Computer Vision Assignment 3

22580530

May 2023

Question 1

I create a GAN (generative adversarial network) in PyTorch and train it on the CIFAR-10 dataset. The model uses a DCGAN (deep convolutional GAN) architecture. The code for the model architecture and training is adapted from a guide in the PyTorch documentation [5]. Various optimizations from [2] are incorporated to produce better results.

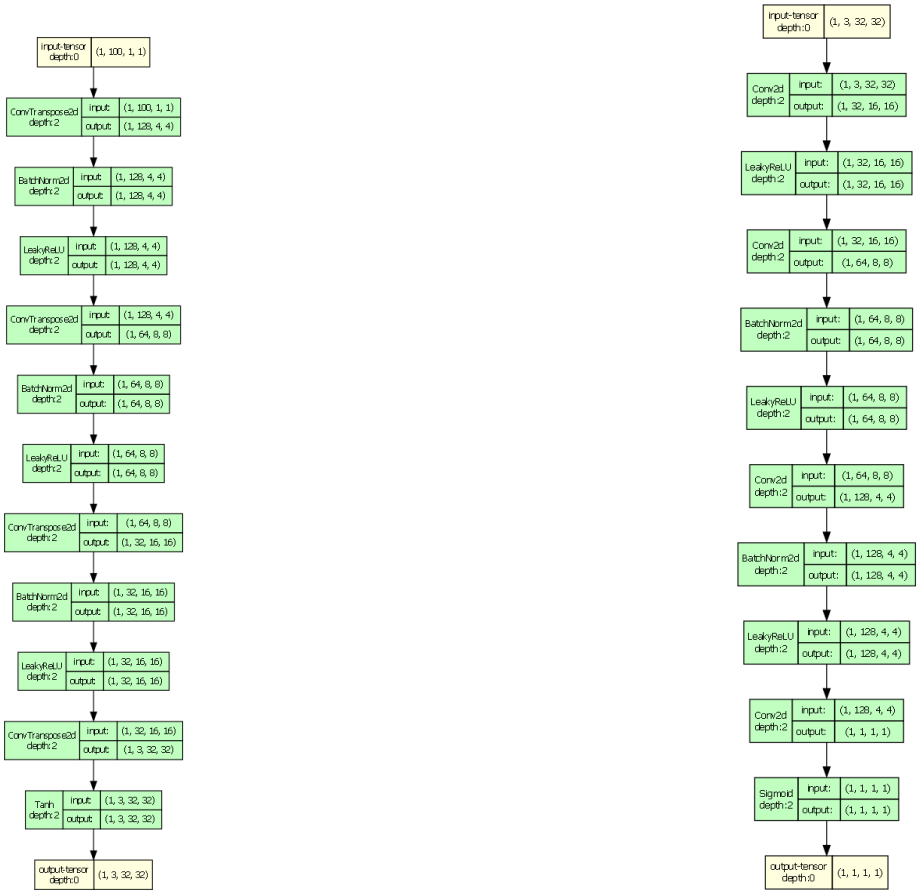
The generator takes a noise vector of length 100 as input. It has three hidden transpose convolution layers, each with a 4×4 kernel, leaky RelU activation function and batch normalization. The output layer is another transpose convolution layer with a 4×4 kernel and a tanh activation function. From left to right, the number of channels is decreased, and the size of the image representation is increased by using a stride of 2 (4 in the first layer) in the transpose convolutions. The output layer produces a $3 \times 32 \times 32$ tensor with values between -1 and 1. The architecture of the generator is shown in Figure 1a

The discriminator is a bit like a mirrored version of the generator. The format of its input is the same as the format of the generator's output. The network consists of three hidden convolution layers with 3×3 kernels, leaky Relu activation and batch normalization. From left to right the number of channels is increased, and a convolution stride of 2 (4 in the last layer) is used to shrink the dimensions of the image representation. The output layer is a convolution layer with a 4×4 kernel which compresses the representation into a single value, which is fed into a softmax activation function to give the probability of the image being real. The architecture of the discriminator is shown in Figure 1b.

The loss function used in training is the binary cross entropy loss. Theoretically, the generator would be trained to minimize $\log(1 - D)$, where D is the output of the discriminator on generated images. However, it is in practie better [2] to train the generator to maximise $\log D$. This can be achieved by flipping the labels when training the generator (labeling fake images as real). Each training iteration proceeds as follows:

1. Pass a batch of real images, labeled as real, through the discriminator, calculate the loss and compute the gradients with backpropagation.

2. Do the same, but with a batch of generated images labeled as fake.



(a) Generator (b) Discriminator

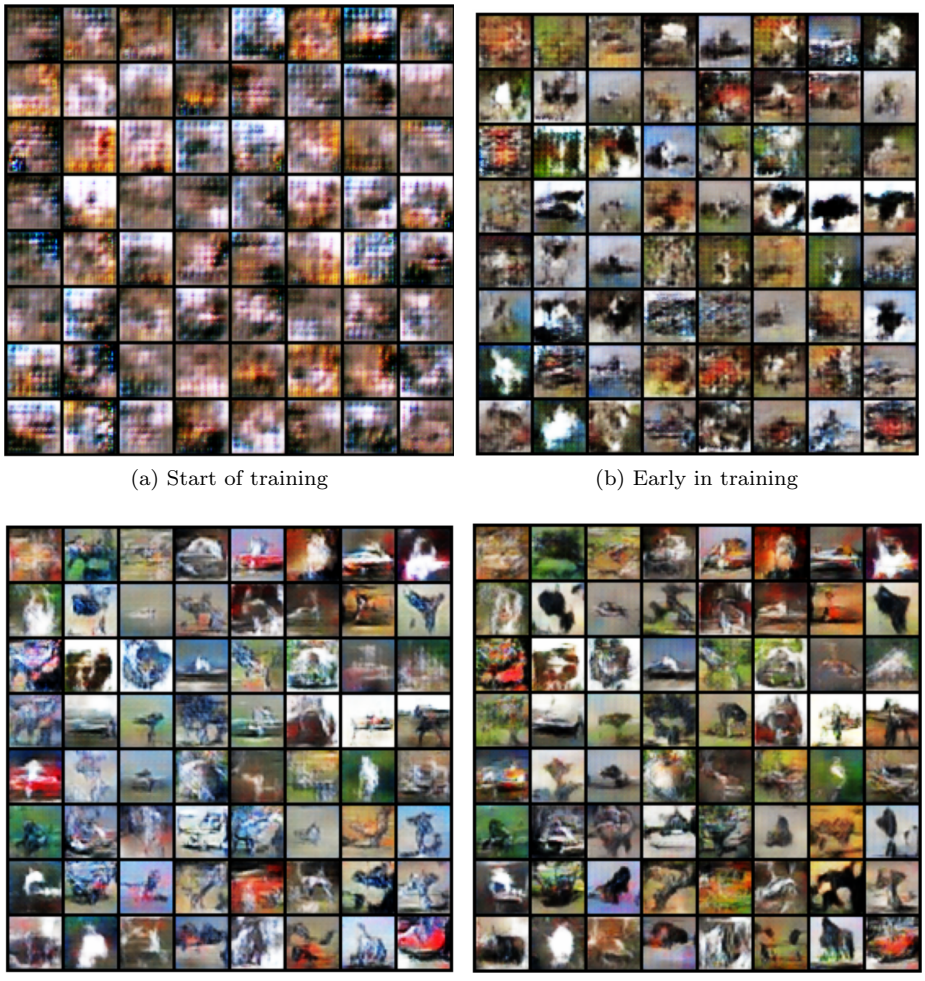
Figure 1: GAN architecture for batch size = 1

3. Update the discrimantor weights.

4. Pass the generated images through the discriminator again, but now labeled as real. Compute the loss and gradients.

5. Update the generator weights.

Images from the CIFAR-10 dataset are preprocessed into normalized $3 \times 32 \times 32$ tensors with values between -1 and 1. The model is trained with a batch size of 64. The generator uses an adam optimizer and the discriminator uses stochastic gradient descent with momentum. Both optimizers use a learning rate of 0.001. The final model is trained for 100 epochs.



(a) Start of training

(b) Early in training

(c) Late in training

(d) End of training

Figure 2: GAN generated images throughout training

Figure 2 illustrates a collection of images generated from the same noise vector at different points in training. In the beginning the generator largely outputs similar looking noise. About a third of the way through training vague shapes take form, but there are no clearly discernible objects. One can also see there is a good amount of diversity in the colors and shapes generated by the model, indicating that the model is trying to generate images from different classes. After roughly two thirds of the total training time, the images become more defined. By the end of training, some of the images are admittedly still quite abstract. However, a few of them clearly depict objects such as ships or birds. It seems that the generator is better at creating images for some classes

than others, but is certainly trying to represent more than one class.

I evaluate the model using the idea of "Inception scores". This involves using a separate image classification model to classify the images produced by the generator. The confidence scores may indicate the quality of images. The distribution of predicted labels can be used to detect mode collapse, a common problem with GANs where one class dominates the generated images. I use the tuned SqueezeNet model from the previous assignment. I tune the weights of the latter half of the hidden layers, as this was shown to achieve a good balance between accuracy (84% on the test set) and training time. The model is fed batches of 512 generated images from the beginning of training, about halfway through training and at the end of training. For each batch I record the average confidence of the predicted class and the number of classifications for each class. The results are shown in the following table:

Epoch	Confidence	Class counts									
		bird	car	cat	deer	dog	frog	horse	plane	ship	truck
0	88%	229	0	2	261	1	0	5	5	0	9
50	87%	173	11	5	231	9	19	7	6	19	32
100	84%	191	36	21	150	14	42	15	8	4	31

The mean confidence score is always high, even when classifying nearly random noise. This may indicate that the chosen SqueezeNet model is overconfident in its predictions. Whatever the reason, this metric is unfortunately not useful for measuring the effectiveness of the GAN. The class counts, however, give us some valuable insight. The GAN seems to suffer from at least partial mode collapse, as the vast majority of images are classified as belonging to two out of the ten classes. However, this problem seems to diminish the more the generator is trained, as the dominance of these classes slowly decreases. There may be an alternative explanation: that the SqueezeNet model biases towards the "bird" and "deer" classes when the image is vague or noisy. This would explain why these classes dominate the most at the start of training. Further experiments would need to be performed to determine the exact cause of these results. Nevertheless, the trained generator at least manages to produce images from each of the ten classes, if not in a uniform distribution. It seems to struggle the most with producing "plane" and "ship" images, possibly because of their unique backgrounds (skies and oceans) compared to the other classes. To improve image clarity and escape the (possible) mode collapse, one could try making the model larger, introducing randomness during training through noisy labels or dropout, and training the model for more epochs.

Question 2

a

word2vec refers to a family of neural network architectures [7] for learning vector representations of words in a collection of text, also called a corpus. One of two model architectures can be used to embed words into a vector space: bag-of-words and skip-gram.

In this model [1], word2vec is used to create vector representations of every unique word in the collection of image captions. The unique words are collectively called the vocabulary. From the arguments, I determine that a bag-of-words architecture is specified in this guide.

The bag-of-words architecture learns the embedding for a word by examining the words which frequently appear around it, known as the "context". The architecture consists of:

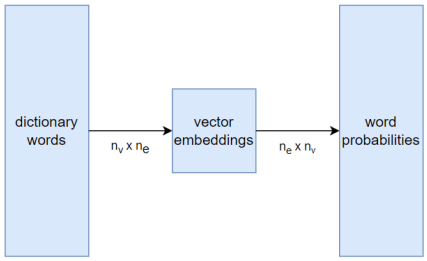


Figure 3: word2vec bag-of-words architecture

- An input layer with as many neurons as the number of words in the vocabulary (vocabulary size n_v).
- A fully connected hidden layer with size equal to the desired dimensionality of the vector embeddings (n_e) and no activation function. This layer outputs the word embeddings.
- A fully connected output layer with size n_v and a softmax activation function.

This is illustrated in Figure 3.

The network is trained using "context pairs" between central words and context words. The context of a word is precisely defined as the n_w words to the left and to the right of the central word, where n_w is the "context window size", as shown in Figure 4. For each unique word in the corpus, word pairs are generated between it and all the words in its context, for every occurrence of the word. During training, one-hot encodings of the words are used. For each context pair, the context word is fed as the input and the central word is the target. Cross entropy is used as the loss function. The network thus learns

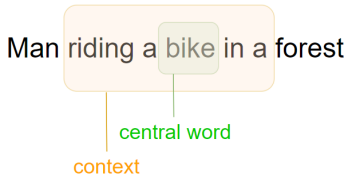


Figure 4: "Context" with a window size of 2

to predict words from their context. Once training is complete, the input and hidden layer can be used as a word embedding layer. Words in the embedding space have the property of having a small distance between them if they are semantically similar, based on their contexts in the training data. This property makes this representation useful for text generation, as in the image captioning model.

The word2vec model is pre-trained on the corpus of image captions in the training data and used as a word embedding layer in the image captioning model. The guide compares two variations of the implementation: freezing the embedding weights while training the image captioning model, and allowing the weights to be tuned. While neither method seems to result in a significant increase in model skill in this example, using a fixed pre-trained embedding layer reduces the number of trainable parameters (and thus the training time) of the image captioning model.

b

The loss function is the categorical cross entropy loss. The loss function is:

$$L_{CE} = -\sum_{i}^{n_c} t_i \log(p_i), \text{ for } n_c \text{ classes.}$$

Here t_i is the target probability and p_i is the model's output probability for a class. In the case of the image captioning model, the input is an image and a sequence of one-hot encoded words. The target is the one-hot encoding of the next word in the caption, with the one at position w. The model attempts to generate the next word. It uses a softmax activation function to output a probability distribution over all the words in the vocabulary. The vocabulary words are the classes in the cross entropy formula, which now simplifies to:

$$L_{CE} = \log(p_w)$$

c

BLEU (bilingual evaluation understudy) is an evaluation metric which is traditionally used for assessing the quality of machine translations, where it is

claimed to align closely with human judgement [4]. A BLEU score is a value between 0 and 1 computed between a candidate string $\hat{y}$ and a reference string y. The formula is constructed as follows:

1. Define the n-gram count as the number of n-grams in $\hat{y}$ which appear in the reference string, including repeated n-grams in $\hat{y}$. An n-gram is a sequence of n words.

2. Define the clipped n-gram count. This is the same as n-gram count, but the count for each unique n-gram is limited or "clipped" to the number of occurences of that n-gram in y.

3. Define the clipped n-gram precision $P_n(\hat{y}, y)$ as the clipped n-gram count divided by the number of n-grams in y, for n-grams of length n.

4. Compute the weighted geometric mean of n-gram precisions for different values of n:

$$G(N, \mathbf{w}) = \exp\left(\sum_i^N \mathbf{w}_i \log P_i(\hat{y}, y)\right) = \prod_i^N P_i(\hat{y}, y)^{\mathbf{w}_i}$$

5. Add a brevity penalty to the metric to prevent it from encouraging short candidate solutions. If c is the length (in words) of $\hat{y}$ and r is the length of y, then:

$$Penalty = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c \leq r \end{cases}$$

   The BLEU metric is the product of the brevity penalty and the weighted geometric mean of the n-gram precisions:

$$BLEU(N, \mathbf{w}) = Penalty \times G(N, \mathbf{w})$$

Typically, and in this model, the BLEU score is implemented with $N = 4$ and uniform weights:

$$BLEU = Penalty \times \prod_i^4 P_i(\hat{y}, y)^{\frac{1}{4}}$$

For evaluating this model, the BLEU score is calculated between the corpus of generated captions and a corpus of reference captions over all images in the test set. Only one reference caption per image is used.

The BLEU score has been shown to align decently well with human judgement in some applications. However, it only considers the overlap in words and phrases between captions, not the legibility or meaning of the candidate captions. Another problem is that the brevity penalty can be insufficient to counteract the metric's bias towards shorter captions [4]. Therefor, I would recommend using other metrics in addition to the BLEU score to evaluate an image captioning model.

Question 3

a

An object detection model may be incorporated into an image captioning model by utilizing the concept of attention. Attention lets the model focus on specific parts of the image when generating each word in the caption. The bounding boxes produced by a detection model can be used as attention regions, which can be fed as input into the image captioning model alongside the full image. The attention regions should be weighted by different weights at each time step of the captioning model, so that the model can focus on different regions for each word it generates. This approach was implemented by du Plessis and Brink [6] using a pre-trained R-CNN object detection model, with promising results.

b

A GAN can be constructed to perform image captioning. Instead of mapping noise to images, the generator can be made to map an input image to a generated text caption. Similarly, the discriminator can be trained to discern real image captions from generated ones. It may be wise to first pass the generator's input image through part of a pre-trained image classification model to generate an embedding. This idea (minus the pre-trained model embedding) was explored by Delbrouck [3], but yielded poor results.

References

[1] Jason Brownlee. *How to Use Small Experiments to Develop a Caption Generation Model in Keras*. Nov. 2017. URL: https://machinelearningmastery.com/develop-a-caption-generation-model-in-keras/.

[2] Soumith Chintala et al. *How to Train a GAN? Tips and tricks to make GANs work*. URL: https://github.com/soumith/ganhacks.

[3] Jean-Benoit Delbrouck, Bastien Vanderplaetse, and Stéphane Dupont. *Can adversarial training learn image captioning ?* 2019. arXiv: 1910.14609 [cs.CL].

[4] Ketan Doshi. *Foundations of NLP Explained — Bleu Score and WER Metrics*. May 2021. URL: https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b.

[5] Nathan Inkawhich. *DCGAN Tutorial*. URL: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html.

[6] Mikkel du Plessis and Willie Brink. "Improving the Performance of Image Captioning Models Trained on Small Datasets". PhD thesis. 2022.

[7] Albers Uzila. *All You Need to Know About Bag of Words and Word2Vec — Text Feature Extraction*. Aug. 2022. URL: https://towardsdatascience.com/all-you-need-to-know-about-bag-of-words-and-word2vec-text-feature-extraction-e386d9ed84aa.