Image Classification of Oxford Flowers-102 Using PyTorch

Y3921282

***Abstract*—Image classification is the task of labeling images into distinct categories. This task is best handled by Convolutional Neural Networks(CNNs); this report presents the use of a CNN model constructed in Python with the use of the PyTorch library to classify 102 types of flowers using the Oxford Flowers-102 dataset. The model achieved an accuracy of 46.9% on the test data, bringing into question the challenges and future direction of CNNs.**

I. Introduction

SINCE the creation of the Perceptron, the first neural network, in 1957[2], machine learning has become an ever-growing field. Today, artificial intelligence (AI) has revolutionised many fields, due to significant advancements in natural language processing, autonomous systems and, notable to this report, image classification. Convolutional Neural Networks(CNNs), a class of neural networks using filters (also referred to as kernels) to allow the neural network to process images as input and learn from their features. To evaluate the use and effectiveness of CNNs, this study applied them to the task of image classification on the Oxford Flowers-102 dataset. This CNN was to be constructed using Python's PyTorch library.

To maintain the integrity of this study.,constraints were imposed: The CNN must not be pre-trained or initialised with pre-set weights and biases and should not be trained on anything other than the train split of the dataset; The CNN may not be constructed by the code from any existing model. These help ensure fairness in evaluating the model's applicability.

II. Method

CONVOLUTIONAL neural networks are deep feed-forward neural networks beginning with an input layer where an image is fed to the network.

Next is the convolutional layer where learnable filters are applied to the input images by computing the dot product between the filters and areas of the image to extract useful features in the form of feature maps.

These maps are then passed to an activation function, introducing non-linearity to the network to allow approximation of complex functions.

The output of this is passed to a pooling layer, which performs down-sampling operations on the output matrix to reduce the feature maps dimensionality which helps retain the most important features while reducing computational load.

The network may be composed of multiple sequences of convolutional and pooling layers, increasing the model's depth. A larger depth allows the model to recognize more complex features.

Once the appropriate depth is reached, the feature map is then flattened into one dimension so it can be processed by fully connected layers which connect a number of input neurons to a differing number of output neurons, allowing the number of output neurons to be reduced to the number of output classes, to successfully classify the input image.

For the model to improve its classification abilities, backpropagation must be applied. A model must have a loss function to assess the model's performance at labeling models. Backpropagation is then used to calculate the gradient of this loss function with respect to the weights and biases of each layer. The calculated gradient is then used in an optimization function which fine tunes the aforementioned weights and biases for improved classification with lower loss.[2]

As Flowers-102 is quite a small dataset, it should be noted that data augmentation was applied to the training set of 1020 images to increase the diversity of the data. This must be done to present the model overfitting to details in the training data, making it worse at labeling unseen data, such as those in the training set[3]. Training images were first randomly cropped and then resized to a 256x256 image, then transformed to be from a random perspective, some randomly mirrored in the y axis, randomly rotated and then applied through a colour jitter algorithm. Not only do these increase diversity but they help to mimic conditions of other cameras, meaning that the model will not opt to classify by the image quality rather than the type of flower. This transformed image is then transformed into a tensor and normalized using the mean [0.485,0.456,0.406] and [0.229,0.224,0.225] the industry standard, calculated over numerous images[1]. This normalisation allows for efficient computation on these tensors by centering the tensors' values around zero with a standard deviation of one. The test data and validation data is resized to 256x256 as to match the input size of the training data, then transformed to a tensor and normalized.

The model constructed for this study was built from the PyTorch CNN boiler plate and then developed on further[4]. The model begins with a convolutional layer with three input channels(the rgb values of each pixel), sixteen output channels and a filter size of three. This is then batch normalised and passed to the activation function. For this model the Rectified Linear Unit (ReLU) activation function is used where $\text{ReLU}(x) = \max(0, x)$. This is used as it prevents the loss of smaller gradients during backpropagation. then passed to a max pooling layer of filter size two and stride of two. Max pooling is a type of pooling which selects the maximum value within the filter and creates a new feature map downsized to only the maximum values in each pooled region.

This is repeated two more times with convolutions of the

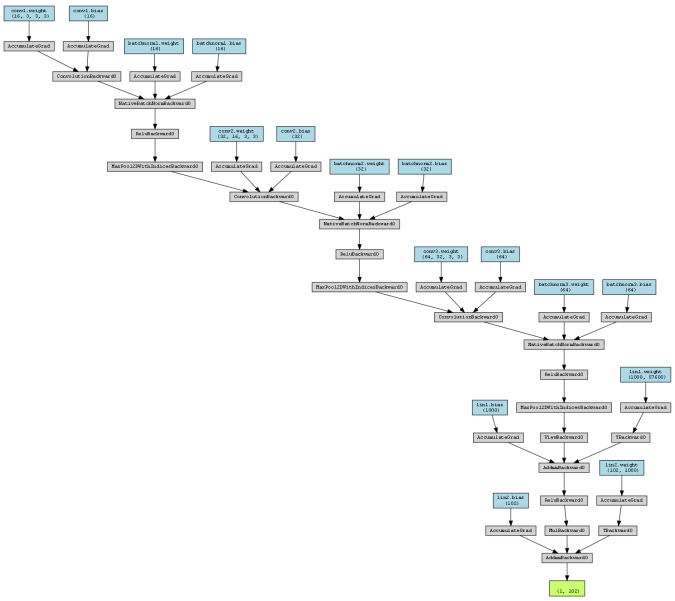

Fig. 1. A diagram showing the CNN architecture

same filter size, increasing the size from 16 to 32 and 32 to 64 and the same activation function, normalisation and then max pooling layers applied.At this point, it was deemed that ample features had been extracted from the input data, without overprocessing, which takes extra computing power and can lead to overfitting if too specific of features are collected.

The output after this is flattened into a one dimensional matrix of length 57600. This allows the feature map to be passed into fully connected linear layers for processing. This is passed to the first linear layer with 57600 input features and 1000 output features, significantly reducing the nodes in the network. The ReLU activation function is once again applied then passed through a dropout layer which is used to randomly zero connections to help avoid overfitting. Finally, this is passed to a linear layer with 1000 input features and 102 output features, each relating to one of the 102 classifications of flower in the dataset.

For training the model, PyTorch's cross entropy loss function was used. This contains the softmax function which converts output feature labels to predicted output probabilities between 0 and 1 giving the advantage that these values have meaning for backpropagation and wont cause recognition of meaningless patterns within the CNN. Cross entropy is calculated by $\text{CrossEntropy}(x) = -\log(p_x)$ where x is the prediction and p_x denotes the predicted probability of class x from the softmax. It should be noted that the log function used is the natural logarithm.Cross entropy is suitable for this model as the logarithmic error allows for larger corrective steps in backpropagation when error is larger, and smaller more precise steps when loss is minute. Backpropagation is done using stochastic gradient descent to optimise parameters, as this saves computation and handles output values of cross entropy loss effectively.

III. Results & Evaluation

Experiment Number	Test Accuracy	Changes made	Issues
1	1%	Original model	No breakout method to prevent overfitting
2	20.9%	Validation breakout added	Overfitting occurs too quickly
3	32.7%	Data augmentation added	model not deep enough to predict more complex features
4	42.3%	Another convolutional layer added	Model training taking too long to improve
5	46.9%	Learning rate increased by x10 and batch normalisation added	Can't breach 50% test accuracy

TABLE I
Experiment Results

DURIGN training, the model calculates its loss over a validation set. The model is set to break out after 1000 training epochs or when the training loss is 20% greater than the validation loss and increasing. This metric is used in comparison with training loss as validation loss increasing while the training loss is increasing is a sign of overfitting (due to the fact the model is not trained on the validation set), signaling that the model has reached its best generalized prediction accuracy. The model with the lowest validation loss is saved to be used on the testing data. A batch size of 32 was chosen to be used for this model as a comfortable balance for model efficiency and performance. For optimization, stochastic gradient descent was used due to its compatibility with cross entropy loss for adaptive step size. Furthermore, due to stochastic gradient descents estimative nature, it majorly saves on computing power and hence improves model efficiency.PyTorch's stochastic gradient descent also allows for weight decay, a measure that controls weight magnitude to counter overfitting. This was included the model to ensure a good final predictivew accuracy.

The model's accuracy was evaluated on a testing set of 6149 images kept seperate from the 1020 training and 1020 validation images to prevent data leakage for an accurate prediction accuracy and trained on an Nvidia 4070 GPU within Visual Studio, taking around 3 hours to reach a breakout.The model achieved a final accuracy of 46.9% after a validation loss triggered breakout. This was achieved with a learning rate of 0.01.

IV. Conclusion & Further Work

IN conclusion, although the performance achieved makes the model useful only as an aid than as a trustworthy classifier, it can still be said that the architecture is a good choice. This is because the main limiting factor of this model is the size of the training data. 1020 images to train on makes it difficult, even with extensive data augmentation, to prepare a CNN to adapt to variations in image. For further work, more complex data augmentation could be added along with more convolutions to extract the more complex features, this could not be done during the study due to limitations on computational power and time.

REFERENCES

[1] Pytorch tutorial: How to normalize an image with mean and standard deviation. https://www.tutorialspoint.com/pytorch-how-to-normalize-an-image-with-mean-and-standard-deviation.
[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
[3] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
[4] PyTorch. Training a classifier. https://pytorch.org/tutorials/beginner/blitz/cifar10$_tutorial.html$.