Architecting and Deploying Optimized GANs with minimal footprint for Fashion Synthesis

Samyak Sarnayak*, Ritik Hariani†, K S Srinivas‡
Department of Computer Science and Engineering, PES University
Bengaluru, India
Email: *samyakssarnayak@pesu.pes.edu, †ritikhariani@pesu.pes.edu, ‡srinivasks@pes.edu

***Abstract*—Fashion generation using Generative Adversarial Networks (GANs) has been very effective at synthesising new clothing items, but the models are large, over-parameterized and are not accessible to an average fashion designer. We propose that using the Lottery Ticket hypothesis (LTH) and Early Bird Tickets (EBT) along with channel pruning on large GANs leads to smaller, optimized models that can be deployed on a client-side web app or a mobile app. We empirically show that LTH can reduce the size of the model by over 55% and EBT can reduce model size by over 16 times (6.2% of the original size) and the inference time of the model by over 21 times (4.7% of the original model). Thus, the pruned models can be run on low power devices or on a client side browser or app with minimal footprint.**

***Index Terms*—Generative Adversarial Networks, Fashion Generation, Pruning, Lottery Ticket Hypothesis, Early Bird Tickets, Channel Pruning**

I. Introduction

Deep learning has recently been employed for many applications within the fashion industry. These include recommendation systems on e-commerce sites, style-matching, personalized customisation, prediction of trends, AR clothing experience, clothing classification. One such domain that has been gaining attention over the last few years includes that of Fashion Synthesis and Generation. Fashion Synthesis involves the processes of generating a variety of new clothing articles for customers in a bulk and within a short span of time. To achieve this feat, Deep learning frameworks such as *Generative Adversarial Nets* can be applied on clothing datasets.

Generative adversarial networks (GANs) [1] is a concept based on game theory for generative learning. GANs aim at training a generator network G to produce samples from the dataset by transforming latent vectors of noise. The feedback from the Discriminator acts as a training signal for G. The Discriminator D is trained to distinguish samples between the generator's output distribution and the real data. The generator network G in turn is then trained to deceive the discriminator into accepting its outputs as equivalent of being real.

GANs are heavy weighted models and training them has proved to be a challenging task. They also require a lot of computation time for training. Methods to improve training on GANs [2] have been developed over the recent years. We propose the use of optimization techniques such as **Lottery Ticket Hypothesis (LTH)** and **Early Bird Tickets (EBT)** to tackle this situation. LTH can be used to minimize the number of the parameters (thereby creating a sparse network) and EBT is an improvement over LTH that also reduces the training time. These techniques involve pruning of the neural network where the parameter counts of trained networks can be reduced by over 80%. The LTH algorithm identifies *winning tickets* which are the weights of higher magnitude by the method of iterative pruning.

Early Bird Tickets is an improvement over LTH which showed that the *winning tickets* can be identified very early in training without having to iteratively train and prune a network multiple times. The winning tickets could be identified with only 6-20% of the total training. To improve the inference time and size of convolution-based models, *channel pruning* can be used to remove unused channels from the convolution layers.

II. Related Work

Generative Adversarial Networks (GANs) [1] were introduced by Goodfellow et al. A GAN comprises of two networks, a generator G and a discriminator D. The generator tries to learn the given data distribution and generates new data points from that distribution. The discriminator tries to determine whether a given image was obtained from the generator or it was obtained from the original dataset. The only input for the generator is a noise vector, known as the *latent vector*, which varies the generator's output as the noise is changed. The generator's goal is to fool the discriminator into outputting that its generated image is in the original dataset while the discriminator maximises the probability of correctly classifying images from the dataset and the images obtained from the generator. This can be represented as a loss function as follows.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

The generator and discriminator can both be neural networks which allows back-propagation of gradients from the discriminator outputs to the generator inputs.

Radford et al. [3] proposed the **Deep Convolutional Generative Adversarial Network (DCGAN)** which uses convolutional layers with ReLU in the generator and with LeakyReLU in the discriminator. Specifically, deconvolution (or convolution transpose) is used in the generator and convolution is used in the discriminator. Between each of the convolution

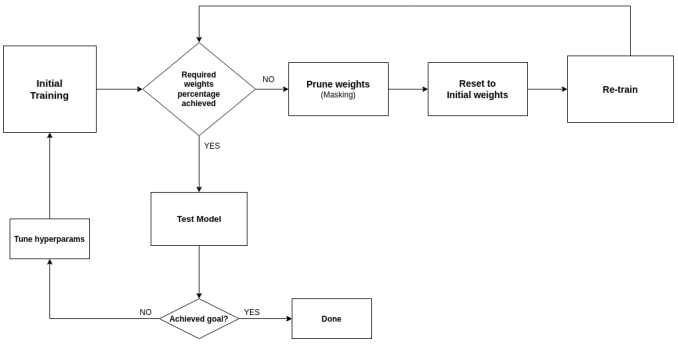

Fig. 1. A model training pipeline based on LTH

and deconvolution layers, batch normalization [4] is used to stabilize the gradients. Radford et al. [3] also provided architecture guidelines for training DCGANs which we use in our work.

Both the discussed GANs are unsupervised i.e., they only need the images themselves to train. Usually, the datasets used to train them also have labels for classification. The generator's outputs can be conditioned on the class label to have more control over the output. Mirza et al. [5] introduced the **Conditional Generative Adversarial Network (CGAN)**. In a CGAN, both G and D are conditioned on the input label y. The loss function then becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{2}$$

Negar Rostamzadeh et al. [6] introduced a dataset of over 290,000 images of fashion articles of size 1360x1360 pixels paired with item descriptions. Each item includes a recommendation by a stylist for a given fashion season. It also contains information about the designer and the brand. [6] synthesised their high resolution images using P-GANs (Progressive GANs) whereas the textual descriptions for each image was created using StackGAN-v1 and StackGAN-v2.

Ziwei Liu et al. [7] introduced DeepFashion, a high resolution clothes dataset with intensively detailed annotations. It contains over 800,000 images, which contain keywords that describe the attributes, clothing landmarks, location, street picture and image of the consumer. To test and present the capabilities of their dataset they proposed a deep model known as FashionNet which was used to predict clothing attributes and landmarks by learning the clothing features. FashionNet was inspired by the traditional VGG-16 network.

Han Xiao et al. [8] introduced Fashion-MNIST, a dataset that consists of 70,000 gray-scale images of fashion products, each of 28 × 28 pixel resolution. It conists of 10 categories each with 7,000 images per category. The dataset has a training set of 60, 000 images and a test set of 10,000 images.

Frankle & Carbin [9] introduced the **Lottery Ticket Hypothesis (LTH)** that proposed to find sparse, pruned sub-networks in a dense trained network. These sub-networks had the special property of being re-trainable allowing them to be as performant as the original model, sometimes even be superior. The LTH method consists of the following steps.

1) Train a network fully for i iterations from random initialisation. These initialisations must be saved.
2) Mask some $p\%$ of the parameters of the network. p is tun-able hyper-parameter and the masking is based only on the magnitude of the weights i.e., smaller weights are masked to 0 while others retained.
3) Set the masked weights to 0 and the remaining weights are re-winded to their initial random values (saved in step 1).
4) Re-trained the new masked network
5) The process is continued iteratively until the desired sparsity is achieved.

Figure 1 shows the pipeline for training a model based on LTH. This method performed well on the experiments showcased by the authors with some networks being pruned upto 80% without losing accuracy. A few major issues were identified in LTH which are the following.

1) The process involves training the models multiple times. Even with one-shot LTH, the model has to be trained twice. In iterative pruning, the models might have to be trained more than 5 times. This leads to a large amount of computation overhead that is very prohibitive to the goal of faster training.
2) Pruned sub-networks are represented by sparse matrices since the pruning is unstructured. Due to the fact that modern hardware such as GPUs and TPUs used for training such networks are optimised for dense matrices, there is little to no speedup achieved during training as well as testing/predicting. Thus, LTH does not aid in the goal of architecting optimised networks to run on low end devices.

Liu et al. [10] introduced a network slimming (pruning) method based on channel-level sparsity (henceforth called channel pruning). In **channel pruning**, a scaling factor γ is associated with every channel in the convolution and convolution-transpose layers. The scaling factors γ are trained along with the network (with some regularization to keep the factors small). After training, the channels with low γ are pruned by removing the entire channel along with its weights. The threshold for γ is chosen based on the desired sparsity percentage. Liu et al. further propose the use of *Batch Normalization* (BN) [4] and the inherent scaling parameter γ present in BN as a substitute for scaling factor. This leads to a method that does not require any changes in the model architecture as most vision models use BN. The pruned models can be further fine-tuned and re-trained to provide similar or higher performance as compared to the original models.

You et al. [11] developed an efficient method for training large networks by combining the works of LTH and channel pruning, this was named **Early Bird Tickets** (EB tickets). EB tickets can be identified very early in training, hence the name Early Bird. Unlike LTH, the networks do not have to be trained completely. Here, the masks are not over the entire network

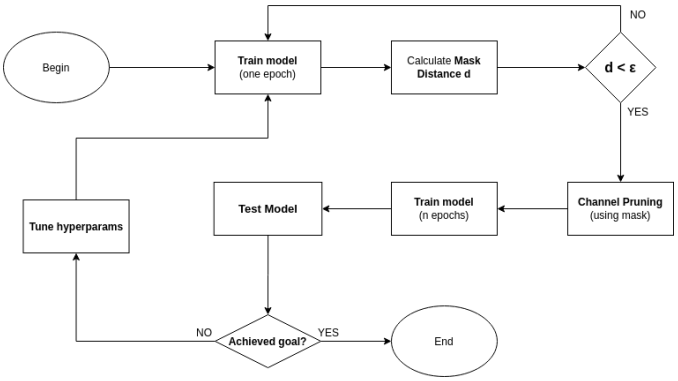

Fig. 2. A model training pipeline based on EBT

weights but the masks are channel-based i.e., entire channels are masked to 0 or 1 from the ideas of channel pruning. The winning tickets are identified early in training by using a large learning rate and monitoring the channel masks by using a metric known as the *Mask Distance* metric, which is simply the hamming distance between channel masks of two consecutive epochs of training. It was shown that the masks are fixed early in training and there is little change as training progresses. A rolling window of the mask distances are kept and based on a set threshold ϵ, when all of the distances in the windows are less than the threshold, the EB training is stopped and masks are fixed. Based on this mask, entire channels are pruned and a new model with only the required channels is made. Figure 2 shows a model training pipeline based on EBT. In summary, this method has the following advantages.

1) Pruned sub-networks are identified early in training (6-20% of original epochs or steps). This implies that networks do not have to be trained multiple times.
2) Pruned sub-networks are not reset to original random initialized weights.
3) Pruned sub-networks are optimized even for modern hardware which work only on dense matrices since entire channels are pruned and the model dimensions themselves change.
4) Thus, EB training reduces the training time, model size, inference time of model as well as the overall computation required for training and testing.

Heusel et al. [12] proposed a method of training GANs to help them converge locally. **Two-Time Update Rule** (TTUR) refers to using dissimilar learning‘ rates for the generator and the discriminator. Specifically, a higher learning rate is used for the discriminator while a lower learning rate is used for the generator. Heusel et al. proved that TTUR leads to better convergence in GANs.

Sinha et a. [13] introduced a simple method of improving performance of GANs called **Top-k training**. The method involves using the discriminator as a scorer i.e., the discriminator outputs are used to judge the generator outputs. Some of the generator outputs which are of low quality i.e., the samples generated by the generator which are easily labelled as fake by discriminator are discarded. The discarded samples do not contribute to the gradients and the generator only learns from the higher quality samples leading to better outputs. We use this method to improve the quality of outputs from our GANs.

III. Methodology

The architecture for Deep Convolutional Generative Adversarial network (DCGAN) was the same as in [3] with a latent vector of 100 length and 5 convolutional layers, each with a 4x4 kernel, along with BN for both for both the networks (G and D). The final output image is of size 64x64. This architecture is named DCGAN64 and is used for the FashionMNIST experiment (final channel size 1) as well as DeepFashion (final channel size 3) with down-scaled images.

An alternate architecture with 6 layers and a final output size of 128x128 was used for the DeepFashion experiment. This architecture is named DCGAN128.

The architecture for our Conditional Generative Adversarial network [5] involves a Generator and a Discriminator conditioned on the input label y. The input vector of class labels to the generator has been encoded using one-hot encoding. Our CGAN model consists of a generator that uses a latent vector of length 100 and 5 convolution layers, each with a 4x4 kernel, stride 2 and a padding of 1 followed by Batch Normalization [4]. The one-hot class vector is combined with the noise vector before giving it as an input to the generator. It uses the ReLU activation function for the intermediate layers and a $tanh$ for the output layer. The discriminator model consists of 5 convolution layers, each with a kernel size of 4, stride 2 and padding of 1 followed by Batch Normalization. The input to the discriminator is the image from the generator along with the an additional channel for every class in the dataset. The Leaky ReLU activation function is used for the intermediate layers along with a sigmoid activation as an output. The output is an image of size 64x64 and is based on the provided input label.

LTH optimization was implemented on the GAN models. This optimization was implemented individually on the generator and discriminator. The weights are initialized using Random initialization. A prune percentile $p\%$ is set as a tun-able hyper-parameter that decides the amount of weight reduction needed. The weights of each generator and the discriminator are masked and pruned upto the percentile p. Once the weights drop below the threshold t, decided by pth percentile of the weights, they are masked to 0 while others are retained. The remaining weights were then re-initialized to the their initial random values and the network model was retrained. This process proceeded iteratively until a sparse enough network that reproduced images with near original accuracy was obtained.

Channel Pruning was implemented by using a mask over all the channels over the network. The pruning was done individually for generator and discriminator as was done with the LTH implementation. The masks had the same shape as the weights of the Batch normalization layers which represent the scaling factors γ. A prune percentage threshold p is set

as a hyper-parameter, this controls the *number of channels to prune* as a percentage. Since entire channels are pruned rather than individual neurons, the final sparsity of the network will be much lesser than p. For example, in one of the experiments, choosing $p = 0.5$ lead to a sparsity of 73.7%. Based on p, the pth percentile t is calculated from the BatchNorm weights. This percentile value t is used as the threshold and the masks M of all channels with γ lower than this are set to 0.

$$w_{bn} = sort(w_{bn}) \tag{3}$$
$$t = w_{bn_{p*total}} \tag{4}$$
$$M_{\gamma<t} = 0 \tag{5}$$

A new network with $N - n$ number of channels, where N is the original number of channels and n is the number of channels to be pruned (in each layer), is created and the parameters of the remaining channels in the original network are copied to the new network. This leads to a network which has significantly lower number of parameters and performs better on traditional hardware.

Early Bird ticket is implemented as follows.

1) Initialize channel pruning masks for every layer
2) Initialize a empty bounded queue Q, with maximum size Q_m to store the mask distances after every epoch
3) After every epoch i, calculate the mask distance as follows.

$$d = \sum |(M_i - M_{i-1})| \tag{6}$$

4) Add d to the queue ensuring that the queue size does not exceed Q_m.
5) If $max(Q) < \epsilon$, stop training and perform channel pruning to get an optimized network.
6) Continue training the optimized network for the remaining epochs.

Two-Time Update Rule (TTUR) [12] is implemented by setting the generator learning rate to 0.0001 and the discriminator learning rate to 0.0004. The generator and discriminator are updated in different steps.

Top-k learning is implemented similarly as in [13].

$$D_{out} = topk(D(G(z)), k) \tag{7}$$

where $topk$ is a function returning the k largest elements of the vector. Following is the procedure for top-k training.

1) Initialize k to the batch size B and set the hyper-parameters γ - the decay rate of k and ν - the minimum value of k.
2) During back-propagation in the generator step, $D(G(z))$ values which are not in the top k samples for the batch are discard and their gradients are not calculated.
3) After every epoch of training, k is updated as follows.

$$k = max(\lfloor \gamma * k \rceil, \nu) \tag{8}$$

Fig. 3. Outputs from the complete DCGAN model trained on FashionMNIST

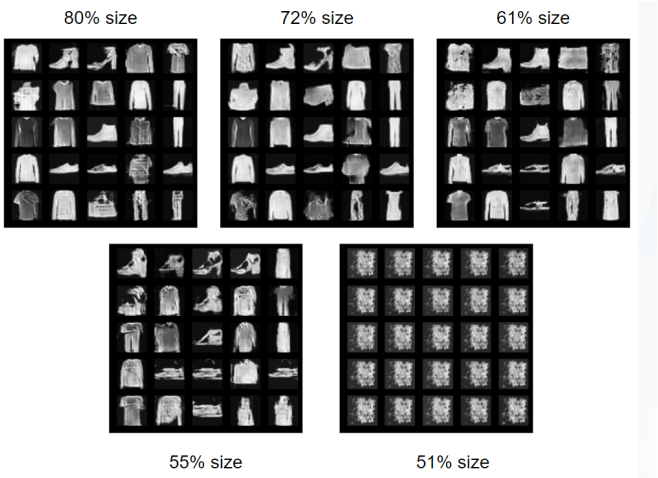

Fig. 4. Outputs from the LTH-pruned DCGAN model trained on FashionMNIST

IV. Experimentation

The DCGAN model was first trained on the FashionMNIST dataset. Binary Cross Entropy was chosen as the loss function for our model. This was optimized using the Adam optimizer. The learning rate for the generator was set to 0.0002 and that of the discriminator was set to 0.0001. The hyper-parameters of the first and second moment for Adam were set to 0.5 and 0.999 respectively. The model was trained for 20 epochs on a batch size of 256. The outputs from this model are shown in Figure 3.

The DCGAN model trained on the FashionMNIST dataset was then first subject to LTH optimization algorithm. The values for the prune percentile p on which the model was

Fig. 5. Outputs from the channel-pruned DCGAN model trained on Fashion-MNIST

experimented were: 80%, 72%, 61%, 55% and 51%. On observation as shown in the Figure 4, it can be seen that the image quality declines on decrease of the value of p. Our model was able to achieve a maximum pruning of weights upto 55% on retaining satisfactory image quality. Upon further pruning to a value of 51%, the image quality decreases drastically and further pruning would be of no help.

This optimization consumed a significant amount of computation/CPU time as it required the whole model (100%) to be re-trained about 5 to 6 times to successfully achieve lower pruning percentiles. Due to this, we implement the EBT optimization, which was initially inspired from LTH, to obtain better results from our model.

Therefore, our DCGAN model earlier trained on the FashionMNIST dataset was next subjected to the EBT optimization algorithm. For the EB Train algorithm, a maximum queue size $Q_m = 3$ and a threshold $\epsilon = 0.1$. From Table I, we can see that the model could be pruned to nearly 6% of the original size with inference time decreased by nearly 21 times. The inference time is calculated by generating images using a batch of 1000 latent vectors. With $p = 0.8$, the model had less than a million parameters. This model, when deployed as a client-only web app, achieved inference times of less than 100ms on CPU [1]. Figure 5 shows outputs of our channel-pruned DCGAN model for pruning percentages of 50%, 70% and 80%. It can be noticed that there is no significant decrease in quality when 70% of the channels have been pruned while pruning 80% of the channels leads to a slight decrease in quality. Thus, we have empirically shown that EBT performs much better than LTH.

[1]This was tested on a consumer laptop with i5-8265U (4 cores, 8 threads) and 8GB of RAM.

TABLE I
RESULTS FROM EARLY BIRD TRAINING OF THE DCGAN64 MODEL.

p	**Parameters**	**Sparsity**	**Inference time**	**Speedup**
0.0	12.6M	100%	26.32s	1x
0.5	3.3M	26.20%	7.06s	3.78x
0.7	1.5M	11.83%	2.47s	10.7x
0.8	0.78M	6.19%	1.23s	21.2x

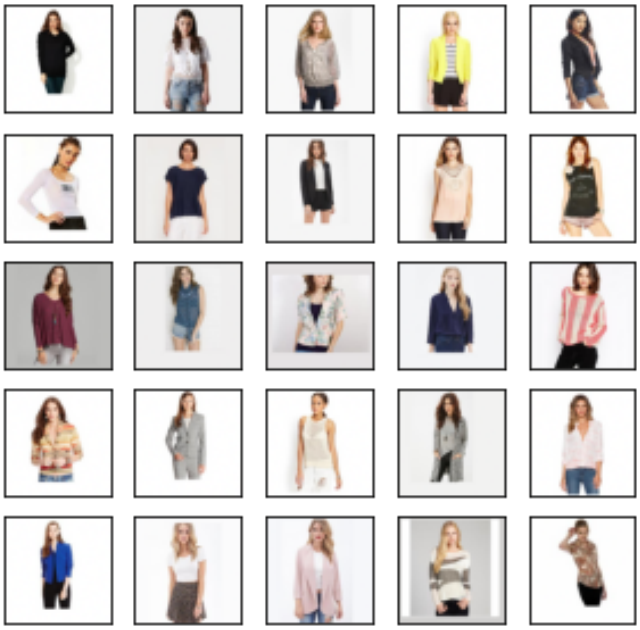

Fig. 6. A sample of images from the DeepFashion dataset.

On obtaining satisfactory results on the FashionMNIST dataset, we embark on training our model on DeepFashion, a larger dataset with images of higher resolution (128 x 128) as compared to FashionMNIST. Figure 6 shows a sample of images from the dataset. We train both the DCGAN64 and DCGAN128 models. The DCGAN64 model is trained by downscaling the images to 64 x 64. The training was done with EBT (channel pruning), label smoothing [2], TTUR [12] and Top-k sampling [13]. The following hyper-parameters were used: $LR_d = 0.0004, LR_g = 0.0001, p = 0.5, \gamma_k = 0.99, \nu_k = 0.5$. The results obtained from the DCGAN64 model are shown in Figure 7 and Figure 8 shows the outputs from DCGAN128. We can see that the DCGAN architecture produces worse outputs on 128x128 due to its inherent limitations.

The optimized models have been successfully deployed in a client-side web application where they are run in real-time on the client's device. The web application can be accessed using the following HTTPS URL: https://fashion.samyaks.xyz/.

V. CONCLUSION AND FUTURE WORK

In this paper, we successfully demonstrated that optimization algorithms and techniques such as Lottery Ticket Hypoth-