Deep Learning – Assignment III

*LeNet5 with Pytorch/Tensorflow

1st Tzu Hsuan Yang
Institute of Data Science
National Cheng Kung University
Tainan, Taiwan
re6114056@gs.ncku.edu.tw

***Abstract*—**在本次的實作中，利用**Pytorch**以及**Tensorflow**實踐**LeNet5 model**，並且使用與前兩份作業相同的**image**資料集，測試實作的**LeNet5**是否能夠正確的分類出**image**，與模型在於訓練的效能還有準確性上是否與**Numpy**版本上有差異。最終在於準確性上透過**Keras**預測出來的結果最好，在**top-1 accuracy**上最佳可以達到**0.1756**的準確性，而在於**top-5 accuracy**可達到**0.4489**的正確性，而**Pytorch**上則與**Numpy**無異。

***Index Terms*—Image Classification, LeNet5, Pytorch, Tensorflow, Keras**

I. Introduction

在本次的實作中，主要任務為實作LeNet5 model用於預測image資料集，其中由於LeNet5 model的限制需要將圖像都轉為32*32的大小，卻要分類出資料集的50個類別，因此在於預測上是具有難度的，主要透過Pytorch以及Tensorflow的Keras這兩大框架實作，並且再將最終的效果與上次作業中的手刻Numpy版本的LeNet5做比較，主要針對模型的accuracy，每一epoch的運算時間，以及整體的空間複雜度做比較。

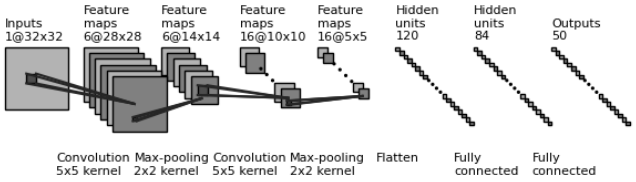

Fig. 1. The Architecture of Lenet-5 .

II. Pytorch version

在本小節中，將提到LeNet5在於Pytorch中是如何被實做出來的，以及細微針對訓練的調整，使得整體的運算效力能夠大幅的提升。

Torch Dataset. 在本次的實作中，將原始的資料透過TorchDataset這個類別，形成Pytorch的Dataset類別，可以提高程式的可讀性，以及使得Dataset模塊化。其中包含基本的getitem以及len這兩個型態，再加上read_file以及data_preproccess，使得每一張影像在於讀取進來後會先透過基礎的影像處理後在放入dataset中。之前都會先將全部的影像讀取進來後，再進行一次平行處理影像到既定格式，導致前處理的時間較長，且若是有資料集更大的問題，則會使得時間拉得越來越長，不利於後續的訓練。

DataLoader. 在建立了Dataset之後就能夠透過DataLoader輕鬆的提取出資料，而在於訓練步驟時，可以輕易的設定batch size，以小批量的方式傳入模型中，且能夠每個epoch後reshuffle所有的數據，降低模型Overfitting的問題，其中更預設使用 Python multiprocessing來加速數據檢索，因此使得整體的訓練速度變得非常的快。

LeNet5 model. 在於模型的建立上遵循原始的LeNet5 model做模型的堆疊，總共包含兩層的Conventional 2D layer，兩層的MaxPooling layer，以及三層的Fully connected layer，在於activation function上皆使用Tanh。

Trian & Test. 在於訓練上由於Training set輸入時需要先設置channel，因此將資料轉換為(channel, height, weight)的形式，而將label做one-hot encoding，在於預測過後將預測值與真實值透過Cross entropy計算loss，並透過Adam優化器去優化模型參數。

III. Tensorflow version

在本小節中，將提到LeNet5是如何透過Tensorflow中的Keras被實做出來的，由於Keras在於framework的設計上已經將多數的功能模組化，使得使用者能夠快速的上手並實踐想要的模型。

LeNet5 model. LeNet5 model透過keras的layer做堆疊，而與pytorch不同的是在於第一層中需要放入input_shape此參數，而預設的順序為(height, weight, channel)，且activation function都是直接設置在Conv與FC中，不需要獨立一層設置，因此在於堆疊層數上可以明顯看出相較於pytorch上來的少很多，在於Dense layer中也只需要輸入Output unit數量，而不需要設置Input因為系統會自動計算，也能夠減少自行計算上的失誤。在於模型的建立上遵循原始的LeNet5 model做模型的堆疊與Pytorch相同，總共包含兩層的Conventional 2D layer，兩層的MaxPooling layer，以及三層的Fully connected layer，在於activation function上皆使用Tanh。

Trian & Test. 由於keras本身沒有如同Pytorch中的dataloader，因此在於前處理上運用原本各自處理的方法進行，而模型能夠透過model.compile、model.fit、model.predict輕易的做到訓練與預測，因此在這邊不多做贅述，也因為模型能夠直接運算categorical cross entropy因此也免去了One-hot encoder的步驟，在於訓練中能夠直接計算loss並透過Adam優化器做更新參數。

IV. Model Compression

State-of-the-art deep learning techniques容易仰賴於大量的參數去達成預測的效果，但是這樣的over-parametrized models難以被應用於大多數的情況下，不管是運算量還是內存，因此在不犧牲準確性的情況下減少內存、電池和硬件消耗，通過減少模型中的參數數量來壓縮模型的最佳技術非常重要。因此在本小節中將介紹本次在於訓練上運用到的兩種Model Compression method，期望能夠在不減少正確性的情況下去降低容量的消耗。

Pruning. Pruning很簡單的來講就是透過剪枝的方式，去減少每一層儲存的參數量，使得網絡能夠輕量化，因此會在神經網路中移除一定比例的weight or bias，不僅可以隨機的修剪參數，也可以特別的針對你認為的哪些參數對您的目標來說是不必要的而做修剪的任務，例如說修剪權重最小的參數或透過accumulated gradient去修剪參數。
而在本次的實作中，簡易的透過隨機的方式去修剪一定數量的參數，在於Conventional layer中修剪20%的參數，而因為FC layer中餐數量本來就相較於Conv中大很多，因此修剪40%的參數作為後續的測試。

quantization. quantization是指以較低的浮點精度的方法執行計算和存儲tensor的技術。透過此操作可以擁有較緊湊的model presentation，且能夠進行高效率運算在各種的硬體設施上。例如說與典型的floatt32模型相比，PyTorch支持INT8的quantization，模型大小以及內存將會減少4 倍，因此計算上通常會快2到4倍。

V. Criterion

A. Categorical Cross-Entropy Loss

是一種常用於多分類任務中的損失函數，也稱為Softmax Loss，因為是將一個Softmax activation加上一個Cross-Entropy loss做計算 ，會將每一個樣本作One-Hot encoding，並乘上預測的機率值，公式如下 [1]：

$$l(x, y) = L = \{l_1, ..., l_N\}^T \tag{1}$$

$$l_n = -\sum_{c=1}^{C} w_c log \frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})} y_{n,c} \tag{2}$$

其中x為輸入的類別，y為真實的類別，w該類別的權重，C為class的數量，N為minibatch的大小，而因為通常都是取所有test loss的加總，因此最後會將l_n中的值加總得到最終Loss。

VI. Experiment

在本章節中將實作的model，主要包含LeNet5 pytorch version以及tensorflow version，此資料集包含63325張影像作為training set，450張影像作為validation set，450張影像作為testing set，並觀察預測的top-1&top-5準確率，本次測試用於20-core server with 512GB memory，再加上NVIDIA Ampere A100的GPU的電腦設備。

A. Evaluation Metric

在於實驗的階段使用的Evaluation Metric是Top-N accuracy。Top-N accuracy 是一種評估分類模型效果的指標，其計算方式是看在模型預測的前 N 個最有可能的分類標籤中，是否包含了真實標籤。Top-N accuracy 的值越高，表示模型的預測能力越好。公式如下：

$$Top - N = \frac{|return\ Top - N\ label \bigcap true\ Top - N\ label|}{Number\ of\ image} \tag{3}$$

B. Result

TABLE I
Hyper-parameter configuration.

Hyper-parameter	learning rate	batch	Epoch
Value	0.01	256	10

TABLE II
Result.

		model		
		LeNet5 Numpy	LeNet5 Torch	LeNet5 TF
Val	top-1	0.02	0.02	0.1222
	top-5	0.1	0.01	0.4244
Test	top-1	0.02	0.02	0.1756
	top-5	0.1	0.1	0.3867
Runtime		9287sec	109sec	40sec

可以由上述的表格中看到，在於預測的結果上Numpy以及Pytorch的版本上都有Overfitting的問題，使得預測結果最終都會預測到同一個類別，因此準確性都是top-1等於0.02，top-5等於0.1，設想的原因可能是因為原始圖片size皆不一致，而為了實驗與測試的簡易性，統一將圖片轉換成32*32的灰階影像，導致影像可能有大量的資訊流失，且在於pytorch上沒有特別在做轉化使得預測結果來的較差，而在Tensorflow中因為使用Keras，在於framework的設計上有許多原生的套件防止模型過擬合，因此在於正確性上就有很明顯的提高，且在於epoch等於10左右即收斂，在於Runtime部分因為Numpy只能使用CPU進行計算，因此在於其餘兩種也都是透過CPU去運算，明顯看出運算時間上在於Pytorch與Keras上都非常的快，但Keras預設為進行多執行緒將所有CPU皆100%運算量，因此對於運算的附載量很大。

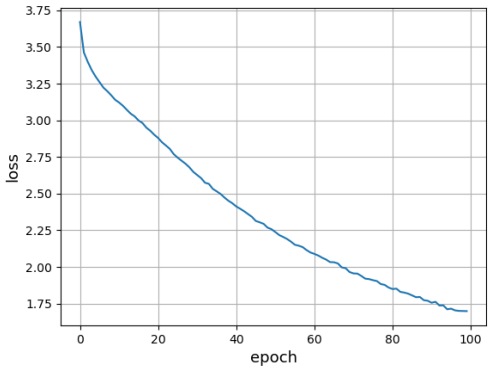

Fig. 2. LeNet5 loss.

在於loss的部分雖然loss呈現穩定的下降趨勢，但可以透過下圖Test accuracy中明顯看出，不管是在於top1 & top5

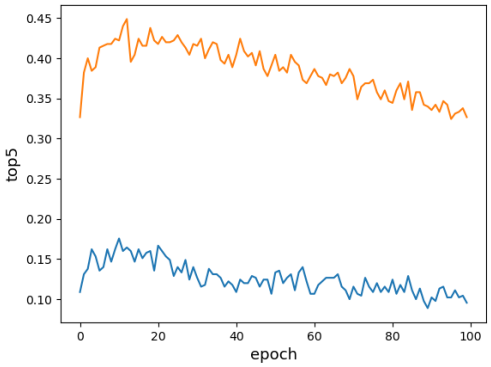

Fig. 3. top1 & top5 accuracy.

accuracy上基本上在於epoch為10左右就幾乎收斂，後續的正確性就開始下降，因此最終選定epoch等於10做結果呈現。

C. Ablation

TABLE III
ABLATION STUDY.

	model			
	LeNet5 Torch	+purning	+quantization	+both
Runtime	115sec	117sec	257sec	292sec

在於此部分測試了兩種Compression method對於實驗的影響，分別是pruning以及quantization，但因為測試的data與之前的前處理方式皆相同，因此在於正確性的部分沒有解決掉在於pytorch上overfitting的部分，而不多加贅述。

在於pruning的部分測試的時間結果幾乎與異且內存也幾乎沒有減少，因為是經由pytorch的prune做pruning，其實踐方式透過添加了名為weight_origand以及weight_mask的新參數，將weight_mask做一個遮罩，裡面由0,1組成，去選取於該權重是否被修剪，而在forward與backward中將它們屏蔽掉，因此目前的修剪實際上需要更多的內存和更多的計算。

在於quantization上運算時間反而多上一倍，查詢後的原因可能為是因為post-training quantized本質上計算時是將int轉換成float計算的，因此中間存在量化和反量化的操作佔絕了些時間，可能會需要在更大型的網路架構上才看的見加速的趨勢。

VII. Conclusion

本次實作了多種不同版本的LeNet-5 model，且由於Pytorch與Tensorflow為兩種不同的AI框架，因此在於實踐上會有許多類似卻功能性不同的方法，在於熟悉的過程中也能夠更廣泛性的去了解不同的框架上的差異。

最終的結果不甚理想，未來可能會需要去回歸到數據本身，何種的前處理方式才能夠使得影響的資訊被提取出來，或者是其他的模型堆疊方式會使得預測準確性更加。

GitHub：
https://github.com/tenyang1999/DL_Assignment3_LeNet-5_with_torch-tf

References

[1] CROSSENTROPYLOSS
https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
[2] LeNet5-MNIST-PyTorch
https://github.com/ChawDoe/LeNet5-MNIST-PyTorch/tree/master
[3] Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning)
https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342
[4] pruning tutorial
https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/7126bf7beed4c4c3a05bcc2dac8baa3c/pruning_tutorial.ipynb#scrollTo=CD3TmeoUNtHy
[5] PyTorch Quantisation
https://colab.research.google.com/drive/1ptzMOHcU5IrtWaSjHvxGYsX6BozcxgF6?usp=sharing#scrollTo=9fL3F-7Rntog