IMLO Assessment Project

Y3912929

***Abstract*—This report gives an account for a project that was executed in the development of a neural network classifier. A deep learning approach was used, where a Convolutional Neural Network (CNN) was designed, built and implemented using the PyTorch framework in order to classify flower species from Flowers-102 dataset. The resulted classification accuracy was approximately 60 percent.**

I. Introduction

IMAGE classification is a fundamental problem in machine learning where the purpose is to classify an image into one of many predefined classes based on its content. The aim of this project is to classify different flower species of the famous Flowers-102 dataset. Initially introduced by Nilsback and Zisserman in 2008, the research of the dataset provided a strong foundation for evaluating classification algorithms and different techniques for classifier designs [1].

The primary vision of this project is to create and train a neural network classifier with 2D dimensional inputs, allowing it to predict the species of a flower based on the image shown. Some of the efficient ways of tackling the problem of data classification include approaches like decision tree algorithms, per-pixel approaches and transfer learning that can reach a very high level of accuracy [2]. However, for the purposes of this project, the decision of proceeding with own design of the model was made.

II. Method

A. Initial Method Idea

While studying deep learning algorithms for image recognition, I came across LeNet-5, a leading CNN architecture (Lecun et al., 1998) specially designed for the recognition of handwritten numerical digits [3]. The LeNet-5 architecture uses a number of different fully connected layers in order to classify image data correctly. Motivated by this finding, I learned how the code presented a coherent, methodological method of coding CCN, step by step, and eventually understood the fundamental strategies underlying proper image recognition. I tried to apply the idea of the LeNet-5 architecture to train the dataset as a first attempt. However, it was soon discovered that the model was not suitable for handling RGB images.

Additionally, a disadvantage of this architecture is that it only accepts input images encoded at 32×32 pixels, while the preferred input size to be standardised at 256×256 pixels.

B. Image Transformation and Preprocessing Steps

In summary, images were variably processed before they reached my model. The images were scaled down to 256×256, rotated randomly up to 45 degrees, and flipped randomly in a horizontal direction. Finally, the images are converted into tensors and normalised using 0.1325 for the mean and 0.3105 for standard deviation.

After this procedure, the dataset was loaded using the transformations mentioned earlier. It is divided into training, validation and test sets. The training and validation sets are created using an 80-20 split of the original data. Dataloaders are created for each of these sets to provide efficient loading during training and evaluation. It is worth mentioning that dataloader wraps the dataset and creates an iterable interface. However, a key point is that the data in training dataloader is being shuffled. It prevents the model from seeing the images in the same sequence every epoch, which helps in generalisation. In the validation and test dataloaders, the data is not being shuffled because shuffling is unnecessary for evaluation.

Validation data becomes a major point through training a model in which it checks the performance of the model in training after each epoch. That would also avoid overfitting to help with early stopping decisions if the model's performance was not improved.

To visually illustrate the preprocessing, a random image from the dataset is selected and displayed before and after the transformations. For the purpose of fetching the original image, the original dataset is created. By doing this, we can get a better understanding of the impact that the transformations have on the images.

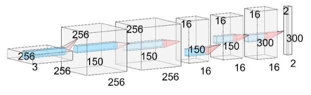

Fig. 1. Diagram of the neural network architecture

C. The Architecture of the Network

After doing a deep research, the decision to use CNN was made. CNNs have slowly emerged as the mainstream algorithm for image classification since 2012. Moreover, CNN architectures are applied to other visual recognition tasks such as object detection, object localization, and semantic segmentation [4].

The network has 4 convolutional with 2 fully connected layers. Images are input into the network in batches of 32, in three channels: RGB. Meaning we pass 32 images per batch in the training, with each individual image sized at 256x256. Each convolutional layer uses a stride of 1, ensuring that every input value is considered. A 5x5 kernel is implemented on the layers. A simplified diagram of the network architecture can be seen in Figure 1.

After each of these, ReLu is applied. This helps the model to learn more complex patters by setting all negative values to zero.

Following ReLU, Max Pooling is used in two different layers. The first layer takes the value of 16, meaning it reduces the spatial dimensions of the feature map by a factor of 16. The second layer takes the value of 8, and does the same but by a factor of 8. The decision to use this approach was due to its effectiveness. The first Max Pooling function aggressively reduces the computational burden while preserving key features. Similarly, the second function further simplifies the feature map, helping the network focus on the most important features and reduce complexity even more.

Then Batch Normalisation is applied, which helps to stabilise and accelerate the training process. Right after, an Adaptive Average Pooling layer is used, to reduce each channel to size of 1x1. This ensures that regardless of the input size, the output size is fixed, making the network more adaptive to different input dimensions.

The output of the adaptive average pooling, then, is flattened with the appropriate function, passed through a linear layer, followed by a dropout layer, activated with a sigmoid, linearly layered again, and finally log-softmaxed. It is worth mentioning that the DroupOut layer between fully connected layers also helps with fighting the overfitting problem.

D. Loss Function and Optimisation

For the loss function, the built-in PyTorch function (NLL-Loss) was applied. Similarly, the optimisation function was Adam algorithm. The decisions were made based on their popularity and success in the field of image classification.

III. Results and Evaluation

A. Evaluation Metrics

Performance of the model at every epoch is stated on the training and validation datasets in terms of accuracy and loss. It gives the accuracy percentage of correctly predicted images, whereas the loss gives the result in numeric form. At the end, we evaluate the model's performance based on the accuracy and loss respectively. As mentioned previously, the early stopping technique is also implemented, based on the validation loss.

B. Various Experiments

Different experiments were made, in order to improve the model. This involved changing the hyperparameters, trying another built-in optimisers, changing the structure of the model, in order to make it more simple, and trying various data transmutation techniques. From my perspective, the results were varying in approximately 2-3 percent, with the highest result being 62.1 % with 125 epochs, which took approximately 3 hours to complete. For any other experiments or results, the code execution time ranged from 30 to 60 minutes, respectively.

Description	Num of Epoch	Accuracy
(1)Changing the channel parameters of the model. Channel parameters: 3x64x64, 64x64 and etc	39	23.3%
(2)Running the modified model without Sigmoid function	26	51.9 %
(3)Running the modified model with STG optimiser	-	0 %
(4)Running the modified model as usual	25	57.9%
(5)Running the modified model without using Early Stopping technique	30	58.4%
(6)Running the modified model as usual	50	58.4%
(7)Running the modified model without using Early Stopping technique	125	62.1%

C. Results and Initial Conclusions

One of the first conclusions was that the drop in the number of channels reduces the complexity of the model, which, in this case, does not provide enough capacity to capture features and patterns from the data, reducing the accuracy. The results with and without Early Stopping technique showed that even though it is designed to avoid overfitting, it might stop the training process too early, resulting in lower accuracy levels. For the optimizer of STG, it may have other settings that are not proper for it, for the current model. I would like to note that before after applying STG, multiple errors and problems came up, which were fixed before the actual experiment.

D. Hardware Component Used

For this given project, a machine with both Intel UHD Graphics and an NVIDIA GeForce RTX 4060 Laptop GPU was used. The conditions were perfect as the video card enhanced the performance, making it ideal for the project.

IV. Conclusion and Further work

In conclusion, this is a relatively good level of progress, with improvements still needed. There is very little to add to this model, according to me, except that simplicity should be an attempt, which was far from the mark. In addition to that, improving Early Stopping technique could also benefit the project, because of the cases where the technique stooped the training early, not allowing the project get high results. Lastly, I also want to note that this project has a lot of potential, especially if one takes into consideration the huge progress made with respect to transfer learning. Utilising pre-trained models in this context could achieve unbelievable performance levels.

References

[1] Nilsback and Zisserman (2008). Automated Flower Classification over a Large Number of Classes. (Accessed: May 2024)

[2] D. Lu and Q. Weng, (2007). A survey of image classification methods and techniques for improving classification performance. International Journal of Remote Sensing, [online] 28(5). Available at: *https://www.tandfonline.com/doi/pdf/10.1080/01431160600746456.* (Accessed: May 2024)

[3] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278–2324. doi:https://doi.org/10.1109/5.726791. (Accessed: May 2024)

[4] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S. and Miao, Y. (2021). Review of Image Classification Algorithms Based on Convolutional Neural Networks. Remote Sensing, 13(22), p.4712. doi:https://doi.org/10.3390/rs13224712. (Accessed: May 2024)