAMPLE $(\{1, \ldots, N\}, N_C)$
for $n$ in $\{1, \ldots, N_C\}$ do
    $S_n \leftarrow$ RANDOMSAMPLE $(\mathcal{D}_{V_n}, N_S)$
    $Q_n \leftarrow$ RANDOMSAMPLE $(\mathcal{D}_{V_n} \backslash S_n, N_Q)$
    $\mathbf{c}_n \leftarrow \frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_n} f_\phi(\mathbf{x}_i)$ end for
J $\leftarrow$ 0
for k in 1, . . . , NC do
    for (x, y) in Qn do
        J $\leftarrow$ J + $\frac{L}{N_C N_Q}$
    end for
end for
```

During testing time, we follow same process for sampling and calculating prototypes, however now instead of computing lose, we would calculate the euclidean distance of query samples from the prototypes computed from Support set. We then output the predicted class for the samples as that prototype class, whose distance is minimum from the embedding of the query sample.

Intuitively, the network f_ϕ starts generating feature vectors such that the samples start getting closer to their prototypes and inter class distance is maximised.

We further experimented with few other metrics like instead of euclidean distance, took the **cosine similarity** between prototype and embedding of query sample. The current network was a encoder, which gave the output as embedding of the input image. A decoder was added in front of the encoder and a **Reconstruction MSE** loss was also added to the loss L.

Since we dont have enough samples, it is very likely that the prototype generated using few samples from the class has a bias associated with it. Also overfitting issue would be there. Hence there was another experiment we conducted in this prototypical setup, where the prototypes, inspired from [29], were modelled as Gaussian Prototypes as follows:

$$c_n = \mathcal{N}(\mu_n, \Sigma_n) \tag{4}$$

where μ_n is the mean as we calculated before over samples, and σ_n is the covariance matrix for the samples and $\mathcal{N}(\mu_n, \Sigma_n)$ is the Gaussian Normal with that mean and covariance. Now a major issue in this was, in our Few shot setup, the covariance matrix cannot be calculated, where the embedding dimension > number of samples available for a class. This comes from the fundamental aspect of how

Covariance matrices are calculated (The determinant of covariance matrix is 0 if the matrix X (all samples feature vector stacked) is a ”tall” matrix or in other words embedding dimension > number of samples). Subsequently, we had to abandon this method, as we know while modelling as Gaussian, we need determinant of covariance matrix(in denominator), which shouldn’t be 0.

6.2 Pseudo Prototype Generator

The major intuition for us to adopt for a generative modelling, was that it would help generate pseudo prototypes for each class which as we saw in previous section was not possible if modelled directly as Gaussian.

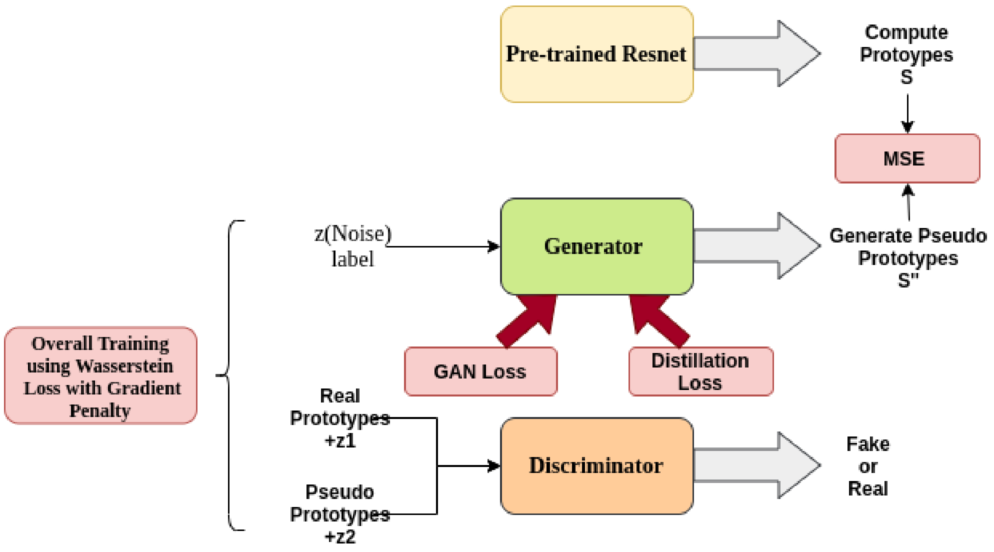

Figure 1: Pseudo Prototype Generator Architecture

Figure 1 shows the architecture of Pseudo Prototype Generator. In this approach we solve our desired FSCIL problem. Let G_t, $\mathcal{D}_t$ denote the Generator and Discriminator at t-th session. As described in section 5, we use that notation for our dataset samples. The network mainly has 3 components:

- **Generator**: The generator takes input as Noise z(sampled from unit normal) concatenated with label and outputs a pseudo Prototype. A generator G_T should be able to generates pseudo prototypes for all classes seen from t=1 to t=T after session T ends.
- **Discriminator**: The discriminator takes input as Real Prototypes perturbed with some Noise z_1 and Pseudo Prototype perturbed with another Noise z_2, both again sampled from Gaussian Normal and it tells which of the sample is fake or real.
- **Pretrained Resnet**: This is our Feature Generator, which takes images as input and generates a feature vector for them $\in R^M$ where M is the dimension of Latent space.

Let C^t denote the prototypes computed for classes at t-th session. The prototype is simply computed as per Equation(2) or average of samples. The MSE loss shown in figure is simple Mean Squared loss between, pseudo prototypes generated by Generator and Prototpyes computed from embeddings coming from Pretrained Resnet.

The distillation loss, for alleviating catastrophic forgetting is also employed, as can be seen in figure, is simply defined as follows:

$$Distil - Loss = MSE(G^t(z, label), G^{t-1}(z, label) \tag{5}$$

Here label runs for all the class labels observed till time $t-1$. Basically Distil-Loss computes the MSE between pseudo prototypes generated by the Current Generator and immediate previous Generator for all the classes observed till previous session. Intuitively, this helps to avoid forgetting, as we are in a sense replaying the past samples. We have to do this, since we dont have access to past original samples(from 1 to t-1) at session t. Define Generator loss:

$$GAN\ Loss + Distil - Loss + MSE \tag{6}$$

We finally write the algorithm used for training the Pseudo Prototype Generator.

Algorithm 2: Pseudo Prototype Generator Training

```
Input: Sequence D(1), ...,D(T) where D(t) = $(x_j^{(t)}, y_j^{(t)})_{j=1}^{|D^t|}$
Require: Feature extractor $F$, Generator $G_0$, Discriminator $\mathcal{D}_0$. All trained end-to-end.
for $t = 1, \ldots, T$
    Step 1: Generate Real Prototypes $C^t$ for classes from $\mathcal{D}_1$ using $F$
    if $t = 1$
        Step 2: Train $G_1$ and $\mathcal{D}_1$ with Real prototypes c $\forall c \in$C$^1$ using MSE+GAN Loss
    else
        Step 3: Train $G_t$ and $\mathcal{D}_t$ with Real prototypes c $\forall c \in$C$^t$ using MSE+GAN+Distil-Loss
end for
```

Intuitively, after every session, Generator is capable enough to produce prototypes for classes that it has seen till session t. After training over a session, the testing is done for all classes observed till that session as follows:

- Generate the feature vector for x_{test} as $F(x_{test})$
- Generate 10 pseudo prototypes for each class observed till now using the Generator G.
- Find the nearest distance prototype from $F(x_{test})$ using Euclidean Distance as metric and output the predicted label as the prototype's class label.

6.3 Revised Pseudo Prototype Generator

The main intuition behind Revising the Pseudo Prototype Generator was the change that Feature Extractor F which was Pre-trained Resnet in previous approach, is now replaced by a learnable Feature Extractor. This is required since we wont be getting Pre-Trained Feature Extractor available always.

We have further added some loses, which help in the training of Feature Extractor and overall GAN model. We first incorporate Elastic Weight Consolidation Loss (EWC) for Feature Extractor according to [12]. In our setup, the EWC Loss at session t is simply defined as follows:

$$EWC = \sum_i F_i * (\theta_i^t - \theta_i^{t-1})^2 \tag{7}$$

where θ_i^t and θ_i^{t-1} are the Feature Extractor parameter at session t and $t-1$ respectively labelled by i and F_i is the Fisher Info Matrix for parameter θ_i. Its construction is simply the squares of gradient with respect to parameter θ_i of our Feature Extractor at time step t averaged over all data samples. (as mentioned in [30]).

Another modification is the dataset augmentation and then dividing it into query set(Q(t)) and support set(S(t)). Intuitively, the support set is used for calculating prototypes. As can be seen in figure, we have further added a MSE2 loss which is defined as follows:

$$MSE2 = MSE(Q(t), S") \tag{8}$$

where S" are the Pseudo Prototypes generated at session t. Intuitively, the Query set is used for avoiding overfitting, and the network will try to produce representation or features of images close to prototypes. (Previously we only had MSE loss between S" and S, which might lead to Generator network being overfitting to Real prototypes)

Finally a small modification is done at the discriminator side. We have implemented a C-Class Classifier too, so a discriminator would also need to tell the class of the prototype in addition to whether its fake or real. This would help in further discriminating the classes. Hence we define final 2 loses: Gen-Loss, Disc-Loss:

$$Gen - Loss = GAN\ Loss + Distil - Loss + MSE \tag{9}$$

$$Disc - Loss = MSE2 + Cross - Entropy + EWC \tag{10}$$

where the Cross-Entropy loss is defined corresponding to C-Class Classifier from Discriminator.

Notation: $F(t)$ is the feature extractor at time t. J = number of iterations for session t.
We finally write the algorithm for training of our Revised Pseudo Prototype Generator.

Algorithm 3: Revised Pseudo Prototype Generator Training

```
Input: Sequence D(1), ...,D(T) where D(t) = $(x_j^{(t)}, y_j^{(t)})_{j=1}^{|D^t|}$
Require: Feature extractor $F_0$, Generator $G_0$, Discriminator $\mathcal{D}_0$. All trained end-to-end.
for $t = 1, \ldots, T$
    Step 1: Augment D(t) to generate S(t) and Q(t)
    repeat for J iterations:
        Step 2: Sample Batch Size from S(t) and Q(t)
        Step 3: Generate Real Prototypes $C^t$ for classes from sampled Batch from S(t) using $F(t)$
        if $t = 1$
            Step 4: Train $G_1$ with Real prototypes c $\forall c \in$C$^1$ using MSE+GAN Loss
                    Train $\mathcal{D}_1$ and $F_1$ with Real prototypes c $\forall c \in$C$^1$ using MSE2+Cross-Entropy Loss
        else
            Step 5: Train $G_t$ with Real prototypes c $\forall c \in$C$^t$ using Gen-Loss
                    Train $\mathcal{D}_t$ and $F_1$ with Real prototypes c $\forall c \in$C$^t$ using Disc-Loss
end for
```

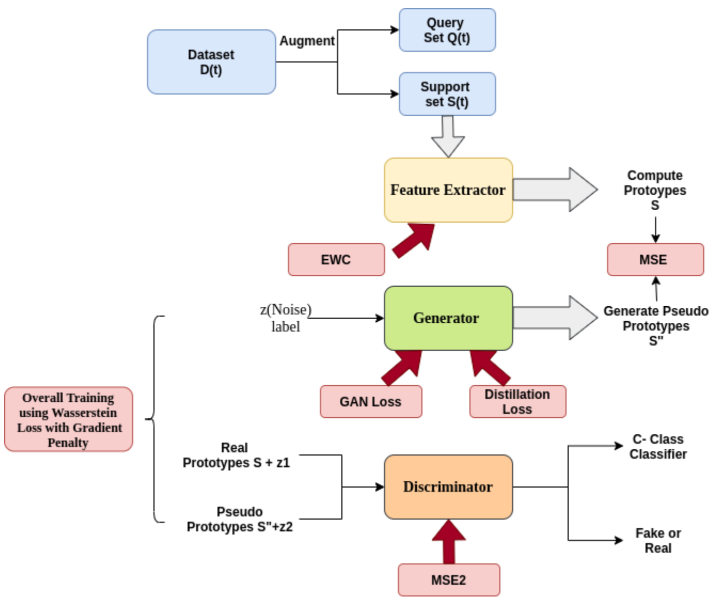

Figure 2: Revised Pseudo Prototype Generator

7 Experiments

In this section we report the results for the Approaches we have devised previously. The Revised Pseudo Prototype Generator model is still in progress , hence we report results only for Approaches 6.1, 6.2

7.1 Prototypical Network with Nearest Mean classifier

7.1.1 Implementation Details

The FSL setup was implemented on Omniglot dataset. Omniglot [31] is a dataset of 1623 handwritten characters collected from 50 alphabets. There are 20 examples associated with each character, where each example is drawn by a different human subject.

The embedding architecture is composed of four convolutional blocks. Each block comprises a 64-filter 3×3 convolution, batch normalization layer, a ReLU nonlinearity and a 2×2 max-pooling layer. When applied to the 28×28 Omniglot images this architecture results in a 64-dimensional output space. Used the same encoder for embedding both support and query points. Training was done via SGD with Adam. The Initial learning rate of 10^{-3} was used and it was reduced by half every 500 episodes. For all experiments we used 20 way 5 shot setup.

7.1.2 Results and Ablation

As mentioned in Approach section, we used some variations with loses to see how the accuracy changes. As seen in table,4, clearly keeping only MSE loss isn't beneficial. The cross entropy loss gives a great boost to accuracy.

Losses	Accuracy
MSE	41%
MSE(Euclidean) + Cross Entropy	99%
Cosine+ Cross Entropy	98%
Reconstruction +MSE + CE	99%

As can be observed in TSNE Plot, incorporating only MSE loss, would mean samples to get close to their prototypes. However, there is no restriction on how other prototypes get spaced out. hence, all the samples get very close to each other in the process and effectively since we are using nearest mean Classification while testing, it can be highly probably a different prototype becomes close to the test sample and hence a low test accuracy.

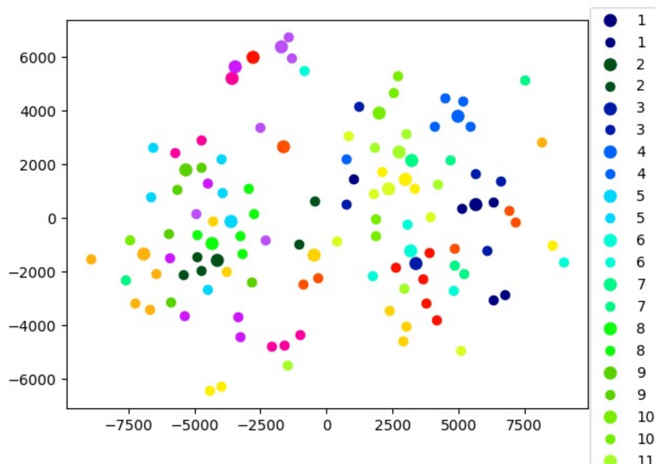

Figure 3: TSNE Plot for MSE Loss. Bigger Dots are the prototypes, smaller dots are the samples

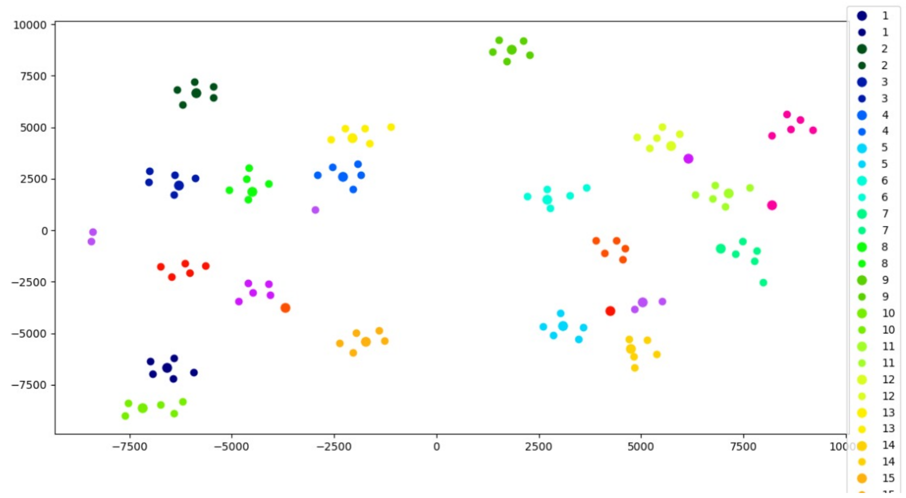
Figure 4: TSNE Plot for MSE + Cross Entropy Loss. Bigger Dots are the prototypes, smaller dots are the samples

We can clearly observe the samples being crowded around the prototypes and all the prototypes being spaced sufficiently apart. Also adding Reconstruction and Cosine metric doesn't bring much difference to accuracy, as the dataset is omniglot which is relatively less complex.

7.2 Pseudo Prototype Generator

7.2.1 Implementation Details

Datasets: We evaluate performance on imagenette [32]. Imagenette is a subset of 10 easily classified classes from Imagenet (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute). We centre crop imagenette images to 224*224, and normalise using standard mean and std deviation values.
Training: We use Pytorch as our framework. For feature extractor, we use the standard Resnet18 pretrained on Imagenet dataset. We train GANs for 1000 epochs. The Adam optimizer is used in all experiments, and the learning rate is 1e-4. Furthermore GAN is trained using standard Wasserstein Loss with Gradient Penalty. The coefficient of all loss terms is set to be 1 except the distillation loss whose weight is increased as we session number increases. This is to ensure that, previous classes are not forgotten and are given due weightage. For our imagenette dataset, we take 4 base classes with 100 samples in each class and subsequently, 2 class in each incremental session with 5 samples for each class (2 way 5 shot). Hence we would have 3 incremental sessions in our setup.
Evaluation The evaluation is done after every session. 50 pseudo prototypes are generated by Generator for all classes observed till that session and nearest prototype is found for the test samples. We finally report class wise accuracy after all sessions are done. Also overall accuracy after each session is also reported.

7.2.2 Results and Ablation

We first observe the overall Accuracy of model after every session.

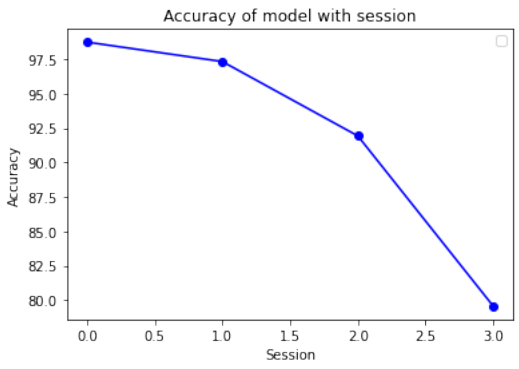

Figure 5: Accuracy of model with session

We can clearly observe that model accuracy drops after every session. This is expected as model would be forgetting previous classes as well as it has to learn from few samples arriving at incremental sessions. At the same time, it is observed that the lowest accuracy which goes is about 80%. This is quite satisfactory as state of art model TOPIC shown in [3] has overall accuracies reaching about 40% after 3 incremental sessions on all datasets. However, their setup is little different (50 base classes and 5 way 5 shot in incremental sessions). Hence we cant compare directly.

Following are the final classwise accuracies of our model after 3rd session. We observe that, due to large number of samples and distillation loss, the base class accuracies are high. For later classes, due to less number of samples and to retain the parameters based on past session generator, we observe a dropping classwise accuracy which is quite expected. At the same time, the latest class accuracy is little higher than just the preceding session classes, since the model is freshly trained on them and hence would produce more accurate prototypes.

Class	1	2	3	4	5	6	7	8	9	10
Final Accuracy(in %)	99.5	94.5	99	93	98	88	49	56.5	51.5	66.5

Table 1: Final Class-wise accuracies of our model

A better idea of the model performance can be got through a confusion matrix shown in Figure 6. We can observe that base classes contribute to lot of false positives and incremental classes have high number of false negatives. In short, the higher classes, are classified as base classes, mainly due to again relatively large number of samples available for base classes.

8 Future Work and Conclusion

In this project, we attempted to solve a Continual Learning setup Problem, specifically Few shot Class Incremental Learning which is ill-posed with dual problem of catastrophic forgetting and overfitting on incremental classes. We explored the past approaches which have been adopted solve either of the above problems and took inspiration from them to solve our FSCIL setup. We started with a FSL setup approach to know its feasibility, followed by a Generative modelling setup and give results on a smaller but a good representative dataset Imagenette for this setup. We analyse its accuracies and conclude them to be quite promising. We finally device a revised Generative model setup to ensure our model gives good accuracies on any dataset. The implementation of this revised model is currently underway and should be completed soon. Following are the future work which would be done in this project:

- Complete implementation of revised Generative Modelling setup and tuning of its various hyperparameters.
- Comprehensive analysis of the model on standard datasets of CUB200, CIFAR100 and Mini-Imagenet.

- Reduce the bias caused by base classes (due to relatively large number of samples present) and subsequently reduce the false negatives for later incremental classes. Would be taking inspiration from approach showed in [23] where they show that there is a bias associated with the FC layer in the network and introduce a Bias correction layer.
- Stabilise the training of GAN to improve its performance.

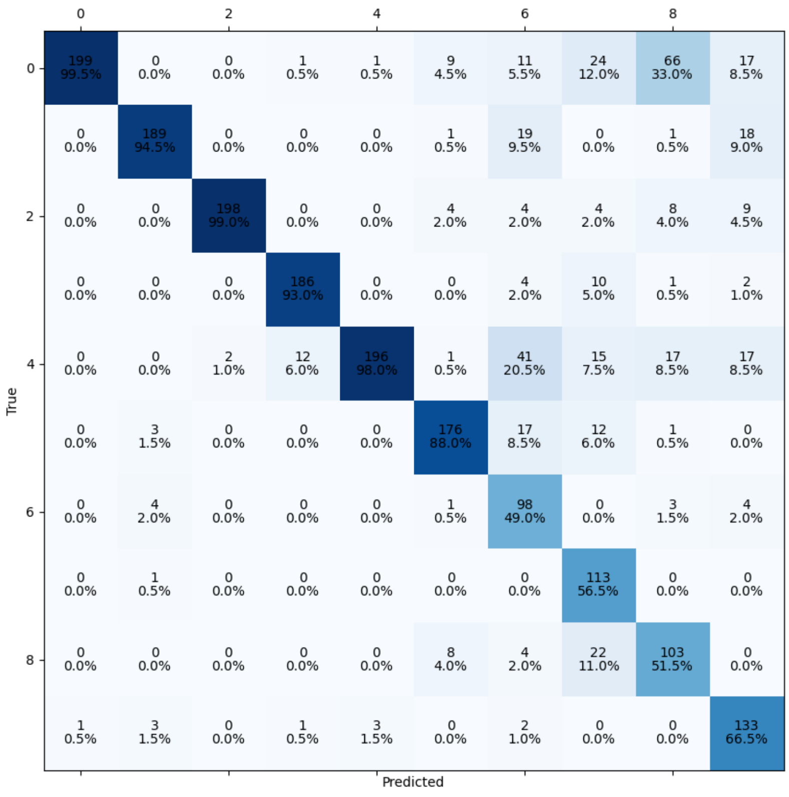

Figure 6: Confusion Matrix for Model after all sessions are completed

References

[1] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, Stefan Wermter *Continual Lifelong Learning with Neural Networks: A Review*

[2] Mengye Ren, Renjie Liao, Ethan Fetaya, Richard S. Zemel *Incremental Few-Shot Learning with Attention Attractor Networks*

[3] Xiaoyu Tao, Xiaopeng Hong, Xinyuan Chang, Songlin Dong, Xing Wei, Yihong Gong *Few-Shot Class-Incremental Learning*

[4] Jake Snell, Kevin Swersky, Richard S. Zemel *Prototypical Networks for Few-shot Learning*

[5] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D. Bagdanov, Shangling Jui, Joost van de Weijer *Generative Feature Replay For Class-Incremental Learning*

[6] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, Daan Wierstra *Matching Networks for One Shot Learning*

[7] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. 2018. *Learning to compare: Relation network for few-shot learning.*

[8] Victor Garcia, Joan Bruna *Few-Shot Learning with Graph Neural Networks*

[9] Hao Cheng, Joey Tianyi Zhou, Wee Peng Tay, Bihan Wen *Attentive Graph Neural Networks for Few-Shot Learning*

[10] Yu-Xiong Wang, Ross Girshick, Martial Hebert, Bharath Hariharan *Low-Shot Learning from Imaginary Data*

[11] Yaqing Wang, Quanming Yao, James Kwok, Lionel M. Ni *Generalizing from a Few Examples: A Survey on Few-Shot Learning*

[12] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, Raia Hadsell *Overcoming catastrophic forgetting in neural networks*

[13] Friedemann Zenke, Ben Poole, and Surya Ganguli. *Continual learning through synaptic intelligence.*

[14] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov *Rotate your networks: Better weight consolidation and less catastrophic forgetting*

[15] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. *Memory aware synapses: Learning what (not) to forget*

[16] Jeongtae Lee, Jaehong Yun, Sungju Hwang, and Eunho Yang. *Lifelong learning with dynamically expandable net- works.*

[17] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Raz- van Pascanu, and Raia Hadsell. *Progressive neural networks.*

[18] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. *Piggyback: Adapting a single network to multiple tasks by learn- ing to mask weights.*

[19] Arun Mallya and Svetlana Lazebnik. *Packnet: Adding multiple tasks to a single network by iterative pruning.*

[20] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. *Overcoming catastrophic forgetting with hard attention to the task.*

[21] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert *icarl: Incremental classifier and representation learning.*

[22] David Lopez-Paz and Marc'Aurelio Ranzato. *Gradient episodic memory for continual learning*

[23] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. *Large scale incremental learning.*

[24] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin *Learning a unified classifier incrementally via rebalancing*

[25] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. *Memory replay gans: Learning to generate new categories without forgetting.*

[26] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim *Continual learning with deep generative replay*

[27] Mengyao Zhai, Lei Chen, Fred Tung, Jiawei He, Megha Nawhal, Greg Mori *Lifelong GAN: Continual Learning for Conditional Image Generation*

[28] Francisco M Castro, Manuel J Mar ́ın-Jimenez, Nicol ́ as Guil, ́ Cordelia Schmid, and Karteek Alahari. *End-to-end incremental learning*

[29] Jogendra Nath Kundu, Rahul Mysore Venkatesh, Naveen Venkat, Ambareesh Revanur, R. Venkatesh Babu *Class-Incremental Domain Adaptation*

[30] Ryo Karakida Shotaro Akaho Shun-ichi Amari *Universal Statistics of Fisher Information in Deep Neural Networks: Mean Field Approach*

[31] Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. *One shot learning of simple visual concepts.*

[32] Link for imagenette dataset