Summative Assessment Individual Project Transcript

This is my individual presentation in which we're going to go through the steps in creating an artificial neural network model for object recognition of images from the CIFAR-10 dataset.

As mentioned, the dataset used for this project is the CIFAR-10 dataset. It's available online and through the Keras library. The dataset consists of 60,000 32 by 32-pixel colour images with associated labels. The dataset includes ten classes with an even distribution of images between them. So 6000 images per class. The training set is already split.

The data is already split into training and test sets, with 50,000 images in the training set and 10,000 images in the test set. The training set has five batches evenly split, so 10,000 batches per 10,000 images per batch.

And each batch has randomly selected images, so there'll be no need for shuffling in the implementation of this model.

You can see by the image to the right that there's some randomly generated images from the data set with their associated labels. Quite pixely because again, they're pretty small, low-quality images. However, you can kind of kind of get a general shape of the objects in the images. It would be interesting to see what the model can detect and what it can't.

So as mentioned, the CIFAR-10 dataset is included in the Keras library. It's part of their inbuilt small datasets.

During the project, the data was initially loaded using the code in the top right of the screen, and during this loading process, it was automatically split into the training data and the test data. X indicating the actual images that the model will be trained or tested on, and Y being associated with the labels.

So there was the associated label and the description as indicated in this image here. So we can see that the description goes in in kind of consecutive order: Airplane, automobile, bird, cat deer, dog, etc. And they have associated labels numerical labels starting from zero and ending in nine. This makes it quite easy to just create a list of class names that is indexable or can be kind of can create a human readable output by putting the label as the index of the of the list. So you can see an example of the or the list down here.
This basically means that we can convert the numerical label to a human readable output. Just making it more understandable.

After this, it was also visualized, 1 or 2 of the images. And we can see in the middle image here the, the data is stored as numpy arrays, with 32 by 32 by 3 shape, this being 32 pixels by 32 pixels, and the RGB colour depth.

On the previous slide, it was mentioned that the data was imported, split into a training set and a test set.

The training set can then further be split into a training set and a validation set. 80% of the data allocated towards the training set, and 20% of the data allocated towards the validation set. This was achieved by importing the train test split method from sklearn, and is demonstrated in the code snippet of this slide.

With this previous training set now split into training and validation, we now have a training set, a validation set and a test set.
The training set is used to build the model and is what is fed into the model for the model to learn patterns.
The validation set is used to select the correct parameters and to tune the parameters of the model.
And the test set is used to evaluate the final model's performance with the finalised selected parameters. A validation set is important because it gives us a way to evaluate the model and modify the parameters, whilst keeping it separate from the test set completely.

We want to keep the test set completely separate from the model development, because any data leakage from the test set into the model can create a lack of generalisation on unseen data. The use of a validation set ensures accuracy isn't mistakenly overestimated.

So the strategy on building and designing a machine learning model, initially starts with a small basic model to get a get a baseline, and a small model is used to reduce computing resource and to help with checking for overfitting of the data. Machine learning models like normalised data, so values between 0 and 1. Therefore the data also needed to be initially normalized to evaluate and train the machine learning model on. I initially did this by just manually dividing the pixel values or the training sets by 255 as that's the maximum value of a pixel.

So convolutional neural network kind of architecture was used for the development of this model as it's quite suited to image recognition and image-based machine learning models. You can see in the top right that there's one of the basic initial models with two convolutional layers, followed by max pool layer, two convolutional layers followed by max pool layer, and then the finalized with flatten and dense layers. Just with a small a small number of filters. So only ten filters used kernel size of three and activation function of ReLU.

So I experimented with different combinations of convolutional layers and max pool layers. This was done to reduce the possibility of overfitting on more epochs. The performance of the model was evaluated using loss and accuracy curves.

This data was saved and then plotted. And basically, what I was looking for was stability as well as, we didn't want the training accuracy to kind of overshoot above the validation accuracy. You kind of want these curves to be very similar. However, if the validation accuracy is less than the training accuracy over time, it's clear evidence that the model is overfitting and isn't able to generalise on unseen data. So another step to reduce the risk of overfitting was to implement data augmentation on the datasets.

So, in this slide you can see an example of data augmentation with just a sample of nine random augmented images of a frog. This was on the left. You can see a code snippet in which I created a data augmentation layer, which was then implemented in later development of models and tuning, just to make make life slightly easier and to help with loading of the data, less data had to be transformed before loading it into the model.
It meant that the model trained for longer because it had to compute the data augmentation during training. However, it made the process a lot simpler. It was just simple augmentation that was done. So random flip, random rotation and random zoom. And you can see on the right an example of how it alters the image.

So more on the strategy of developing the model. Initially the number of layers, number of epochs, and the number of filters in the layers were quite small to kind of reduce the amount of computation power needed to develop the model, and to get those iron out the initial stages of development and investigate overfitting.
However, I saw that the validation accuracy started to plateau and sit around between 50 and 60%. So I started to investigate the number of filters or neurons in each layer, with just ten epochs. This managed to increase validation accuracy to around 62% on one of the models. However, you know, it still was struggling.
So then I increased the number of epochs that, the model was fit on, and this further increased the validation accuracy to 70%. So, I played around with the number of filters and neurons in each layer, you know, increasing from 10 to 32 and 64, and just experimented and had a little look at what others did to get a feel for how this would improve it.

So here we can see an overview of the final architecture of the neural network created. So as mentioned earlier, it was a convolutional neural network that I was basing this on. And this was a result of various implementations, hyperparameter tuning experiments and research online.

So if we go through the architecture from top to bottom, we can see that in the top box around here is the input shape layer and the data transformation layer. So we have an input layer. This is just defining the input shape of the images, the rescaling layer just rescaling or normalizing the image pixel values to be between 0 and 1. As mentioned machine model machine learning models prefer data of that format as well as the data augmentation layer. So this is just transformation layers, input layers to help streamline the machine learning model process.

Next we have three stacked convolutional or conv 2D layer blocks. So, each layer block includes a convolutional layer, a batch normalisation layer, a convolutional layer also followed by another batch normalisation layer, a max pooling layer and a dropout layer.

The convolutional layer is the main layer in a convolutional neural network. And it helps extract features, from the filters or using filters. So I added some padding to these convolutional layers to conserve the input shape into the convolutional layer and hence preserve information.

The batch normalisation layer was implemented throughout the model, to basically normalise the activations of layers and help stabilise training process and reduce overfitting. So, it was a good regularization method. The maxpool layers, were used to reduce the dimensions of the feature maps, and also the dropout layer was another way of regularizing the model. So the dropout layer, um, basically randomly drops nodes or neurons, which alters the connectivity throughout the network. The final block was basically the fully connected layers. So the flattened layer basically just converts the output of the convolutional layers into a one dimensional, or a single dimensional layer, which is what the format that the dense layers need their inputs to be in.
And then the dense layers were just there to help with classification of the extracted features.

Once the model was developed, the validation and training set were then combined into one training set and the model was finalised and finally trained on the full training data and evaluated on the on the test set. You can see here that the evaluation on the test set, although it's where the evaluation or val accuracy and val loss, it was evaluated on the test set. And we can see that it had a final validation accuracy of 0.84 or 84%. So this is the accuracy that the model achieved on unseen data.

One of the activation functions used in the neural network was ReLU which is short for rectified linear unit.
What this activation function basically does is allow positive values to pass through unchanged and negative values are set to zero.
It is useful for avoiding common problems like. The vanishing gradient problem, however, has some drawbacks, such as the dying ReLU problem, in which neurons become inactive and therefore can't learn any on further iterations.

So, the activation function used in the output layer of the model is SoftMax. And it's quite common to use in multi-class classification. It basically transforms the output of the neural network into a probability distribution over the input classes, helping to classify.

The loss function implemented was sparse categorical cross entropy. It's a common loss function used for multi-class classification when the categories or labels are provided as integers.

So to reflect and summarize the model building, the evaluation or final evaluation of the model could have been more robust using other metrics like precision and recall, and also confusion matrix to evaluate if any categories were causing trouble for the model.

The final model could have been trained on more epochs. The final model was trained on 50, but with more epochs it could have had better accuracy.
A lot of manual experimentation was done when developing the model, so techniques such as cross validation and ensemble learning may have gotten better results sooner.
And generally building a model, start simple and build your way up. Add complexity as you go, or take it away based on results from the validation data.

References

Krizhevsky, A. (2009). CIFAR-10 and CIFAR-100 datasets. Toronto.edu. Available from: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed on 15th January 2025].

Keras. (N.D.). Keras documentation: CIFAR10 small images classification dataset. keras.io. Available at: https://keras.io/api/datasets/cifar10/ [Accessed on 15th January 2025].

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. ‘O’Reilly Media, Inc.’

Wang, Z.J. et al. (2021). CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. IEEE Transactions on Visualization and Computer Graphics, 27(2), pp.1396–1406. doi:https://doi.org/10.1109/TVCG.2020.3030418 [Accessed 20th January 2025].

Dumane, G. (2020). Introduction to Convolutional Neural Network (CNN) using Tensorflow. [online] Medium. Available from: https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83 [Accessed 20th January 2025].

GeeksforGeeks (2024). ReLU Activation Function in Deep Learning. GeeksforGeeks. Available from: https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/ [Accessed 19th January 2025].

Bala, C. (2023). Softmax Activation Function: Everything You Need to Know. Pinecone.io Available from: https://www.pinecone.io/learn/softmax-activation/ [Accessed 19th January 2025].

Chand, S. (2023). Choosing between Cross Entropy and Sparse Cross Entropy — The Only Guide you Need! [online] Medium. Available at: https://medium.com/@shireenchand/choosing-between-cross-entropy-and-sparse-cross-entropy-the-only-guide-you-need-abea92c84662 [Accessed 19th January 2025].