**Transcript Of Individual Presentation:**

**Neural Network Models for Object Recognition**

**(Track 2)**

**Word Count: 2504**

# Cover Slide:

Hello all. My name is Fahad and it gives me great pleasure to be presenting this individual program named as, Neural Network models in object recognition. In the presentation, I will guide you through the steps of designing, training, and evaluating deep learning models, i.e., CNN and transfer learning ones, on the task of object recognition in the images on one of the existing training datasets, CIFAR-10.

# Slide 1: Introduction

In this project, we look into two major directions of performing deep learning: the first is constructing a model on our own based on Convolutional Neural Networks (CNNs), and the second is taking advantage of transfer learning by training MobileNetV2 that has been pretrained on ImageNet. The two models were measured based on their performance on the correct classification of images into one of the ten categories of objects.

Our interest in this project lies in this degree of functionality being applied in the real world in terms of facial recognition systems, autonomous vehicles, and smart surveillance. The selection of CIFAR-10 as a data set was not accidental it is widespread in academic projects, it is not so large so it cannot be complicated to handle, and it has a diverse range of common objects (Krizhevsky, 2009). The third tool of implementation is TensorFlow, Keras, and other libraries of visualization which can be applied to analysis and assessment.

**Slide 2: EDA: Exploratory Data Analysis**

I had conducted Exploratory Data Analysis (EDA), to interpret the data and make sure that the data was balanced, as well as fit to be trained, before starting model building. The CIFAR-10 is a collection of color images that has 60,000 images with a 32x32 pixels dimension and three color components (RGB). This gets equally split into 10 different classes, with 6000 images each of airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck (Krizhevsky, 2009).

By means of visualizing the data through bar graphs and sample pictures, I confirmed that the data is balanced, as there is an equal count of samples within classes. This balance is significant in training unbiased and with respect to fair measure evaluation such as precision, recall and F1-score. A balanced dataset will help in ensuring that the model is not too skewed to a particular class and this aspect comes to play in many real-life applications where class imbalance is present.

I also picked random samples of each of the classes to note whether or not there was variation in classes. As an example, the word "automobile" may have cars in varying angles, lightings and in various colors. Such intra-class variation makes the task of classification more complicated but also allows the model to be more generalized when addressed adequately.

**Slide 3: Preprocessing**

Preprocessing is the most basic part of machine learning pipeline, not least of all in computer vision. In this project, the preprocessing was done in a number of steps:

1. Normalisation of pixel value: Pixel values were normalised to 0 and 1 within an 0-255 range. It is a typical procedure invoked in deep learning as it balances the optimization procedure and speeds up convergence (Brownlee, 2022).

2. Label encoding: Labels were encoded as numerical data, using the sparse\_categorical and not one-hot label encoding, since it allows using multi-class classification tasks and integers as labels. This encoding is effective together with the sparse loss functions of TensorFlow.

3. Image resizing: Compared to MobileNetV1, MobileNetV2 prefers more robust input resolutions and therefore CIFAR-10 image rescaling was set to 96x96 pixels (by default 32x32 pixels). This resizing enables an adjustment to the architecture of the pretrained model keeping the aspect ratio intact as much as possible (Howard et al., 2017).

4. Shuffling and augmentation: To enhance the strength of model and reduce overfitting the data was shuffled and augmented (depending on the application) using horizontal flipping and random cropping. The method approximates the look of a more varied dataset, resembling variations seen in real pictures.

**Slide 4: Data Partitioning**

The data was thereafter stratified into three:

1. 40,000 images for training
2. 10,000 images for validation
3. 10,000 images for testing

The model was fitted on the training set and during the training, hyperparameters were tuned and the performance could be monitored using the validation set. The trained model tested on the final test set was used to assess the quality of performance of the model on new data.

This is a three-way split which is a good practice that enables the detection of over fitting/ under fitting at an early stage of development. When a model gets great tops and low accuracies on the validation set to those on the training set, chances are that it is over fitting. Conversely, poor results on both sets indicate the underfitting, which usually involves architecture alteration or improved training of methodologies.

Moreover, the presence of a separate test set--which is in no way connected to the training and tuning process--guarantees quality end performance benchmark, that would indicate how the model would perform in reality when it is deployed in a live environment (TensorFlow Docs, 2023).

**Slide 5: Custom CNN Architecture**

A custom Convolutional Neural Network (CNN) was the first model used. The keras sequential API was used to design the architecture whereby it gives a clear and easy layer stacking architecture.

Among the architecture there were:

* Convolutional layers (Conv2D) equal to 2; activation function ReLU
* MaxPooling layers to shrink the dimension in space
* Flatten layer to convert feature maps with vectors
* A Dense layer with 128 neurons and after that a Dropout layer with a rate of 0.5
* Final fully connected layer containing 10 neurons that is activated using softmax as classification layer

The architecture works on the best practices of CNN, and it allows extracting low and mid features of images such as edges, corners, and textures (Brownlee, 2022). Dropout layer contributes towards avoiding overfitting as random neurons are disabled in every training epoch.

Although this is a quite basic CNN, its performance on CIFAR-10 is not that bad, and can also be used as a baseline against more complex models.

**Slide 6: Transfer Learning with MobileNetV2**

The second was a transfer learning; whereby the MobileNetV2 architecture was employed, and it was pretrained on ImageNet dataset (Howard et al., 2017; Deng et al., 2009). This allows us to take learned features on a very large dataset and refine it on smaller more focused domain specific dataset such as CIFAR-10.

Steps taken:

1. Load MobileNetV2 without class classification tops
2. Stop the base layers training during the first stage by freezing them
3. Insert a Global Average Pooling (GAP) in it, then a Dense(128) followed by Dropout and Dense(10) layer and add a softmax activation into it
4. Unfreeze 50 previous layers, and modified them so that the model will be adjusted to CIFAR-10 image collection

The method makes it less time consuming in training and also less computationally expensive with a good level of accuracy. It is specifically applicable when there is a small amount of data to work with but a small amount of processing power available, e.g. mobile applications or an edge computing system.

**Slide 7: Training Process**

The training process has been greatly crafted so as to maximize performance and reduce overfitting and time-cost. The Adam optimizer was used to train both models, custom CNN and MobileNetV2, since it is an adaptive optimizer (adaptive learning rates) and converges easily in deep learning tasks and applications. It is particularly useful it we deal with noisy gradients and sparse data as in image classification.

The loss function was sparse categorical crossentropy, which is suitable in multi-class classification applications where each label is integer. It is also effective when utilised with sparse encoded labels, and assists in reduction of error prediction between the real label and the probability product that was given by the model.

I set the batch size to 64 because at this level it creates a trade-off between good computation and training stability. The smaller batch would probably take too much training time and a bigger one would lead to memory problems or instability of convergence. Generalization also depends on the batch size, when smaller batching normally induces a regularization effect because of noisy gradient estimates.

The custom CNN model was trained on 20 epochs whereas mobileNetV2 was trained in two stages:

1. The initial epochs (10) were a training phase with frozen base layers to keep the pretrained knowledge.
2. Fine-tuning (10 extra epochs) with freezing the final 50 layers, which enables the proficiency-specific Pattern adjustment (Howard et al., 2017).

Also, there was use of early stopping, that watched over the loss in validation. Training would automatically halt in case of validation performance ceased to improve after several epochs straight. Such an approach avoids overfitting and preserves computational resources (TensorFlow Docs, 2023).

Random seeds were fixed and the whole pipeline, including data loading and training, was covered with explicit logging of training numbers including loss, accuracy and times of the epochs.

The training pipeline in a nutshell was made efficient, stable, and generalizable which allowed both models to learn successfully as well as minimized the chance of overfitting or underfitting.

**Slide 8: Model Evaluation – Accuracy and Loss**

The performance of models was also determined by various measures after training, such as test accuracy, training and validation loss, and accuracy curves over epochs.

Its test accuracy reached approximately 72%, which is not bad considering that CNN was trained entirely using a CIFAR-10 dataset. It however had some characteristic drawbacks in terms of slower convergence as well as moderate overfitting which are depicted in its learning curves. After the 6th epoch, the validation loss start stabilising, and a small difference started emerging between the training and validation accuracy implying that the model was already starting to memorize the training set.

Conversely, the transfer learning model MobileNetV2 scored much well with a test accuracy of about 85%. Training and validation loss rates rapidly dropped in the early epochs with faster convergence in this model on its first few epochs. Such a fast learning would not be possible without the pretrained weight that is based on ImageNet since, at that point, they already have strong feature representations (Deng et al., 2009; Chollet, 2017).

Also, the fact that the training and validation curves are close in MobileNetV2 showed that the model was quite generalized, and it was not being overfit. That symbolizes a significant benefit of transfer learning it builds on the previous knowledge, which eliminates the necessity of huge amounts of data and prolonged training.

Moreover, the learning curves gave me an idea of model behavior. In the case of CNN, the curves began to separate at the last point implying inability to generalize to some extent after some time limit. Alternatively, the results obtained in MobilNetV2 were consistent, with a low training loss, and a low validation loss, which demonstrated a remarkable match to the new domain without the common hazards exhibited by the overfitting phenomena as well as the standstill observation in models.

**Slide 9: Confusion Matrix Analysis**

The confusion matrix is an excellent diagnostic method of examining the performance of a classification model beyond mere accuracy. It presents the number of true positive, false positive, true negative, and false negative of each class.

In the case of both models, and especially MobileNetV2, the matrix reflected good diagonals, meaning that many assurances were obtained in relation to the correct predictions in most of these categories. A good example is that items such as airplane, ships and automobiles possessed good precision and recall. These are generally classes that are more marked visually and are clear cut to classify by the model.

Nevertheless, significant misclassifications were still recorded, particularly, between semantically or visually, similar classes. The model sometimes at times mixed up:

* Cats and dogs
* Deer and horses
* Birds and frogs, particularly when their silhouettes merged together because of their shapes

The above errors also match the errant sense of humans talked of by the human eye in terms of visual similarity, pose or background clutter when the images are of low-resolution thereby making these classes inherently hard to partition (Krizhevsky, 2009).

The other lesson that the confusion matrix offers is the type of errors distribution. In some classes this model preferred greater precision at the expense of lesser recall or vice versa. This asymmetry could provide indication of the sort of under prediction or over prediction of particular classes that the model may be making.

When designing a model to a specific application such as a healthcare system or a security system it is very important to re-tune the model so as to reduce such false positives and false negatives. Next steps that might help to solve this problem are fine-tuned decision threshold or adding class-specific loss weights.

**Slide 10: Single Image Test**

In order to get more insight on real-world application possibility, I ran the trained MobileNetV2 model over one unseen image of the CIFAR-10 test set. This is done to analyse the degree of performance of the model with batched inference removed, approximating a similar deployment situation, e.g. real-time prediction in an app or an embedded system.

The model produced an estimated class label as well as a confidence score, (the probability spread of all 10 classes.) The top-1 prediction was corresponding to the actual label in the majority of cases, and the confidence level surpassed 8090 % in most cases.

This proved that the model was robust where single inference was present. More to the point, it demonstrated that MobileNetV2 can essentially perform quite well with batch classification in experimental scenarios but is also sufficiently predictable in single image predictions that can be crucial in its deployment as the part of an edge AI application such as smartphone or IoT devices (Howard et al., 2017).

Besides, I evaluated the confidence of the model even in erroneous predictions. The second highest prediction matched the correct class in the given cases, which serves as evidence of top-k accuracy metrics useful for future improvements of the given type of machine. This is used to construct confidence limits or ensemble strategies to identify uncertain prediction.

Altogether, this single-image test filled the niche between theoretical achievements and practical implementation proving the applied model to be successful not only in terms of numbers.

**Slide 11: Strengths and Weaknesses**

**Strengths:**

* **Transfer learning** significantly boosts performance and reduces training time.
* MobileNetV2 is **lightweight and efficient**, ideal for mobile devices.
* Models show **strong generalisation** even on small datasets.

**Weaknesses:**

* MobileNetV2 offers **limited flexibility** for architectural changes.
* **Resizing CIFAR-10 images** for pretrained models introduces interpolation artifacts.
* The **custom CNN** is prone to overfitting without proper regularisation like dropout and early stopping.

**Slide 12: Code Implementation**

Python, TensorFlow, and Keras were used to run the entire code. The project covered:

1. Loading, preprocessing of dataset
2. CNN and MobileNetV2 Architecture setup
3. Callback training (e.g. early stopping)
4. Evaluation scripts with confusion matrices, accuracy/loss curves as well as individual image predictions

The code was split up, tidy, and reusable- this means it is a strong foundation to be used, in the future, in real-life systems or be developed.

**Slide 13: Conclusion**

In conclusion, it is obvious that deep learning is rather effective in object recognition, especially as a combination of transfer learning, thought-out preprocessing, and assessment procedure.

It was evident that the MobileNetV2 transfer learning model scored above the custom CNN in terms of accuracy and convergence. Validation sets, dropout, and early stopping were crucial in their model-building process with regard to generating generalisable models.

This project limits the overall conclusion to the fact that transfer learning is an outstanding approach in the case of small datasets and CIFAR-10 study is one of the great benchmarks to estimate classification models.

**References**

Brownlee, J. (2022). How to Develop a CNN From Scratch for CIFAR-10 Photo Classification. Machine Learning Mastery.  
<https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/> (Accessed: 11 July 2025)

Chollet, F. (2017). Deep Learning with Python. Manning Publications. Available at: <https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf> (Accessed: 11 July 2025)

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Available at:  
<https://ieeexplore.ieee.org/document/5206848> (Accessed: 12 July 2025)

Howard, A. G., et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.  
<https://arxiv.org/abs/1704.04861> (Accessed: 12 July 2025)

Keras CIFAR-10 example. Available at: <https://keras.io/api/datasets/cifar10/> (Accessed: 13 July 2025)

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. University of Toronto. Available at: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> (Accessed: 13 July 2025)

TensorFlow/Keras Documentation. Transfer learning and fine-tuning guide.  
<https://www.tensorflow.org/tutorials/images/transfer_learning> (Accessed: 13 July 2025)