Individual Presentation Transcript

Hello and welcome to this presentation where we will be examining neural network models for object recognition.

In recent years, artificial intelligence has become increasingly prevalent in our lives. Deep Learning is a subset of AI and machine learning that relies on algorithms inspired by the functions of the human brain otherwise known as artificial neural networks or ANNs. ANNs have been causing a huge impact in many industries excelling in tasks such as speech recognition, instant language translation as well as the development of self-driving cars amongst other things. In this presentation, we will be focusing on another of its use cases, image recognition.

So, in this presentation, I will be developing a neural network using the CIFAR-10 image dataset for image recognition. CIFAR-10 is a widely used collection of images that are used to train machine learning and computer vision algorithms. The dataset consists of 60000 low-resolution colour images in 10 different classes. You can see over here a snapshot of the dataset, and you can see the 10 different classes starting from airplane and ending in truck. The dataset is divided into 50,000 training images and 10,000 test images.

I will be using a class of deep neural networks called convolutional neural networks or CNNs. CNNs have a unique architecture, which is particularly adept at processing data with a grid-like topology, such as images. At the core of the CNN is the convolutional layer, which employs filters or kernels. These filters slide across the input image, carrying out convolutional operations that detect localised features like edges, shapes, and textures. After the convolutional layer, there are the pooling layers. Their role is to progressively reduce the spatial dimensions of the extracted feature maps. This dimensionality reduction is important as it decreases the number of parameters, which minimises computational complexity. Finally, there is the fully connected layer. Here, neurons are intricately linked to all activations from the previous layer, bringing together all the information that the model has learned using the localised features and global patterns. It's within this layer that the final classification is determined (Alzubaidi et al, 2021).

So, our objective will be to develop and train a convolutional neural network using the CIFAR 10 training set, alongside a validation set, to accurately predict image classification on the test dataset. But why are we using CNN, well CNNs are particularly effective in image classification. Some advantages of CNN include their ability to learn spatial hierarchies of features automatically and adaptively from images which is beneficial for capturing the essential characteristics of images. (Huang, 2022). CNNs also have a history of being applied successfully in object classification across various domains including medical imaging and facial recognition (Ruofan, 2022). On the other hand, it’s important to note the limitations of CNN, first it requires large datasets for training to avoid overfitting which can be limiting in domains where data is scarce (Mikolajczyk & Grochowski, 2018). It can also be computationally intensive, especially if you are just using your local system (Schneider et al, 2021).

So, I’ll start by first inspecting and visualising the data. I decided to examine the shape of the training and test data and the corresponding labels. This will provide insight into the dimensions and structure of the data. So, for the training data here, the 50,000 represents the number of training samples in the dataset, the 32,32 represents the dimensions of each image, so each image is 32 pixels in height and width. And finally, the three signifies the number of channels in the images. Since CIFAR images are coloured, the channels correspond to the RGB (red, green blue) components of each pixel and the same happens for the shape of the test data. For the label data, the 50,000 matches the number of training samples and 1 represents the single label that is associated with each sample. I have also included a 3x3 grid of the first 9 images and put the corresponding label at the top, so you can see the first image is a frog, and the second a truck.

I have visualised the distribution of the classes in the training and test set here and you can see from the diagrams all the classes are perfectly evenly distributed. Now we can move on to preparing the data. I needed to normalise the data. This means adjusting the range of pixel intensity values found in the RGB images, so they fall between 0 and 1 and we do this by dividing by 255. This ensures all features are operating on a similar scale which helps with learning and training speed, and also prevents bias during weight initialisation (Zhang, S et al, 2021). Secondly, I categorised the data meaning I turned the 10 distinct classes into integer format. This will make sure it is compatible with classification algorithms, and loss functions and also make it compatible with one of the activations functions we will be using.

Lastly, before we dive into the model details, it's crucial to establish a validation set. This involves dividing our original training data into a revised training set and a separate validation set. The validation set serves as an unbiased evaluation of the models' performance during the training process by showing how the model performs with unseen data. It lets us know if the model is overfitting to the training data, and it also allows for the adjustment of hyperparameters without using the test set which we can reserve for final evaluation to ensure an unbiased assessment of the model’s performance (Yamashita et al, 2018). I have used the test train split function in the sci-kit learn library to create my validation set, the code can be seen at the top here and I’ve set the split to 20% which is a reasonable amount for it to effectively learn and generalise.

Moving on to the model discussion, I’ll first introduce the architecture of my initial baseline model and then detail the enhancements made through three stages, leading to the development of my final model. The foundational architecture remains consistent across models, so I will dive into the details of this now but then only highlight the specific modifications in the subsequent models.

So, this is the baseline model, I’ve got the code on the left and a visual representation of the model on the right here. This is a sequential model, meaning the layers are stacked linearly. In terms of the design elements, the architecture is inspired by the Visual Geometry Group or VGG style. This architecture is renowned for its simplicity and depth, use of small filters, stacking of convolutional layers and increase in depth while reducing spatial dimensionality (Simonyan & Zisserman, 2014).

My model contains three convolutional blocks, each composed of two convolutional layers, the layers are the yellow blocks here, and these layers use 3x3 filters, ReLU activation and padding is set to same which just keeps the spatial dimensions of the output the same as the input when the stride is 1. The first block has two convolutional layers with 32 filters each, the second block has two convolutional layers with 64 filters each, and the third has two convolutional layers with 128 filters each. This doubling pattern of the filters is a characteristic of VGG, where the depth of the network increases as the spatial dimensions are reduced (Simonyan & Zisserman, 2014). Following each set of convolutional layers, there’s a max pooling layer which is the red block here with a pool size of 2x2. This serves to reduce the spatial dimensions by half which reduces the number of parameters and computations in the network (Zhao & Zhang, 2024). This is also a common feature in VGG-style networks to introduce spatial hierarchy (Simonyan & Zisserman, 2014).

After the convolutional blocks, we get to the head of the model. The model flattens the output which is the green block here turning the 2D feature maps into a 1D feature vector, which is then passed to two fully connected layers or the dense layers, which is the blue block with ReLU activation. The first dense layer has 128 units, and the final Dense layer has 10 units coinciding with the 10 classes with a 'SoftMax' activation to output probabilities of the classes.

In terms of the optimiser, the model is compiled with the Stochastic Gradient Descent or SGD optimiser, and we are using ‘categorical cross-entropy’ as the loss function. The performance metric which will be recorded is ‘accuracy’. The model is trained using mini-batch sizes of 128 for 25 epochs, and it uses the validation data provided to monitor the performance after each epoch.

I want to delve into the rationale behind my selection of the chosen activation functions. I used ‘ReLU’ short for Rectified Linear Unit as the activation function for the convolutional layers and the initial dense layer of my model. Now ReLU is a very simple activation function and operates under a straightforward principle. For any given input, it outputs the value directly if it is positive, if the input is negative, the output is zero (Agarap, 2018) and here is a visual representation here on the right. So, this not only makes the model more computationally efficient, but it also adds non-linearity into our model which helps the model perform better with complex datasets which are more realistic (Agarap, 2018). This property also helps deal with the vanishing gradient problem which other activation functions like sigmoid and hyperbolic tangent have which is when gradients shrink to very small values during the training process which significantly slows down the learning process. Essentially ReLU is computationally efficient it has been shown to perform well in deep learning models helping them to train faster and achieve better performance (Cabanilla et al, 2024).

In the head of the model, in the final dense layer, the model uses a SoftMax activation function. Now SoftMax is typically used in the final layer of a in multi class classification models when you want to predict the probability the input belongs to one of the classes. This is because SoftMax converts a vector of values into a probability distribution, the outputs are a range from 0 to 1 and sum to 1. Essentially, it is taking the raw output scores from the previous layer, and it turns them into probabilities using the exponential function to ensure its positive. Because the scores are also normalised, this makes it possible to interpret the output of the probability distribution. Due to the exponential function, the differences between the scores are magnified, for example if one score is slightly higher then another, the corresponding probability will be significantly higher. This helps the network be more decisive about which class is the most likely. Essentially SoftMax provides meaningful interpretation of the model’s prediction making it clear which class is the most probable (Nwankpa et al, 2021).

I’ll now explain the choice of optimiser and loss function. The loss function measures model performance by calculating the discrepancy between its predictions and the actual data and the optimiser adjusts the model to minimise loss by using the loss function's feedback to update parameters (Hasan et al, 2022).

For the loss function, I have chosen categorical cross-entropy which is perfectly suited for multi-class classification. It works by comparing the difference between the model’s predicted probabilities, which we derive from the SoftMax activation function and the actual class distribution from the labels. It penalises incorrect class probabilities and guides the model towards accuracy (Kerkhof et al, 2023). The optimiser, SGD, is a variant of gradient descent where instead of using the whole dataset, it uses mini batches. This approach not only brings computational efficiency for large datasets, but it also adds an element of randomness that can be beneficial.

Finally, in terms of epochs, I’ve chosen to use 25. An epoch refers to one complete pass through the entire training dataset so essentially, I’m specifying how many times the learning algorithm will work through the entire dataset. Each epoch represents an opportunity for the network to learn and adjust its weights. With each pass through the dataset, the optimiser uses the loss function to update the model's weights to minimise the loss. However, you have to be careful as to many will lead to overfitting and will be computationally intensive, 25 was a good fit for me, the accuracy was still high and didn’t take too long to train.

So, I’ve gone through the core architecture of the model and now I’ve printed the model summary which details the layers and the parameters of the model. The parameter column indicates the number of parameters that are learned for each layer during the training process. These parameters are essentially the weights that the neural network adjusts through backpropagation as it learns from the data. The total number of parameters in the model indicates the model's capacity to learn from data. A model with more parameters has a higher learning capacity, which means it can model more complex functions. However, it also runs the risk of overfitting, especially if not enough data is available (Lu et al, 2022). The total parameter counts of 550,570 and that’s quite substantial so I would say this has a high capacity to learn.

So, after the model has finished running, I can evaluate its performance. The text in the box here is the output from the final epoch and the two graphs plot the cross-entropy loss and classification accuracy respectively for the training and validation data so we can compare performance. So, from the final epoch, I can see the accuracy for the training data was 0.69, meaning it was predicting correctly 69% of the images and the validation set was 0.57. This suggests the model is overfitting to the training data and it’s not performing as well on unseen data and that’s also supported by the fact that the average validation loss is higher than the training data loss. When we look at the diagram, we can see it was performing similarly on the validation data up until 10 epochs and then it started to plateau and fluctuate so we need to adjust this model.

For my second model, I chose to incorporate dropout layers which tackle against overfitting. Dropout, a regularisation technique, effectively forgets randomly selected neurons during training by setting their outputs to zero. This process, which can be thought of as a temporary amnesia, reduces the network's dependency on specific neurons, encouraging it to learn more general features. It's as if the network trains multiple versions of itself, each time with a different set of neurons turned off, thereby enhancing its ability to generalise to new data. This strategy makes the model more robust and less likely to overfit (Srivastava et al, 2014). I've set the Dropout rate to 20%, applying it after the max pooling layer and the first dense layer, to improve the network's generalisation capabilities on unseen data. I’ve got the model summary on the left here and a visual representation of the model right and you can see the dropout comes, its this green block here after the Max pooling layer and the first dense layer.

Now I’m going to rerun the model, everything else is the same, I’ve just added the dropout layers. And you can see even with this one change, the training accuracy has jumped up to 0.85 and the validation accuracy to 0.79. So, there is still some overfitting, but the model is performing significantly better on both the training and validation data. You can see the real significance of Dropout here; it makes sure the model doesn’t rely on a specific set of features but rather learns multiple ways to achieve the same outcome which makes the model more robust. From the graphs you can see the plot between training loss and accuracy and validation loss and accuracy is much more aligned. The model is learning through the epochs and there is convergence and less fluctuation with the validation data, so it is performing better on unseen data.

Now there is another way to reduce overfitting. We’ve already applied regularisation, but another way is to add more data. It would be difficult to source new data, but we can generate new data from our existing data. We do this through a process called data augmentation. In data augmentation, we change the inclination of our existing images for example you can see with this kitten picture we can rotate it, or we can flip the picture. For a human it’s the same, but for a machine it’s a different picture. Data augmentation artificially expands the training images. This can lead to a more robust model as it simulates a form of diversity that the model would encounter in the real world (Ying, 2019). I’ve put the code on the left here, so my augmentations are randomly rotating by 10 degrees, random horizontal flips, and random translations of the images along the width and height by 10%.

So when we evaluate this model, we see that the training accuracy has fallen down to 0.76 and the validation accuracy to 0.77. Now this may seem like the model has performed worse because of the lower accuracy but you can see there is no overfitting and its actually performed better on unseen data. Now the drop in accuracy could be because we have added increased complexity to the training data by adding more images which makes it harder to fit but this is realistic and I’m more confident in this model being consistent with unseen data which is what is important because we want to evaluate on the test data. The curves are interesting, you can see for accuracy the model performed consistently better on the validation data then the model. The closeness in the final epochs suggests the model has a good balance between learning from the training data and generalising new data.

Moving on to the final iteration of the model, I am going to incorporate batch normalisation (BN), this is a technique that builds on our strategy of normalising input data by standardising features across all layers, leading to easier and faster model convergence. BN adjusts the weights within each layer for each mini batch by calculating their mean and variance, extending the consistency principle applied to the input layer across the model's subsequent layers. You can think of these as standardising parts at every station of a factory line, ensuring each data point is uniformly processed to prevent bottlenecks. By standardising each layer, BN enhances the model's stability and efficiency in learning, leading to quicker and improved performance (Bjorck et al, 2018). With the integration of BN, I'm increasing the training epochs to 50 and reducing the learning rate to achieve smoother and more efficient optimisation, enhancing learning quality without sacrificing speed and I’ve put the code here below.

So here is a visual representation of the final model, so you can see I’ve added batch normalisation after every convolutional layer, it’s the red block here and I’ve also added it after the first dense layer. After each of these sections, BN is standardising the data making sure it fits a common standard making the model learn faster and quicker.

The final model summary shows the inclusion of batch normalisation, these parameters—encompassing means, variances, scales, and shifts—allow for the normalisation of each feature map individually. As a result, alongside other architectural enhancements, the model's total parameters have increased to 552,674.

When I run this final model, you can see the accuracy on the training was 0.85 and on the validation data it was 0.83. The average loss was also very similar between training and validation data, so I am happy with that performance, I wanted it to be on at least 0.8 for validation accuracy. You can see from the plots it is quite up and down when performing to unseen data but its aligning with the training data and there seems to be no overfitting. I’m happy to use this model to evaluate the test data.

So now we can evaluate on our test data, and you can see here we got around 0.82 accuracy and 0.549 average loss which is very similar to the performance before which is exactly what you want. We can conclude our model learned from the training data and is generalising its predictions to unseen data quite well. The accuracy is relatively high, and I would say this a successful training process.

I’ve also printed the classification report here, this provides a detailed analysis of the performance of the model. It has the precision score which is the proportion of true positive results in all positive predictions and the recall which is the true positive results out of all actual positives and then the F1 which is weighted average of precision and recall. So, looking at some interesting cases, class 6 which was frog had a precision score of 0.62 which means only 62% of the instances predicted as class 6 were correctly identified. As for recall, class 3 which was cat and class 5 which was dog had a score of 0.58 and 0.68 respectively. This is saying the model only correctly identified 58% of cats and 68% of dogs, perhaps it was getting them mixed up as they were the most similar classes in the data. However overall, there are high values of f1 score throughout the model which suggests the model performed fairly well across all classes.

In conclusion, this analysis has progressed through the development and refinement of a CNN model, starting from a baseline model and undergoing three subsequent iterations for improvement. Initially, the baseline model achieved a validation accuracy of 59.9%. The introduction of Dropout layers significantly enhanced performance, elevating accuracy to 79.6%. Although the incorporation of data augmentation slightly reduced accuracy to around 77%, it was effective in reducing overfitting and enhancing performance to unseen data. The subsequent addition of batch normalisation layers further improved validation accuracy to 82%. This improvement was mirrored in the test data, where a comparable accuracy was observed.

Finally, I wanted to end with some ideas for further work and reflections on key learnings I have acquired. So for further work, I think a deeper analysis of the model's performance on the individual classes rather than solely focusing on overall accuracy is a good way to improve the model. Notably, our model performed worse in distinguishing cats and dogs. This would suggest a valuable opportunity to refine the model’s ability to classify these categories. Secondly evaluating the model on other datasets would offer insights into its generalisability and robustness which would be instrumental in understanding the model's applicability to a broader range of images. Finally, we could explore the practical applications of our model in a real-world scenario. For example, given its capability to identify trucks, one intriguing possibility is integrating the model into autonomous driving systems improving their ability to recognise and respond to trucks in the road.

In terms of reflections on the entire process, this project has taught me valuable lessons about model development and the importance of the iterative process and hyperparameter experimentation. I have seen how simpler models can sometimes outperform more complex ones, highlighting how model design is not always about complexity. Furthermore, I have learned about the creative external avenues you can take to improve model performance, such as data augmentation. Overall, this project has taught me the necessity of adopting a flexible and experimental approach to model building. Each dataset interacts uniquely with models which requires a bespoke approach to refinement for each dataset.

Secondly, I’ve evolved my understanding of model evaluation. Previously I would have emphasised high training accuracy, interpreting this as the hallmark of success. However, this project has shifted my perspective underscoring the importance of a model's performance on unseen data. It was a reminder to me that although I complete this work in an academic context, the ultimate objective is to develop models that function effectively in real-world scenarios. I’ve come to appreciate that the true measure of a model’s generalisation capability lies in the alignment of the training and validation performance. A model that performs consistently well on both training and unseen data is more likely to be robust and applicable in real-life situations.

And finally, I saw the difference between theoretical knowledge and practical application. Theoretical knowledge provides a solid understanding, yet applying these concepts in practice can reveal discrepancies between theory and real-world performance. There were times when I would find information online and incorporate it into my code expecting better performance, only to find out it would make my model perform considerably worse. It reminded me that although models aim to simplify reality, the unpredictability and complexity of real-world data present significant challenges. Moreover, the opaque nature of these models makes it difficult to pinpoint why certain outcomes occur highlighting the complexities involved in understanding and improving model performance.

So that is the end of my presentation, I hope you enjoyed listening to me.

Reference List

Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaria, J., Fadhel, M.A., Al-Amidie, M., Farham, L. (2021) Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data (53). DOI: https://doi.org/10.1186/s40537-021-00444-8

Agarap, A.F. Deep Learning using Rectified Linear Units (ReLU) Reprint. DOI: https://doi.org/10.48550/arXiv.1803.08375

Cabanilla, K.I.M., Mohammad, R.Z. & Lope, J.E.C. (2024) Neural networks with ReLU powers need less depth. Neural Networks 172. DOI: https://doi.org/10.1016/j.neunet.2023.12.027

Balaji, S. (2020) Binary Image Classifier CNN using TensorFlow. Available from: https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 [Accessed 3rd Feb 2024]

Bot Penguin (2024) Softmax Function. Available from: https://botpenguin.com/glossary/softmax-function [Accessed 4th February 2024].

Bjorck, J., Gomes, C., Selman, B. & Weinberger, K.Q. (2018) Understanding Batch Normalization. Advances in neural information processing systems 31. DOI: https://doi.org/10.48550/arXiv.1806.02375

The CIFAR-10 dataset (2009) Available from: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 3rd January 2024]

Gosmar.eu (2020) Neural networks and speech recognition. Available from: https://www.gosmar.eu/machinelearning/2020/05/25/neural-networks-and-speech-recognition/. [Accessed 12th February 2024].

Hassan, E., Shams, M.Y., Hikal, N.A. & Elmougy, S. (2022) The effect of choosing optimiser algorithms to improve computer vision tasks: a comparative study. Multimedia Tools and Application. 82: 16591-16633. DOI: https://doi.org/10.1007/s11042-022-13820-0

Huang, K. (2022). “Image Classification Using the Method of Convolutional Neural Networks”. 2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS). Dalian, China, 11-12th December 2022 pp 827-832. DOI: 10.1109/TOCS56154.2022.10016070

Kerkhof, M., Wu, L., Perin, G., & Picek, S. (2023) No (good) loss no gain: systematic evaluation of loss functions in deep learning- based side – channel analysis. Journal of Cryptographic Engineering. 13: 311-324. DOI: https://doi.org/10.1007/s13389-023-00320-6

Lu, Yongtao, Huo, Y., Yang, Z., Niu, Zhao, M., Bosiakov & Lei, L. (2022) Influence of the parameters of the convolutional neural network model in predicting the effective compressive modulus of porous structure. Frontiers in Bioengineering and Biotechnology. DOI:10.3389/fbioe.2022.985688

Machine Learning Mastery (2020) A Gentle Introcution to the Rectified Linear Unit (ReLU) Available from: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ [Accessed 4th February 2024].

Mikolajczyk, A. & Grochowski, M. (2018) “Data Augmentation for Improving Deep Learning in Image Classification Problem”.2018 IEEE International Disciplinary PhD Workshop. Katowice, Poland, 09-12 May 2018. DOI: 10.1109/IIPHDW.2018.8388338

Mo, R. (2022). A Survey of Image Classification Algorithms based on CNN. Highlights in Science, Engineering and Technology. Volume 15. DOI: <https://doi.org/10.54097/hset.v15i.2222>

Murray, C. (2017) Building a Facial Recognition Pipeline with Deep Learning in Tensorflow. Available from: https://medium.com/hackernoon/building-a-facial-recognition-pipeline-with-deep-learning-in-tensorflow-66e7645015b8. [Accessed 12th February 2024].

Nwankpa, C.E., Ijomah, W., Gachagan, W. & Marshall S. (2020) Activation Functions: comparison of trends in practice and research for deep learning. 2nd International Conference on Computational Science and Technology 17-19th December. Jamshoro: Pakistan.

Schneider, M., Amann, R. & Mitsantisuk. C. (2021) “Waste object classification with AI on the edge accelerators”. 2021 IEEE International Conference on Mechatronics. Kashiwa, Japan, 07-09 March 2021 pp 1-6. DOI: 10.1109/ICM46511.2021.9385682.

Shinde, Y. (2022) Custom Data Augmentation using Keras Image Data Generator. Available from: https://medium.com/geekculture/custom-data-augmentation-using-keras-imagedatagenerator-7cfd58e54171 [Accessed 4th February].

Simonyan, K. & Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations San Diego. May 7-9th. Oxford: University of Oxford.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014) Dropout: A Simple way to Prevent Neural Networks from Overfitting. Journal of Machine Learning. 15(56): 1929-1958.

Yamashita, R., Nishio, M., Kinh Dian Do, R. & Togashi, K. (2018) Convolutional neural networks: an overview and application in radiology. Insights into Imaging. 9: 611-629. DOI: https://doi.org/10.1007/s13244-018-0639-9

Ying, X. (2019) An Overview of Overfitting and its Solutions. Journal of Physics Conference Series. 1168. DOI: doi:10.1088/1742-6596/1168/2/022022

Zhao, L. & Zhang, Z. (2024) A improved pooling method for convolutional neural networks. Scientific Reports. 14. DOI:https://doi.org/10.1038/s41598-024-51258-6

Zhang, S., Ku, B & Ko, H. (2021) Analysis of normalization effect for earthquake events classification. The Journal of the Acoustical Society of Korea. 40: 130-138. DOI: https://doi.org/10.7776/ASK.2021.40.2.130.