Multitask Learning with Generative Adversarial Networks

Ben Carr

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2017

Abstract

Generative modelling has undergone a paradigm shift with Ian Goodfellow's conception of the *Generative Adversarial Network* (GAN)[1]. In the three years since, this architecture and its numerous variants have made breathtaking advances in many domains, particularly in image synthesis. It is now possible for GANs to generate photo-realistic, hitherto unseen images, involving complex natural objects and scene dynamics. Until this year, a major drawback of GANs was their large sensitivity to hyperparameters and unstable training dynamics. These issues were effectively alleviated with the Wasserstein GAN[2] (WGAN) and its extension WGAN-GP[3].

This work aims to build upon these advancements by exploring the effect that *multitask learning* (MTL) has on GANs. MTL is a method of forcing models to simultaneously learn different, but related, tasks, thus increasing the generalisation power of their hidden representations. This has been proven to be very successful for discriminative models, as seen in the work of Rich Caruana[4].

This is where we begin, with MTL in discriminative networks, focusing on a form of MTL that shares weights between a main task classifying images from CIFAR-100 and an auxiliary task classifying the same images, only with coarser labels. We then turn to GANs, laying some foundational theory into how they work and how they are evaluated. This motivates our choice of using FID evaluation over the previously popular inception score and using WGAN-GP as a base architecture. This choice of WGAN-GP is later affirmed when comparing results against vanilla GAN architectures. We then extend this WGAN-GP model with different forms of class conditioning and MTL. Of these proposed models, many successfully condition upon class for CIFAR-100, i.e. the class of the generated images can be controlled.

Our most successful model works by conditioning similarly to Auxiliary Classifier GAN[5] but, inspired by our initial investigations into MTL, with an additional auxiliary classifier in the discriminator that classifies coarse labels. We incorporate this "MTL trick" into a state of the art ResNet WGAN to find a significant improvement in FID and image diversity.

Acknowledgements

I thank my supervisor Peter Bell for being a great soundboard and source of inspiration. I wish him all the best with George.

I thank Leo for his help and look forward to one day visiting him in Hunan.

I thank Buster, who, despite his cruciate ligament surgery, was faithfully by my side through the late nights.

I thank Jane & Ed for making my undergraduate studies possible.

I thank my family for the unending cups of tea and support.

Last but not least, I thank my friends, for the weekend we're finally about to have.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ben Carr*)

This one goes out to Dave

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Abbreviations

MTL	Multitask Learning
STL	Single Task Learning
GAN	Generative Adversarial Network
G	Generator
D	Discriminator
C	Classifier
MLE	Maximum Likelihood Estimation
WGAN	Wasserstein GAN
WGAN-GP	WGAN with Gradient Norm Penalisation
cGAN	Conditional GAN
cWGAN	Conditional Wasserstein GAN
AC-GAN	Auxiliary Classifier GAN
AC-WGAN	Auxiliary Classifier Wasserstein GAN
SC-GAN	Separate Classifier GAN
SC-WGAN	Separate Classifier Wasserstein GAN
IS	Inception Score
FID	Fréchet Inception Distance
SGD	Stochastic Gradient Descent
BN	Batch Normalisation, often referred to as Batchnorm
IN	Instance Normalisation
CIN	Conditional Instance Normalisation

Mathematical Symbols

p_G	Probability distribution of the generator's output
$p_{\mathbf{z}}$	Probability distribution of random noise
p_{data}	Probability distribution of training data
λ	Soft-sharing hyperparameter
b	Number of layers that are shared
$\mathbf{z}$	Data point sampled from $p_{\mathbf{z}}$
$\mathbf{x}$	Data point sampled from p_{data}
$\mathbf{y}$	Class of data point sampled from p_{data}
ReLU	Activation function with $f(x) = max(0,x)$
LeakyReLU	Activation function with $f(x) = \alpha x$ if $x < 0, x$ otherwise
$\mathcal{L}$	Loss function
θ	Generator's parameters
w	Discriminator's parameters
ϕ	Classifier's parameters
s	Frequency that the classifier is trained
$\mathcal{D}_{JS}(p\|\|q)$	Jenson-Shannon divergence between two distributions p and q
$\mathcal{D}_{KL}(p\|\|q)$	Kullback-Leibler divergence between two distributions p and q
$\sup(x)$	The least upper bound of x, can be thought of as the maximum
$\inf(x)$	The greatest lower bound of x, can be thought of as the minimum
η	Gradient penalisation hyperparameter
c	Weight clipping constant
$\|f\|_L \leq 1$	f is 1-Lipschitz, its gradient does not exceed a magnitude of 1
$\|f\|_L \leq K$	f is K-Lipschitz, its gradient does not exceed a magnitude of K

Chapter 1

Introduction

This thesis explores the intersection of *generative adversarial networks* and *multitask learning*. Before stating specific contributions we will give some background and motivation for both of these topics. This chapter also aims to motivate our choice of dataset, CIFAR-100, and provide context by discussing related contemporary work.

1.0.1 Supervised and Unsupervised Learning

Until recently, most major advancements in deep learning have been in *supervised learning*. These successes have come from a wide array of domains, perhaps the most spectacular from vision[6][7][8][9] and speech[10][11][12]. A supervised learning model learns how to classify after training on large quantities of labelled data. However, a vast proportion of the data collected in the world today is unlabelled, for example, most of the endless stream of images, video and audio uploaded to the internet. It is thus beneficial for A.I. systems to be designed such that they can harness this data in their learning. The prime example of *unsupervised learning* that we have is in ourselves. Our intelligence is, for the most part, learned through unsupervised observation, not by being taught the name for every event and object. For example, we don't need to be told that an object continues to persist after it has been obscured from view, we learn this through observation. It is commonly believed that to someday achieve artificial general intelligence, huge developments in unsupervised learning will have to be made.

1.0.2 Generative Adversarial Networks

Recently, the most promising step in unsupervised learning has been in generative modelling with generative adversarial networks (GANs), introduced by Goodfellow *et al.*[1]. These are deep neural networks that strive to generate new samples that look as if they've come from the training data. Loosely speaking, the measure of their success is in how "realistic" their generated samples are. To quantify this for an image is hard, and will be discussed later. This generation is usually done without the aid of labels, so is indeed a form of unsupervised learning. However GANs can benefit greatly from the addition of class information, not only in performance but also in allowing control of their output. We will mainly be exploring these *supervised GANs* in this work.

The adversarial element of a GAN comes from the fact that there are actually two networks in the model, both pitted against each other, as depicted in Figure 1.1. The *generator* network learns to create new samples whereas the *discriminator* network learns to identify whether a sample has been generated or if it is a real one from the dataset. By training these two networks in parallel the discriminator gets better at spotting fakes while the generator gets better at fooling the discriminator by generating more realistic fakes, eventually the quality of these generated samples can be very high. The situation is often dramatised as an art forger and police detective pitted against each other. As the police detective gets better at spotting a forgery, so too must the forger get better in fooling her/him.

One might reasonably ask why these generative models are worth researching. We have no scarcity of data in the world, why generate more? Why are GANs such an important step forward for unsupervised learning? Three important reasons to pursue GANs are:

- **Reinforcement Learning**. GANs can be incorporated into reinforcement learning agents to allow them to efficiently generate future possible states, conditioned on the current state. From these states, the agent can plan its actions accordingly. Additionally, for dangerous environments, by imagining states and actions, an agent can virtually explore an environment without risk of damage.

- **Semi-Supervised Learning**. GANs can help fill in missing data for semi-supervised tasks. This is of large practical value given that most data is unlabelled or only partly labelled.

- **Representation Learning**. To generate realistic samples, GANs must learn

powerful hidden representations of the data. These representations, once captured, can be used downstream for a variety of other models and tasks, e.g. for classification.

1.0.3 Class Conditioning with GANs

As mentioned, without labels, in a purely unsupervised setting, there is no way to control what class of image a GAN will generate. It might learn to generate diverse images of each class, but there is no way of knowing beforehand which inputs will generate which class. To overcome this, a GAN can be conditioned on by the class information, in both the discriminator and generator. "Conditioned by" here can be as simple as using the class vector as an additional input into the network. When successful, the generator learns a relationship between the class input and its output. It learns this relationship to better fool the discriminator, since the discriminator has now learnt from real images what the relationship should be. For example, the discriminator might "know" that a perfect image of a dog is actually a fake since the generator was supposed to generate a cat. After this relationship has been successfully learnt it is possible to control the class of the images that the generator will output.

This is more than just a neat utility, these class conditioned GANs often outperform unsupervised GANs, as witnessed in Table 5.1.

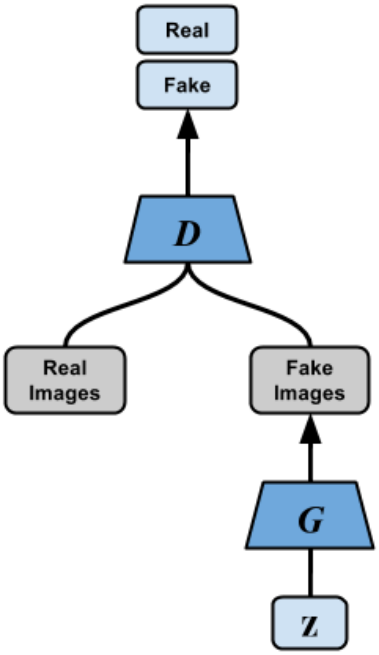


Figure 1.1: The GAN architecture proposed by Goodfellow *et al.* in 2014. The generator G transforms random noise $\mathbf{z}$ into fake samples. The discriminator D classifies samples as either real or fake.

1.0.4 Finding Good Representations

GANs typically have far fewer parameters than required to perfectly capture the data they are being trained on. For instance, they cannot memorise a whole training set of high-dimensional images without orders of magnitude more parameters than they are given. Instead, they are forced to capture the most pertinent characteristics of the data in generalisable representations. The data is then reconstructed by combining these

representations stochastically. This can be viewed as a form of lossy compression. For instance, a typical GAN trained on Imagenet might compress 200GB of pixels down to 100MB of weights.[1] The quality of these hidden representations and how efficiently they're used to generalise the data, determines the quality of the generated images.

Finding powerful representations, i.e. *Representation Learning*, is a central challenge of machine learning, not just for GANs. For deep neural networks, representations are captured in a hierarchical structure, deeper layers learn increasingly more abstract features.[2] Features that are too specific to the training data are not as useful, the model is effectively just memorising. If this happens we say the model has overfit. A classifier that has overfit will have poor performance on a test set, a generator that has overfit will generate samples that look nearly identical to samples in the training set. Capturing diverse and complex features motivates the design of deep and wide networks with lots of parameters. However, this runs the risk of overfitting as the networks are free to use their large capacity to simply memorise from training examples.

Many techniques for avoiding overfitting and for improving the generalisation power of learned representations have been discovered. ℓ_2 regularisation, otherwise known as weight decay[13], penalises large weights in the objective function. This encourages the network to learn representations with smaller weights that are assumed to be simpler and less specific. Dropout[14] randomly switches off a proportion of the activations with each pass during training, then at testing, uses them all. This forces the network to effectively learn an ensemble of sub-networks and stops weights from cooperating in memorisation of the training set. Dropconnect[15] generalises Dropout by switching off weights instead of activations. Data augmentation is a method of exploiting invariances in the dataset to artificially increase its size. For example, images that are horizontally mirrored should be classified the same for most tasks, so the dataset can be artificially doubled by flipping each image. This has a regularizing effect as the model learns more general features that can cope with variances in the data. This has recently been shown to be effective if the augmentation is applied on feature space, as opposed to the inputs[16].

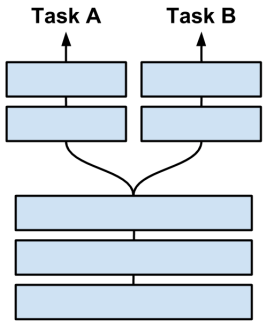
[1] Compression performance is an appealing way of quantifying intelligence. A good compressor needs to spot and capture regularities well, it needs to be "smart". Some believe that finding a compressor that can efficiently compress a snapshot of Wikipedia is our most promising route to artificial general intelligence, http://prize.hutter1.net/ (accessed, 13th August 2017).

[2] The terms *feature* and *representation* will be used interchangeably.

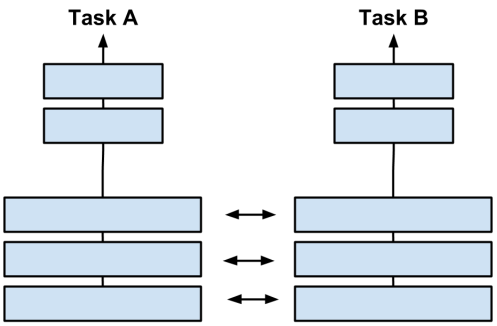
1.0.5 Multitask Learning

Another way to extract better hidden representations and avoid overfitting is through multitask learning (MTL). Most machine learning tasks are narrowly focused and trained on in isolation with a single task, we will call this *single task learning* (STL). MTL models on the other hand, train on multiple tasks in parallel. The general idea being that if there is some relatedness between the tasks then there could be a transfer of training signals that would benefit some or all of the tasks. For STL, if the labelling does not describe the important aspects of the data well enough, if the labels are too simple or broad, the training signals will not guide the learning to a good region in parameter space, i.e. the model will not learn good features. MTL can help fix that. Rich Caruana[4] describes MTL as: "[improving] generalization by leveraging the domain-specific information contained in the training signals of related tasks".

There are many different forms of MTL, the most common approach in neural networks is *hard-sharing*, first proposed by Caruana[4]. This is where a set number of layers are shared across multiple tasks, as seen in Figure 1.2a. It has been shown that the risk of overfitting in the shared layers is an order of N smaller than overfitting in the task-specific layers, where N is the number of tasks[17].



(a) **Hard-sharing**. The first three layers are shared, the final two layers are task-specific.



(b) **Soft-sharing**. Parameters in the first three layers are kept close by minimising the distance between them.

Figure 1.2: Different forms of parameter sharing across two tasks.

Another approach is *soft-sharing*, as depicted in Figure 1.2b. Here, the distance between the parameters of the tasks are regularised by way of penalisation terms in their respective loss functions. In hard-sharing the distance between parameters across tasks is kept at zero whereas soft-sharing admits other distances. In this light, soft-sharing can be seen as a generalisation of hard-sharing. Different regularisation schemes have

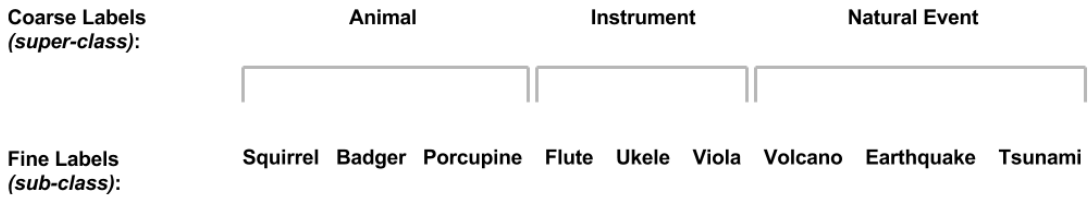


Figure 1.3: Example of class hierarchy

been proposed such as regularising the ℓ_2 norm[18] or trace norm[19], it is not clear to us when any one method should be preferred.

1.0.6 Multitask Learning in this Thesis

A central aim of this project is to implement multitask learning on GAN architectures and test whether performance gains can be made. We hypothesise that the same benefits as seen in MTL classification models will apply for GANs also, thus we explore MTL applied to classification before applying MTL to GANs.

Soft-sharing via regularisation of the ℓ_2 norm has been selected as the particular variety of MTL to be focused on, where possible. Soft-sharing has potential benefits over hard-sharing since it allows the network more flexibility in setting its weights, sometimes it may not be preferable to copy the other networks weights exactly. Hard-sharing does have the advantage that there are far fewer parameters in the overall model (half as many in the shared layers), this can reduce training time significantly. The classification experiments were not prohibitively slow however, so this extra training time was not a major concern. ℓ_2 norm regularisation was selected over trace norm regularisation since it is simpler to implement and interpret.

As for choosing which auxiliary tasks to employ, we decided to use an auxiliary task that classifies a coarser set of labels than the main task. This has proved effective in other work[20] and has the advantage that no extra data is required, it can be applied to any dataset with labels that can be divided into a super-class/sub-class hierarchy, like in Figure 1.3.

1.0.7 Which Dataset to Work with?

Since GANs have been most successfully applied to image generation, it makes sense to pick vision tasks for this project. Teterwak *et al.*[20] showed that two-task MTL classification using hard-sharing can be successfully applied to a variety of image

datasets: ImageNet[21], Caltech-256[22] and CIFAR-100[23]. In each case the *main* task was standard classification and the *auxiliary* task, was also classification, but over a superset of the original classes, i.e. a coarser set of labels. Based on their success, these three datasets are good candidates.

Caltech-256 suffers from class imbalance, with a minimum of 81 images/class, and is not popular in GAN-literature so we decided against using it. CIFAR-100 was chosen over Imagenet for a variety of reasons. Firstly, the images are small. GANs have a recognised problem when tasked with generating images of high resolution, where locally the images look good but globally the images are incoherent. This is why images of ten-legged, six-eyed dogs are not uncommon in the literature, e.g. [24]. Smaller images are also easier to compute with. Imagenet has higher, mixed resolutions and would require down sampling.

CIFAR-100 is also easier to work with than Imagenet. There is perfect class balance and all images have the same resolution. There is also a natural choice of auxiliary task labels, the set of 20 coarse labels the dataset comes with, along with the 100 finer labels.

A challenge of using this dataset is the lack of benchmarks in the GAN-literature. There are plenty for CIFAR-10, which is far easier to train GANs on given there are 10x as many examples for each class (another challenging aspect of CIFAR-100). This means we didn't know which models were going to work. GANs are very sensitive to their architecture, their hyperparameters and the task they're given. A GAN doing well on CIFAR-10 does not make any guarantees for CIFAR-100. For example, we found the Auxiliary Classifier GAN[5]s (AC-GANs) performance to suffer significantly when re-tasked from CIFAR-10 to CIFAR-100. Note, we couldn't simply side-step these challenges by working with CIFAR-10 since there is no sensible way of forming fine labels (for the main task) and coarse labels (for the auxiliary task). This would mean we couldn't implement the form of MTL used in [20].

Since there are no benchmarks or results for CIFAR-100, it is an open question as to what the state-of-the-art is and whether a GAN can be successfully class conditioned on this dataset.

1.0.8 The CIFAR-100 Dataset

The CIFAR dataset[23] contains 120,000 small images of resolution [32x32]. Each pixel is represented by a 3-digit RGB value, thus, each image is represented by a

[3x32x32] array of size $3 \times 32 \times 32 = 3072$ numbers. 60,000 of the images (referred to as CIFAR-10) are each labelled with one of 10 different classes, with 6000 images per class. The other 60,000 images (referred to as CIFAR-100) are each labelled with one of 100 different classes, this time with 600 images per class. The 100 classes are grouped into 20 super-classes, as shown in Figure 1.4. For this project, for classification experiments, CIFAR-10 and CIFAR-100 are each split into a training set of 40,000 images, a validation set of 10,000 images and a test set of 10,000 images. For GAN experiments, CIFAR-10 and CIFAR-100 are each split into a training set of 50,000 images and a test set of 10,000 images, as is standard in the literature. Although we primarily focussed on CIFAR-100, CIFAR-10 was occasionally used. As it turned out, the test set wasn't used for either case. Only validation accuracies were reported for classification experiments and the GAN experiments had no need for the test set. We didn't report test accuracies for the classification experiments since we weren't striving for anything close to state-of-the-art results, only for proof of concepts.

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figure 1.4: The hierarchy structure of the CIFAR-100 classes. Image courtesy of https://www.cs.toronto.edu/~kriz/cifar.html (accessed, 11th May 2017).

Considering MTL classification on the CIFAR dataset, with CIFAR-100 as the main task, there are two pairings of tasks that present themselves as clear candidates for 2-task MTL:

1. CIFAR-100 could be used exclusively, as done in Teterwak*et al.*[20], with the main task being 100-way classification and the auxiliary task being 20-way classification of the supersets.

2. CIFAR-100 with 100-way classification could be used as the main task and CIFAR-10 for the auxiliary task.

The 1st case is preferred over the 2nd, regardless of which performs better, since it does not require an additional dataset. The 2nd case will be considered briefly for classification but not at all for GANs (although this is an interesting avenue). There are many different ways to aggregate the 100 classes of CIFAR-100 into supersets, the 20 supersets given to us with the data are pretty arbitrary. Thus there are many different tasks we could conceive of for the auxiliary task. Investigating the effects of different types of supersets is out of the scope of this project but is touched upon by Teterwak *et al.*[20].

It is worth mentioning that the CIFAR-100 dataset is challenging. The 100 classes include objects as diverse as bicycles, kangaroos, forests and televisions. The images show a wide variability in shape, colour and scene dynamics that are not displayed in some datasets such as the LSUN bedroom set[25], the CUB-200 dataset of birds[26], the LFW dataset of faces[27] or CIFAR-10. Part of the issue lies in the fact that 32x32 images are severely limited in the amount of information they can carry, e.g. birds can be indistinguishable from planes at that resolution. CIFAR-10 doesn't suffer because of this since its 10 classes are all fairly distinct. The state-of-the-art in classification on CIFAR-100 is achieved by a model with 34.4M parameters, trained for 1600 epochs and has an error of 15.85%[28]. Compare this with the state-of-the-art error for CIFAR-10, 2.75%, achieved by a smaller version of the same model.

1.0.9 Contributions

This thesis makes the following main contributions:

1. We explore soft-sharing MTL for classification on CIFAR-100 with coarse label classification as an auxiliary task. We take a look at various hyperparameters: the soft-sharing hyperparameter λ, the number of shared layers b and the size of the dataset. We compare this auxiliary task with one that classifies CIFAR-10 and with a STL baseline. We find significant improvements over the baseline for both auxiliary tasks, in keeping with the results of [20] for hard-sharing. We also find hard-sharing to marginally outperform soft-sharing. To our knowledge this is the first time this kind of soft-sharing MTL has been investigated on CIFAR or any other vision dataset.

2. We show several GAN architectures that can successfully class condition on CIFAR-100, allowing control of the generator's outputs. We provide evidence that class conditioning upon an architecture called Wasserstein GAN with Gradient Penalisation[3] (WGAN-GP) as opposed to the original GAN[1] gives large performance gains, in terms of training stability and image quality but not in terms of convergence speed where it is found to be a lot slower. To our knowledge this the first time successful class conditioning of CIFAR-100, without mode collapse, has been reported.

3. We find that adding a second auxiliary task to the discriminator of an Auxiliary Classifier Wasserstein GAN (AC-WGAN) can significantly improve image quality. The first auxiliary task classifies 100 fine labels, as in the original architecture, and the second auxiliary task classifies 20 coarse labels. This gives ≈5 Fréchet inception distance (FID) improvement over normal AC-WGAN and ≈2 over Wasserstein GAN (WGAN) without class conditioning. We apply this MTL technique to a strong ResNet WGAN architecture that [3] trained to achieve very close to state-of-the-art on CIFAR-10 and find similar performance gains, setting a tentative state-of-the-art result for CIFAR-100. The resulting samples are recognisable and show diversity across all 100 classes with little sign of mode collapse.

1.0.10 Related Work

MTL hard-sharing was first formalised by Rich Caruana[4] (1997) and is now used extensively in a variety of forms throughout the deep learning field. Successful applications of MTL can be seen in speech recognition[29], vision[30], natural language processing[31] and drug discovery[32]. Most germane to our work is that of Odena *et al.* with AC-GAN[5] (to be discussed shortly) and that of Teterwak *et al.*[20] with their work on MTL hard-sharing with CIFAR-100, in which, they show the benefits of an auxiliary task that classifies coarser labels than the main task. They also explore the effect of various hyperparameters on hard-sharing. We do a similar analysis with MTL soft-sharing in Chapter 2. Soft-sharing is becoming a more common method of MTL, seeing success in applications such as dependency parsing[18] and facial recognition[19]. To our knowledge there isn't any pre-existing work on using soft-sharing with an auxiliary task that classifies coarse labels. In an overview of MTL in deep learning[33], Ruder advocates a paradigm shift in MTL away from hard-sharing,

which has dominated for two decades, towards more sophisticated methods such as soft-sharing. It should be mentioned that methods of MTL exist, which are much more sophisticated than our implementation of ℓ_2 soft-sharing, such as sluice networks[34] or cross stitch networks[35].

The GAN model was initially proposed by Goodfellow *et al.* in 2014[1]. This, and the flurry of architectures that quickly followed, was a breakthrough in deep generative models, achieving many state-of-the-art results in image generation. However, the original GAN architecture was extremely sensitive to its hyperparameters, making them notoriously difficult to train. Two years on (2016), Salimans *et al.*[36] showed a variety of techniques, e.g. feature matching, which could improve training stability and yielded strong results on CIFAR-10. This year (2017), perhaps the most significant step forward has been made since the seminal GAN paper with the conception of Wasserstein GAN (WGAN) by Arjovsky *et al.*[2].

WGAN minimises the Wasserstein distance, otherwise known as the Earth Mover's distance, between the generators distribution and the target distribution. This is in contrast with the original GAN, which minimises the Jenson-Shannon divergence. [2] shows that switching to this new distance provides much smoother gradients in the discriminator and has a dramatic improvement on training stability, although, the possible architectures to pick from are still limited. Quickly following the WGAN paper, Gulrajani *et al.*[3] showed how a small change to the WGAN architecture, a penalisation of the norm of the discriminator gradients instead of clipping them as done before, further improves training stability and now permits a wide array of architectures to choose from (providing they don't use batch norm in the discriminator). For example, with this architecture, WGAN-GP, it is possible to use very deep networks, Gulrajani *et al.* successfully used a 101-layer ResNet[37] in both the generator and discriminator. Along with increased stability, the WGAN-GP architecture boasts state-of-the-art results for CIFAR-10. Most of our models in this work are built upon WGAN-GP but use shallower and thinner networks than those used to achieve state-of-the-art.

Another relevant variant is Conditional GAN[38] (cGAN). In cGAN, the generator and discriminator are class conditioned by simply using the class vectors as additional inputs to the first layer of the generator and the last layer of the discriminator. If successful, the discriminator learns what each class should look like by training on the real images, this then guides the generator to generate images that are of the same class as its class vector inputs. cGAN and slight variants of, have proven to work well on a variety of applications. For example, [39] generates images conditioned on text, [40]

generates faces based on binary vectors of attributes like "sunglasses" or "moustache" and [41] generates stylised images conditioned on a style-map.

Auxiliary Classifier GAN[5] (AC-GAN) similarly achieves class conditioning but without using the class vector as an additional input to the discriminator. Instead, the discriminator has an auxiliary output that classifies the images, and so, AC-GAN incorporates a form of MTL. The softmax cross entropy of the loss of this output is then minimised by both the discriminator and generator, resulting in a class conditioned model with controllable output images. [5] boasted state-of-the-art (at the time) inception scores for CIFAR-10 with this architecture. This form of conditioning has become a standard way of class conditioning with GANs, it is how [3] achieve their state-of-the-art results with supervised WGAN-GP for instance.[3] To the best of our knowledge, there are no forms of AC-GAN that use a second auxiliary classifier like in our most successful model.

There are other GAN architectures that utilise some form of MTL. Before AC-GAN, Odena[42] and Salimans *et al.*[36] both (independently) adapted the original GAN discriminator to reconstruct class labels, creating Semi-Supervised GAN (SGAN). This can be seen as a type of MTL in the discriminator, with hard-parameter sharing in all layers but the last, where the main task is classifying the probability of a sample being fake and the auxiliary task is classifying the sample as one of N classes. The generators of these models were kept the same as the original GAN. For both cases SGAN was shown to increase image quality and data efficiency. Note, AC-GAN differs from these architectures in its generator, where it conditions upon class in the same way as cGAN.

InfoGAN[43] follows a similar architecture, except the auxiliary task in the discriminator instead reconstructs the latent variables from which the samples are generated and the variational information between the generator and the auxiliary task is regularised. This allows control of latent variables such as the rotation angle of a MNIST digit. A graphical display of these differences in architecture can be seen in Figure 1.5.

[3] At least that is what is claimed in the paper, the source code shows that they also use a conditional form of batch normalisation that the original AC-GAN did not use. This is discussed further in the *Results and Discussion* Chapter

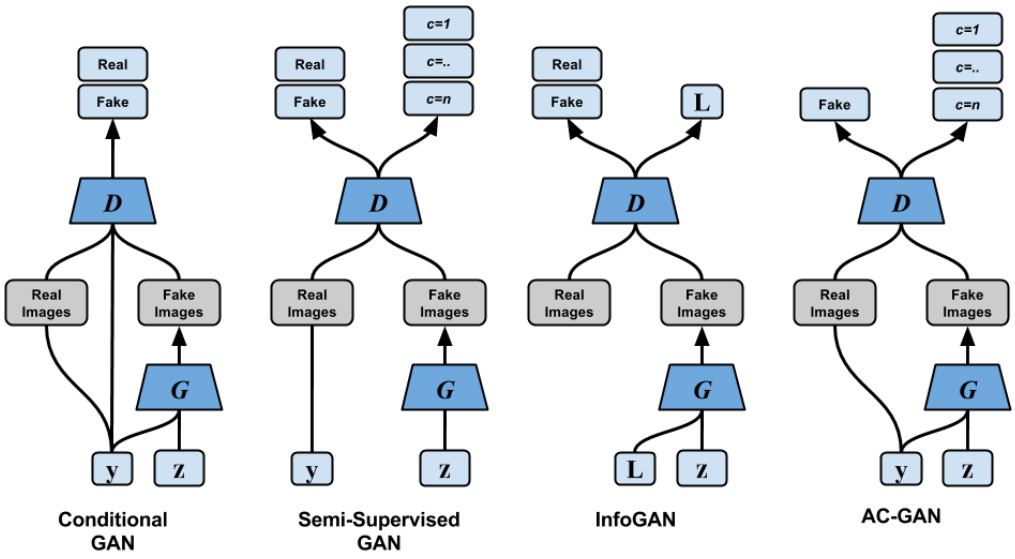


Figure 1.5: Comparison of some different class conditional GAN architectures. For: generator G, discriminator D, random noise **z**, class vector **y** and latent variables **L**. All models can control the class of their outputs, learning a conditional distribution for each class.

1.0.11 Overview of Thesis

The next chapter of this thesis, *Multitask Learning with Classification*, explores soft-sharing MTL for classification of CIFAR-100. Although classification is not our primary focus in this work, this section leads to some insights in hyperparameter selection and serves to show the feasibility of extending MTL to generative models. We also provide evidence that hard-sharing is preferable to soft-sharing for CIFAR-100.

In the third chapter, *Generative Adversarial Network Theory*, we move on to GANs, looking at their theoretical justifications and evaluation techniques. We give theoretical motivation for our choice in Wasserstein GAN (WGAN) as our base architecture, in particular WGAN with gradient penalisation (WGAN-GP). We also motivate our choice of primary evaluation, FID. Thereafter, we outline several methods of class conditioning, most of which involve forms of MTL.

The fourth chapter, *GAN Implementation Details*, puts forward and motivates the specific architectures that were experimented with. This includes 9 WGAN and 4 normal GAN architectures, all of which fall into one of four categories: baseline without class conditioning, e.g. WGAN-GP or normal GAN; simple conditioning by concatenation of the class vector, e.g. Conditional WGAN (cWGAN); conditioning by an auxiliary classifier in the discriminator, e.g. Auxiliary Classifier WGAN (AC-WGAN); conditioning by a separate classifier network, e.g. Separate Classifier WGAN (SC-WGAN). All architectures in the AC-WGAN and SC-WGAN categories rely on MTL, the others are STL models. We include these varieties without MTL as points of comparison.

The fifth chapter, *Results and Discussion*, reviews the results of our experiments. We look at each category in turn, finding many of the WGAN architectures to successfully condition on class without mode collapse. This is in stark contrast to the normal GAN architectures, which all suffer from severe mode collapse. This supports our theoretical justifications we laid out in the fourth chapter. We find the most promising results in the AC-WGAN category, where our proposal of an additional auxiliary task in the discriminator is found to boost performance significantly.

As a final experiment we extend Gulrajani *et al.*'s state-of-the-art ResNet WGAN architecture[3] with the same MTL technique and find significant improvements in FID but not in inception score. From qualitative analysis it is clear that samples from our proposed model display much more diversity. We hypothesise that the reason for the discrepancy in evaluation scores is that inception score does not pick up on image

diversity as well as FID does.

The last chapter, *Conclusion*, summarises and critiques the thesis and looks forward to possible continuations of this work.

Chapter 2

Multitask Learning with Classification

> Knowledge is power. Knowledge shared is power multiplied.
>
> Robert Noyce

Multitask learning (MTL) has several known mechanisms by which it works. These seem to have been first identified by Caruana[4].

- **Implicit Data Amplification:** MTL can effectively increase the size of the dataset. This can occur in a direct manner, if the tasks are being trained on different datasets. Or it can be indirect, if the tasks are being trained on the same dataset. In this case, data amplification occurs since the tasks have their own data-dependent noise associated with them. Extra tasks mean the model can learn representations that have less data-dependent noise. The effect is similar to adding more data.

- **Eavesdropping:** For some tasks, certain representations are easier to learn than they are for other tasks. This can depend on the quality of the labels, whether they can transmit good training signals through the model. In MTL, tasks can "eavesdrop" on other tasks to learn representations that they are struggling on.

- **Representation Bias:** Hidden representations that benefit one task can often benefit another. MTL biases the training to learn those representations that benefit multiple tasks. This increases the generalisation power of the model in facing new tasks.

- **Attention Focusing:** If data is scarce or noisy, it can be difficult for a model to discern between irrelevant and relevant representations. Training with extra tasks provides extra evidence as to which representations are actually useful.

Before applying MTL to a GAN architecture and running lots of very time-consuming experiments to explore their hyperparameter space, it is efficient to explore MTL applied to *classification* of CIFAR-100. This also serves to test the suitability of CIFAR as a dataset to perform MTL on.

2.0.1 MTL Classifier Implementation Details

All models were implemented in TensorFlow 1.0.0[44]. We emphasise that the goal of these models was to explore MTL, not achieve state-of-the-art accuracies. However, they were given enough capacity to perform reasonably well ($\approx 50\%$ on CIFAR-100) to make the results meaningful. Since we were not concerned with state-of-the-art we did not run test set evaluations or perform sophisticated statistical tests. We instead drew our conclusions solely on validation set learning curves for accuracy and error.

All models used a four layer convolutional neural network with dimensions of [64, 128, 128, 128] and single strided filters of 5x5 throughout. A standard softmax cross-entropy layer was used at the end. 2x2 max pooling was applied after the first, second and fourth layers. No fully connected layers were used as it was decided that the small performance gains they potentially offered were not worth the large compromise in training time.[1] Also, fully connected layers do not appear in a lot of GAN architectures and it is desirable to mirror those design choices as much as possible.

Weight decay was applied at a rate of 5e-5. Dropout[14] was applied with keep-probability 0.5 for all layers. Batch normalisation[46] was applied after each layer, this was found to be very important in getting the models to converge within a reasonable time. No data augmentation was implemented.

Training ran for 200 epochs. For simplicities sake, stochastic gradient descent (SGD) was used. The learning rate started at 0.01 then was annealed to 0.001 and 0.0001 at epochs 66 and 133 respectively. The batch size was 256.

Each task for MTL had exactly the same architecture except in the final softmax layer, which depended on the number of classes being classified. Soft-sharing of a set number of layers, see Figure 1.2b, was implemented via a simple ℓ_2 norm regularisa-

[1] Papers like [45] have demonstrated that they may not be necessary.

tion term, which was added to the loss functions of both tasks:

$$\lambda \left\| \mathcal{W}_{\text{main},b} - \mathcal{W}_{\text{aux},b} \right\|_2 \tag{2.1}$$

Where $\mathcal{W}_{\text{main},b}$ are the weights of the main network in the first b layers and $\mathcal{W}_{\text{aux},b}$ are the weights of the auxiliary network in the first b layers. λ and b are both hyperparameters that will be investigated. For all MTL models, the main task was 100-way classification of CIFAR-100 and the auxiliary task was 20-way classification on CIFAR-100s coarse labels, unless stated otherwise.

The training regime had the tasks train on batches in random ordering till each dataset had been passed through exactly once. At this point the data was shuffled, reset and a new epoch commenced. This meant that each epoch took twice as long to run as an epoch for an equivalent STL model.

To check the implementation of a model is sound, it is desirable to replicate a known result. Unfortunately, state-of-the-art on CIFAR is achieved by models with far more parameters than we can afford to have. One "good" baseline model was ran anyway, to check that the accuracies were in the right ballpark. This took a lot longer to train so couldn't be used for other experiments. This model was the same as the previous baseline, except widened to have dimensions [64, 128, 256, 512], data-augmentation (random cropping to 24x24 and horizontal flipping, applied to 25% of the data in each epoch) was performed and linear response normalisation was applied after the first two layers (using TensorFlow default parameters). This model got a maximum of 61.2% on the validation set but did not appear to have converged after 600,000 iterations, see Figure 2.1. Very little time was spent on the hyperparameter search since the objective here was not to achieve state-of-the-art. Another advantage the other models had is that they were all trained on 50,000 images from CIFAR-100 whereas this model was trained on 40,000, in keeping with our other experiments, as described in Section 1.0.8. For reference, the state-of-the-art as of 2016 can be seen in Table 2.1.

2.0.2 Comparing Auxiliary Tasks

We now compare the two auxiliary tasks proposed in Section 1.0.8. The first is 20-way classification of CIFAR-100, the second is 10-way classification of CIFAR-10. The networks followed the implementation details laid out in Section 2.0.1, with $b = 3$ shared layers using a soft-sharing value of $\lambda = 0.1$.

Model	**Accuracy**
Wide ResNet[47]	0.795
Let's keep it simple[48]	0.749
ResNet-110L[49]	0.748
Fractional Max-Pooling(12M)[50]	0.736
Scalable Bayesian Optimization Using Deep Neural Networks[51]	0.726
All you need is a good init[52]	0.723
ResNet-1202L[49]	0.721
Batch-normalized Max-out Network In Network(k=5)[53]	0.711
Highway[54]	0.678
Deeply Supervised Network[55]	0.654
FitNet[56]	0.650
Our Classifier	**0.612**

Table 2.1: Test set accuracies for state-of-the-art classification on CIFAR-100 as of 2016, according to [48].

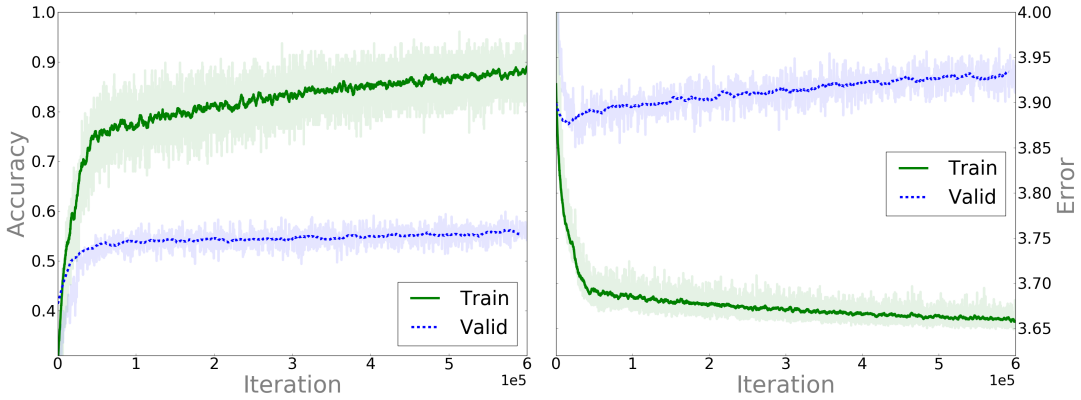


Figure 2.1: A "good" baseline model (STL) was trained as a sanity check to see if its performance is in the same ballpark as state-of-the-art models. Although the model doesn't seem to have fully converged after 600k iterations, it scores a maximum validation accuracy of 61.2%, which isn't too far from state-of-the-art in 2016, see Table 2.1. Considering it hasn't converged, is trained on a smaller dataset (40,000 images not 50,000) and has had little hyperparameter optimisation, this indicates that the model is likely sound.

From the results shown in Figure 2.2, it is clear that MTL improves generalisation power. It is not a surprise that using the additional dataset, CIFAR-10, in the auxiliary task is better than using the same dataset only with coarser labelling. The extra images of CIFAR-10 offer more useful training information to the main task, i.e. better training signals are passed via the shared layers, from which, better representations are learned.

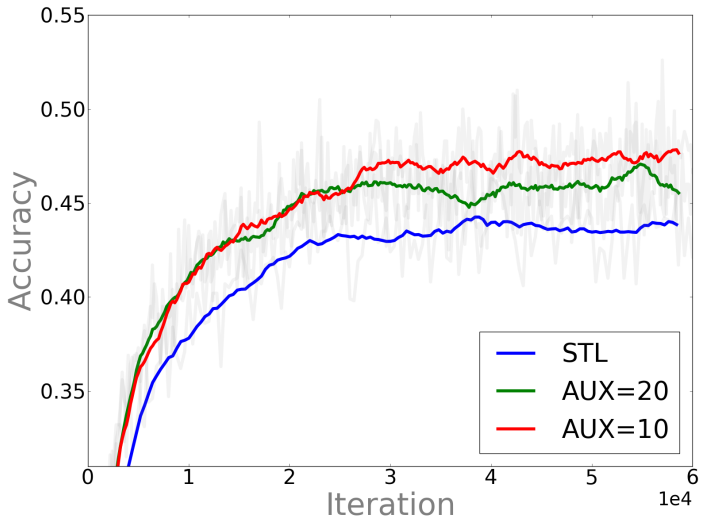


Figure 2.2: Here we compare the performance of two MTL auxiliary tasks, proposed in Section 1.0.8, against a STL baseline. AUX=20 corresponds to using a 20-way CIFAR-100 classification auxiliary task while AUX=10 corresponds to using 10-way CIFAR-10 classification as its auxiliary task. Accuracy scores are of the main task (100-way classification on CIFAR-100) measured on the validation set. Both forms of MTL improve performance with AUX=10 doing slightly better.

2.0.3 Comparing MTL Hyperparameters

We now turn to the effect of two hyperparameters: the value of b, i.e. the the number of layers to share information across, and the value of the soft-sharing hyperparameter λ. We also investigate the effect of reducing the dataset size. Unless otherwise stated, 100% of the data was used along with hyperparameter values of $b = 3$ and $\lambda = 0.1$.

2.0.4 Number of Shared Layers b

From Figure 2.3 we see that the number of shared layers b is significant. The validation *accuracies* show that $b = 2$ and $b = 3$ do equally well but the validation *errors* clearly show that $b = 3$ does better. This discrepancy comes from the fact that the accuracy

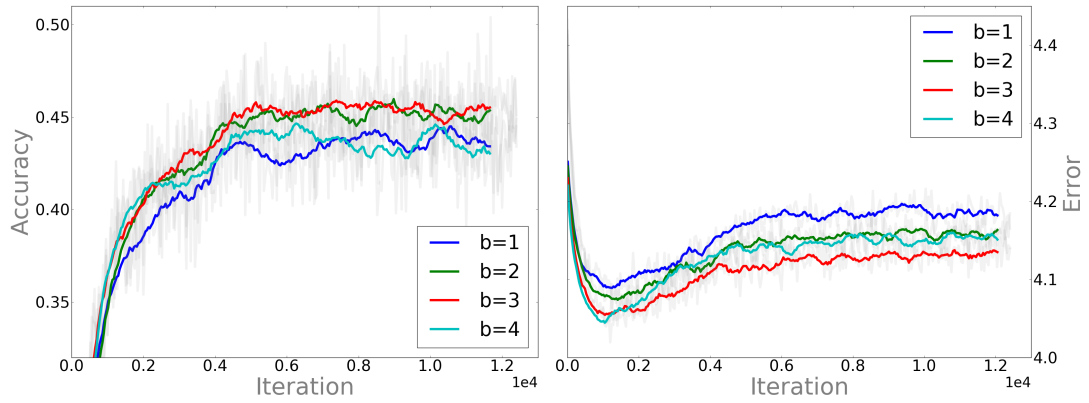


Figure 2.3: Validation set performance for different values of b, the number of shared layers. The cross-entropy errors show significant overfitting even as the validation accuracy gets better. This can occur as long as the predicted scores of the model don't cross the threshold where the predicted class changes. From a combination of accuracy and error performance, $b = 3$ was found to be best.

and softmax cross-entropy error are measuring quite different things. The accuracy metric is derived from a hard threshold of the softmax outputs of the final layer, it only considers which class scored highest. The softmax cross-entropy takes into consideration the outputs for all the classes, not just the one that happened to score highest and become the final prediction. The fact that they are measuring different quantities explains why it is possible for the error to get worse while the accuracy improves. However, the fact that the error quickly gets worse is a sign of overfitting and/or too high a learning rate, we found reducing the learning rate did help but at too much of a compromise to training time. Since we already had employed quite a few methods of regularisation (e.g. dropout and weight decay) and were testing out another method (soft-sharing MTL) we didn't feel the need to add further forms of regularisation.

2.0.5 Soft-sharing Hyperparameter λ

Also significant, the effects of the value of λ can be seen in Figure 2.4. $\lambda = 0.1$ was found to achieve the highest validation accuracy. The effect λ has on the distance between the respective model's weights is shown in Figure 2.5, where, as expected, higher λs correspond to closer weights. As the learning rate anneals, the approximative lower bound of how close the weights can be is also annealed, as witnessed by the sharp steps for the higher λs. For example, as the learning rate goes from 0.01 to 0.001 (a third of the way through training), this allows the $\lambda = 1$ models to refine their weights

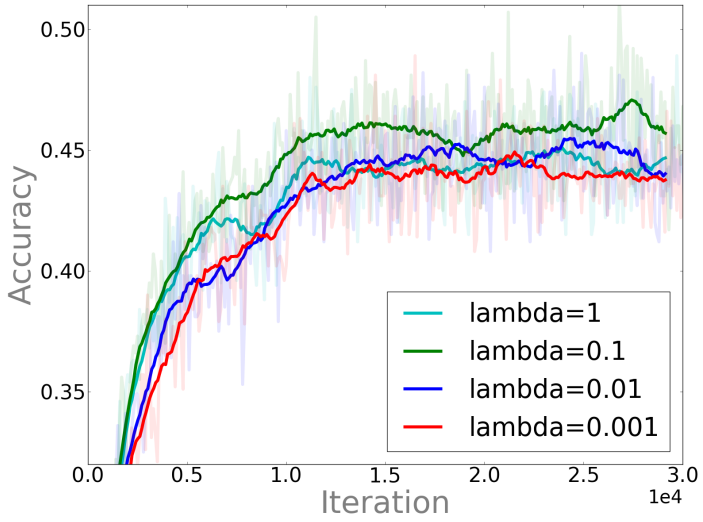


Figure 2.4: Validation set performance for a range of λ values. λ is a MTL-hyperparameter that controls the amount of soft-sharing that occurs, see Equation 2.1. There is $\approx 2\%$ range in accuracy for the values tested. $\lambda = 0.1$ clearly does best.

to be an order of magnitude closer together. It follows that, since we can never have a learning rate of 0, we can never have weights that are identical with soft-sharing. This is a potential drawback of soft-sharing and shows that soft-sharing is not a true generalisation of hard-sharing.

2.0.6 Size of Dataset

It is interesting to see if MTL is a benefit for reduced datasets. This informs us as to how data efficient MTL is in comparison to the baseline. To test this, the training data was randomly sampled to reduce it down to 75%, 50% and 25% of its original size. No effort was made to maintain an equal balance of classes, as is the case in the full dataset. The results can be viewed in Table 2.2.

We see that MTL improves upon the baseline for each dataset size, conferring similar increases in accuracy at each size (except 0.75 where STL and MTL both score 46%). This further bolsters the evidence that MTL is working.

2.0.7 Soft-Sharing Versus Hard-Sharing

As a final experiment for this chapter, we compared hard-sharing and soft-sharing. The results can be seen in Figure 2.6. The soft-sharing model used the best hyperparameters found, namely, $b = 3$ and $\lambda = 0.1$. The hard-sharing model bifurcated after the third

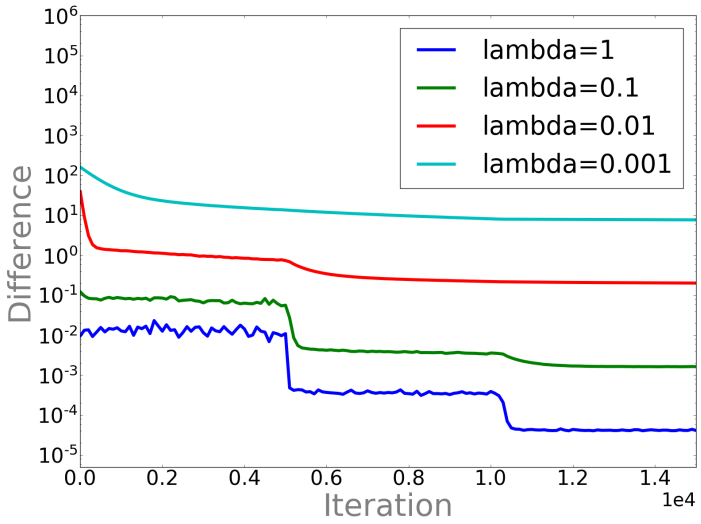


Figure 2.5: The l_2 norm of the difference between the parameters in the shared layers. That is, $\left\|\mathcal{W}_{\text{main},b} - \mathcal{W}_{\text{aux},b}\right\|$. This term is added to the losses of both tasks, as described in Section 2.0.1. It is clearly seen that increasing λ decreases the difference in weights and that the learning rate controls a lower bound of this difference, as evidenced by the sharp drops as the learning rate is annealed.

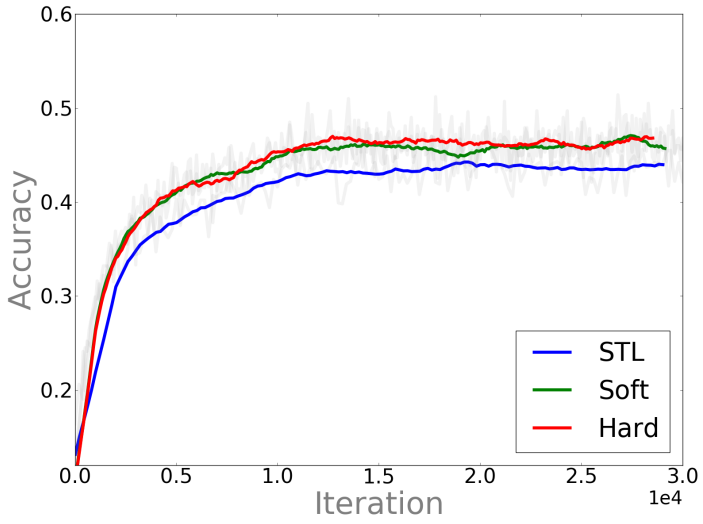


Figure 2.6: Validation set performance of hard-sharing against soft-sharing. The soft-sharing model uses $b = 3$ and $\lambda = 0.1$, the hard-sharing model shares across its first three layers. All other aspects of training are the same. While both forms of MTL do significantly better than the baseline, hard-sharing appears to do marginally better.

layer. All other hyperparameters and aspects of training were kept equal. From the validation performance, it appears that hard-sharing might be marginally better in this case. This was unexpected, since the higher values for λ were found to be detrimental to performance and as λ is increased you might expect the soft-sharing models to converge to the hard-sharing models. Although, as previously shown, soft-sharing does not quite generalise hard-sharing since it does not permit weights to be identical. It would be interesting to investigate this result further, for example, comparing the two forms of MTL for different values of b, λ and dataset size, to see if the outcome changes.

2.0.8 MTL Classification Conclusion

We have seen that soft-sharing MTL works well in reducing overfitting on CIFAR-100, with $\approx 4\%$ improvement over the baseline for AUX=20 (as seen in Table 2.2). However, the models used so far have been quite weak, and have overfitted significantly, as evidenced in the gap between the training and validation curves of our "good" baseline in Figure 2.1. It is unlikely that the benefits of MTL would be so significant alongside other forms of regularisation that also reduce this gap.

%	STL	MTL
100	0.47	0.51
75	0.46	0.46
50	0.41	0.44
25	0.35	0.36

Table 2.2: Maximum validation accuracies achieved for varying dataset sizes. MTL scores are from the main task where AUX=20 was the auxiliary task. Although comparing maximums is not a robust form of evaluation, soft-sharing MTL clearly increases generalisation performance.

Getting MTL working on classification bodes well for our hopes of incorporating it into a GAN. Ideally, the classification models would have mirrored the discriminators architecture as much as possible, e.g. in dimensions. This would make decisions of how to apply MTL to the discriminator more straightforward. Sadly, given the size of discriminator that we used, this would have been too much of a compromise in terms of computation time.

It would have also been preferable to do a wider search of the hyperparameter space, by a random search, which is more effective than a grid search[57]. A better set of MTL hyperparameters, i.e. b and λ values, in classification would help in guiding the selection of hyperparameters for MTL with GANs. Again, it was decided this

would take too long to implement and run.

Drawing conclusions from a single run of a model is dangerous since the random initialisation of its weights can sometimes play a significant part in what area of the error surface the training converges to. It is thus common practice to run the same experiment multiple times with different random seeds and report the median. However, most of the outcomes of this chapter were quite clear cut and its unlikely a different initialisation would have effected this. We decided that reporting the results of varying dataset sizes to be more worthwhile for showing that MTL was indeed working.

Chapter 3

Generative Adversarial Network Theory

> What I cannot create I do not understand.
>
> Richard Feynman

We now turn to the theoretical foundations of GANs, justifying the architectural decisions we have made. To be specific, we will explore the standard GAN, the Wasserstein GAN (WGAN), the conditional GAN (cGAN) and the Auxiliary Classifier GAN (AC-GAN). Motivation will be given as to why WGAN with gradient penalisation (WGAN-GP) was chosen as our base architecture.

We consider extending cGAN and AC-GAN to a Wasserstein model, yielding cWGAN and AC-WGAN. We also propose a new version of AC-WGAN where the discriminator is given a second auxiliary task (bringing it to a total of three tasks). Furthermore, we propose a new category of class conditional GAN, SC-WGAN and propose some variations therein. Two methods of evaluating GAN performance are also discussed, the *inception score* (IS) and the *Fréchet inception distance* (FID).

3.0.1 Before GANs: Maximum Likelihood Estimation

Previously, the most common way of creating a generative model was by *maximum likelihood estimation* (MLE). For a generative model p_θ with parameters θ (usually a deep neural network with weights), the MLE objective is,

$$\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log(p_\theta(\mathbf{x}^{(i)};\theta)) \tag{3.1}$$

Where $\mathbf{x}^{(i)}$ are samples from the dataset. Maximising this is equivalent to minimising the *Kullback-Leibler divergence*, denoted $\mathcal{D}_{KL}$, between the model's distribution p_θ and the target distribution of the data p_{data}.

As [2] points out, minimising the KL-divergence is only well defined if the model manifold (the space that the generated samples form) and the support of the true data distribution (subspace within the full data space where the data actually lies) have a non-negligible intersection. Otherwise the KL-divergence contains a division by zero and returns infinity, which is impossible to minimise. To get MLE to work, the typical solution is to add Gaussian noise to increase the models manifold, ensuring that there is an intersection between the distributions. It turns out that the amount of noise for this remedy to work is unavoidably large, enough to cause noticeable blurriness in generated images. This motivates an alternative approach.

A different tactic is to have a function $G(\mathbf{z})$ that transforms random noise $\mathbf{z}$ into samples, which look like that of the original data. This doesn't give us a direct approximation of the distribution p_{data} like with MLE but it does let us sample for free. To generate a sample we just plug a $\mathbf{z}$ into G, no Monte Carlo methods necessary. This is the approach GANs take. The difficult part is in how $G(\mathbf{z})$ is made to approximate the real data, this is where the adversary D plays its part.

3.0.2 The Original GAN

We will now look at a more formal definition of the original GAN architecture introduced by Goodfellow *et al.*[1].

We define the generator G as a transformation from a random variable $\mathbf{z}$ to the data space that we are training on. Here we are training on CIFAR images with 32x32 RGB values, giving us 32x32x3=3072 dimensions. Before passing through the model these pixels were scaled to lie in the range [-0.5, 0.5]. For the experiments ran here, $\mathbf{z}$ was drawn from a 128 dimensional multivariate normal distribution with zero mean and identity covariance. This is in keeping with most GAN papers we have read. We define discriminator D as a transformation from the data space to a number in the range [0, 1], the probability that an image is real. For instance, the discriminator outputs 0 for images it has 100% confidence are fake and 1 for images that it has 100% confidence are real.

The architecture is usually viewed from a game-theoretic perspective in which the training is viewed as a non-cooperative two-player game, the discriminator D versus

the generator G. More specifically, it is a mini-max game, whereby the discriminator is maximising a value function while the generator is minimising it. A solution to this objective is called a *Nash Equilibrium*. In practice, solving analytically for a Nash Equilibrium is intractable so the value function is trained on by approximative, gradient-descent procedures.[1] The original value function proposed was,

$$\min_D \max_G V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\Big[\log D(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}\Big[\log\Big(1-D(G(\mathbf{z}))\Big)\Big] \tag{3.2}$$

Where p_{data} is the probability distribution of the data and $p_{\mathbf{z}}$ is the probability distribution of the random variable $\mathbf{z}$. Looking at the behaviour of D and G in turn, we see that for D to minimise V it will try and assign values of 0 for real images, $\mathbf{x} \sim p_{\text{data}}$, since this will minimise $\log D(\mathbf{x})$. Conversely, D will try and assign 1 to the outputs of G for $\mathbf{z} \sim p_{\mathbf{z}}$ since that will minimise $\log\Big(1-D(G(\mathbf{z}))\Big)$. To maximise V, G will try and output values that D will assign to 0 since this maximises $\log\Big(1-D(G(\mathbf{z}))\Big)$.

Initial attempts to train this value function found that the gradients of $\log\Big(1-D(G(\mathbf{z}))\Big)$ would often vanish as the discriminator saturates. To get around this, it was discovered that it is better to *maximise* $\log D(G(\mathbf{z}))$ since the gradients are stronger[1]. The final training algorithm is outlined in Algorithm 1.

Goodfellow *et al.* went on to prove that G's output distribution p_G will converge to p_{data} via minimisation of the *Jenson-Shannon divergence*, denoted $\mathcal{D}_{JS}$, between the two distributions[1]. This is in contrast to MLE which directly minimises the $\mathcal{D}_{KL}$, as described in Section 3.0.1. However, this proof for G only holds given enough time and with a model of sufficient capacity. Thus, typical of deep learning papers, this proof doesn't give any guarantees for "real-world" neural networks.

The biggest drawback of the original GAN architecture is its instability in training and high sensitivity to hyperparameters. They are too sensitive to either the generator or discriminator dominating. This often leads to a *mode collapse* where the generator gets stuck producing the same image (from the same mode of p_G) for many values of $\mathbf{z}$. When this happens, the gradients of the discriminator aren't strong enough for the generator to recover. An intuitive approach to tackling this is to monitor the statistics of the loss functions of G and D then annealing the training of either that starts to win, or, train the loser repeatedly until it catches up. However, these approaches have shown

[1] Heusel *et al.* have proved that if the generator and discriminator are permitted to have different learning rates, along with some other weak conditions, then GANs and WGANs are guaranteed to converge to a Nash equilibrium using either SGD or Adam. They call this the two time-scale update rule (TTUR).

Algorithm 1 Training algorithm used for GAN.

Require: α, the learning rate. β_1, β_2, Adam parameters. m, batch size. w_0, initial discriminator parameters. θ_0, initial generator parameters.

procedure TRAIN GAN
  for $k = 0, ..., 200{,}000$ **do**
    for $i = 0, ..., m$ **do**
      Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
      Sample real images: $\mathbf{x} \sim p_{\text{data}}$
      $\mathcal{L}_D^{(i)} \leftarrow \log D_w(\mathbf{x}) + \log\Big(1 - D_w(G_\theta(\mathbf{z}))\Big)$
      Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
      $\mathcal{L}_G^{(i)} \leftarrow \log\Big(1 - D_w(G_\theta(\mathbf{z}))\Big)$
    $\mathcal{L}_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} L_D^{(i)}$
    $\mathcal{L}_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} L_G^{(i)}$
    $w \leftarrow \text{Adam}\Big(\nabla_w \mathcal{L}_D, \alpha, \beta_1, \beta_2\Big)$
    $\theta \leftarrow \text{Adam}\Big(\nabla_\theta \mathcal{L}_G, \alpha, \beta_1, \beta_2\Big)$

no sign of overcoming mode collapse. An example mode collapse is given by Figure 5.9.

3.0.3 Evaluating GAN Quality

Evaluating the performance of GANs is challenging. The loss functions of G and D for standard GANs do not relate to image quality, only to how well they are doing against their adversary. Furthermore, it is intractable to calculate precisely any useful kind of distance between the generators distribution and the real distribution that is being modelled. This would be the ideal evaluation. Instead, approximate approaches are taken.

Human evaluation is an intuitive option. However, getting a low-variance estimate of performance requires a large amount of samples which is expensive in terms of time and money. Additionally, as discovered by Salimans *et al.*[36], human evaluation is very sensitive to how the task is posed.

A cheaper and easier alternative, also discovered by Salimans *et al.*, is to use an *inception score* (IS). This relies on the inception model[58] to calculate the conditional class distribution $p(y|\mathbf{x})$ where y is the class and $\mathbf{x}$ is a generated image.[2] If meaningful

[2]We downloaded a pretrained inception model from http://download.tensorflow.org/

features are represented in the samples then the entropy of this distribution should be low, i.e. there should be little uncertainty in which classes are represented. Conversely, if the generated samples display a good amount of diversity then the marginal $\int p(y|\mathbf{x} = G(\mathbf{z}))d\mathbf{z}$ should have a high entropy. The inception score reflects these two entropy conditions by taking the KL-divergence of the two distributions,

$$\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathcal{D}_{KL}\left(p(\mathbf{y}|\mathbf{x})||p(\mathbf{y})\right)\right]\right) \tag{3.3}$$

Thus, a higher IS should correspond to higher quality images. It is exponentiated to make the differences between scores more apparent. This inception score has become the benchmark form of evaluation in GAN papers. For some reference of state-of-the-art inception scores see Table 5.1.

A related scoring method, only recently proposed (June 2017), is the *Fréchet inception distance* (FID). It takes into account the statistics of the real data[59] and compares them directly with the generated samples. This has been shown to pick up disturbances in sample quality better than the original inception score, see Figure 3.1. It works by passing both the generated samples $\mathbf{x}_G$ and the real data $\mathbf{x}_{\text{data}}$ through the first three layers of the inception model to retrieve an encoding of each in feature space. This yields two encodings $f(\mathbf{x}_G)$ and $f(\mathbf{x}_{\text{data}})$ that are assumed to be normally distributed, this is the weakest assumption since Gaussians have maximal entropy. If the generator's distribution is optimal, i.e. $p_G = p_{\text{data}}$, then these two Gaussians should be equal. By calculating the mean and covariance of the two encodings $f(\mathbf{x}_G)$ and $f(\mathbf{x}_{\text{data}})$ we can estimate these Gaussians. The FID is then defined to be the Fréchet distance between these two Gaussians, defined as,

$$\text{FID}(\mathbf{x}_G, \mathbf{x}_{\text{data}}) = \|\mathbf{m}_G - \mathbf{m}_{\text{data}}\|_2^2 + \text{Tr}(\mathbf{C}_G + \mathbf{C}_{\text{data}} - 2(\mathbf{C}_G\mathbf{C}_{\text{data}})^{\frac{1}{2}}) \tag{3.4}$$

Where $\mathbf{m}_G$, $\mathbf{m}_{\text{data}}$ are the means and $\mathbf{C}_G$, $\mathbf{C}_{\text{data}}$ the covariances of the two Gaussians. For GANs in this project, both FID and IS are calculated for all models. Although we expect that FID benchmarks will become the standard, it is useful to compare inception scores against previous results.

Another reason for trusting FID over IS, is how dataset dependent IS is. Let $\mathbf{x}_1$ be samples from one dataset $Data_1$ and let $\mathbf{x}_2$ be samples from another, $Data_2$. An ideal

`models/image/imagenet/inception-2015-12-05.tgz` (accessed, 3rd May 2017) and used OpenAI's code for calculating the inception score, https://github.com/openai/improved-gan/tree/master/inception_score (accessed, 3rd May 2017). Since that inception model is trained on 64x64 images, the generated samples were upsampled from 32x32 by bilinear interpolation before evaluating.

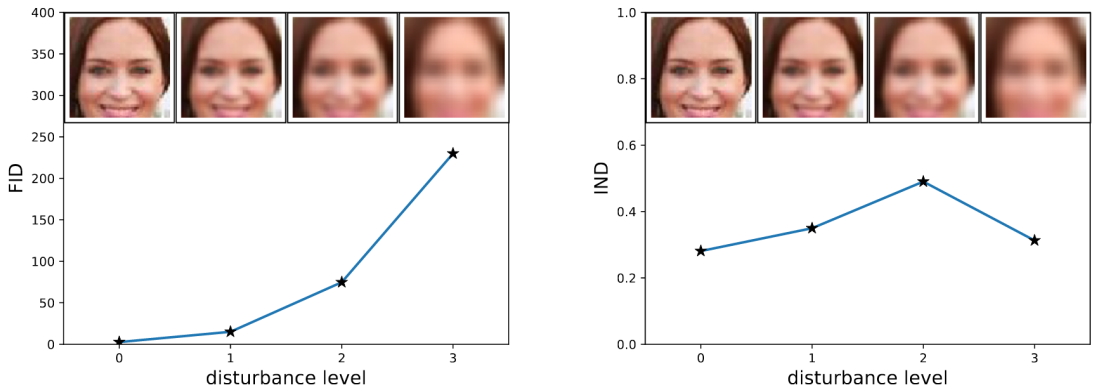


Figure 3.1: FID is the *Fréchet inception distance*, IND is equal to $m - inception\ score$ where m is the inception score's upper bound. Both values should rise as the picture gets more disturbed but the inception score fails badly on the last step while the FID detects the disturbance well. The disturbance here was caused by a Gaussian blur. Heusel *et al.*[59] go on to present similar results for 5 other types of disturbance. Image courtesy of [59].

evaluation metric $\mathcal{E} : \mathbf{x}, Data \rightarrow \mathbb{R}$ would have,

$$\mathcal{E}(\mathbf{x}_1, Data_1) = \mathcal{E}(\mathbf{x}_2, Data_2) \tag{3.5}$$

In other words, samples from one dataset should score the same as samples from another when compared to their respective datasets. In this way, the ideal metric would be agnostic of the dataset.

We experimented with this, finding that for 50,000 samples, CIFAR-10 scored 11.23 and CIFAR-100 scored 14.69 on IS. Contrast that to 5.85 and 5.29 on FID using 5000 samples. We see that inception scores can only be compared relative to a benchmark for a specific dataset whereas FID appear to be comparable across datasets. It would be interesting to test this out with other datasets, including some non-vision ones.

3.0.4 Wasserstein GAN

As mentioned, the standard GAN brings the distribution of the generator towards that of the real data by minimising the JS-divergence and MLE minimises the KL-divergence. The type of distance that is being minimised is important, it dictates how smooth the gradients are, and thus, which sequences will converge. The *Wasserstein GAN*[2] (WGAN) uses the *Wasserstein distance*, otherwise know as the Earth Mover's

distance.[3] Intuitively, the Wasserstein distance between two distributions is the least amount of work (mass $\times$ distance) required to shift one distribution into another. This is shown in Figure 3.3. Arjovsky *et al.* go on to show a simple example in which one distribution will not converge to another under any of the tested metrics (which included KL-divergence and JS-divergence) but does under the Wasserstein distance. Although the example was contrived, it gives some theoretical motivation for why it may be a better choice.

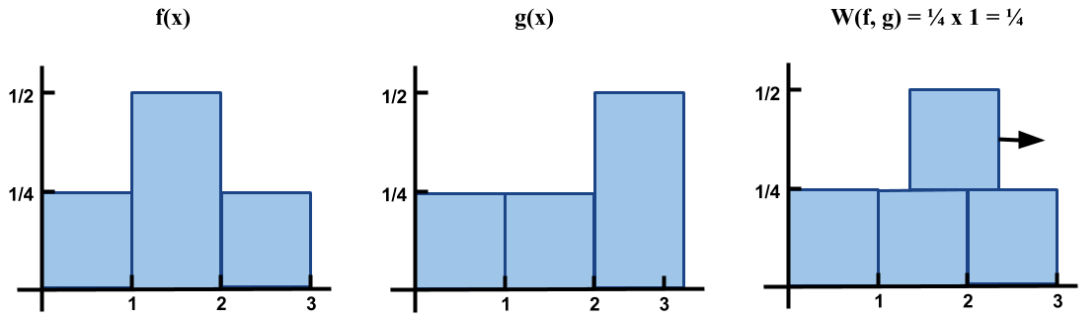


Figure 3.3: The Wasserstein distance measures the least amount of work (mass $\times$ distance) to shift one distribution into another. Here we see a simple discrete case where the least amount of work required to move f into g is to shift 1/4 of mass 1 unit along. Therefore, $W(f,g) = 1/4 \times 1 = 1/4$.

The formal definition of the Wasserstein distance between p_G and p_{data} is,

$$\mathrm{W}(p_G, p_{\text{data}}) = \inf_{\gamma \in \Pi(p_G, p_{\text{data}})} \mathbb{E}_{x,y\sim\gamma}||x-y|| \tag{3.6}$$

Where $\Pi(p_G, p_{\text{data}})$ is the set of all joint distributions with marginals $\int \gamma dx = p_G$ and $\int \gamma dy = p_{\text{data}}$. The inf symbol stands for the *infimum* and is a mathematical form of a minimum (and can be thought of as such).

However, computing the Wasserstein distance here is intractable. Intuitively this makes sense, the Wasserstein metric is the least amount of work required to transform between the distributions, finding this means searching through the infinite possible transformations that take p_G to p_{data}. Fortunately, an equivalence from analysis can help, a special case of Kantorovich-Rubinstein duality (a proof of which can can be found in appendix B),

$$\mathrm{W}(p_G, p_{\text{data}}) = \sup_{\|D\|_L \leq 1} \Big(\mathbb{E}_{\mathbf{x}\sim p_G}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] \Big) \tag{3.7}$$

[3] Care must be taken with the acronyms in this project as to whether we are dealing with a GAN or a **W**GAN.

Where in our case, x corresponds to images and $D(\mathbf{x})$ is the output of the discriminator. The sup symbol is the *supremum* and can for all practical purposes be thought of as the maximum. $\|D\|_L \leq 1$ is a restraint on D, requiring it to be *1-Lipschitz* or equivalently, that the magnitude of D's gradient may never exceed 1.

3.0.5 Enforcing the Lipschitz Constraint by Clipping Weights

Finding this supremum over all 1-Lipschitz functions is still intractable but now it is much easier to approximate. We can ensure a neural network is K-Lipschitz, i.e. $\|D\|_L \leq K$, for some unknown constant K, if we clip its weights to the range $[-c, c]$ for some known constant c. This doesn't tell us what K is but just that it exists. Clipping the weights like this doesn't quite solve our Lipschitz issue since we want the supremum over 1-Lipschitz functions not K-Lipschitz functions.

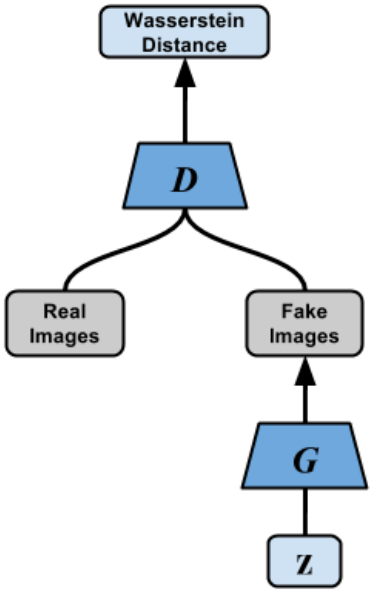


Figure 3.2: The WGAN architecture proposed by Arjovsky *et al.*[2].

However, since dividing a K-Lipschitz function by K gives a 1-Lipschitz function, and since equation 3.7 is linear, if we find the supremum over K-Lipschitz functions then we have found $K\mathrm{W}(p_G, p_{\text{data}})$. This scalar factor of K is unimportant in terms of optimisation, it just means the gradients are scaled by K. This can be accounted for by adjusting the learning rate, so K gets absorbed into hyperparameter tuning. Thus, it is enough to clip the weights of the discriminator by some c and converge it to an approximation of the supremum of D to give us a good approximation of the Wasserstein metric.[4] From this discussion, we can see an immediate issue with clipping the weights like this, it introduces a complex dependency between c and the optimisation hyperparameters.

We found that a useful way to think of the discriminators training objective is to imagine what would happen if the Lipschitz constraint was removed. This would allow the discriminator network to have an unbounded gradient. The training objective would encourage the discriminator to assign higher and higher values to the fake images and 0

[4] You may notice that the discriminator is no longer really discriminating, it outputs arbitrarily high outputs for the Wasserstein estimate, not probabilities between 0 and 1 like in the original GAN. Hence for WGANs, it is often referred to as a critic instead.

to the real images. The supremum of equation 3.7 would be unbounded and the training would never converge. By placing the constraint that the gradient of the function must be bounded by some K, the supremum of equation 3.7 is bounded and the training converges towards that bound. This training can not overfit, it'll just keep getting closer and closer to the true Wasserstein distance.

3.0.6 Deriving the Training Steps

To train G we wish to take the gradient of this Wasserstein metric with respect to the parameters of G, call them θ. This is equivalent to,

$$\begin{aligned}\nabla_\theta \mathrm{W}(p_G, p_{\text{data}}) &= \nabla_\theta \Big(\mathbb{E}_{\mathbf{x}\sim p_G}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] \Big) \\ &= -\mathbb{E}_{\mathbf{x}\sim p_G}[\nabla_\theta D(\mathbf{x})] \end{aligned} \tag{3.8}$$

This is just the gradient of equation 3.7. The second term disappears since it has no dependency on θ. Now, for $\mathbf{x} \sim p_G$ we have $\mathbf{x} = G(\mathbf{z})$ for some $\mathbf{z} \sim p_\mathbf{z}$, giving us,

$$\mathbb{E}_{\mathbf{x}\sim p_G}[D(\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[D(G(\mathbf{z}))] \tag{3.9}$$

Where $p_\mathbf{z}$ is the distribution of the random variable $\mathbf{z}$ (e.g. for experiments in this project $p_\mathbf{z}$ is a 128 dimensional Gaussian). Thus, equation 3.8 becomes,

$$\nabla_\theta \mathrm{W}(p_G, p_{\text{data}}) = -\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}[\nabla_\theta (D(G(\mathbf{z})))] \tag{3.10}$$

This gives us our desired gradients for training. The training process can be broken down into two steps:

- Approximate $\mathrm{W}(p_G, p_{\text{data}}) = -\mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}}\Big[\nabla_\theta (D(G(\mathbf{z})))\Big]$ by training D to convergence
- Update the generator's parameters θ by gradient descent with $-\nabla_\theta \mathrm{W}(p_G, p_{\text{data}})$.

WGAN offers increased training stability, [2] argues that this is due to the Wasserstein metric offering a higher level of continuity with respect to the generators parameters. However, optimisation is still not straightforward as the weight clipping value c interacts unpredictably with the cost function of the discriminator[3]. This often leads to vanishing or exploding gradients, especially in deep networks. These issues can sometimes be mitigated by batchnorm but not always[3]. Regardless, it is clearly preferable to have an architecture that is stable enough to work without batchnorm.

3.0.7 Enforcing the Lipschitz Constraint by Gradient Penalisation

Gulrajani *et al.*[3] proposed a new method of enforcing the K-Lipschitz condition called gradient penalisation (GP), yielding a slightly modified **WGAN-GP** architecture. Instead of clipping the weights to restrain D's gradient, a new term in the loss of the discriminator is introduced that directly penalises the norm of D's gradient. This term is softly enforced with a hyperparameter η.[5] The objective becomes,

$$L_D = \sup_{\|D\|_L \leq 1} \Big(\mathbb{E}_{\mathbf{x}\sim p_G}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[f(\mathbf{x})] \Big) + \eta \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}} \Big[\big\| \nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}} \big\|_2 - 1)^2 \Big] \tag{3.11}$$

Where the new variable $\hat{\mathbf{x}}$ is soon to be defined. The -1 in this new term encourages the gradient towards a norm of 1, which [3] showed to be the optimal gradient for the discriminator. Now, ideally the gradient norm would be restrained over D's entire loss surface, however this would require sampling $\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}$ over an intractably large area. Instead, [3] found it sufficient to only sample $\hat{\mathbf{x}}$ from straight lines between samples from p_G and samples from p_{data}. Intuitively this makes sense since it is only that part of the discriminators surface that is relevant if the model is successfully training and minimising the Wasserstein distance.

So D's objective is no longer just minimising the Wasserstein distance, as in equation 3.7, but is also keeping the gradient of D close to 1 with respect to samples on a straight line between points from p_G and p_{data}.

By removing the requirement of weight clipping, the complex interaction between c and D's loss is removed. Instead, the gradient is smoothly regularised, resulting in robust stability across a wide variety of architectures, including deep networks of 101-layers[3]. The training algorithm for WGAN-GP is given in Algorithm 2.

One nice feature of WGAN is that the discriminator's loss corresponds fairly well to image quality, as shown in [2]. This is because the discriminator's loss is an approximation of the Wasserstein distance between generated samples and real samples. This applies for WGAN-GP as well. However, such an evaluation is limited in its potential since you cannot readily compare the discriminator loss of one WGAN model to another (and certainly not to a different type of GAN). The discriminator loss is relative to that particular WGAN architecture and cannot be used as a comparison against other discriminator losses. It could be possible to consistently use a powerful, pre-trained WGAN discriminator for evaluation, similar to the inception score. We know of no efforts in this direction.

[5] The original paper actually calls the parameter λ but we've already used this notation for our soft-sharing hyperparameter.

Algorithm 2 Training algorithm used for WGAN-GP.

Require: α, learning rate. β_1, β_2, Adam parameters. m, batch size. n_D, number of iterations to train discriminator for one generator iteration. η, penalisation rate of discriminators gradient. w_0, initial discriminator parameters. θ_0, initial generator parameters.

procedure TRAIN WGAN
  for k = 0, ..., 200,000 **do**
    for t = 0, ..., n_D **do**
      for i = 0, ..., m **do**
        Sample noise: $\mathbf{z} \sim p_z$
        Sample real image: $\mathbf{x} \sim p_{\text{data}}$
        Generate image: $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z})$
        Sample number: $\varepsilon \sim U[0,1]$
        Calculate interpolate image: $\hat{\mathbf{x}} \leftarrow \varepsilon\mathbf{x} + (1-\varepsilon)\tilde{\mathbf{x}}$
        Calculate gradient penalisation term: $GP \leftarrow (||\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})||_2 - 1)^2$
        $\mathcal{L}_D^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) + D_w(\mathbf{x}) + \eta GP$
      $\mathcal{L}_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_D^{(i)}$
      $w \leftarrow \text{Adam}\Big(\nabla_w \mathcal{L}_D, w, \alpha, \beta_1, \beta_2\Big)$
    Sample m noise samples: $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p_{\mathbf{z}}$
    $\mathcal{L}_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} \log\Big(1 - D_w(G_\theta(\mathbf{z}^{(i)}))\Big)$
    $\theta \leftarrow \text{Adam}\Big(\nabla_\theta \mathcal{L}_G, \theta, \alpha, \beta_1, \beta_2\Big)$

3.0.8 Class Conditioning GANs

So far we have only looked at GANs which deal with unlabelled data in a purely unsupervised setting. It is possible however, to modify the architecture slightly and make use of the rich information captured in the labels, as mentioned in Section 1.0.3. Indeed, this is crucial to how we implement MTL to some of our proposed GANs, it allows the networks to differentiate between the two CIFAR-100 tasks. After all, the two tasks in question (discussed in Section 1.0.8) differ only in their labelling. So if we want any hope of incorporating this form of MTL into a GAN, we must use class conditioning.

3.0.9 Conditional GAN

The simplest way to class condition a GAN, introduced by [38], is to incorporate the class information in the first layer of the generator (before the convolutions) and the last layer of the discriminator (after the convolutions). This is done by concatenation of the real data's labels with a layer of the network's (sometimes actually with multiple layers, as done in [39]). Figure 1.5 shows a very high level comparison of the original conditional GAN[38] (cGAN) against other varieties of class conditioned GAN.

The hope is that the generator will learn modes over its input space that align well with the class information, giving us control over where the modes are formed. This enables us to control what type of samples are generated by G, we just have to feed in the corresponding class vector concatenated with random noise.

This is more than just convenient, it can significantly increase the quality of the samples being produced. The difference in performance can be witnessed in Table 5.1, where the difference between state-of-the-art unsupervised and supervised GANs is clear. Part of the reasoning for this boost in performance may be the fact that class conditioning forces the GAN to learn modes over all of the classes. Without class conditioning it is common for GANs to skip certain classes when generating samples, for example [60] showed an example of a GAN trained on MNIST generating 37 1s and only one 2 out of 100 samples.

3.0.10 Conditional WGAN

Does this form of conditioning work identically with WGAN and WGAN-GP as it does for cGAN? Nothing was found in the surrounding literature that addresses this.

Since the generator does not fundamentally change in a WGAN (same architecture and loss function), we can assume that it will learn the conditional distributions over the classes in the same way. The discriminator plays a very different role however. In the original GAN, the discriminator is truly discriminating. In a WGAN the discriminator is estimating the Wasserstein distance between generated samples and real samples, this estimate is then used to obtain gradients to train G. The outputs of D are no longer probabilities between 0 and 1 but are relatively unconstrained (there is only the Lipschitz constraint).

The new class information changes the gradient formulation (equation 3.10) to incorporate a joint distribution prior $p_{\mathbf{z},\mathbf{c}}$. Also, $D(\mathbf{x},\mathbf{c})$ and $G(\mathbf{x},\mathbf{c})$ now have extended domains. Equation 3.10 then becomes,

$$\nabla_\theta \mathrm{W}(p_G, p_{\text{data}}) = -\mathbb{E}_{\mathbf{z},\mathbf{c}\sim p_{\mathbf{z},\mathbf{c}}}\Big[\nabla_\theta D(G(\mathbf{z},\mathbf{c}),\mathbf{c})\Big] \tag{3.12}$$

The question becomes, does this additional class information help D estimate the Wasserstein distance better, giving better gradients for G? Or does it provide no benefit? In which case, $D(\mathbf{x}) \approx D(\mathbf{x},\mathbf{c})$. We could see no reason why this wouldn't work in the same way as it does in cGAN. We have omitted the training algorithm since it is so similar to that of WGAN-GP, Algorithm 2.

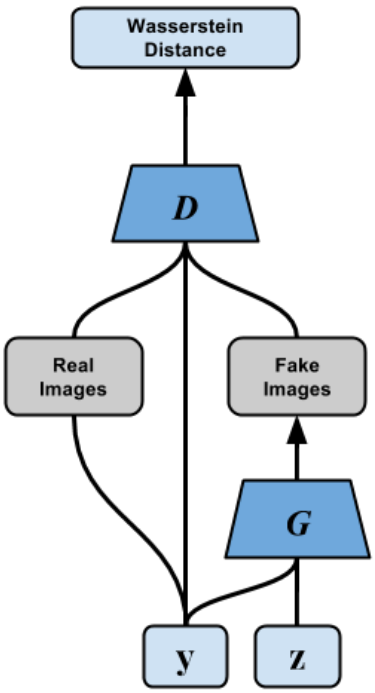


Figure 3.4: A Wasserstein extension of the Conditional GAN architecture proposed by Mirza *et al.*[38].

3.0.11 Auxiliary Classifier WGAN

The AC-GAN paper[5] does not propose a novel way of class conditioning but rather the combination of two pre-existing methods, namely, cGAN conditioning in the generator and SGAN[42][36] conditioning in the discriminator. We didn't have concerns that this architecture would fail for the Wasserstein case since [3] achieved a very strong IS on CIFAR-10 by doing something very similar. We now outline how the architecture works.

The discriminator trains to classify images (both real and generated) as well as estimate the Wasserstein distance between real and fake images like it usually would. To do this, the discriminator has two outputs, bifurcating in the final layer. The total

discriminator loss is the same as the original only with an additional classifier loss term. The classifier loss is a standard softmax cross-entropy loss that is calculated from both a batch of real and a batch of fake images. Since it is being calculated on two batches, it gets halved before combining with the discriminator loss. Likewise, the generator loss has the same classifier loss added on. Otherwise, the generator is conditioned upon in the same way as cWGAN. To clarify, the losses of the discriminator and generator are,

$$\mathcal{L}_D \leftarrow \mathcal{L}_D + (\mathcal{L}_{CR} + \mathcal{L}_{CF})/2$$
$$\mathcal{L}_G \leftarrow \mathcal{L}_G + (\mathcal{L}_{CR} + \mathcal{L}_{CF})/2$$

Where $\mathcal{L}_{CR}$ is the loss of the discriminator in classifying real images and $\mathcal{L}_{CF}$ is the loss of the discriminator in classifying fake images. All other aspects of the model are the same as the baseline. An outline of the full training regime is given by Algorithm 3.

The class conditioning occurs because the generator is encouraged to generate images that reduce the classifier loss, so it generates images that are easy to discern and hence match with its class vector input $\mathbf{y}$.

One issue with AC-WGAN is how to add the classifier loss to the generator and discriminator loss. Should the contributions be scaled differently? This might make sense since the discriminator is being trained 5 times for every one generator iteration, so the training may be more balanced if the classifier loss contribution to the discriminator is scaled by 0.2.

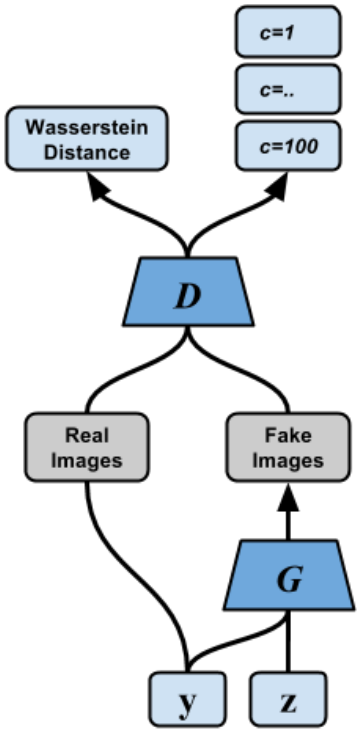


Figure 3.5: A Wasserstein extension of the Auxiliary Classifier GAN architecture proposed by Odena *et al.*[38].

3.0.12 MTL-based Extension to AC-WGAN

In light of our investigations into MTL, one interesting extension to AC-WGAN, is to add a third task to the discriminator, a second auxiliary task, which classifies the coarse labels of CIFAR-100. This provides extra hard-sharing between related tasks, so has the potential to confer MTL-benefits such as data amplification and eaves-dropping. Data amplification may be particularly useful in this case since there is a scarcity of

Algorithm 3 Training algorithm used for AC-WGAN.

Require: α, learning rate. β_1, β_2, Adam parameters. m, batch size. n_D, number of iterations to train discriminator for one generator iteration. η, penalisation rate of discriminators gradient. n_c, number of classes. ω, controls the contribution of the classifiers loss. w_0, initial discriminator parameters. θ_0, initial generator parameters. ϕ_0, initial classifier parameters.

procedure TRAIN AC-WGAN
  for $k = 0, ..., 200{,}000$ **do**
    for $t = 0, ..., n_D$ **do**
      for $i = 0, ..., m$ **do**
        Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
        Sample real image with class: $\mathbf{x}, \mathbf{c} \sim p_{\text{data}}$
        Generate image: $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z}, \mathbf{c})$
        Sample number: $\varepsilon \sim U[0,1]$
        Calculate interpolate image: $\hat{\mathbf{x}} \leftarrow \varepsilon\mathbf{x} + (1-\varepsilon)\tilde{\mathbf{x}}$
        Calculate gradient penalisation term: $GP \leftarrow (||\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})||_2 - 1)^2$
        $\mathcal{L}_D^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) + D_w(\mathbf{x}) + \eta GP$
        Classifier loss on real image: $\mathcal{L}_{CR}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\mathbf{x}), \mathbf{c})$
        Classifier loss on fake image: $\mathcal{L}_{CF}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\tilde{\mathbf{x}}), \mathbf{c})$
      $\mathcal{L}_{CR} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CR}^{(i)}$
      $\mathcal{L}_{CF} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CF}^{(i)}$
      $\mathcal{L}_C \leftarrow \left(\mathcal{L}_{CR} + \mathcal{L}_{CF}\right)/2$
      $\mathcal{L}_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_D^{(i)} + \omega\mathcal{L}_C$
      $w \leftarrow \text{Adam}\left(\nabla_w \mathcal{L}_D, w, \alpha, \beta_1, \beta_2\right)$
    for $i = 0, ..., m$ **do**
      Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
      Sample real image with class: $\mathbf{x}, \mathbf{c} \sim p_{\text{data}}$
      Generate image: $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z}, \mathbf{c})$
      Classifier loss on real image: $\mathcal{L}_{CR}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\mathbf{x}), \mathbf{c})$
      Classifier loss on fake image: $\mathcal{L}_{CF}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\tilde{\mathbf{x}}), \mathbf{c})$
    $\mathcal{L}_{CR} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CR}^{(i)}$
    $\mathcal{L}_{CF} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CF}^{(i)}$
    $\mathcal{L}_C \leftarrow \left(\mathcal{L}_{CR} + \mathcal{L}_{CF}\right)/2$
    Sample m noise samples: $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p_{\mathbf{z}}$
    Sample m classes: $\{\mathbf{c}^{(i)}\}_{i=1}^{m} \sim p_{\text{data}}$
    $\mathcal{L}_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D_w(G_\theta(\mathbf{z}^{(i)}, \mathbf{c}^{(i)}))\right)$
    $\mathcal{L}_G \leftarrow \mathcal{L}_G + \omega\mathcal{L}_C$
    $\theta \leftarrow \text{Adam}\left(\nabla_\theta \mathcal{L}_G, \theta, \alpha, \beta_1, \beta_2\right)$

data for each class in CIFAR-100. These benefits could
lead to better hidden representations in the discriminator, which would confer better Wasserstein gradients to the generator and allow it to generate higher quality images.

To update AC-WGAN as such, the losses become,

$$\mathcal{L}_D \leftarrow \mathcal{L}_D + (\mathcal{L}_{CRM} + \mathcal{L}_{CFM} + \mathcal{L}_{CRA} + \mathcal{L}_{CFA})/4$$
$$\mathcal{L}_G \leftarrow \mathcal{L}_G + (\mathcal{L}_{CRM} + \mathcal{L}_{CFM} + \mathcal{L}_{CRA} + \mathcal{L}_{CFA})/4$$

Where $\mathcal{L}_{CRM}$ and $\mathcal{L}_{CFM}$ are the classifier losses for the main task and $\mathcal{L}_{CRA}$ and $\mathcal{L}_{CFA}$ are the classifier losses for the auxiliary task.

3.0.13 Separate Classifier WGAN

Another form of class conditioning, which we couldn't find evidence for in the literature, is the same as AC-WGAN, only, instead of an auxiliary classifier in the discriminator there is a separate classifier network. This classifier trains to discriminate the classes of real images and fake and is trained with respect to its own parameters as well as those of the generators. We call this model the Separate Classifier WGAN (SC-WGAN). In the same way as AC-WGAN, class conditioning occurs as the generator is encouraged to generate images that minimise the classifiers loss. In this architecture, the discriminator is no longer being class conditioned on (although this would be an interesting variant that we didn't test) but the generator is. The concern of how often to train the classifier losses for the generator and discriminator remains.

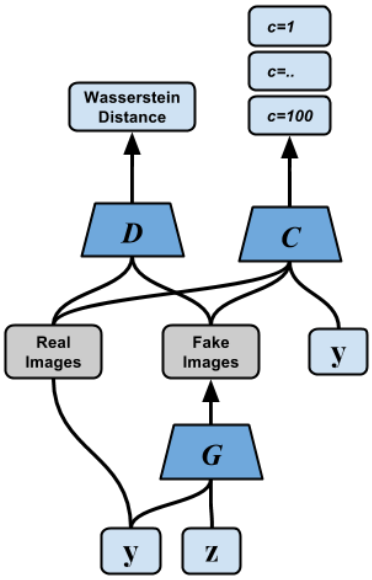


Figure 3.6: Our proposed Separate Classifier WGAN (SC-WGAN) architecture.

That there isn't any attention to SC-GANs in the literature might suggest they are outperformed by AC-GAN architectures. With an additional classifier network (or two), they are also slower to train. Still, we felt that they are an interesting form of class conditioning, especially for a Wasserstein discriminator.

For a normal AC-GAN, the discriminator is doing as the name suggests, discriminating, so adding an auxiliary task (or two) that is also discriminating seems like a

natural extension. However in the WGAN, the discriminator is no longer really discriminating, it's performing a regression of the Wasserstein distance. To extend it to also discriminate classes is a bigger leap than it is for AC-GAN. This can be either a good thing or a bad thing. For MTL we require tasks to be related but not too similar for optimal performance. How far from this "goldilocks zone" our auxiliary classifier task(s) lie with respect to estimating the Wasserstein distance is hard to gauge. If the task(s) are not related enough, SC-WGAN may prove to be a better option since the discriminator is no longer performing MTL. Hence, we hypothesised that this form of class conditioning may be especially good for the Wasserstein discriminator in particular.

The losses for SC-WGAN are,

$$\mathcal{L}_C \leftarrow (\mathcal{L}_{CR} + \mathcal{L}_{CF})/2.$$
$$\mathcal{L}_D \leftarrow \mathcal{L}_D$$
$$\mathcal{L}_G \leftarrow \mathcal{L}_G$$

The classifier loss $\mathcal{L}_C$ gets minimised with respect to the classifier's parameters ϕ and the generator's parameters θ but not the discriminator's. The algorithm is given in Algorithm 4.

3.0.14 Incorporating MTL into SC-WGAN

A variant of SC-WGAN that we tested was with soft-sharing MTL between the classifier and discriminator. Since the two networks are performing related but different tasks and learning similar but different features, they are good candidates for benefiting from MTL. As Chapter 2 explores, soft-sharing in this way is a sensible choice of applying MTL.

Another way of incorporating MTL into SC-WGAN is to have two separate classifiers, one classifying fine labels and one classifying coarse labels. This is a similar architecture to the extension of AC-WGAN mentioned earlier, with two auxiliary classifier tasks. A drawback of this form of SC-WGAN is that with 4 different neural networks, it is quite cumbersome to train.

3.0.15 Other Ideas for Incorporating MTL into a WGAN

There are many other conceivable variants of SC-WGAN (and AC-WGAN) that involve forms of MTL. The ones proposed are slightly arbitrary. One could also envision

Algorithm 4 Training algorithm used for SC-WGAN.

Require: α, learning rate. β_1, β_2, Adam parameters. m, batch size. n_D, number of iterations to train discriminator for one generator iteration. η, penalisation rate of discriminators gradient. s, classifier train frequency. n_c, number of classes. w_0, initial discriminator parameters. θ_0, initial generator parameters. ϕ_0, initial classifier parameters.

procedure TRAIN SC-WGAN
  for k = 0, ..., 200,000 **do**
    for t = 0, ..., n_D **do**
      for i = 0, ..., m **do**
        Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
        Sample real image with class: $\mathbf{x}, \mathbf{c} \sim p_{\text{data}}$
        Generate image: $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z}, \mathbf{c})$
        Sample number: $\varepsilon \sim U[0,1]$
        Calculate interpolate image: $\hat{\mathbf{x}} \leftarrow \varepsilon \mathbf{x} + (1-\varepsilon)\tilde{\mathbf{x}}$
        Calculate gradient penalisation term: $GP \leftarrow (||\nabla_{\tilde{\mathbf{x}}} D_w(\hat{\mathbf{x}})||_2 - 1)^2$
        $\mathcal{L}_D^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) + D_w(\mathbf{x}) + \eta GP$
      $\mathcal{L}_D \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_D^{(i)}$
      $w \leftarrow \text{Adam}\Big(\nabla_w \mathcal{L}_D, w, \alpha, \beta_1, \beta_2\Big)$
    Sample m noise samples: $\{\mathbf{z}^{(i)}\}_{i=1}^{m} \sim p_{\mathbf{z}}$
    Sample m classes: $\{\mathbf{c}^{(i)}\}_{i=1}^{m} \sim p_{\text{data}}$
    $\mathcal{L}_G \leftarrow \frac{1}{m}\sum_{i=1}^{m} \log\Big(1 - D_w(G_\theta(\mathbf{z}^{(i)}, \mathbf{c}^{(i)}))\Big)$
    $\theta \leftarrow \text{Adam}\Big(\nabla_\theta \mathcal{L}_G, \theta, \alpha, \beta_1, \beta_2\Big)$
    if k mod $s == 0$ **then**
      for i = 0, ..., m **do**
        Sample noise: $\mathbf{z} \sim p_{\mathbf{z}}$
        Sample real image with class: $\mathbf{x}, \mathbf{c} \sim p_{\text{data}}$
        Generate image: $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z}, \mathbf{c})$
        Classifier loss on real image: $\mathcal{L}_{CR}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\mathbf{x}), \mathbf{c})$
        Classifier loss on fake image: $\mathcal{L}_{CF}^{(i)} \leftarrow \text{CrossEntropy}(C_\phi(\tilde{\mathbf{x}}), \mathbf{c})$
      $\mathcal{L}_{CR} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CR}^{(i)}$
      $\mathcal{L}_{CF} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{CF}^{(i)}$
      $\mathcal{L}_C \leftarrow \Big(\mathcal{L}_{CR} + \mathcal{L}_{CF}\Big)/2$
      $\phi \leftarrow \text{Adam}\Big(\nabla_\phi \mathcal{L}_C, \phi, \alpha, \beta_1, \beta_2\Big)$
      $\theta \leftarrow \text{Adam}\Big(\nabla_\theta \mathcal{L}_C, \theta, \alpha, \beta_1, \beta_2\Big)$

MTL being employed in the generator by having two independent class conditioned GANs, one conditioned on coarse labels and one conditioned on fine labels. Then a MTL scheme like hard-sharing or soft-sharing could be set up between their respective generators and/or discriminators. However, training multiple GANs in parallel wasn't realistic within our time frame.

Chapter 4

GAN Implementation Details

Here we outline the technical details of the experiments that were run. The motivation for *why* we ran these experiments is established in the previous chapter.

All models shared the same hyperparameters wherever possible, these are displayed in Table 4.2. Weights were initialised according to a uniform distribution $U[-k,k]$ where $k = \frac{\sqrt{6}}{\sqrt{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}$ as recommended by Glorot *et al.*[61]. Where fan_{in} and fan_{out} are the number of connections coming in and out of the neuron being initialised. It would have been interesting to investigate the effects of different initialisations to the stability and performance of WGANs, especially in the discriminator where batchnorm wasn't used. We say this because batchnorm is supposed to reduce sensitivity to the initialisation scheme used.

Here we outline the 9 different WGAN architectures that were tested, one baseline WGAN and 8 with some form of conditioning. Of these 8, 6 included some form of MTL. Table 4.1 summaries the different forms of MTL and class conditioning used. All WGANs build upon the WGAN-GP architecture[3], for brevity the "-GP" is omitted from model names. All models follow the default hyperparameters mentioned in Tables 4.2 and 4.3 unless stated otherwise. The code was implemented in TensorFlow 1.0.0[44] and ran on a NVIDIA 1070 GPU.

4.0.1 Wasserstein GAN Implementation

The code for this baseline model is forked from one of the WGAN authors github repo.[1] It follows training Algorithm 2. As a sanity test, we ran this model on CIFAR-10 and found that it faithfully reproduced the expected inception score of ≈ 6[3].

[1] Martin Arjovsky's, https://github.com/martinarjovsky/WassersteinGAN (accessed 28th May, 2017).

Method	MTL	Classifier Type	Conditioning Location	Other
WGAN				
cWGAN.a			*G&D*	
cWGAN.b			*G&D*	Extra layer for class vector
AC-WGAN.a	✓	1 auxiliary	*G&D*	
AC-WGAN.b	✓	2 auxiliary	*G&D*	
AC-WGAN.c	✓	2 auxiliary	*G&D*	Bifurcation occurs a layer earlier
SC-WGAN.a	✓	1 separate	*G&C*	
SC-WGAN.b	✓	1 separate	*G&C*	Soft-sharing between C and D
SC-WGAN.c	✓	2 separate	*G&C*	

Table 4.1: Overview of different WGAN models implemented, showing the differences in how they were class conditioned. For generator G, discriminator D and classifier C.

Data	**Optimiser**	**Learning Rate**	β_2	β_2	**Iterations**	**Batch Size**
CIFAR-100 (50,000)	Adam	1e-4	0.5	0.9	2e5	64

Soft-sharing λ	D**-Iterations**	**GP** η	**z-Dim**	**Conv-Filters**	**LeakyReLU-**α	**Strides**
0.1	5	10	128	5×5	0.2	2×2

Table 4.2: Default hyperparameters that were used across all GAN experiments. β_1 and β_2 are hyperparameters for Adam, D-iterations denotes the number of times the discriminator optimiser is trained relative to one iteration of the generator, soft-sharing λ is the hyperparameter that controls the rate of soft-sharing, GP η is the parameter that controls the gradient-norm penalisation of D and **z**-dim is the dimension of the noise variable that is passed to the generator.

	Activations	1st Layer	BN	2nd Layer	BN	3rd Layer	BN	4th Layer	BN	Final Activation
G	ReLU	FC 4096	✓	DeConv 256	✓	DeConv 128	✓	DeConv 64		Tanh
D	LeakyReLU	Conv 64		Conv 128		Conv 256		FC 4096		None
C	LeakyReLU	Conv 64		Conv 128	✓	Conv 256	✓	FC 4096		Softmax

Table 4.3: Network dimensions of all GANs unless specified, for generator **G**, discriminator **D** and classifier **C**. Not all models use classifiers. **BN** stands for batch normalisation, or batchnorm. Note, DeConv layers aren't technically deconvolutional layers, they are upsample convolutional layers (convolutional layers that increase the image size).

4.0.2 Conditional WGAN Implementation

cWGAN.a is a WGAN with simple conditioning in the generator and discriminator, as done in [38]. Class vector **y** is concatenated with the noise vector **z** before passing to the generator (before the upsample convolutional layers). Similarly, **y** is concatenated with the discriminator's penultimate layer before the output layer (after the convolutional layers). All other hyperparameters are default.

cWGAN.b is exactly the same as cWGAN.a except in the discriminator, where the **y** vector is passed through a fully connected layer with 256 units before concatenating. This to test whether the model benefits from more interaction between the conditional information in **y** and the dense codes within the network. Both models otherwise follow the WGAN training Algorithm 2.

4.0.3 Auxiliary Classifier WGAN Implementation

AC-WGAN.a is an extension of AC-GAN[5] to the Wasserstein architecture. This is not a novel extension, [3] use an AC-WGAN with a ResNet[37] architecture in both the generator and the discriminator to achieve (very close to) state-of-the-art results on CIFAR-10. AC-WGAN in [3] uses two different hyperparameters for the classifier loss contribution. They used rates of 1 and 0.1 for the generator and discriminator respectively. We decided to keep this choice as simple as possible, leading us to choose 1 for both. This is a clear avenue of future investigation. An outline of the full training regime of AC-WGAN is given by Algorithm 3

AC-WGAN.b is the same but with two auxiliary classifier outputs in the discriminator, one for 100-way classification as in AC-WGAN.a and one for 20-way classification. The losses are combined in a linear fashion, as outlined in Section 3.0.12. All other details remain consistent with AC-WGAN.a.

AC-WGAN.c is the same as AC-WGAN.b only the bifurcation point isn't in the final layer like for the other two architectures, but in the second last last layer. This means each task has a dedicated convolutional layer. This allows the final convolutional layers more capacity to focus on their respective tasks but could reduce MTL benefits since less parameters are shared. It also hurts computation time since it is effectively adding two extra 256-dimensional convolutional layers.

4.0.4 Separate Classifier WGAN Implementation

SC-WGAN.a is similar to AC-WGAN but classification of the images is carried out through a separate network, as opposed to being an auxiliary task of the discriminator. The discriminator is the same as in the baseline WGAN and the generator is conditioned upon in the same way as cWGAN. The classifier is trained every s^{th} iteration, where s is a hyperparameter to be tuned. In this project we used $s = 2$ and did not investigate further. The classifier is trained to classify fake and real images using a standard cross-entropy loss. Also every s^{th} iteration, the generator's parameters are trained with respect to the classifier's loss on fake images. In other words, the generator updates its parameters so that the images it generates are more discernible to the classifier. An outline of the training regime is given by Algorithm 4.

SC-WGAN.b is the same as SC-WGAN.a only with soft-sharing between the first two layers of the discriminator and classifier with $\lambda = 0.1$.

SC-WGAN.c is the same as SC-WGAN.a only with two separate classifiers. One classifier is the same as in SC-WGAN.a, the other classifies the images with the coarser 20-way labelling of CIFAR-100. The training is alternated with equal probability between the two classifiers.

4.0.5 Normal GAN Implementation

To help understand the effect of the Wasserstein loss, a variety of normal GANs were tested as a comparison. To switch from a WGAN to a normal GAN (in line with the advice given by [62]), the following steps were taken:

1. $L_G \leftarrow \text{CrossEntropy}\Big(D(G(\mathbf{z})), 1\Big)$
2. $L_D \leftarrow \text{CrossEntropy}\Big(D(G(\mathbf{z})), 0\Big) + \text{CrossEntropy}\Big(D(\mathbf{x}), 1\Big)$
3. D-Iterations $\leftarrow 1$

4. Discriminator given a final layer sigmoid activation

5. Batchnorm added to each layer of the discriminator

For image $\mathbf{x} \sim p_{\text{data}}$ and noise $\mathbf{z} \sim p_{\mathbf{z}}$. All hyperparameters were otherwise kept the same.

The following models were tested (also shown are the corresponding WGANs they were converted from):

1. **GAN** (WGAN)

2. **cGAN.a** (cWGAN.a)

3. **AC-GAN.a** (AC-WGAN.a)

4. **SC-GAN.a** (SC-WGAN.a)

Note, these experiments cannot really hope to show conclusively that for class conditioning on CIFAR-100, WGANs are better (or worse) than GANs. This is since there has been no attempt to tune hyperparameters. Models should really be compared best against best. Or, models can be compared in terms of stability, after being tested with a wide variety of hyperparameters. Both of these types of comparison require lots of hyperparameter tuning and therefore lots of experiments. The experiments here can only provide limited evidence towards the stability and performance of the different architectures. This is a point of complexity of GAN-research in general. Models that offer state-of-the-art results can be harder to train (e.g. AC-GAN), models that offer increased stability can perform worse (e.g. the original WGAN). Both varieties are often interesting and important in their own right.

Since there appears to be nothing in the GAN-literature on CIFAR-100 it is interesting to see how the strong performers on CIFAR-10 do when re-tasked to CIFAR-100. Ideally all of the top, state-of-the-art GANs would be reproduced then tested on CIFAR-100. This is clearly out of the scope of this project, however, we decided we had time to try it for one model. AC-GAN[5] was the natural choice since it would allow comparison with our AC-WGANs. Note, the AC-GAN from [5] has quite a different architecture from our AC-GAN.a. The original AC-GAN sees the following main changes (a complete list can be found in Appendix A of [5]):

1. A thinner Generator

2. Three extra convolutional layers in discriminator

3. Three convolutional layers with stride 1x1 in discriminator
4. No final FC layer in discriminator
5. Kernel of 3x3 for all convolutional layers in discriminator
6. Dropout in discriminator with keep probability of 0.5
7. Activation noise at all layers (adding Gaussian noise to the outputs of activations as recommended in [36])

We call our reproduction, **AC-GAN.b**.

Unfortunately, the authors did not include the full list of hyperparameters. They specified three possible learning rates for the generator, three possible learning rates for the discriminator and three possible activation noise standard deviations. They then stated that they achieved their (then) state-of-the-art results by doing a grid search over the 27 possible combinations of these hyperparameters. Without the exact hyperparameters or source code, this makes the model hard to reproduce. Without running lots of experiments, if an attempted reproduction comes out with a negative result it is hard to ascertain whether this is due to some slight deviation in our code or a slight difference in hyperparameters. This issue is especially pertinent when dealing with original, non-Wasserstein GANs since they are notoriously sensitive to hyperparameters and suffer badly from mode collapses (where lots of points in $\mathbf{z}$-space all generate the same image).

4.0.6 Calculating FID and IS

Both the FID and IS both require loading the Inception graph at runtime. The FID also requires loading and calculating statistics of the true data at runtime. Because both forms of evaluation take fairly long ($\approx$ 3 mins), during training, the IS and FID were calculated every 2000 iterations. Both were calculated using 5000 generated samples. This is the recommended amount for FID according to [59] but [36] recommend 50,000 for IS. We decided that this would have taken too much compute time given as it is our secondary form of evaluation. It is also the same sample size as [3] use in their source code.

Chapter 5

Results and Discussion

Here we discuss the results of the models outlined in Chapter 3 and 4. We also provide evidence that the networks are not memorising training examples but are learning generalisable features of the data. To understand the impact that class conditioning has on image quality we compare the models to a baseline WGAN without class conditioning. We look at each category of GAN in turn and then do a comparison of the best from each category. The categories are cWGAN, AC-WGAN and SC-WGAN. The FID was found to be far less noisy and more consistent than the IS. This is evidenced by the faint grey lines in the graphs that indicate the true, unsmoothed data.[1] For this reason and others pointed out in Section 3.0.3, we draw our conclusions from the FID and use the IS for supporting evidence.

For conciseness, we have omitted a complete set of generated images from this section and direct the reader to appendix A. A high level overview of the WGAN architectures, along with samples of the first 10 classes can be found there. It is from samples like these that we checked to see if the models had successfully class conditioned. This method of checking might not seem a robust form of evaluation since the classes are difficult to identify in nearly all cases. However, as the difference between Figures 5.1 and 5 demonstrates, successful class conditioning turned out to be easy to identify. We do concede the possibility that some classes might have been partly conditioned, or conditioned poorly, and we failed to notice. We do not add images of the GAN samples as they were generally very poor and not interesting.

Unfortunately we didn't have time to train all of our models to convergence. The two we chose to train for an extra 2e5 iterations were: the WGAN baseline and the

[1] Smoothing was performed by a simple convolution. The moving window size was 10 for FID and 20 for IS.

Figure 5.1: Samples generated from AC-WGAN.a where class conditioning succeeded. Each row clearly contains a separate class, even if those classes are hard to identify.

second strongest of our class conditioned models, AC-WGAN.b. The reason we didn't choose AC-WGAN.c, our strongest, was because it took substantially longer to train and performed only marginally better than AC-WGAN.b.

5.0.1 cWGAN Results

Both varieties of cWGAN failed to successfully class condition and the quality of images was slightly worse than the baseline according to FID and IS. This method has proved successful for simpler tasks like MNIST[38] but it evidently does not scale well to a task like CIFAR-100. There are many other variants of cWGAN that we could have

Figure 5.2: Samples generated from cWGAN.a where class conditioning failed. Each row should be of the same class but here there doesn't appear to be any such grouping. For instance, the top row should be images of apples.

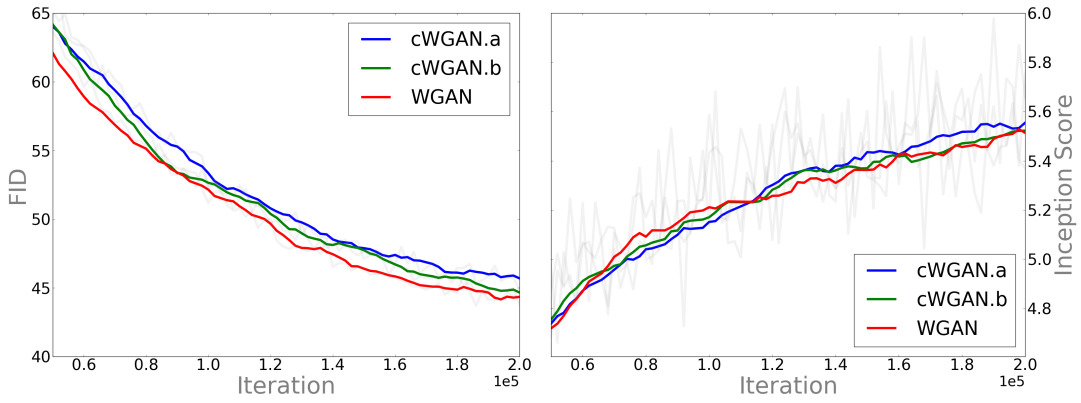


Figure 5.3: FID and IS results for cWGAN.a and cWGAN.b against a WGAN baseline. There was no qualitative evidence of successful class conditioning, as witnessed in Appendix A. The FID results indicate a slight detriment in performance of both cWGAN models.

tried with more time. For example it might have worked if we'd concatenated every layer with class information.

5.0.2 Auxiliary Classifier WGAN Results

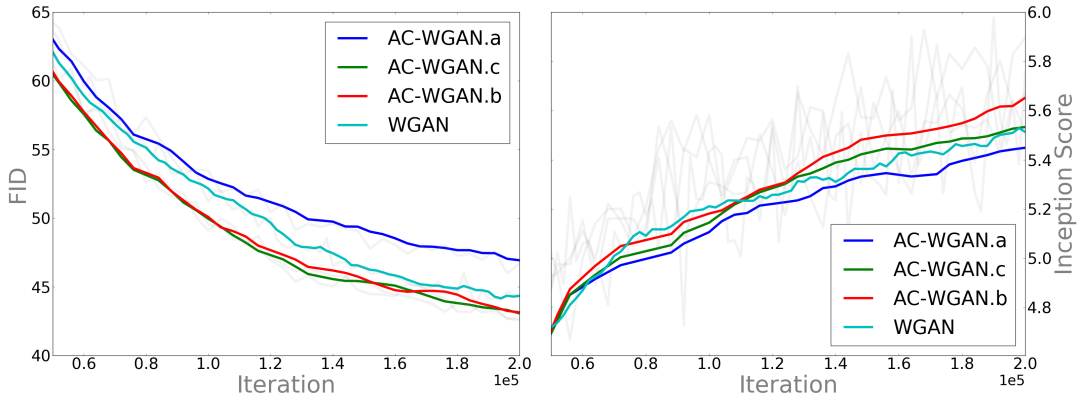


Figure 5.4: FID and IS results for AC-WGAN.a, AC-WGAN.b and AC-WGAN.c against a WGAN baseline. All AC-WGAN models were successful in conditioning on class, as witnessed in Appendix A. Both FID and IS indicate that our proposed MTL variants, AC-WGAN.b and AC-WGAN.c, generate better images.

All varieties of AC-WGAN succeeded in class conditioning. AC-WGAN.a, the most simple Wasserstein version of AC-GAN[5], was outperformed by the baseline by a margin of ≈3 FID and ≈0.1 IS. This is in contrast to [3]'s CIFAR-10 results,

which showed a WGAN-GP with AC-GAN conditioning outperform their WGAN-GP baseline significantly (by ≈0.6). A few important differences might explain this.

Firstly, CIFAR-10 is much easier to train and condition on, there are 10x the images per class and there are a tenth as many classes to condition on and learn distributions over. Odena *et al.*[5] highlight that this is much more difficult. To get round the issue for a 1000-class Imagenet dataset, instead of training one network on 1000 classes, they trained 100 on 10, reporting that a reduction in classes/model was crucial in escaping mode collapse.

Another difference our AC-WGANs had with [3] is that both their discriminator and generator were relatively large ResNets[37] whereas we had relatively simple convolutional networks. Thirdly, the type of batch normalisation they used was a conditional variant of batchnorm called conditional instance normalisation, described in Appendix C.[2]

AC-WGAN.b and AC-WGAN.c on the other hand, outperformed AC-WGAN.a and WGAN, as evidenced by both IS and FID. Clearly the addition of the extra auxiliary task helped, presumably for the same reasons MTL worked on classification in Chapter 2. From the IS, it appears that AC-WGAN.c did marginally better than AC-WGAN.b, although the FID does not support this so we are not drawing that conclusion.

5.0.3 Separate Classifier WGAN Results

Two varieties of SC-WGAN succeeded in class conditioning, SC-WGAN.a and SC-WGAN.b. However, we couldn't get SC-WGAN.c to work, with training breaking down around the 1e5 mark, as seen in Figure 5.6. Although the other two models did converge, they were outperformed by the baseline, see Figure 5.5. The poorer performance here could be down to poor hyperparameter choice, especially in how we balanced the classifiers training, where we had little intuition to guide us. It also might be partly because the generator for SC-WGAN.a and SC-WGAN.b is receiving gradient updates from an extra network than AC-WGAN (an extra two networks for SC-WGAN.c). These extra networks will increase the level of total noise in the gradient updates, which could lead the generator to a poor area of parameter space. This is what motivated the design of SC-WGAN.b, to keep the classifier and discriminator parameters close to one another which would perhaps reduce the noise of the gradi-

[2]This switch in normalisation scheme is not outlined in their paper but is included in their released code https://github.com/igul222/improved_wgan_training (accessed 5th August 2017).

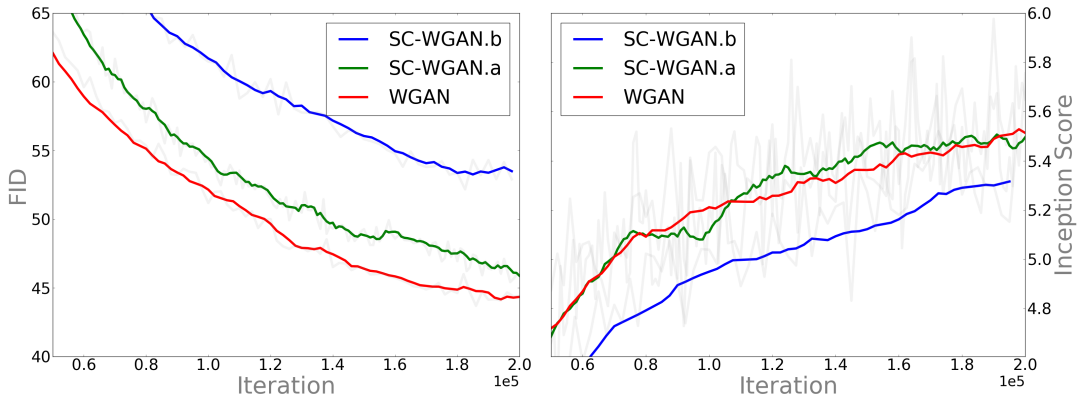


Figure 5.5: FID and IS results for SC-WGAN.a and SC-WGAN.b against a WGAN baseline. Both SC-WGAN models were successful in conditioning on class, as witnessed in Appendix A, although the baseline achieved a better FID.

ents. We were surprised to see that this actually harmed performance. We were also surprised by the terrible performance of SC-WGAN.c and don't rule out the possibility that this was an issue with implementation. It would be interesting to investigate further with different hyperparameters.

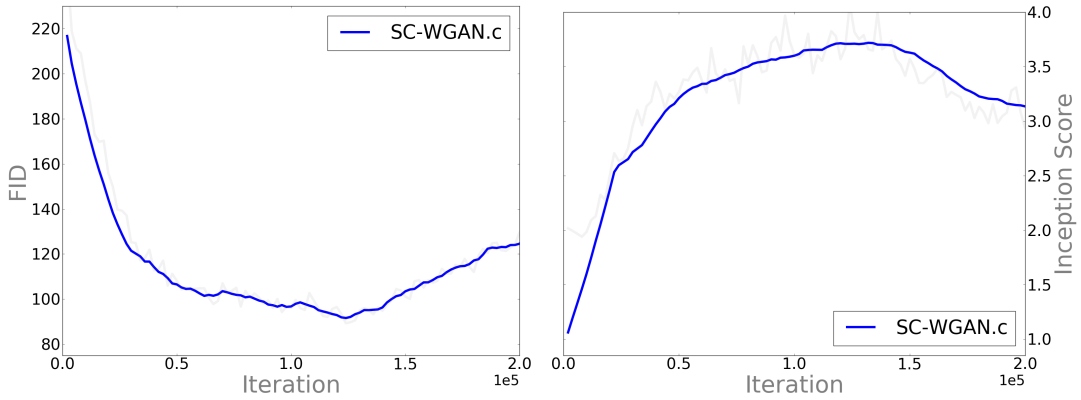


Figure 5.6: FID and IS results for the failed SC-WGAN.c model, graphed separately so not to skew the scale on the axis. The image quality of the samples is very poor, see appendix A.

5.0.4 Normal GAN Results

To test the effect that the Wasserstein loss and gradient penalisation had on performance, we tested out a few of the same architectures but with the original GAN loss[63] instead. This required a few small changes to the models, as outlined in Chap-

ter 4.0.5. We found that these architectures were highly unstable, with all of them quickly suffering from mode collapse. This was also true for the direct reproduction of the AC-GAN model from [5], which we denote **AC-GAN.b**. The other models were denoted: **AC-GAN.a**, a non-Wasserstein version of AC-WGAN.a; **GAN**, a non-Wasserstein version of WGAN; **cGAN**, a non-Wasserstein version of cWGAN.a and **SC-GAN**, a non-Wasserstein version of SC-WGAN.a.

Their poor performance is not too surprising since GANs are notoriously unstable, which is why WGAN and WGAN-GP were such breakthroughs. They do however train much faster as Figure 5.7 demonstrates. In Figure 5.9, it is interesting that the conditional models still retained their ability to condition on class despite mode collapse, the mode collapses seemed to only happen within a class. This means that the models still generate some diversity (100 different images at least) even if every class has mode collapsed. If the quality of those 100 images are good, then this could potentially be a way for conditional GANs to cheat the IS and FID evaluations. It depends on how well those scoring methods capture diversity. This phenomenon could be a benefit of class conditioning in general. If class conditioning can contain a mode collapse to a single class then for our case, with CIFAR-100, this means a mode collapse cannot "spread" to over 1/100th of the generated samples.

After the poor performance of AC-GAN.b, we had concerns for our implementation, leading us to check the model on CIFAR-10 to see if the results agreed with [5]. Again, we found quite severe mode collapse, but this time the quality of images was high and some classes did not have any signs of mode collapse. The model topped out at an IS of 5.4, compared to [5]s score of 8.25. Note, the IS is an exponential so small changes in image quality can lead to large changes in score. If our implementation did not suffer from mode collapse it would have scored much higher, possibly in the region of 8.25. [5] does not provide a full list of hyperparameters that they used to achieve their top score, they only provide the 27 possible options that they grid searched over. It is possible that one of those 27 hyperparameter configurations would have matched our AC-GAN.b and scored 5.4. We admit it is also possible that we missed a crucial element in the implementation. There also weren't any generated CIFAR-10 samples provided so we weren't able to qualitatively compare the models. All of this doubt would of course be alleviated if they had provided code to reproduce the experiments, a practice that we hope will become the norm for machine learning papers.

Although we didn't spend any time tuning hyperparameters for these models, and although we had some concerns for our AC-GAN.b implementation, there is clear

evidence here for the superior stability of the WGAN-GP architecture on CIFAR-100. This is in line with [3]'s findings, where they give multiple architectures that perform well for WGAN-GP and not at all for the normal GAN.

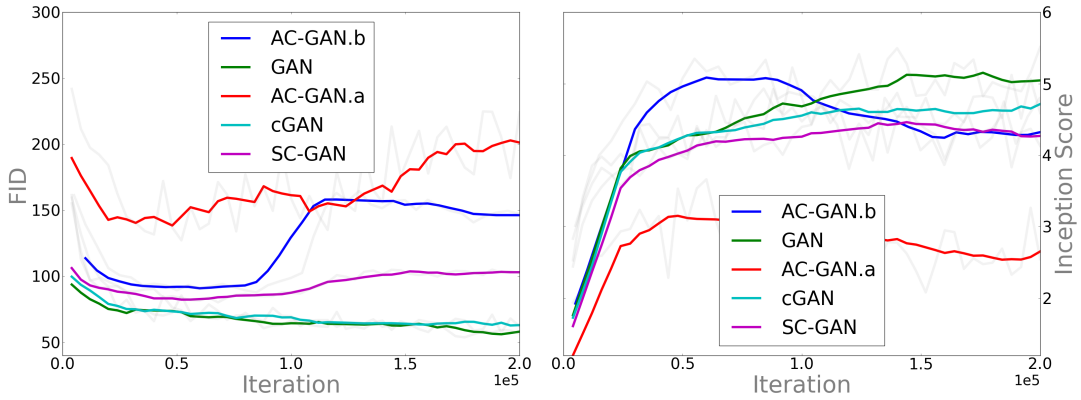


Figure 5.8: FID and IS results for normal GAN experiments. The image quality of the samples was found to be very poor with severe mode collapse in all cases.

5.0.5 Comparison of Best Models

A comparison of the best models of each category is given in Figure 5.11 where AC-WGAN.b is shown to come out on top (again we chose to use AC-WGAN.b and not AC-WGAN.c because it trained quicker and only did marginally worse). Since the curves have clearly not converged over 2e5 iterations we extended the training of two models, we ran the baseline WGAN and AC-WGAN.b for an extra 2e5 iterations, see Figure 5.10. Over this extended training time the models converge in a predictable fashion and we see no change in outcome, AC-WGAN.b gives an improvement of $\approx$1 for FID and $\approx$0.2 for IS.

It would have also been interesting to further compare AC-WGAN.b against AC-WGAN.a, since AC-WGAN.a follows a standard form of class conditioning and it makes sense to compare one class conditioned model versus another, rather than against a WGAN without class conditioning. However, given that AC-WGAN.b outperformed AC-WGAN.a by a significant margin over 2e5 iterations (as evidenced by Figure 5.4), we decided that it was highly unlikely that the outcome was going to change over another 2e5 iterations so the experiment was omitted for sake of time.

5.0.6 Evidence that the Models are not Memorising

Figure 5.9: Samples from a mode-collapsed AC-GAN.b (different from AC-WGAN.b) at iteration 70,000, trained on CIFAR-100. Each row corresponds to a different class. Despite mode collapse, the class conditioning still prevails, with separate mode collapses occurring within separate classes.

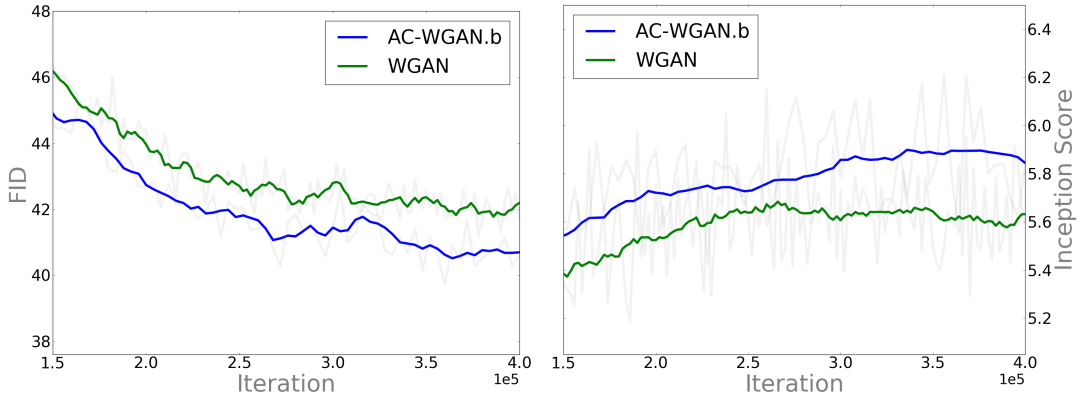


Figure 5.10: FID and IS results up to 4e5 iterations for AC-WGAN.b and the WGAN baseline. The outcome has not changed from the first 2e5 iterations.

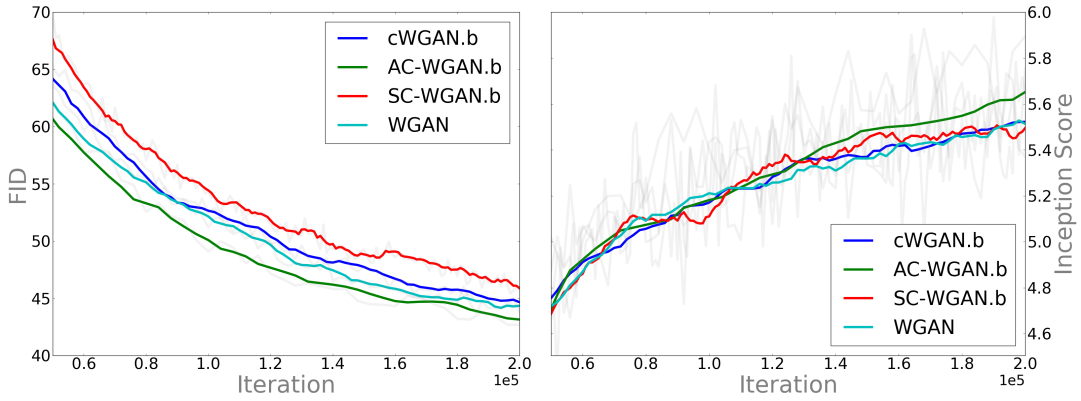


Figure 5.11: FID and IS results for AC-WGAN.a, cWGAN.b and AC-WGAN.c against a WGAN baseline. All AC-WGAN models were successful in conditioning on class, as witnessed in Appendix A. Both FID and IS indicate that our proposed MTL variants, AC-WGAN.b and AC-WGAN.c, generate slightly better images than the baseline WGAN and significantly better images than the standard AC-WGAN.a.

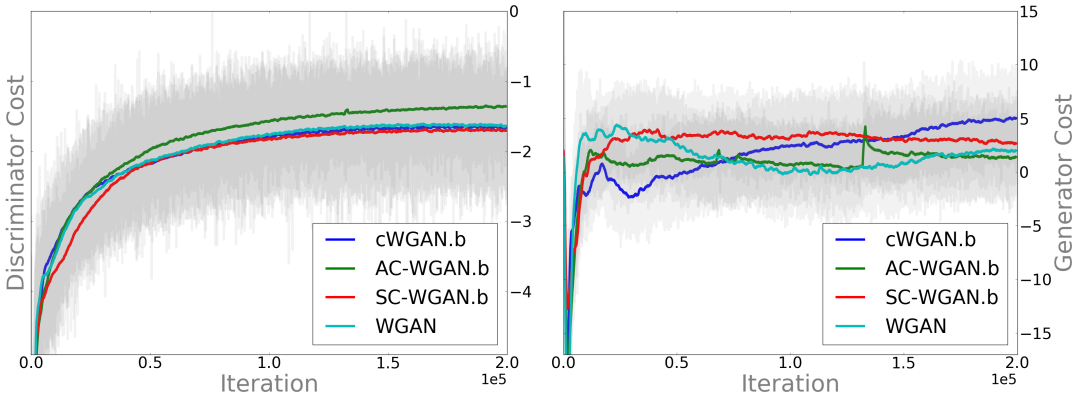


Figure 5.12: Discriminator and Generator costs for cWGAN.b, AC-WGAN.b and SC-WGAN.b against a WGAN baseline. The discriminator costs indicate that the AC-WGAN.b model generates images that are closest to the real data (in terms of Wasserstein distance). The generator cost is hard to interpret but it appears all models are converging towards 0 apart from cWGAN.b.

Unsupervised Method	**Inception Score**	**Supervised Method**	**Inception Score**
ALI[64](in [65])	5.34	**AC-WGAN.a(ours)**	**5.90**
BEGAN[66]	5.62	SteinGAN	6.35
WGAN(ours)	**6.00**	DCGAN(with labels, in [67])	6.58
DCGAN[62] (in [67])	6.16	Improved GAN	8.09
Improved GAN (-L+HA)[36]	6.86	AC-GAN[5]	8.25
EGAN-Ent-VI[68]	7.07	SGAN-no-joint[67]	8.37
DFM[65]	7.72	WGAN-GP ResNet	8.42
WGAN-GP ResNet[3]	7.86	SGAN[67]	8.59
Real Data[36]	11.24	Real Data[36]	11.24

Table 5.1: Inception scores for state-of-the-art unsupervised and supervised GANs on CIFAR-10. These are not comparable to results in this project since we are working with CIFAR-100, a different dataset, of which there are no benchmarks available. Although we do include our two sanity-test experiments, which were conducted on CIFAR-10. FID here would be preferable but, as of writing, there are no benchmarks for that either. This collection of results is courtesy of Gulrajani *et al.* (2017)[3].

Method	FID	IS	MTL	Successful Conditioning	Classifier Type	Conditioning Location	GAN FID	GAN IS
AC-WGAN.c	**42.55**	5.80	✓	✓	2 auxiliary	*G & D*		
AC-WGAN.b	42.68	5.93	✓	✓	2 auxiliary	*G & D*		
WGAN	43.69	**5.98**					53.70	5.51
cWGAN.b	43.80	5.76				*G & D*		
cWGAN.a	44.83	5.90				*G & D*	57.34	5.12
SC-WGAN.a	45.42	5.82	✓	✓	1 separate	*G & C*	80.21	4.73
AC-WGAN.a	45.98	5.87	✓	✓	1 auxiliary	*G & D*	127.17	3.66
SC-WGAN.b	52.17	5.55	✓		1 separate	*G & C*		
SC-WGAN.c	89.28	4.08	✓		2 separate	*G & C*		

Table 5.2: Comparison of our different WGAN models on CIFAR-100, ordered by FID, for generator G, discriminator D and classifier C. Note, these scores are maximum scores and do not capture the variance of the results, the graphs are more enlightening in this regard. The final columns include comparative GAN results where available.

To prove the generator isn't memorising training samples, many papers show nearest neighbours of generated samples with respect to the training data. Nearest here is defined by a simple ℓ_2 norm of the difference in pixel intensities. Figure 5.13 displays an example of a nearest neighbour.

We performed the nearest neighbourhood search after converting the images to grey-scale so the network couldn't cheat by simply changing the colour. However there are issues with this approach. Namely, the network could cheat by memorising training examples and then outputting them only with parts of the image translated. This would fool the nearest neighbour search since even though it'd be the same data, the pixels wouldn't match.

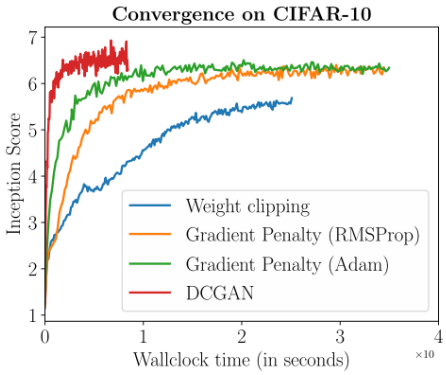


Figure 5.7: Wall-clock comparison of different GANs training on CIFAR-10. DCGAN[62] is similar to our normal GAN architecture. WGAN (weight-clipping) and WGAN-GP converge significantly slower. DCGAN also achieves a higher IS. Image courtesy of [3].

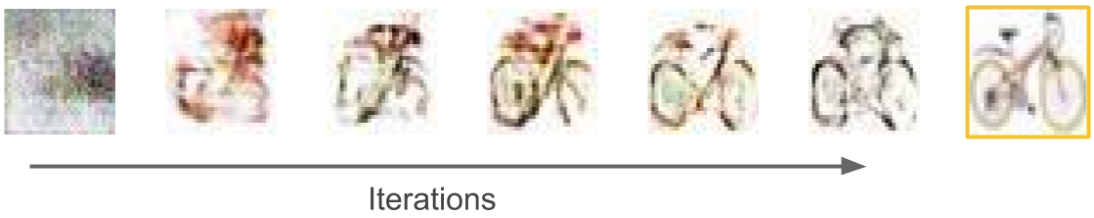


Figure 5.13: The evolution of the generators output for a fixed noise value **z** between iterations 200 and 200,000 for the baseline WGAN-GP alongside the nearest neighbour (far right) of the final output. Although the nearest neighbour shares many close similarities, there are clear differences in the bike, e.g. the colour of the tyres.

5.0.7 Interpolating in Latent Space

A better method of proving that the generator has learnt something useful is to interpolate between two points in latent space, i.e. **z**-space. An overfit model that has memorised different training examples will yield discrete transitions in observed images as **z** is interpolated between two points.

We applied a very simple method of linear interpolation, across a single **z**-dimension. The results of some cherry picked **z**-interpolations can be seen in Figures 5.14 to 5.18. The Figures shown are generated from AC-WGAN.b but similar results were obtained from all successful models. For each Figure, the class variable was kept constant as was all but one dimension of the noise variable **z**. This one **z**-dimension was shifted in 0.5 increments from -2.5 to 2.5 to obtain 10 different images.

Many of the dimensions didn't seem to represent semantically meaningful features while some of them definitely did (hence the cherry-picking) but none of the tests showed any sign of discrete transitions. For example the **z**-dimension that was shifted in Figure 5.14 captures a colour change, the **z**-dimension that was shifted in Figures 5.15 and 5.16 (the same dimension gave both results) appears to capture a resizing or scaling of the object. This proves that the network is disentangling semantically meaningful features from the data. The same **z**-dimension can control qualitatively similar features across multiple classes, as the examples of the rescaled apple 5.15 and bottle 5.16 show. The network isn't simply memorising training examples, each point on the interpolated line between two distinct points in the latent space yields different "realistic" images. The fact that the effects of the different dimensions of **z** are usually not interpretable and in general don't obey a nice 1-1 mapping with recognisable features is to be expected as the transformation from **z** space to image space is highly non-linear.

It is worth qualifying, the interpretations we have made here are qualitative and up to debate, there is definitely a danger of reading too much into cherry picked samples.

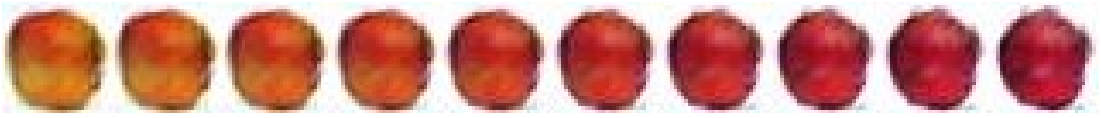


Figure 5.14: Interpolating a single dimension in **z** space, between orange apples and red apples. This dimension appears to control an aspect of colouring.

Figure 5.15: Interpolating a single dimension in **z** space, between small apples and large apples. This dimension (the same one as Figure 5.16) appears to control scaling. In examples with backgrounds and foregrounds, it appeared to only scale foreground objects, evidence that the models are disentangling scene dynamics.

Figure 5.16: Interpolating a single dimension in **z** space, between small bottles and large bottles. This dimension (the same one as 5.15) appears to control scaling, along with lighting of the background.

5.0.8 Criticism of Results

To gather statistically meaningful results it is preferable to re-run experiments and report the median. Since we chose to experiment with a broad range of architectures rather than focus on one, we didn't have time to perform a robust analysis of the results. This is why our results are lacking error bars on FID or IS. We felt that since we weren't attempting to achieve state-of-the-art results, the precision of our evaluations could be compromised slightly. We however acknowledge that this is a drawback of our methodology.

Another drawback is the fact that we didn't train all of the models to convergence. It is dangerous to extrapolate learning curves into future epochs. For instance, sometimes regularisation doesn't reveal its benefits until late into the training, so there is a

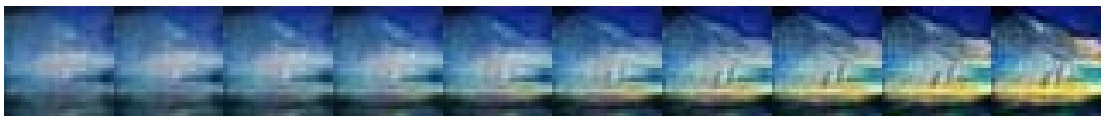


Figure 5.17: Interpolating a single dimension in **z** space, between a normal sky and a sunset. The network isn't simply plastering some yellow on to the scene, there is a source and direction to the the light. The silhouette of the landscape in the foreground gets noticeably sharper and there is some reflection in the clouds.

Figure 5.18: Interpolating a single dimension in **z** space, between two different houses. The house slowly rotates, gets bigger and grows a porch.

possibility that some of the models that had MTL aspects to them might have eventually outperformed the baseline, e.g. AC-WGAN.a. However it is still worthwhile to investigate the performance of models within a fixed number of iterations and 2e5 iterations is a reasonable length of training time.

Another undesirable aspect of our methodology is the fact that we report maximum scores in our tables. A preferable method of evaluation would be to take the models at these maximum scores and generate another set of samples on which the FID and IS are reported on, as done in [5] for example. This is analogous to evaluating on a test set in supervised learning. Again, we didn't feel compelled to do this since we weren't striving for state-of-the-art. The learning curves in the graphs are more enlightening and where we draw our conclusions from.

5.0.9 ResNet WGAN Results

As a final experiment we tested the cutting edge ResNet architecture used by Gulrajani *et al.*[3] that achieved very close to state-of-the-art on CIFAR-10. The full specification of the model is given in Appendix D. Class conditioning was incorporated in the same way as AC-GAN, with the addition of conditional instance normalisation[69] (CIN) in place of vanilla batchnorm. The theory of CIN is briefly discussed in Appendix C.

We ran two experiments:

1. One experiment trained the exact same architecture as [3] on CIFAR-100 instead of CIFAR-10. This served as our state-of-the-art benchmark. Since there are no

published GAN results for CIFAR-100, this is about the best we could do. We denoted this architecture **ResNet.a**.

2. One experiment, the same as 1. only with our "MTL trick" included. Namely, we added an extra auxiliary classifier task to the discriminator that classified coarse labels. We denoted this architecture **ResNet.b**.

The FID and IS results can be seen in Figure 5.19. Samples from the first 10 classes from the fully trained models can be seen in Figures 5.20 and 5.21.

Clear from both the FID and IS results, the training of ResNet.b takes a significantly longer time to converge. However, the FID results quite clearly suggest that ResNet.b converges to a better area of parameter space. This is not shown on the IS graph, where the two architectures score the same. Qualitatively, it appears that ResNet.a suffers from mode collapses within many classes, e.g. the bears in the fourth column of Figure 5.20. This occurs noticeably less in the samples viewed from ResNet.b, e.g. the bears in Figure 5.21 show much more diversity. Although this qualitative evaluation is not robust, we use it to hypothesise that the MTL in ResNet.b helps reduce mode collapse and increase diversity. As [59] show, the FID captures image diversity better than the IS, which could explain the discrepancy of the evaluations. [59] also show a range of distortions that the FID picks up on but the IS does not, so it is also possible that ResNet.b produces less distorted images.

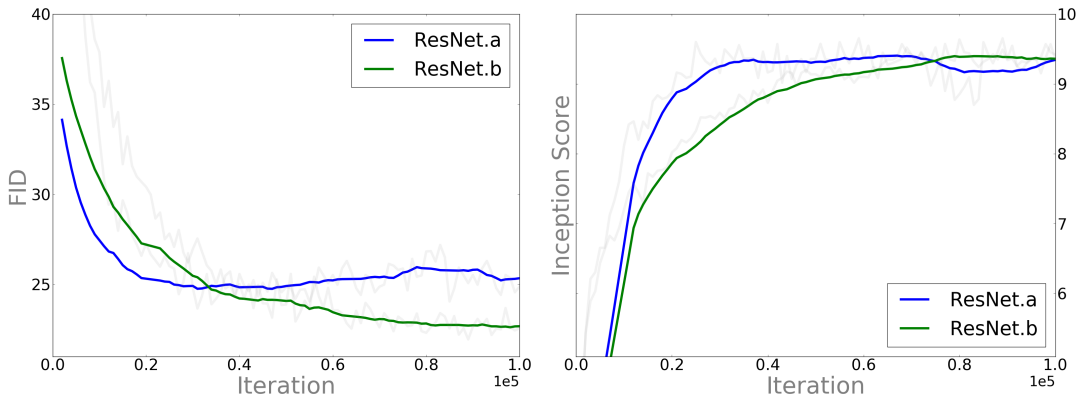


Figure 5.19: The MTL trick applied to a state-of-the-art ResNet WGAN architecture on CIFAR-100. ResNet.a is a strong model introduced by [3], ResNet.b is the same model but with a second auxiliary task in the discriminator that classifies coarse labels. Interestingly, the FID shows a significant improvement whereas the IS does not. We hypothesise that this is due to a lack of sensitivity to image diversity in the IS metric.

Figure 5.20: Generated images of the first 10 classes of ResNet.a. There is evidence of "mild" mode collapse in many of the classes, e.g. the aquarium fish in the second column are all similar as are the bears in the fourth column.

Figure 5.21: Generated images from ResNet.b. There appears to be significantly less mode collapse when compared to ResNet.a, e.g. the fish in the second column display much more variety, as do the bears in the fourth column.

Chapter 6

Conclusion

In the first part of this work we explored the effects of MTL on classification of CIFAR-100 images. We saw that for two different auxiliary tasks, one classifying with coarse labels and one classifying a different dataset, CIFAR-10, MTL gave considerable benefits over the baseline. For the coarse label auxiliary task, this was shown to be true for 100%, 75%, 50% and 25% of the data. Hard-sharing performed slightly better than soft-sharing, where soft-sharing was found to work best with hyperparameters of $\lambda = 0.1$ and $b = 3$. These findings are in keeping with the results of Teterwak *et al.*, where they experimented with hard-sharing MTL architectures on image classification[20]. We don't know of any comparative work on this form of soft-sharing.

After laying some foundational GAN-theory in Chapter 3, we proposed a variety of different WGAN variants that involve class conditioning and MTL. Despite the large number of classes in CIFAR-100, several architectures in the AC-WGAN and SC-WGAN categories were found to allow control over the generator's outputs. The simpler cWGAN variant was found unsuccessful however. We also found that using the WGAN-GP loss gave significant benefits over the normal GAN loss, with higher image quality and little sign of mode collapse. We proved through nearest neighbour searches and latent space interpolations that the WGANs were not memorising training examples but had learned useful representations that were used to produce novel images.

Our most successful models, AC-WGAN.b & AC-WGAN.c, relied on two auxiliary classifier tasks, as opposed to the original AC-GAN's single auxiliary classifier. This extra auxiliary task classified coarse labels, as explored in Chapter 2. The benefits of this additional multitask learning allowed these models to outperform the vanilla

AC-WGAN.a as well as the WGAN baseline. This spurred us to apply this "MTL trick" to a state-of-the-art architecture, Gulrajani *et al.*'s ResNet WGAN-GP, which has the second best IS for CIFAR-10. We found that this gave significant improvements for FID but gave the same IS. After qualitative assessment it was clear that our model gave a more diverse range of samples within each class, doing a much better job at avoiding mode collapse. We conclude from this that the IS does not capture image diversity well. For this reason and others pointed out in Section 3.0.3, we hope that researchers will switch to FID as a primary evaluation technique, bringing about stronger comparisons between generative models.

6.0.1 Future Work

There are many natural continuations to the work presented here. These were not pursued due to time or computational constraints.

A clear one is hyperparameter optimisation, something we did very little of. For example, it would be interesting to try out different values for s, the classifier training frequency. We feel that we did not give the SC-WGAN models much of a chance in this regard. It would also be interesting to explore the effect of different granularities of coarse labelling for the auxiliary tasks. In particular for AC-WGAN.b and AC-WGAN.c, our most successful models. The 20 coarse labels given to us in CIFAR-100 are quite arbitrary. Related to this, an important experiment, that we didn't run, would be to set the auxiliary task(s) to exactly match the main task and compare this against the auxiliary tasks that we used. This would tell us how important it is to have differing granularities of labels across the tasks. Perhaps it is enough to take a simple ensemble of identical tasks.

There are lots of other forms of MTL that could be incorporated into a GAN or WGAN architecture. For example: we could have extended AC-WGAN to have three auxiliary classifier tasks, each classifying a different hierarchy of classes; we could have tried four tasks, all either soft-sharing or hard-sharing; we could have used the trace norm for regularisation instead of the ℓ_2 norm, etc. Given the huge amount of candidates, just blindly testing out different permutations of "MTL-GAN" is not the ideal approach to exploring the model space. It would be preferable to have a better understanding of how MTL fundamentally works and how it effects the hidden representations that are captured. This, in conjunction with a better understanding on the training dynamics of GANs would guide model-design in a much more directed

fashion. We attempted to do this with our precursor investigation into MTL for the discriminative case, but we only really scratched the surface.

From a more empirical perspective, it would have been interesting to push our strongest model, ResNet.b, to train on the same 1000 class Imagenet problem that the original AC-GAN failed to class condition on. The authors of the AC-GAN paper state that: *"Building a single unified model that could generate diverse samples from all 1000 classes would be an important step forward."*[5]. Given that ResNet.b outperforms the original AC-GAN significantly, it could well achieve this.

Appendix A

Extended WGAN Results

Shown are random generations of the first ten classes from CIFAR-100, each row of images is a different class. Classes were selected this way (as opposed to random) to help show that the "easy" classes weren't cherry picked. For the baseline model, without any attempted class conditioning, the ordering of the images has no significance. Also shown are the best FID and IS scores achieved of each architecture after 200,000 generator-iterations of training.

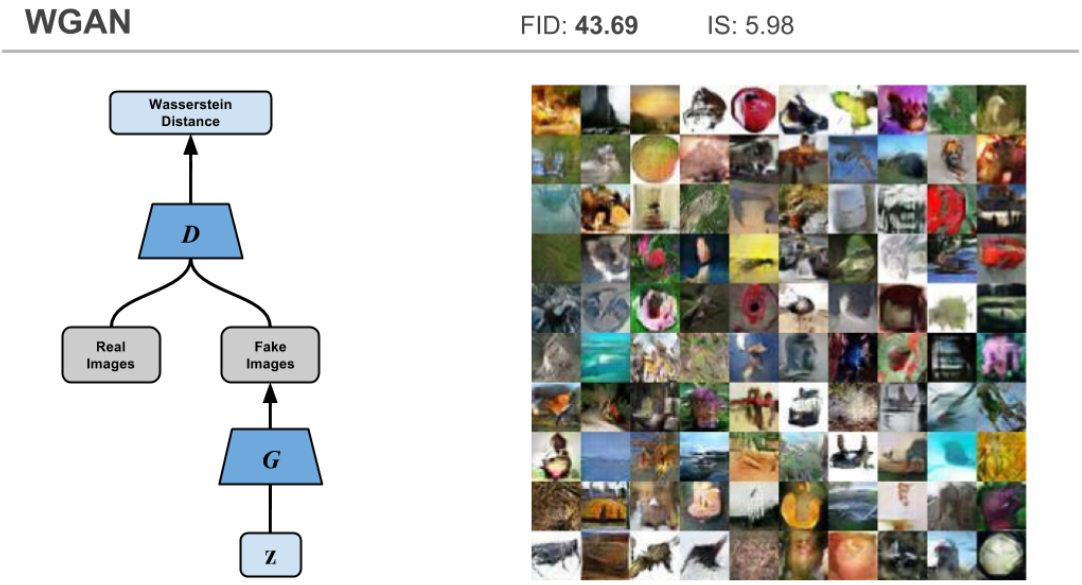


Figure A.1: WGAN (baseline) results.

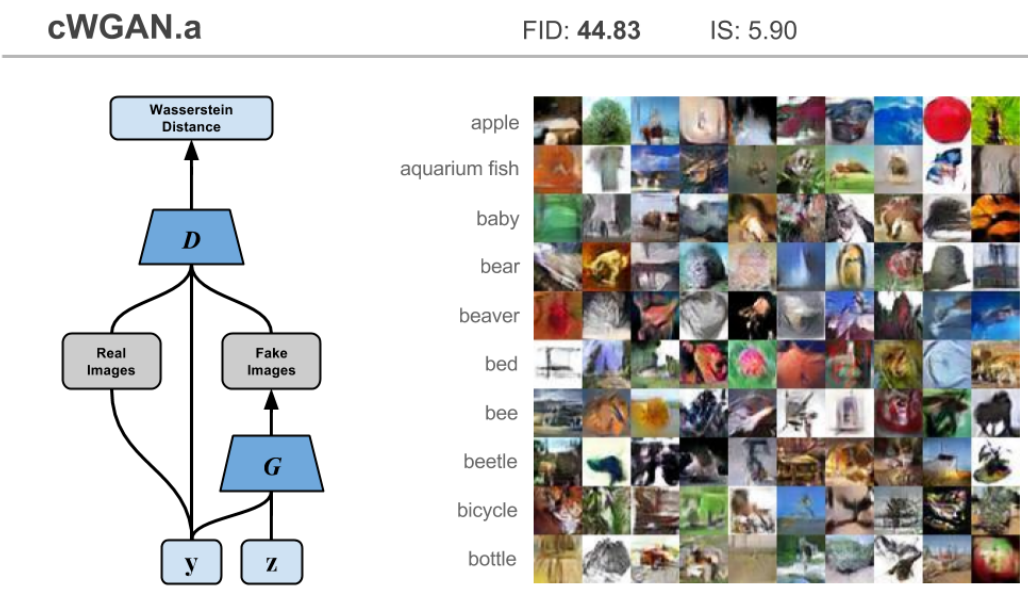


Figure A.2: cWGAN.a results.

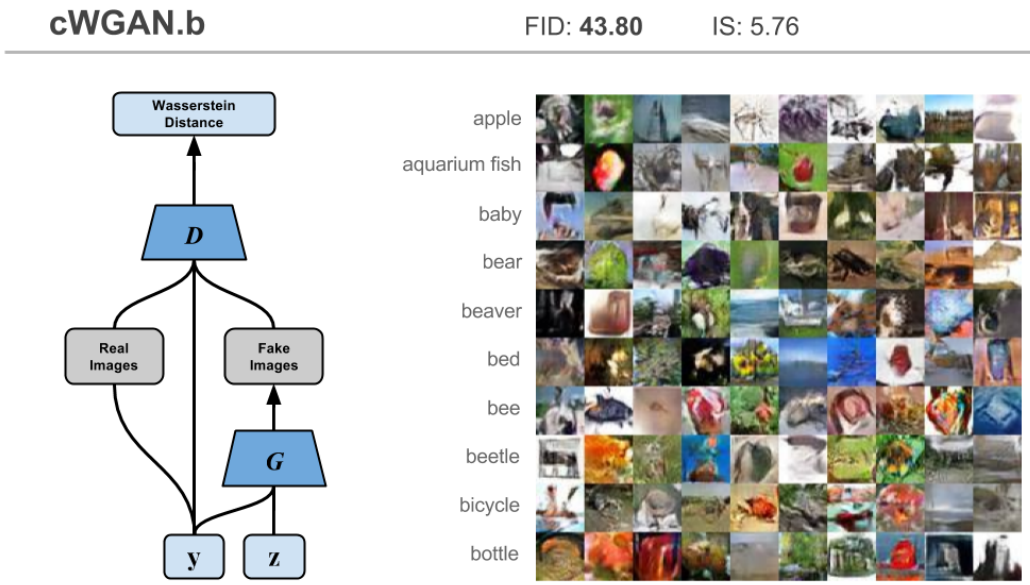


Figure A.3: cWGAN.b results.

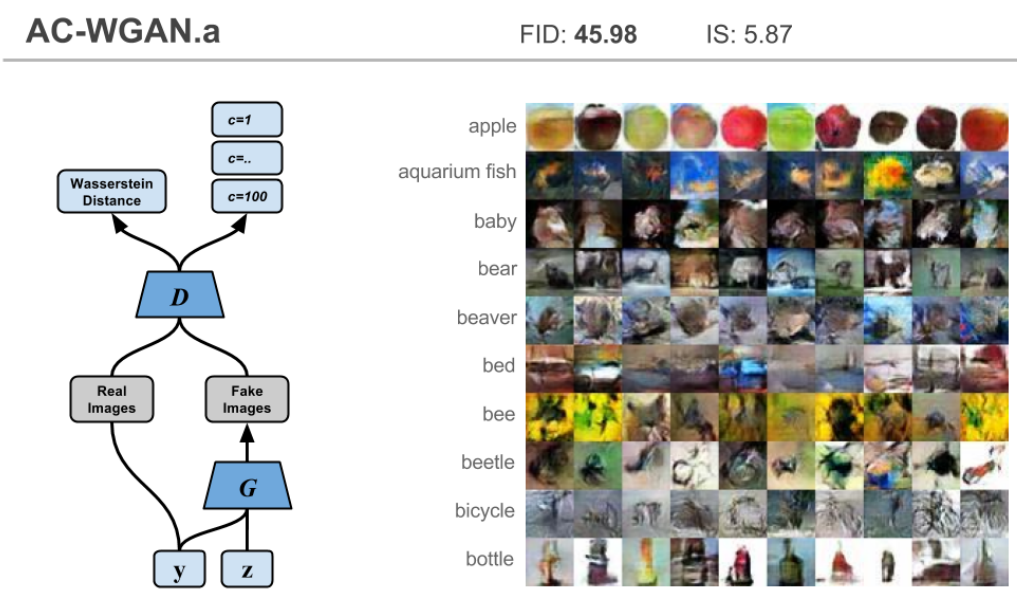


Figure A.4: AC-WGAN.a results

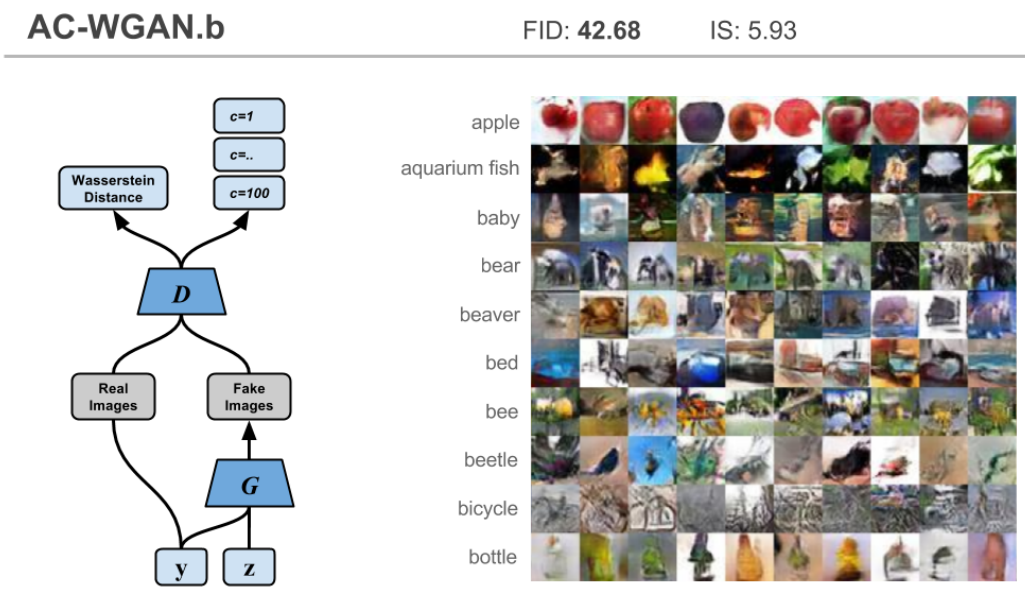


Figure A.5: AC-WGAN.b results.

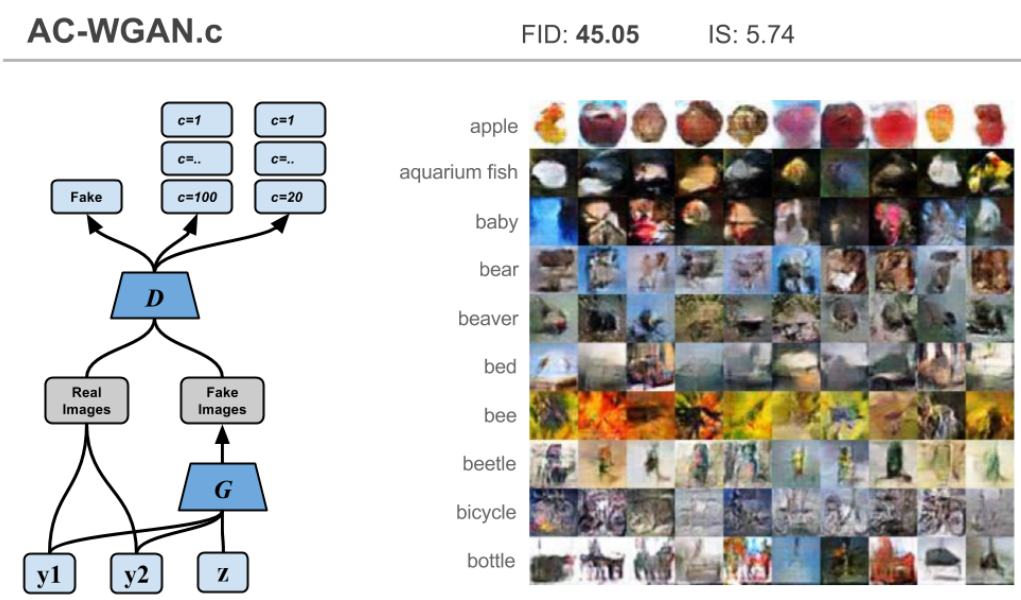


Figure A.6: AC-WGAN.c results.

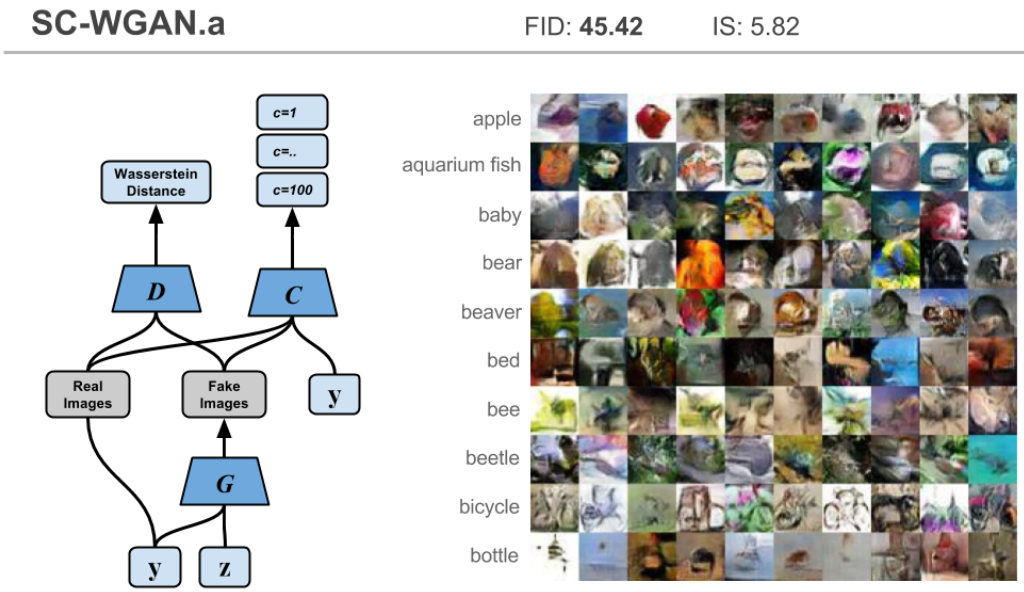


Figure A.7: SC-WGAN.a results.

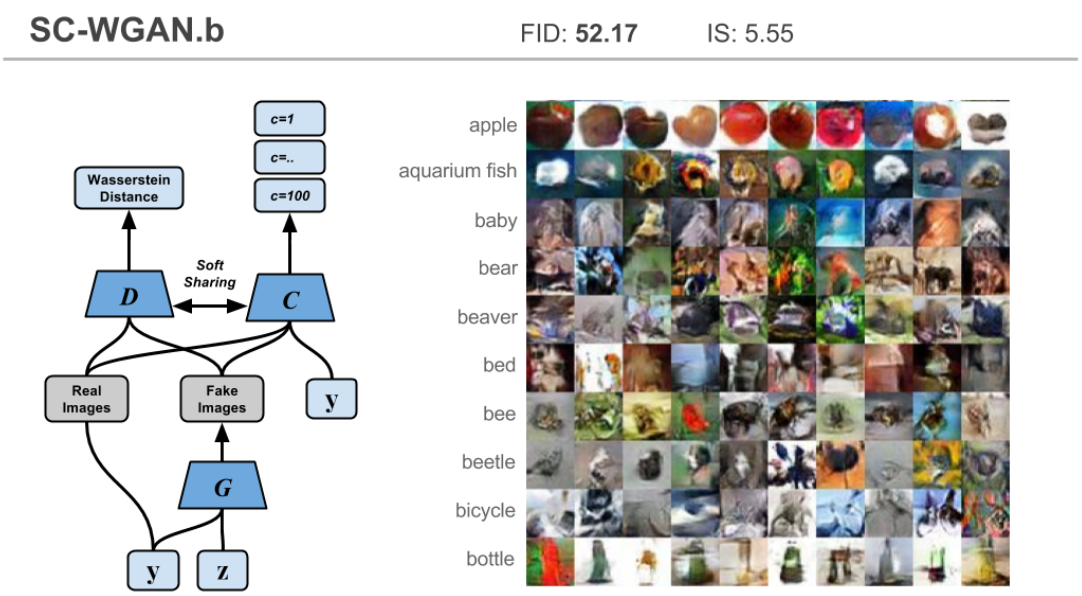


Figure A.8: SC-WGAN.b results.

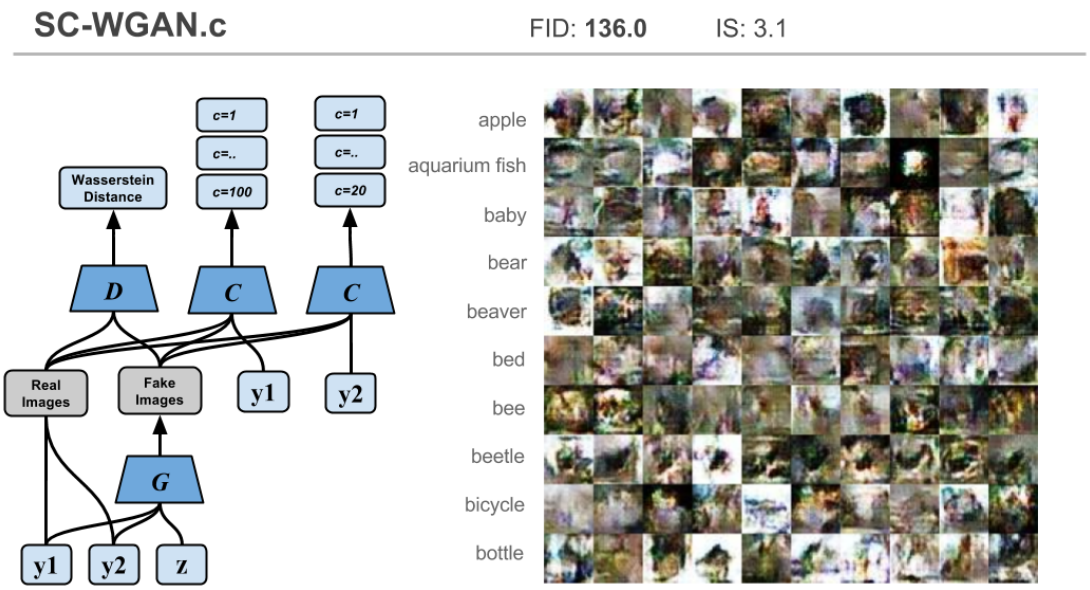


Figure A.9: SC-WGAN.c results.

Appendix B

Kantorovich-Rubenstein Duality

The WGAN architecture minimises the Wasserstein distance of the generator's output distribution p_G and the real distribution of the data p_{data}. The Wasserstein distance is defined as,

$$\mathrm{W}(p_G, p_{\text{data}}) = \inf_{\gamma \in \Pi(p_G, p_{\text{data}})} \mathbb{E}_{x,y\sim\gamma}||x-y|| \tag{B.1}$$

Where $\Pi(p_G, p_{\text{data}})$ is the set of all joint distributions with marginals $\int \gamma dx = p_G$ and $\int \gamma dy = p_{\text{data}}$. γ can be thought of as a *transport plan* that transforms one distribution into another[1]. Intuitively, the Wasserstein distance is the amount of work required in the optimal transport plan. There are infinite possible transport plans, finding the infimum in B.1 directly (using linear programming) suffers heavily from the curse of dimensionality and is completely intractable for high dimensional data-spaces like images. Fortunately this equation has a dual form, an equivalence, which helps overcome this tractability issue,

$$\mathrm{W}(p_G, p_{\text{data}}) = \sup_{\|f\|_L \leq 1} (\mathbb{E}_{x\sim p_G}[f(x)] - \mathbb{E}_{x\sim p_{\text{data}}}[f(x)]) \tag{B.2}$$

$\sup_{\|f\|_L \leq 1}$ is the supremum over all 1-Lipschitz functions, that is, over functions with gradient less than or equal to 1 everywhere. This is a special case of a theorem in mathematical analysis called *Kantorovich-Rubenstein duality* and in a way, lies at the heart of the WGAN. This equivalence is simply stated in the original WGAN paper

[1] Kantorovich's work on optimal transport, and specifically it's implications to Economics, was heavily scrutinised by the Soviet Union as they feared it's capitalist leanings.[70]

[2] but since it is so crucial, we will give a proof here, interpreted from Villani's book, *Optimal Transport: Old and New*[70].

To simplify the notation, let $\pi = \Pi(p_G, p_{\text{data}})$. We then have,

$$\mathrm{W}(p_G, p_{\text{data}}) = \inf_{\gamma \in \pi} \mathbb{E}_{x,y\sim\gamma}||x-y|| \tag{B.3}$$

Removing the constraint $\gamma \in \pi$ we can restate this as,

$$= \inf_{\gamma} \mathbb{E}_{x,y\sim\gamma}||x-y|| + \begin{cases} 0 & \text{if } \gamma \in \pi \\ \infty & \text{otherwise} \end{cases} \tag{B.4}$$

The optimal γ hasn't changed here since any $\gamma \notin \pi$ is punished by the ∞ term. Now we can rephrase the conditional part (the part in the single sided brackets) with an additional function $f : \mathbb{R} \to \mathbb{R}, x \mapsto k$. B.4 becomes,

$$= \inf_{\gamma} \left\{ \mathbb{E}_{x,y\sim\gamma}||x-y|| + \sup_{f} \left\{ \left(\mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{x\sim\gamma} f(x) \right) - \left(\mathbb{E}_{t\sim p_{\text{data}}} f(t) - \mathbb{E}_{y\sim\gamma} f(y) \right) \right\} \right\} \tag{B.5}$$

To see that the conditional part of B.4 is equivalent to the new terms in B.5, let $\gamma \in \pi$. Then by the definition of π,

$$x, y \sim \gamma \implies x \sim p_G \text{ and } y \sim p_{\text{data}}$$

This implies that,

$$\mathbb{E}_{s\sim p_G} f(s) = \mathbb{E}_{x\sim\gamma} f(x)$$

and,

$$\mathbb{E}_{t\sim p_{\text{data}}} f(t) = \mathbb{E}_{y\sim\gamma} f(y)$$

Thus the new terms in B.5 vanish. This holds for any $\gamma \in \pi$. If $\gamma \notin \pi$ then the supremum over f will trivially be ∞ since the terms don't vanish and f can be any real function. Thus, B.4 and B.5 are equivalent.

Since the first expectation term in B.5 does not rely on f we can move the sup to the front, giving us,

$$= \inf_{\gamma} \sup_{f} \left\{ \mathbb{E}_{x,y\sim\gamma}||x-y|| + \left(\mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{x\sim\gamma} f(x) \right) - \left(\mathbb{E}_{t\sim p_{\text{data}}} f(t) - \mathbb{E}_{y\sim\gamma} f(y) \right) \right\} \tag{B.6}$$

Finding the inf and sup here is a bilevel optimisation problem. Solving this requires finding the optimal solution of the inner optimisation $\sup_f$ for every value of the outer optimisation $\inf_\gamma$. Then from these solutions, taking the one that is optimal for the outer optimisation. The next step is to use the minimax-principle (the proof of which is omitted for brevity) that allows us to swap the order of the inf and sup without changing the answer. This can only be done if the $\sup_f$ optimisation is convex and the $\inf_\gamma$ optimisation concave. The $\sup_f$ optimisation is trivially convex since the function being optimised is constrained to be either 0 or ∞. To see if the concavity condition is met it is easiest to go ahead and swap the sup and inf around first, giving us,

$$
\begin{aligned}
&= \sup_f \inf_\gamma \left\{ \mathbb{E}_{x,y\sim\gamma}||x-y|| + \Big(\mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{x\sim\gamma} f(x)\Big) - \Big(\mathbb{E}_{t\sim p_{\text{data}}} f(t) - \mathbb{E}_{y\sim\gamma} f(y)\Big) \right\} \\
&= \sup_f \left\{ \mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{t\sim p_{\text{data}}} f(t) + \inf_\gamma \Big\{ \mathbb{E}_{x,y\sim\gamma}||x-y|| + \mathbb{E}_{y\sim\gamma} f(y) - \mathbb{E}_{x\sim\gamma} f(x) \Big\} \right\} \\
&= \sup_f \left\{ \mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{t\sim p_{\text{data}}} f(t) + \inf_\gamma \mathbb{E}_{x,y\sim\gamma}\Big[||x-y|| - (f(x)-f(y)) \Big] \right\} \qquad \text{(B.7)}
\end{aligned}
$$

By the definition of concavity we can deduce that the inf optimisation is indeed concave.

Now consider the case where f is 1-Lipschitz. By the definition, this means,

$$
\begin{aligned}
f(x) - f(y) &\leq ||x-y|| \\
||x-y|| - (f(x)-f(y)) &\geq 0
\end{aligned}
$$

The infimum of the expectation $\inf_\gamma \mathbb{E}_{x,y\sim\gamma}$ is thus equal to 0. For the case where f is not 1-Lipschitz we have,

$$
\begin{aligned}
f(x) - f(y) &> ||x-y|| \\
||x-y|| - (f(x)-f(y)) &< 0
\end{aligned}
$$

The infimum is then free to be $-\infty$. Thus, we require f to be 1-Lipschitz at which point the inf term vanishes. We can absorb this condition into the sup term, giving us our desired form,

$$
= \sup_{||f||\leq 1} \Big\{ \mathbb{E}_{s\sim p_G} f(s) - \mathbb{E}_{t\sim p_{\text{data}}} f(t) \Big\}
$$

Appendix C

Conditional Instance Normalisation

Here we outline the differences between *batch normalisation*, *instance normalisation* and *conditional instance normalisation*. We follow the treatment given by Huang *et al.* in [71].

C.0.1 Batch Normalisation

Batch normalisation[46] (BN) is a method of normalising the output distribution of individual feature channels of a layer. This has been found to ease training significantly, often reducing sensitivity of hyperparameter selection and remedying vanishing/exploding gradients of deep architectures. BN was originally designed for discriminative networks although it has become a crucial element of certain generative models, e.g. the original GAN architecture[62].

We now formalise BN in the case where the inputs are images. Let x be an input batch of images, $x \in \mathbb{R}^{N\times F\times W\times H}$. Where N is the size of the batch, F is the number of feature channels, H is the height of the image and W is the width of the image. BN normalises the mean and standard deviation of each feature channel,

$$BN(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right)+\beta \tag{C.1}$$

Where $\gamma,\beta \in \mathbb{R}^F$ are trainable parameters that are learned by the network; $\mu(x),\sigma(x) \in \mathbb{R}^F$ are the mean and standard deviations of the feature channels, calculated across the

current batch,

$$\mu_f(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nfhw}$$
$$\sigma_f(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nfhw}-\mu_f(x))^2+\varepsilon} \tag{C.2}$$

Where $f \in \{1,...,F\}$ and $\varepsilon << 1$ is some small number to prevent floating point issues. This normalisation scheme is applied only during training, during inference there is a minor difference. The mean and variance vectors used are estimated during training and not calculated over a test mini-batch. This is because inference should always be calculated independently for each individual sample, using batch-statistics is "cheating".

C.0.2 Instance Normalisation

Surprisingly, Ulyanov *et al.*[72] found that a small simplification of BN can improve performance. Instance normalisation (IN) applies the same affine transformation to x as Equation C.1 but it treats a batch as having only a single element. μ and σ are then calculated as,

$$\mu_{nf}(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{fhw}$$
$$\sigma_{nf}(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{fchw}-\mu_f(x))^2+\varepsilon} \tag{C.3}$$

Each mean and variance are calculated for a single instance of x. This means that instance normalisation can be applied in exactly the same way during training and inference.

C.0.3 Conditional Instance Normalisation

Dumoulin *et al.*[69] proposed an extension, conditional instance normalisation (CIN), which uses a different set of learned parameters γ^c and μ^c for each class c,

$$CIN(x,c) = \gamma^{(c)}\left(\frac{x-\mu_c(x)}{\sigma_c(x)}\right)+\beta^{(c)} \tag{C.4}$$

For the case of CIFAR-100, this requires 100x more normalisation parameters than IN. As in IN, CIN is applied the same for testing and training. Remarkably, a network with

CIN can generate images of different classes using the *same* convolutional parameters but *different* CIN parameters[71].

Appendix D

ResNet WGAN

This is an outline of the architecture used in our final experiment with the ResNet WGAN. The model is copied from the architecture used to achieve SOTA on CIFAR-10 by Gulrajani *et al.*[3]. They did not include details of the architecture in their paper but did release source code for it. The model relies on *residual learning*, introduced by He *et al.*[49]. Residual learning allows layers to learn residual mappings with reference to the inputs of the layer before, as depicted in Figure D.1. This is very effective at easing the training of deep architectures.

The generator is class conditioned via CIN and the discriminator via an auxiliary classifier which classifies the images by fine labelling. Figure D.2 shows the architecture for the model used in [3], for our proposed model there is a second auxiliary classifier in the discriminator that classifies coarse labels. The layer dimensions can be found in Table D.1.

All convolutional layers used 3x3 filters with biases. The weights were initiated as in our other WGAN experiments, see Chapter 4. The Adam optimiser was used with $\beta_1 = 0$, $\beta_2 = 0.9$ and an initial learning rate of 2e-4 that was decayed linearly to 0 over 100,000 training iterations. The discriminator used a batch size of 64 and the generator, 128. The discriminator was trained 5 times for every one iteration of the generator. The inception score was

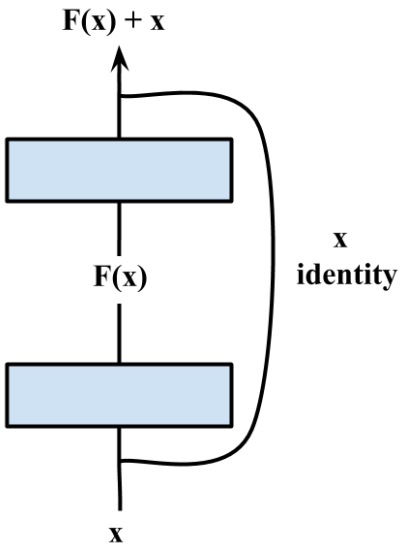


Figure D.1: Residual learning, where $F(x)$ is the transformation of a layers weights and activation function.

	1st Layer	2nd Layer	3rd Layer	4th Layer	5th Layer	6th Layer	7th Layer	8th Layer
Generator	FC 2048	DeConv 128	Conv 128	DeConv 128	Conv 128	DeConv 128	Conv 128	Conv 3
Discriminator	Conv 128	Conv 128	Conv 128	Conv 128	Conv 128	Conv 128	Conv 128	Conv 128

Table D.1: Layer dimensions for the ResNet WGAN.

calculated every 1000 generator iterations and 50,000 samples were used in calculating the score. FID was calculated with 5000 samples as in our other experiments. The random noise was drawn from a 128 dimensional multivariate Gaussian with 0 mean and identity covariance. The auxiliary classifier's loss was added to the normal discriminator loss after scaling by 0.1. For our proposed variant, with two auxiliary tasks, both auxiliary losses were scaled by 0.05 instead.

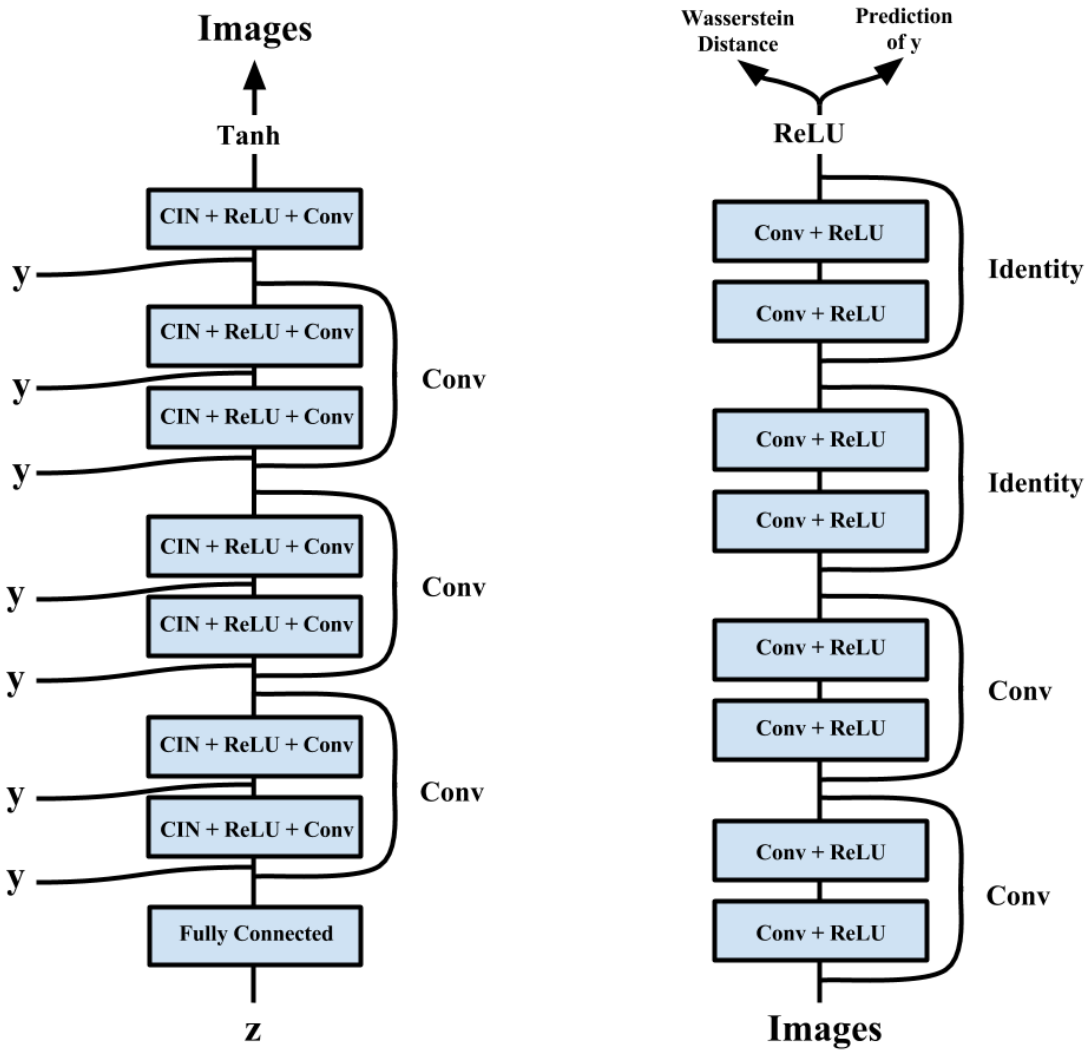


Figure D.2: ResNet WGAN architecture for generator (left) and discriminator (right). Where CIN is conditional instance normalisation, as described in Appendix C. The residual mappings are sometimes required to be convolutional mappings that either upsample or downsample. Like AC-GAN the discriminator has an auxiliary classifier task. For our proposed architecture there is a second auxiliary classifier task that classifies coarse labels.

Bibliography

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *Iclr 2015*, pp. 1–11, 12 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[2] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv:1701.07875*, p. 30, 2017. [Online]. Available: https://arxiv.org/abs/1701.07875http://arxiv.org/abs/1701.07875

[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," 3 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[4] R. Caruana, "Multitask Learning," *Machine Learning, 28*, vol. 75, no. September, pp. 41–75, 1997. [Online]. Available: http://www.cs.cornell.edu/~caruana/mlj97.pdf

[5] A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis With Auxiliary Classifier GANs," *arXiv*, pp. 1–14, 2016. [Online]. Available: http://arxiv.org/abs/1610.09585

[6] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation," *Advances in neural information processing systems*, pp. 1799–1807, 2014. [Online]. Available: http://arxiv.org/abs/1406.2984

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, 2015, pp. 1–9. [Online]. Available: https://arxiv.org/abs/1409.4842

[8] C. Ement Farabet, C. Couprie, L. Najman, and Y. Lecun, "Learning Hierarchical Features for Scene Labeling." [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/farabet-pami-13.pdf

[9] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25 (NIPS2012)*, p. 1–9, 2012. [Online]. Available: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[10] W. Chen, D. Grangier, and M. Auli, "Strategies for Training Large Vocabulary Neural Language Models," *arXiv*, p. 12, 2015. [Online]. Available: http://arxiv.org/abs/1512.04906

[11] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/38131.pdf

[12] T. Sainath, A. R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8614–8618, 2013. [Online]. Available: https://pdfs.semanticscholar.org/c282/a1f1613a477ec30d3ab93bb862d6271bb02c.pdf

[13] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," *Advances in Neural Information Processing Systems*, vol. 4, pp. 950–957, 1992. [Online]. Available: http://0-citeseerx.ist.psu.edu.innopac.up.ac.za/viewdoc/summary?doi=10.1.1.41.2305

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[15] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," *Icml*, no. 1, pp. 109–111, 2013. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/papers/icml2013_wan13

[16] T. DeVries and G. W. Taylor, "Dataset Augmentation in Feature Space," 2 2017. [Online]. Available: http://arxiv.org/abs/1702.05538

[17] J. Baxter, "A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling," *Machine Learning*, vol. 28, no. 1, pp. 7–39, 1997. [Online]. Available: http://link.springer.com/10.1023/A:1007327622663

[18] M. S. Schlichtkrull and A. Søgaard, "Cross-Lingual Dependency Parsing with Late Decoding for Truly Low-Resource Languages," *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 220–229, 2017. [Online]. Available: https://arxiv.org/pdf/1701.01623.pdfhttp://arxiv.org/abs/1701.01623

[19] Y. Yang and T. M. Hospedales, "Trace Norm Regularised Deep Multi-Task Learning," *Iclr*, no. 2014, pp. 1–5, 6 2017. [Online]. Available: http://arxiv.org/abs/1606.04038

[20] P. Teterwak and L. Torresani, "Shared Roots: Regularizing Neural Networks through Multitask Learning," pp. 1–11, 2014. [Online]. Available: file:///Files/B3/B3786ACB-ABCC-4FE5-91F4-8894255F97A8.pdf

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A large-scale hierarchical image database." *Cvpr*, pp. 248–255, 2009. [Online]. Available: http://ieeexplore.ieee.org/iel5/5191365/5206488/05206848.pdf?arnumber=5206848%5Cn

[22] G. Griffin, a. Holub, and P. Perona, "Caltech-256 object category dataset," *Caltech mimeo*, vol. 11, no. 1, p. 20, 2007. [Online]. Available: http://authors.library.caltech.edu/7694

[23] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *... Science Department, University of Toronto, Tech. ...*, pp. 1–60, 2009. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+Multiple+Layers+of+Features+from+Tiny+Images#0

[24] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based Generative Adversarial Network," *Nips*, no. 2006, pp. 1–15, 2016. [Online]. Available: http://arxiv.org/abs/1609.03126

[25] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop," *arXiv*, pp. 1–9, 6 2015. [Online]. Available: http://arxiv.org/abs/1506.03365

[26] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," 2011. [Online]. Available: http://www.vision.caltech.edu/visipedia/papers/CUB_200_2011.pdf%5Cn

[27] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts Amherst Technical Report*, vol. 1, pp. 07–49, 2007. [Online]. Available: http://vis-www.cs.umass.edu/lfw/lfw.pdf

[28] X. Gastaldi, "Shake-Shake regularization," *arXiv*, pp. 1–10, 2017. [Online]. Available: http://arxiv.org/abs/1705.07485

[29] P. Bell, P. Swietojanski, and S. Renals, "Multitask Learning of Context-Dependent Targets in Deep Neural Network Acoustic Models," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 2, pp. 238–247, 2 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7747518/

[30] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 11-18-Dece. IEEE, 12 2016, pp. 1440–1448. [Online]. Available: http://ieeexplore.ieee.org/document/7410526/

[31] R. Collobert and J. Weston, "A unified architecture for natural language processing," in *Proceedings of the 25th international conference on Machine learning - ICML '08*. New York, New York, USA: ACM Press, 2008, pp. 160–167. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1390156.1390177

[32] B. Ramsundar, S. Kearnes, K. Edu, P. Riley, D. Webster, D. Konerding, and V. Pande, "Massively Multitask Networks for Drug Discovery." [Online]. Available: https://arxiv.org/abs/1502.02072

[33] S. Ruder, "An Overview of Multi-Task Learning in Deep Neural Networks," 6 2017. [Online]. Available: http://arxiv.org/abs/1706.05098

[34] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Sluice networks: Learning what to share between loosely related tasks," 5 2017. [Online]. Available: http://arxiv.org/abs/1705.08142

[35] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch Networks for Multi-task Learning," 4 2016. [Online]. Available: http://arxiv.org/abs/1604.03539

[36] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *Nips*, pp. 1–10, 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[37] S. Wu, S. Zhong, and Y. Liu, "Deep residual learning for image steganalysis," pp. 1–17, 12 2017. [Online]. Available: http://arxiv.org/abs/1512.03385

[38] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *CoRR*, pp. 1–7, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[39] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative Adversarial Text to Image Synthesis," *Icml*, pp. 1060–1069, 5 2016. [Online]. Available: http://arxiv.org/abs/1605.05396

[40] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *Proceedings of The 33rd International Conference on Machine Learning*, p. pp. 1558–1566, 12 2016. [Online]. Available: http://arxiv.org/abs/1512.09300

[41] D. Wang and Q. Liu, "Learning to Draw Samples: With Application to Amortized MLE for Generative Adversarial Learning," pp. 1–13, 11 2016. [Online]. Available: http://arxiv.org/abs/1611.01722

[42] A. Odena, "Semi-Supervised Learning with Generative Adversarial Networks," *Icml*, pp. 1–3, 2016. [Online]. Available: http://arxiv.org/abs/1606.01583

[43] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets," *arXiv:1606.03657 [cs.LG]*, no. Nips, pp. 1–14, 6 2016. [Online]. Available: http://arxiv.org/abs/1606.03657

[44] Alistarh, "Communication-Efficient Machine Learning, with Applications to Large-Scale Summarization," *None*, vol. 1, no. 212, p. 19, 3 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[45] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," *Iclr*, pp. 1–14, 2015. [Online]. Available: http://arxiv.org/abs/1412.6806

[46] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[47] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *Arxiv*, pp. 1–15, 5 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[48] S. H. Hasanpour, M. Rouhani, and M. Fayyaz, "Let ' s keep it simple , Using simple architectures to outperform deeper and more complex architectures." [Online]. Available: https://arxiv.org/abs/1608.06037

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 11-18-Dece, pp. 1026–1034, 2016. [Online]. Available: https://arxiv.org/abs/1502.01852

[50] B. Graham, "Fractional Max-Pooling," pp. 1–8, 12 2014. [Online]. Available: http://arxiv.org/abs/1412.6071

[51] J. Snoek, O. Rippel, and R. P. Adams, "Scalable Bayesian Optimization Using Deep Neural Networks," *Icml*, pp. 0–3, 2 2015. [Online]. Available: http://arxiv.org/abs/1502.05700

[52] D. Mishkin and J. Matas, "All you need is a good init," *Iclr*, pp. 1–8, 11 2015. [Online]. Available: http://arxiv.org/abs/1511.06422

[53] J.-R. Chang and Y.-S. Chen, "Batch-normalized Maxout Network in Network," *Arxiv*, 11 2015. [Online]. Available: http://arxiv.org/abs/1511.02583

[54] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv:1505.00387 [cs]*, 5 2015. [Online]. Available: http://arxiv.org/abs/1505.00387%5Cn

[55] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-Supervised Nets," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.5185

[56] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for Thin Deep Nets," *Iclr*, pp. 1–13, 12 2015. [Online]. Available: http://arxiv.org/abs/1412.6550

[57] J. Bergstra JAMESBERGSTRA and U. Yoshua Bengio YOSHUABENGIO, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012. [Online]. Available: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[58] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 12 2016. [Online]. Available: http://arxiv.org/abs/1512.00567

[59] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium," 6 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[60] Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse, "ON THE QUANTITATIVE ANALYSIS OF DECODER- BASED GENERATIVE MODELS." [Online]. Available: https://arxiv.org/pdf/1611.04273.pdf

[61] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," pp. 249–256, 2010. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[62] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv*, pp. 1–15, 11 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

[63] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in Neural Information Processing Systems 27*, pp. 2672–2680, 2014. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[64] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville, "Adversarially Learned Inference," *arXiv*

preprint arXiv:1606.00704, pp. 1–15, 6 2016. [Online]. Available: http://arxiv.org/abs/1606.00704

[65] J. Zhao, M. Mathieu, Y. Lecun, and F. Artificial, "Improving generative adversarial networks with denoising feature matching," *Iclr*, no. 2014, pp. 1–16, 2017. [Online]. Available: https://openreview.net/pdf?id=S1X7nhsxl

[66] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: Boundary Equilibrium Generative Adversarial Networks," *arXiv*, pp. 1–9, 3 2017. [Online]. Available: http://arxiv.org/abs/1703.10717

[67] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie, "Stacked Generative Adversarial Networks," *Iclr*, pp. 1–25, 12 2017. [Online]. Available: http://arxiv.org/abs/1612.04357

[68] B. Dai, S. Fidler, R. Urtasun, and D. Lin, "Towards Diverse and Natural Image Descriptions via a Conditional GAN," 3 2017. [Online]. Available: http://arxiv.org/abs/1703.06029

[69] V. Dumoulin, J. Shlens, and M. Kudlur, "A Learned Representation For Artistic Style," *Style Transfer*, no. 2016, p. 14, 10 2016. [Online]. Available: http://arxiv.org/abs/1610.07629

[70] C. Villani, "The Wasserstein distances," in *Optimal Transport, old and new*, 2008, pp. 93–111. [Online]. Available: http://link.springer.com/10.1007/978-3-540-71050-9_6

[71] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization," 3 2017. [Online]. Available: http://arxiv.org/abs/1703.06868

[72] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis," 1 2017. [Online]. Available: http://arxiv.org/abs/1701.02096