Deep Learning

Usman Ali

April 16, 2024

Applications of Deep Neural Networks in Robotics

Deep learning is a subset of machine learning that involves the training of artificial neural networks [3]. Neural networks are computational models that mimic the information processing of human brains. They are composed of interconnected nodes or neurons that collaborate to identify patterns and make decisions. Computational models consisting of several processing layers can acquire representations of data with various levels of abstraction through deep learning [5]. This is achieved through the use of Deep Neural Networks (DNNs) which consist of multiple layers of interconnected nodes, each layer processing input from the previous one to build increasingly complex representations. This layered structure allows DNNs to learn from vast amounts of data and perform tasks such as image and speech recognition, language translation, and many others with remarkable accuracy.

Robotics applications for deep learning are numerous, and utilise a wide variety of models, for example, discriminative and classification models. Such models have made significant advancements in extracting meaningful interpretations from video footage and still images. A demonstration of such advancements is that of Ouyang et al [4], who used a single DNN to address four separate elements of pedestrian detection: classification, articulation and motion handling, occlusion handling, and feature extraction. This work has helped to ensure accurate pedestrian detection which is indispensable for the development of autonomous vehicles, facilitating the safe use of such machines among humans as well as being important in the domain of intelligent video surveillance.

Unsupervised learning models are an additional type of model used in the field of robotics. Deep spatial autoencoders are an example of unsupervised learning models, particularly known for their ability to construct state spaces without explicit supervision. This is exemplified by the approach of Finn et al [1], automating the extraction of visual features to achieve visuomotor control. The extraction of features enabled the robot to learn motion skills using reinforcement learning methods, and continuously improve on such skills. This dual approach of unsupervised feature extraction followed by reinforcement learning facilitates the development of complex robotic actions, such as pushing a free-standing toy block, picking up a bag of rice using a spatula, and hanging a loop of rope on a hook at various positions.

Recurrent models are a class of neural networks that are proficient at learning in order to predict complex interactions. Recurrent models make use of their internal state enabling them to maintain information from previous inputs [5]. As a result, these models are useful in simulating the impacts of time in a changing environment, thanks to their knowledge of current and previous states. One usage of recurrent networks is demonstrated by Jain et al [2], where models are trained to predict traffic manoeuvres in human-driven vehicles to improve collision avoidance systems. The architecture involved made use of Long Short-Term Memory (LSTM) units to remember information over extended periods, which is crucial in tasks that require understanding sequences such as predicting traffic patterns. The model resulted in an average manoeuvre anticipation of 3.5 seconds before they occurred at a precision of 90% attesting to the effectiveness of this type of deep learning in real world machines.

The aforementioned paragraphs show a number of practical uses of deep learning within robotics. It is evident that these methods are playing an evolutionary role in the development of various technologies as well as improving on previously existing methods as shown by Ouyang et al [4] and Jain et al [2]. The example of the robot in [1], shows how deep learning can be used to enable object manipulation in a controlled environment. However, transferring this process into the real world can be difficult due to the unpredictability of environments and significantly more factors being in play, which highlights some of the difficulties of deep learning in general. Such obstacles underline the inherent difficulties in applying deep learning, notably the substantial datasets needed for training and the extensive computational and manual effort required to achieve adaptability in these advanced systems.

DNN classification Problem

As has been discussed, DNNs have significant capability in the field of image classification due to their ability to extract complex features. The DNN classification problem involves training these networks to accurately recognise and differentiate between various classes within a dataset. The dataset used was CIFAR-10, consisting of 60,000 32x32 images belonging to 10 different classes. One of the reasons for choosing this dataset is due to its size, facilitating reasonably fast experiments without putting significant strain on the computer's resources, making it suitable for the machine being used to carry out these simulations. Additionally, the dataset presents images that pose a substantive challenge for the model, with a complexity that could lead to misclassification even by human observers.

Methodology

The methodology used began with a simple MLP model, establishing a foundational point to build upon and obtaining a baseline measure of the performance of the model when classifying images for the CIFAR-10 dataset. More layers and nodes were incrementally added, increasing the complexity of the network after realising the need to extract more complex features in order to improve the accuracy of the image classification model. This gradual evolution in the network architecture was driven by the aim of increasing accuracy, enabling one to see the effects of different depths of network as well as how changes in hyperparameters — such as learning rate, kernel size, and the number of filters — can affect the model too. As a sidenote, ideally a grid based approach may be more suitable but with the available equipment and considering the large number of hyperparameters to be tested, this approach was not possible within the timeframe.

Details and Results

Initial Design

The initial design was made up of a flat input layer converting the 32x32 pixel images into a 1D array, followed by two hidden layers consisting of 128 nodes, allowing the model to learn a hierarchy of featurs suitable for this dataset, and an output layer of 10, reflective of the number of classes. Each hidden layer makes use of the ReLU activation function and an optimizer of SGD is used.

Table 1: Comparison of test accuracy and loss with different batch sizes

Batch Size	32	128	256
Accuracy	0.500	0.459	0.432
Loss	1.422	1.512	1.624

The first parameter to be tested and tuned was batch size; the number of training samples that are processed prior to updating the internal parameters of the model. The results in Table 1 show an increase in batch size resulting in a decrease in test accuracy as well as a greater loss. It seems more frequent updates lead to better generalisations on the test set. It seems that varying the batch size balances the trade-off between computational efficiency and model precision, as smaller batch sizes take longer to compute due to the more frequent updating.

Convolutional Neural Network

The next step in the process was to evolve the model into a very simple CNN in order to compare this model with the previous MLP. The simple CNN model consisted of one convolutional layer with 32 filters and a dense layer with 64 nodes. Another model identical to the previous one was also used but with 512 nodes in the dense layer to see what impact the number of nodes has on image classification.

From table 2, it is evident that as the number of nodes increases, the accuracy of the model also improves. This can be attributed to the increased capacity of the dense layer to learn complex patterns and representations from the data. With more nodes, the model has a greater ability to capture and process a

Table 2: Comparison of test accuracy and loss with different number of nodes

Number of nodes	64	512
Accuracy	0.515	0.550
Loss	1.389	1.268
Average time per epoch (s)	4	6

wider range of features, allowing for more accurate image classification. However, it's important to note that increasing the number of nodes also comes with a higher computational cost, demonstrated in the increase in the average time per epoch.

Optimizers

The optimizer is another hyperparameter that can be altered and result in significant differences in accuracy. All previous models used SGD as the optimizer but others such as Adam and RMSprop exist. Optimizers dictate how the model updates its weights in response to the error observed during training, aiming to minimise the loss function.

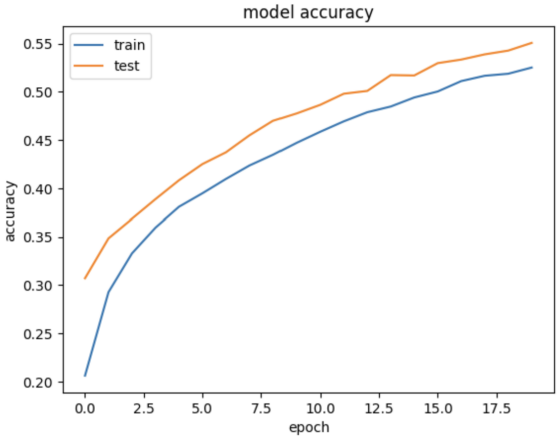

Figure 1: Model using SGD

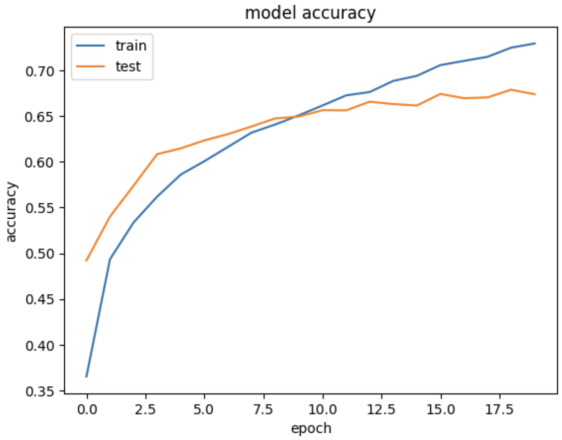

Figure 2: Model using Adam

Figure 1 shows a more gradual improvement in training accuracy when SGD is used compared to Adam, this is because SGD updates the weights uniformly and this results is slower convergence. Meanwhile, Adam makes use of values to adjust the learning rate for each weight individually allowing for more precise weight updates and faster convergence.

Convolution filters

Another parameter to be investiagted is the effect of the number of convolution filters used. The previous model made use of 32 filters, this was compared with a model using 8 filters. Consequently, the lower number of filters resulted in a reduction in the accuracy from 0.665 to 0.634. Using fewer filters in a convolutional layer results in a reduced capacity to extract and represent diverse features from the input image, thus hindering the network's ability to discriminate effectively between various features.

Pooling

Pooling in neural networks is a technique that reduces the dimensions of the feature maps, thus decreasing the number of parameters and computations needed in the network. Max pooling involves taking the maximum value from a group of neurons in the feature map, effectively summarising the most prominent features while discarding less informative ones. This process helps to reduce overfitting by providing an abstracted form of the features. Average pooling calculates the average of the values in the pooling windown instead of taking

the maximum. The average pooling model had an accuracy of 0.649, slightly worse than the max pooling model's accuracy of 0.665. The reason for this difference is due to max pooling's ability of being better in extracting salient features which is useful for tasks such as image classification.

Another intersting simulation with a larger max pooling window of size 4x4 ended up with a better test accuracy of 0.696, forcing the network to encapsulate the most salient features in a coarser representation, potentially enhancing the model's ability to generalise from the training data to the unseen test data. However, a window that is too large could result in the loss of important information having a negative impact on the model's performance.

Adding more convolutional layers

Adding more convolutional layers to a CNN enables the network to learn a hierarchical representation of features, starting with simple edges and textures and building up to complex patterns. Deeper layers can combine features from previous layers to form more abstract representations, which is very useful in image classification. However, adding more layers can lead to overfitting which can be resolved by regularisation techniques such as dropout. Dropout works by randomly deactivating a proportion of neurons in a neural network during each training iteration, which helps prevent the network from becoming overly dependent on specific neurons. The deeper network used made use of four convolutional layers consisting of 32 and 64 filters, as well as two instances of max pooling, and made use of dropout resulting in a test accuracy of 0.701.

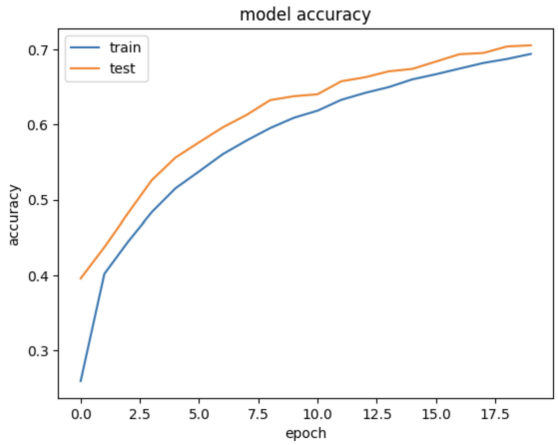

Figure 3: Model without batch normalisation

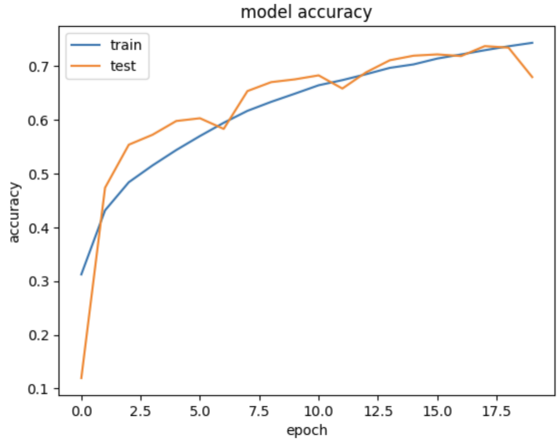

Figure 4: Model with batch normalisation

The effect of batch normalisation can be seen in figures 3 and 4. Batch normalisation results in a much faster rise in accuracy during the earlier epochs, showing how normalisation can increase learning speed. Moreover, there is much less of a gap between the training and testing data showing how this technique reduces overfitting.

Other regularisation techniques

Lasso regression (L1) adds absolute value of magnitude of coefficient as penalty term to the loss function. Ridge regression (L2) adds the squared magnitude of coefficient as the penalty term instead. Elastic net regression (L1_L2) combines both penalties, distributing weights and preserving feature selection qualities. As can be seen in the code repository, these regression methods proved to be significantly advantageous in reducing overfitting.

Table 3: Comparison of test accuracy and loss with different regularisation techniques

Regulariser	L2	L1	L1_L2
Accuracy	0.769	0.773	0.816
Loss	1.056	1.889	1.360

Learning Rate

Table 4: Comparison of test accuracy and loss with different learning rates for Adam

Learning rate	0.001	0.0001	0.00001
Accuracy	0.775	0.700	0.468
Loss	0.717	0.859	1.498

Table 4 shows using a lower learning rate than 0.001 with the adam optimizer results in worse accuracy and loss due to the network weights being updated at a slower pace, taking longer to reach the best local minima in the same number of epochs resulting in slower convergence.

Further simulations were carried out and can be viewed in the code repository; these simulations included experimenting with data augmentation as well as using different activation functions. Potential future simulations could involve testing larger learning rates to see if the trend of faster learning continues or does the learning rate become too large that it overshoots the optimal minima, leading to unstable training and potentially higher loss. Moreover, some simulations could be redone with significantly more epochs to see if the model generalises better to the test data or if it becomes overfitted to the training data, which is often indicated by a decrease in test accuracy even as training accuracy continues to increase.

References

[1] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[2] Ashesh Jain, Hema S Koppula, Shane Soh, Bharad Raghavan, Avi Singh, and Ashutosh Saxena. Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. *arXiv preprint arXiv:1601.00740*, 2016.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[4] Wanli Ouyang and Xiaogang Wang. Joint deep learning for pedestrian detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2056–2063, 2013.

[5] Harry A Pierson and Michael S Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.

A Source Code

For the source code and additional resources, visit the repository at GitLab Repository.