Designing a Deep Learning Model for Image Classification

Introducing a Modified LeNet Model based on CNN Architecture for Image Classification

I. Introduction

As part of the broader domain of computer vision, image classification refers to the processes of categorization and labelling of images into pre-defined categories (or classes) based on their content, given a set of pre-defined rules. It is an important feature of computer vision and has many real-world applications, like in autonomous driving – where image classification is used to map and recognise objects in real time.

This paper explores the methodology in creating a deep learning-based image classification model using convolutional neural network (CNN) architecture. The proposed model is an adaptation of the widely known LeNet-05 model, introduced by LeCun et al in 1998[1]. The models are created in Python using the Tensorflow 2.10.1 package.

The main objective of this project is to train an image classification model to categorise images from a pre-existing dataset into the respective classes using CNN. The LeNet model is introduced as a baseline approach, after which a modified version with adjustments to structure and hyperparameters is introduced, to enhance findings and improve model accuracy.

II. Data & Preliminary Analysis

The data for this project includes a privately curated set of 13 394 images in .JPEG format, classified into ten distinct categories (or classes).

The dataset was partitioned into training and validation sets with a 70:30 ratio, wherein 70% of the images were allocated towards training the model, and 30% were allocated to the validation process. Within the validation set, the data was split further with an 80:20 ratio, wherein 20% of the images within the validation set were now allocated towards testing (i.e., prediction of the classes) in the model.

Fig. 1 shows the visualisation of image data, as generated by a matplotlib graph. The images are categorized into ten distinct categories, depending on the objects present in the images.

Given the nature of the dataset, training this model is more reminiscent of training an object recognition model, wherein the computer is trained to recognize the broad category to which an object in the image belongs. For instance, in Fig. 1, the computer classifies the image of the dog as belonging to the class 'n02102040'. If this class consists of just images of dogs, then essentially the model is being trained to recognise images of dogs as 'n02102040'.

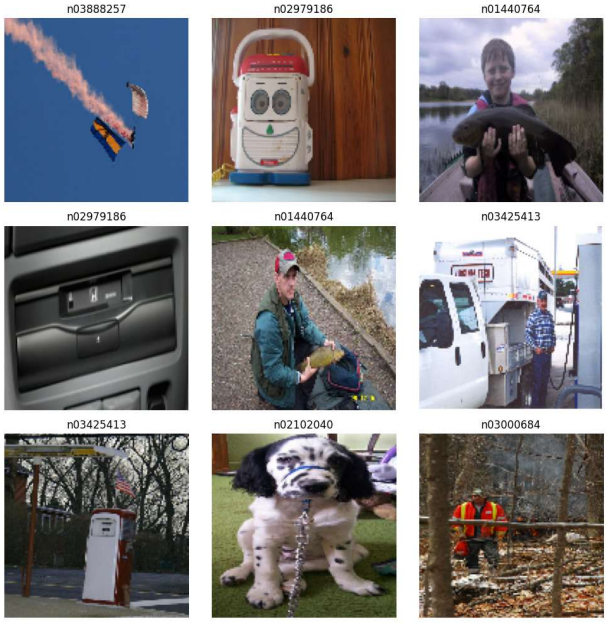

Fig. 1. Visualisation of Image Data

III. Methods

Image classification is performed by first building a LeNet model that can assign labels to images such that the labels denote the class to which the image belongs. An image presented as input in the model is analysed, and a label corresponding to the class of the image is returned. This model is then improved upon through structural changes and hyperparameter tuning, and the resulting modified model is compared to the baseline for evaluation.

A. Experimental Setup

The hardware configuration used consists of a device running Windows 11 and equipped with an NVIDIA GeForce RTX 3060 Laptop GPU, which was used to train the model with many epochs.

The model was built using the Python programming language, in Python v3.9.16 through a conda-based virtual environment built specifically for training models on the equipped GPU.

TABLE I. Python Packages Used for Modelling

Package Name	Version	Purpose of Use
Tensorflow	2.10.1	Model-building
Keras	2.10.0	Model-building
Numpy	1.24.2	Creating arrays of data to be used with the model
Matplotlib	3.7.1	Plotting graphs which visualize data & model performance metrics

Table 1. Details of Python packages used for modelling.

B. Model Structure

A deep model based on CNN architecture usually includes four types of layers, which are as follows:

The Convolution layer applies a set of kernels to the image to produce a set of output features. Multiple convolution layers help to extract multiple feature sets, which help improve classification outcomes.

Each layer has a corresponding Activation function, which in essence, defines how the weighted sum of the input is transformed into an output from node(s) in the layer.

Each convolutional layer is created with a corresponding Pooling layer, which reduces the dimensionality of the output from the previous layer in a structured manner. It helps in reducing overfitting and compresses the number of data and parameters as well. While the original LeNet-05 model uses an AveragePooling layer, the modified model uses a MaxPooling layer, as image classification in this instance requires greater emphasis on presence and shape of the object.

The Flattening layer, as the name suggests, 'flattens' the output from the preceding convolutional & pooling layers by converting them into one-dimensional arrays, which will then be passed through the fully connected layer(s).

The Dense layer is a fully connected layer, in which each neuron receives input from all neurons in the previous layer to produce a single output value, which in this case is a classification label.

The Output layer is the last layer in the model, and has as many neurons as there are number of classes in the dataset. The appropriate activation function is chosen depending on the expected output from the model.

C. Training Process

The size of each batch in the training process is set to 32, such that training and validation occurs in batches of 32 images. The number of epochs in the training process is set to 25, to allow for observation of improvement trends in the model.

D. The LeNet-05 Model

The architecture of the LeNet-05 model consists of two convolution layers, two corresponding pooling layers, one flattening layer and three dense layers.

The first convolutional layer consists of 6 filters with a 5x5 kernel and produces 6 feature maps of size 28px*28px. The ReLU activation function is used, which improves classification accuracy due to its properties.

The second layer consists of 16 filters with a 5x5 kernel and produces 16 feature maps of size 10px*10px. The ReLU activation function is used here as well.

With regards to the pooling layers corresponding to each convolutional layer, the AveragePooling2D method is used, which needs no parameters.

The first dense layer has 120 neurons and uses the ReLU activation function, while the second layer has 84 neurons and also uses the ReLU activation function. The third dense layer This layer uses the softmax activation function, such that the class with the highest probability will be the predicted class for the image. This function is preferred in this context as we deal with classification of images.

After the model is constructed, it is compiled using an optimization function, which measures the performance of the model in the classification process by measuring the differences between the predicted outputs of the model and the true output of the training data. The LeNet-05 uses a Stochastic Gradient Descent (SGD) optimization function, which has a fixed learning rate and is less memory-intensive to run.

IV. Experiments

A. Modifying the LeNet-05

Here is where the LeNet-05 architecture was modified, taking into account the needs for this particular dataset.

The LeNet-05 model was developed to be used with MNIST dataset of handwritten digits, and the parameters of this model were more suited for the characteristics of the images in the dataset (which were grayscale images of size 32px*32px). The images used in the current dataset are more complex, and thus would require some structural modifications, alongside some hyperparameter tuning to be more effective.

a) Hyperparameter Tuning

The parameters of the existing convolutional layers were modified, such that the new layers now consist of sixty-four filters with a 3x3 kernel and produce 64 feature maps of size 128px*128px. The ReLU activation function is used.

A third convolutional layer was added, consisting of 32 filters with a 3x3 kernel and produces 32 feature maps of size 128px*128px. The ReLU activation function is used to improve the classification accuracy.

One of the Dense (Output) layers has been removed. The model now has two Dense layers, with one layer having 64 neurons and uses the ReLU activation function, and the second and final layer having 10 neurons (corresponding to the number of classes in the dataset) with a ReLU activation function.

The model is now compiled using the Adaptive Moment Estimation (Adam) optimization function, which has a dynamically adjusted learning rate, is more robust to outliers, and is faster and more efficient for larger datasets. It is, however, more memory-intensive than the SGD, but the trade-off may be bearable if it provides higher classification accuracy.

B. Comparisons of Evaluation Metrics between the models

1) Test Accuracy

TABLE II. Test Accuracy of Both Models

Model Name	Test Accuracy
LeNet-05	0.5136
Modified LeNet	0.6321

There is a remarkable increase in the test accuracy when the modified LeNet model is introduced, which is expected given the hyperparameter adjustments performed.

2) Loss Function

The loss function is used to find error or deviation in the training process. Both models use a similar loss function, which is ideal for comparisons of performance.

a) LeNet-05

OVERFITTING / UNDERFITTING

In the LeNet-05 model, there is some overfitting occurring.

While the training loss decreases as the number of epochs increases, the validation loss decreases only until a certain turning point as indicated in Fig. 4, after which it rises steeply. It seems that the model is being subject to over-fitting when there are more than 10 epochs in the training process.

While this model was run with 20 epochs, it seems it would have been sufficient to train the model with just 10 epochs to ensure maximum accuracy and minimum over-fitting.

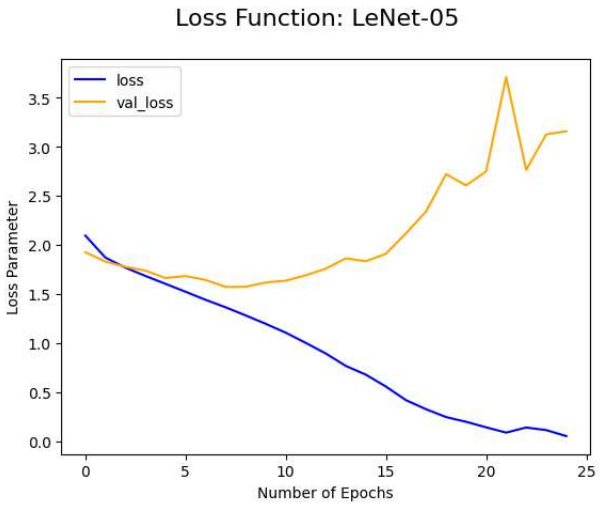

Fig. 2. Loss Function from the LeNet-05 model.

REPRESENTATIVENESS OF THE DATASET

The loss function can also provide insights into the representativeness of the training and validation dataset. and can indicate if the training dataset and validation dataset both have proportional statistical characteristics.

It is generally assumed that the training and validation datasets will have proportionally statistical characteristics, given how they are generated, but because the dataset for this project has been provided in the form of training and validation datasets, this step is included as a precaution.

While the larger gap between the training and validation loss graph lines beyond the turning point can be attributed to over-fitting, the portion below the turning point indicates that the representativeness of the datasets is indeed maintained, given the narrow gap between the graph lines. This is demonstrated in Fig. 5, where the loss function is magnified so as to measure the gap between the graph lines.

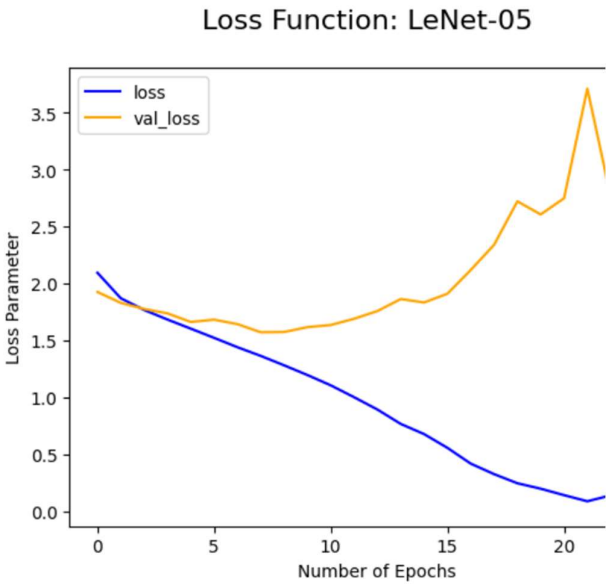

Fig. 3. Magnified image of the Loss Function from the LeNet-05 model to show the gap between the training loss & validation loss graphs.

b) Modified LeNet

OVERFITTING / UNDERFITTING

There seems to be quite a lot of over-fitting in the modified LeNet model as well.

The training loss does decrease as the number of epochs increases, but the validation loss reaches the turning point within a fewer number of epochs (approximately 5, as compared to 10 epochs on the LeNet-05), after which the gap between the training loss graph and validation loss graph widens significantly – a key indicator of over-fitting in the model.

While this model was run with 20 epochs, it seems that this model "trains faster" in just 5 epochs – perhaps due to the use of the Adam optimization function (which has a dynamically adjusted learning rate). Perhaps it would be sufficient to train the model with just 5 epochs to ensure maximum possible accuracy and minimum overfitting.

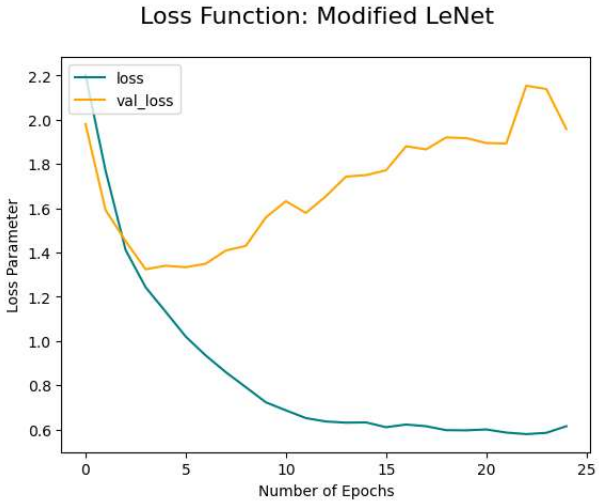

Fig. 4. Loss Function from the modified LeNet model.

REPRESENTATIVENESS OF THE DATASET

While the larger gap between the training and validation loss graph lines beyond the turning point can be attributed to over-fitting, the portion below the turning point indicates that the representativeness of the datasets is indeed maintained, given the narrow gap between the graph lines. This is demonstrated in Fig. 4, where the loss function is magnified so as to measure the gap between the graph lines.

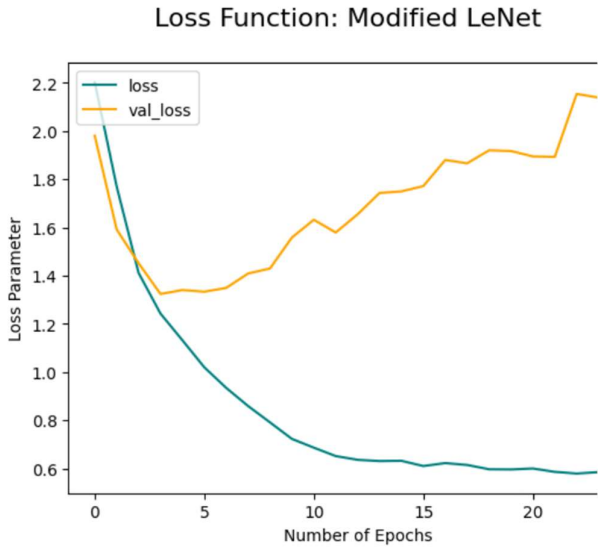

Fig. 5. Magnified image of the Loss Function from the modified LeNet model to show the gap between the training loss & validation loss graphs.

3) Accuracy Function

The accuracy function maps the change in accuracy of classification of training and testing data over the entire training process.

While it is expected that the training accuracy improves as the number of epochs increases, the validation accuracy indicates how well the model can predict the classification of images after it has been trained.

a) LeNet-05

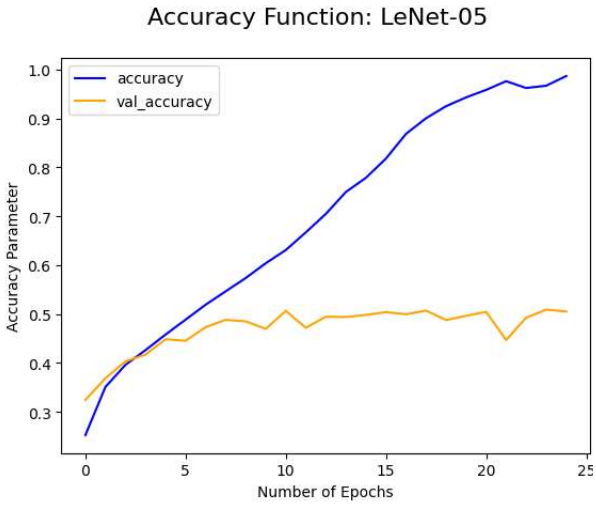

Fig. 6. Accuracy Function from the LeNet-05 model.

For the LeNet-05 model, it seems that the validation accuracy floats about in the range of 40-50%. This indicates that the trained model can accurately classify about 40-50% of images in the dataset. This indicates that despite the simplicity of this model, it can still correctly classify the images about half the time.

It is also noteworthy that the test accuracy is quite close to the validation accuracy shown in the graph, at just about 51%.

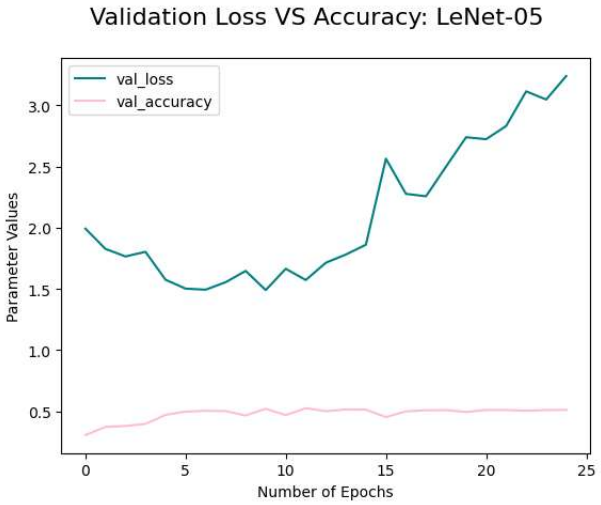

Fig. 7. Comparing Validation Loss & Validation Accuracy for the LeNet-05 model.

As the loss curve shows higher than the accuracy curve for the validation dataset, this indicates that the model probably makes big errors in most of the data, which does not bode well for the classification capability of the model. This does reflect in the test accuracy of the model, where only about half the image data is correctly classified by the model.

b) Modified LeNet

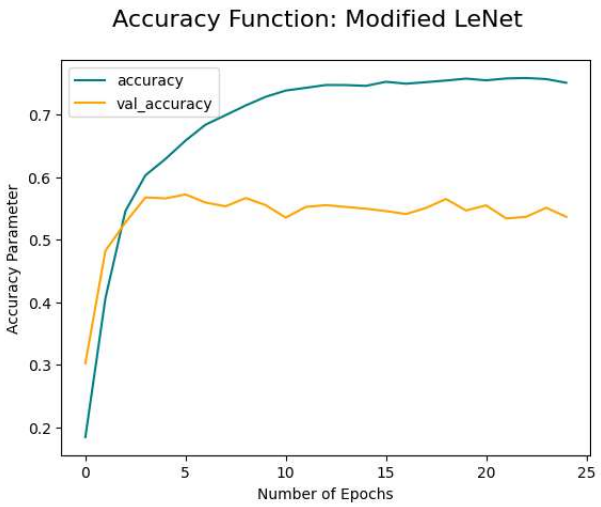

Fig. 8. Accuracy Function from the modified LeNet model.

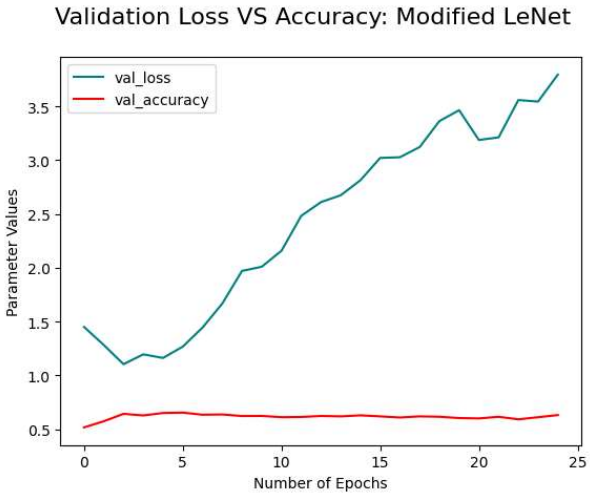

Fig. 9. Comparing Validation Loss & Validation Accuracy for the modified LeNet model.

We observe a similar problem as in the LeNet-05 model here. The loss curve is still higher than the accuracy curve, indicating that the model can make big errors in most of the data. While the test accuracy of this model is much better than that of the LeNet-05, this is far from the ideal scenario with high accuracy and low loss.

V. Reflection

This project is a very elementary attempt at creating an image classification model that can recognize distinct features in an image and classify them into their respective categories. While the classification accuracy of the model was not very satisfactory as intended, it would be interesting to observe how this model would behave when introduced to other image data outside the provided dataset.

However there are some key concerns regarding the viability of the model, with the main concern being that both models have high loss and low accuracy in classification, which was observed in the validation stage. This suggests that there could be room for further improvement in the model, perhaps by adding more convolutional layers to extract more features, or by simply adding more data to the validation dataset to observe if this was merely due to insufficient data.

More hyperparameter tuning might also be essential, especially with respect to the final output layer of the modified LeNet – it would be worth observing if modifying the ReLU activation function to a softmax activation function can help improve classification, given its nature. Some finer adjustments can also be tried out with respect to the Dense layers, by modifying the number of neurons in the layers. However, given the simplicity of the model, it is quite commendable to see 50-60% test accuracy levels.

References

[1] X. Li and Y. Guo, 'Adaptive Active Learning for Image Classification', in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA, Jun. 2013, pp. 859–866. doi: 10.1109/CVPR.2013.116.

[2] Y. Jingyi, S. Rui, and W. Tianqi, 'Classification of images by using TensorFlow', in *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, Xi'an, China, Apr. 2021, pp. 622–626. doi: 10.1109/ICSP51882.2021.9408796.

[3] A. B and K. M T, 'Convolutional Neural Network Based Image Classification And New Class Detection', in *2020 International Conference on Power, Instrumentation, Control and Computing (PICC)*, Thrissur, India, Dec. 2020, pp. 1–6. doi: 10.1109/PICC51425.2020.9362375.

[4] S. Mondal, K. Agarwal, and M. Rashid, 'Deep Learning Approach for Automatic Classification of X-Ray Images using Convolutional Neural Network', in *2019 Fifth International Conference on Image Information Processing (ICIIP)*, Shimla, India, Nov. 2019, pp. 326–331. doi: 10.1109/ICIIP47207.2019.8985687.

[5] P. Zhang, 'Satellite Image Classification Based on Convolutional Neural Network', in *2021 3rd International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, Guangzhou, China, Dec. 2021, pp. 754–757. doi: 10.1109/IAECST54258.2021.9695931.

[6] Ramsundar, B., & Zadeh, R. B. (2018). TensorFlow for deep learning : from linear regression to reinforcement learning / Bharath Ramsundar and Reza Bosagh Zadeh: From linear regression to reinforcement learning (First edition.). O'Reilly Media.