[**Slide 1] Introduction**

Hello everyone! Today, we will discuss how we build our own Convolutional Neural Network or CNN model for object recognition.

Our task is to build a CNN that can recognize objects within images. The dataset we are using here is the CIFAR-10 dataset. This dataset is widely used for image recognition. It contains 60,000 images for 10 different object classes.

Throughout this presentation, we will walk you through our approach to building and training a CNN model by discussing key decisions and results along the way. We hope you will gain a deeper understanding of how our CNN works and how we use it for object recognition.

[**Slide 2] Dataset and Validation Set Creation**

In this slide, we will discuss how we split the data for our model. The CIFAR-10 dataset is a collection of 60,000 small, 32x32 pixel colour images. These images are divided into 10 different classes such as cars, birds, horses, etc...

We needed to split this dataset into three parts, a training set, a validation set, and a test set to train in our model. The training set is used to train the model. The validation set is used to tune the model's parameters and prevent overfitting.

And the test dataset is used to evaluate the model's performance on unseen data.

The code snippet in this slide shows how we loaded the CIFAR-10 dataset using TensorFlow and split it into training and testing sets. The validation set was later created from a portion of the training set.

**[Slide 3] Model Architecture**

In this slide, we will discuss the architecture of our CNN model. Our model begins with a convolutional layer that has 32 filters and is followed by a batch normalization layer. This pattern is repeated but the second convolutional layer is followed by a max pooling layer and a dropout layer. This layer combination is helpful for us to reduce overfitting and improves our model's ability to generalize.

We can also see the number of filters in the convolutional layers increases. We start with 32 filters and then move to 64 filters. Finally, we increased the filters to 128. This increment in complexity allows our model to learn more complicated patterns of the data.

After the convolutional layers, we flatten the data and pass it through a fully connected layer with 128 units. This is followed by another batch normalization layer and a dropout layer.

Finally, we have a dense layer with 10 units, corresponding to the 10 classes in our dataset. This layer uses the softmax activation function, which outputs a probability distribution over the 10 classes, allowing us to make a prediction.

The code snippet on this slide shows how we defined this architecture using TensorFlow's Sequential model API.

**[Slide 4] Activation and Loss Functions**

In this slide, we'll discuss the activation and loss functions used in our model.

The Rectified Linear Unit, or ReLU, is used as the activation function in the convolutional and dense layers of our model. ReLU is a popular choice for these types of layers because it helps to reduce the vanishing gradient problem, which can delay the training of deep neural networks.

For the output layer, we use the softmax activation function. Softmax is ideal for multi-class classification tasks like ours, as it outputs a probability distribution over the classes, allowing us to make a prediction.

Finally, we use sparse categorical crossentropy as our loss function. This is a suitable choice for multi-class classification tasks, as it measures the error between the true class labels and the predictions made by the model.

**[Slide 5] Training the Model**

In this slide, we'll discuss how we trained our model.

Our model was trained for 40 epochs. An epoch is one complete pass through the entire training dataset. The number of epochs is a hyper-parameter that defines the number of times the learning algorithm processes the entire training dataset.

We chose to train our model for 40 epochs based on its performance on the validation set. It is important to monitor the performance of the model on the validation set to ensure that the model is learning effectively and not simply memorising training data, which can lead to overfitting.

To prevent overfitting, early stopping is also implemented. Early stopping is a method of specifying an arbitrarily large number of training epochs and stopping training when the performance of the model stops improving on the holdout validation dataset. In our case, training is stopped if the validation loss does not improve after three epochs.

The code snippet on this slide shows how we trained our model for 40 epochs and validated it on the test set, with early stopping implemented.

**[Slide 6] Neural Network Design**

In this slide, we'll discuss the design of our Neural Network.

We used the Keras library with TensorFlow as the backend for our model. Keras is a high-level neural network API written in Python and executable on top of TensorFlow. It has been developed with a focus on enabling rapid experimentation, allowing easy and fast prototyping.

We chose a Convolutional Neural Network (CNN) for our task of image classification. CNNs are particularly good at picking up on patterns in the input image, like lines, gradients, circles, squares, which are then used to understand the image on a larger scale. CNNs are especially good at understanding the spatial relationships between pixels, which makes them ideal for image classification tasks.

Batch normalisation and dropout were implemented to improve model performance and control overfitting.

Batch regularisation is a technique to improve the speed, performance and stability of artificial neural networks.

Dropout is a regularisation technique that approximates training a large number of neural networks with different architectures in parallel.

Finally, the Adam optimiser was used for efficient gradient descent.

Adam is an optimisation algorithm that can be used as an alternative to classical stochastic gradient descent methods and iteratively updates the network weights based on the training data.

This design approach allowed us to build a robust and efficient model for our object recognition task.

**[Slide 7] Accuracy Analysis**

This slide describes the accuracy of our model.

Our model achieved a high training accuracy of approximately 90%. This means that our model was able to correctly classify 90% of the images in the training set.

Validation accuracy was slightly lower, at 87%. This is the accuracy of the model on the validation set, a set of images that the model has not seen during training. The fact that the validation accuracy is close to the training accuracy indicates that our model generalised well to unseen data.

Dropout layers and early stopping were used to suppress overfitting. Overfitting occurs when a model learns too much on training data and performs poorly on unseen data. The use of drop-outs and early stops helped to maintain a balance between training and validation accuracy.

The plot diagram on this slide shows training accuracy, test accuracy, loss and validation loss across epochs. As you can see, training and validation accuracies are very close throughout the training process. This is a good sign and shows that our model has learnt the underlying patterns in the data well without simply memorising the training data.

This concludes our accuracy analysis. Our model performed well on both the training and validation sets, and was shown to perform well on real-world data.

**[Slide 8] Conclusion**

In conclusion, through this assignment, we've gained a deeper understanding of Convolutional Neural Networks, or CNNs, and their application in object recognition.

We've learned about the importance of having a validation set. A validation set allows us to evaluate our model's performance during training, which is crucial for tuning the model's hyperparameters and preventing overfitting.

We have also explored various techniques to improve the performance of the model, such as the use of drop-out layers and early stopping. Dropout layers help to prevent overfitting by randomly setting a percentage of the input units to zero at each update during training. Early stopping is a form of regularisation used to avoid overfitting when training learners in iterative methods such as gradient descent. This is done by terminating the learning process before the learner passes the point of overfitting.

Finally, we've successfully implemented a CNN for object recognition and achieved good accuracy on both the training and validation sets. This indicates that our model is capable of generalizing well to unseen data, making it a robust solution for object recognition tasks.

This exercise has been a valuable learning experience in understanding and implementing CNNs for image classification tasks.

**References**