Comparative Analysis for Object Detection using Transfer Learning

Department of Information and Communication Technology, MIT

Semester V Mini Project Report Submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Honours Degree

Of

Bachelor of Technology

in

Computer and Communication Engineering

by

Aanya Shantaram

220953438

Under the guidance of

Dr. Smitha N. Pai,
Professor and Head- Dept of ICT
Manipal Institute of Technology
Manipal | Karnataka | 576104

Date of Submission: 26th December 2024

Table of Contents

1) Abstract

This project evaluates the accuracy and performance of a number of convolutional neural network architectures for object detection using the CIFAR-10 dataset. Architectures like ResNet (ResNet-50 and ResNet-101), VGGNet (VGG-16 and VGG-19), MobileNet (MobileNetV2 and MobileNetV3) and InceptionNetV3 are compared for their accuracy and computational efficiency. For this purpose, transfer learning is employed where the models are pretrained on the ImageNet dataset using TensorFlow and Keras

2) Introduction

Object detection is a vital task in computer vision. It identifies and locates multiple objects in an image, and assigns them to precise bounding boxes. It does both image classification and localisation. It is widely used in domains such as autonomous driving, healthcare and surveillance.

CIFAR-10

The CIFAR-10 (Canadian Institute for Advanced Research) dataset is used in this experiment. It contains 60,000 images split into 50,000 training images and 10,000 test images, with 10 classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Training Set:

50,000 images

Each image is a 32x32 color image (RGB), resulting in a shape of (32, 32, 3).

Images are divided into 10 classes, with 5,000 images per class.

Test Set:

10,000 images

Same structure as the training set, with 1,000 images per class.

Transfer Learning

Training a high-accuracy model from scratch on the CIFAR-10 dataset can be challenging due to its complexity and relatively small image size. Hence transfer learning is used.

Transfer Learning refers to reusing a pretrained model on a new problem.

A machine can use the knowledge learned from a prior assignment to increase prediction about a new task in transfer learning. In this experiment multiple CNN models pre-trained on the ImageNet dataset are used to classify CIFAR-10 images. Here the top layers are replaced by custom dense layers for this classification.

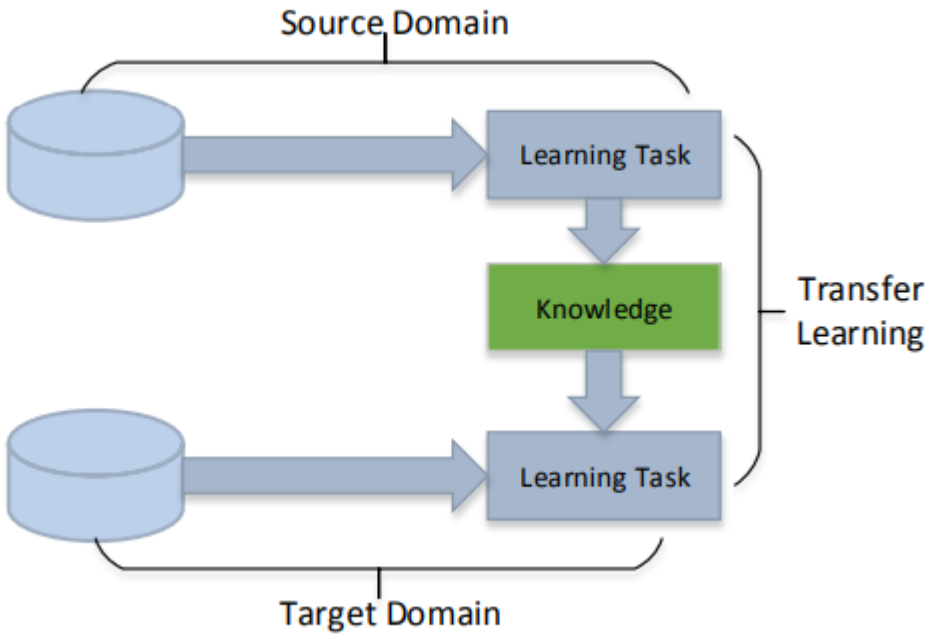
ImageNet

The ImageNet dataset is a large-scale visual database designed for use in object recognition and classification research. It contains over 14 million labeled images which has more than 20,000 categories. Pre-training on ImageNet enables models to generalize well when fine-tuned for smaller, task-specific datasets like CIFAR-10.

In this study, the performance of 9 models pre-trained on the ImageNet dataset are evaluated based on their performance on the CIFAR-10 dataset: ResNet-50, ResNet-101 MobileNet, MobileNetV2, MobileNetV3Small, MobileNetV3Large, VGG16, VGG19 and InceptionNetV3. These are state-of-the-art deep CNN architectures and since training them requires vast datasets and computational resources, transfer learning is used to save these resources.

3) Literature Review

Transfer learning has emerged as one of the important techniques in deep learning, especially for object detection. The idea of using pre-trained models trained on large-scale datasets, such as ImageNet, helps researchers adapt these models to specific tasks with reduced training times and improved performance.



Steps in transfer Learning: (Krishnan L)

1. Pretrained Model: Select a large deep learning model that is trained on a large amount of data preferably related to the object detection task.

2. Remove or Modify the Output Layer: The output layer of the pre-trained model is usually designed for a specific classification task. In object detection, you need to replace this layer with a new one that suits your task. The new output layer will have neurons corresponding to the number of object classes you want to detect.

3. Freeze Pre-trained Layers: Prevent the pre-trained weights from getting updated too quickly by freezing some of the initial layers of the network. These layers capture low-level features that are generally transferable across tasks.

4. Fine tuning: Train the modified model on your labeled object detection dataset. During training, both the new output layer and some of the earlier layers will adapt to the specific characteristics of your objects.

5. Evaluation and Fine-tuning: After the training process, evaluate the model's performance on a validation dataset. If necessary, fine-tune hyper-parameters and continue training.

Training deep convolutional networks from scratch is computationally expensive, requiring significant resources and large labeled datasets. Transfer learning addresses this by utilizing pre-trained feature extractors, drastically reducing both the time and data required.
Models pre-trained on datasets like ImageNet capture generalized features such as edges, textures, and object shapes, which can be fine-tuned for various specific tasks. This adaptability ensures better generalization even for smaller datasets.
Most successful object detection frameworks, such as Faster R-CNN and YOLO, use pre-trained backbones like ResNet or MobileNet for feature extraction, demonstrating the power of transfer learning in achieving competitive accuracy.

This benchmark is used in the ImageNet dataset, having over a million labeled images over 1,000 categories for training robust convolutional networks. ResNet, VGG, and MobileNet trained on ImageNet learn transferable features.

Frameworks used for Transfer Learning

PyTorch: Torchvision library contains a suite of pre-trained models that makes the workflow of transfer learning much easier.
Detectron2: This is a framework built on PyTorch, which supports advanced object detection pipelines and fine-tuning pre-trained models.
TensorFlow: TensorFlow Hub has pre-trained models that are easy to modify for object detection. For this study, Tensorflow will be used.

4) Technologies

1. Google Colab

Google Colab is a free cloud-based environment that allows people to work on machine learning experiments and research. It offers the space for writing, running, and sharing code using Jupyter notebooks written in Python. One of the most prominent features of Colab is the availability of free GPUs and TPUs for running computationally demanding tasks. It is also integrated with Google Drive which facilitates easy storage and sharing of notebooks among different users.It also has pre-installed libraries such as TensorFlow, Keras which are used in the project.

2. Keras and Tensorflow

TensorFlow provides a comprehensive set of tools for building and deploying machine learning models, including support for deep learning.
Keras was used as the primary framework to build, train, and evaluate deep learning models in an efficient manner in this project. Its API provided the convenience of using both pre-trained and custom architectures for CNN. Importing pre-trained models like ResNet, MobileNet, and EfficientNet from tensorflow.keras.applications allowed the use of transfer learning to reduce training time and effort.

3. ResNet

- ResNet(Residual Network) is a CNN architecture that was published in 2015 by researchers at Microsoft Research.
- It effectively solved the problem of the vanishing/exploding gradient.
- ResNet has multiple residual blocks, allowing gradients to flow through skip connections.
- Available in different depths: ResNet-50, ResNet-101, etc., where the number 50 and 101 indicate the total number of layers.
- Each block has convolutional layers, batch normalization, and an identity shortcut connection.
- Efficiently deep, enabling it to learn more complex patterns without it affecting the performance negatively much.

4. MobileNet

- MobileNet is a CNN architecture which was published in 2017.
- Mobilenet is a model which does the same convolution as done by CNN to filter images but in a different way than those done by the previous CNN.
- It uses the idea of depth convolution and point convolution which is different from the normal convolution as done by normal CNNs.

- This increases the efficiency of CNN to predict images and hence it's very efficient when computational resources are limited.
- MobileNetV2: Introduced inverted residuals and linear bottlenecks, further optimizing for speed and memory.
- MobileNetV3: Combines MobileNetV2's architecture with squeeze-and-excitation modules and neural architecture search.

5. ***VGG16***
   - VGG16 was published in 2014 and is one of the simplest (among the other cnn architectures used in Imagenet competition).
   - This network contains 16 layers in which weights and bias parameters are learnt.
   - A total of 13 convolutional layers are stacked one after the other and 3 dense layers for classification.
   - The number of filters in the convolution layers follow an increasing pattern (similar to decoder architecture of autoencoder).
   - The informative features are obtained by max pooling layers applied at different steps in the architecture.
   - The dense layers consist of 4096, 4096, and 1000 nodes each.
   - The cons of this architecture are that it is slow to train and produces the model with very large size.

6. ***VGG19***

   VGG19 is a similar model architecture as VGG16 with three additional convolutional layers, it consists of a total of 16 Convolution layers and 3 dense layers.

7. ***InceptionNet***

   - InceptionNet, also known as GoogleNet, consists of 22 layers and was the winning model of the 2014 image net challenge.
   - Inception modules are the fundamental block of InceptionNets. The key idea of inception module is to design good local network topology (network within a network)
   - These modules or blocks acts as the multi-level feature extractor in which convolutions of different sizes are obtained to create a diversified feature map
   - The inception module also consists of 1 x 1 convolution blocks whose role is to perform dimensionality reduction.

- By performing the 1x1 convolution, the inception block preserves the spatial dimensions but reduces the depth. So the overall network's dimensions are not increased exponentially.
- Apart from the regular output layer, this network also consists of two auxiliary classification outputs which are used to inject gradients at lower layers.

5) Methodology

Data SetUp

1. Kaggle Dataset Access:

Kaggle API was configured using kaggle.json to download the CIFAR-10 dataset (cifar-10.zip). The dataset was extracted, and .7z files (containing training data) were handled using the py7zr library.

2. Dataset Structure:

Training images (train.7z) were organized in a folder, and labels were provided in a CSV file (trainLabels.csv).

3. Label Mapping:

Labels are mapped to numerical values using a dictionary ({'airplane': 0, 'automobile': 1…).

Data Preparation

1. Image Processing: Training images were loaded as NumPy arrays using PIL, and a dataset was created by matching image IDs with labels. Image data (X) and labels (Y) were converted into NumPy arrays for further processing.
2. Train-Test Split: The dataset was split into 80% training and 20% testing subsets using train_test_split.
3. Normalization: Pixel values were scaled to a range of 0-1 to improve model convergence.

Simple Neural Network Model

1. Architecture: A simple CNN was built using TensorFlow and Keras with 2 convolutional layers with ReLU activation, followed by max-pooling layers and a flattening layer to convert feature maps into a 1D vector. Also with dense layers for feature learning and classification (with 10 output nodes and softmax activation).

2. Compilation: The model was compiled using the Adam optimizer, sparse categorical cross-entropy loss, and accuracy as the evaluation metric.
3. Training: The model was trained on scaled training data (X_train_scaled) for 10 epochs with a validation split of 10%.
4. Evaluation: The model achieved a test accuracy score on the test data (X_test_scaled).

General Steps for each model in the process

1. Importing the Pre-trained Model:
    - Import the desired pre-trained model (egs ResNet50, MobileNet, etc) from `tensorflow.keras.applications`.
    - Use `weights='imagenet'` to leverage weights pre-trained on the ImageNet dataset.
    - Set `include_top=False` to exclude the final dense layer for classification. This is used for transfer learning and makes it so only the convolutional base is remaining, where the convolutional layers serve as a feature extractor.
    - Define the input_shape as per the dataset being used (egs (256, 256, 3) for 256x256 RGB images).

2. Freezing the Convolutional Base:
    - Freeze the layers of the pre-trained model (`convolutional_base.trainable = False`) to retain the learned features during the initial training phases.

3. Building the Model:
    - Add layers to prepare the input (egs `UpSampling2D` or `Resizing` if resizing is required).
    - Add the pre-trained convolutional base as the feature extractor.
    - Flatten the extracted feature maps using a Flatten layer or reduce dimensionality using `GlobalAveragePooling2D.`
    - Add dense layers with decreasing numbers of neurons (egs: 128, 64) for fine-tuning the feature extraction.
    - Include `BatchNormalization` layers to stabilize and speed up training.
    - Use `Dropout` for regularization to prevent overfitting.
    - Add a final dense layer with softmax activation for classification into the 10 classes.

4. Compiling the Model:
   - ❖ Choose an optimizer (egs: RMSprop, Adam) with a small learning rate (egs 2e-5 or 1e-4) to prevent extreme updates to the pre-trained weights. Typically one would choose RMSprop for deep models and Adam for more lightweight ones.
   - ❖ Use `sparse_categorical_crossentropy` for the loss function since the labels are integers.
   - ❖ Include accuracy as a metric for evaluation.

5. Training the Model:
   - ❖ Fit the model using the training dataset (X_train_scaled, Y_train) and validate on `validation_split=0.1`.
   - ❖ Train for 10 epochs.

6. Evaluating the Model:
   - ❖ Evaluate the model's performance on the test dataset (X_test_scaled, Y_test).
   - ❖ Print the test accuracy.

7. Visualizing Results:
   - ❖ Plot training and validation loss over epochs to understand training behavior.
   - ❖ Plot training and validation accuracy to ensure the model is learning effectively.

Differences between models

1. ***Resnet (ResNet50, ResNet101)***

   Uses larger input sizes (e.g., 256x256) so requires more UpSampling2D layers for CIFAR-10 images which are of size (32,32)

2. ***MobileNet (MobileNet, MobileNetV2, MobileNetV3Small, MobileNetV3Large)***

   Smaller input sizes (32x32 for CIFAR-10) are acceptable for this model. Here `GlobalAveragePooling` is used to reduce feature maps instead of flattening them.

3. ***VGG (VGG16,VGG19)***

   Smaller input sizes (32x32 for CIFAR-10) are acceptable for this model too. It has fewer dense layers compared to others and relies heavily on the final fully connected layers.

4. ***InceptionV3***

   It requires resizing the input to larger dimensions of 150 x 150. Here `GlobalAveragePooling` is used to reduce the dimensionality of feature maps.

6) Results

Model	**Accuracy (in %)**
Simple base neural network	67.90
ResNet50	93.73
ResNet101	94.58
MobileNet	79.42
MobileNetV2	90.46
MobileNetV3Small	79.18
MobileNetV3Large	81.63
VGG16	58.46
VGG19	56.95
InceptionNetV3	92.54

Table : Observed accuracies of models

1. ***Simple Neural Network: 67.9%***

   This is a very simple NN which lacks the complexity or depth to extract meaningful features from the dataset. So while it performs much better than random guessing, its simple architecture limits its performance.

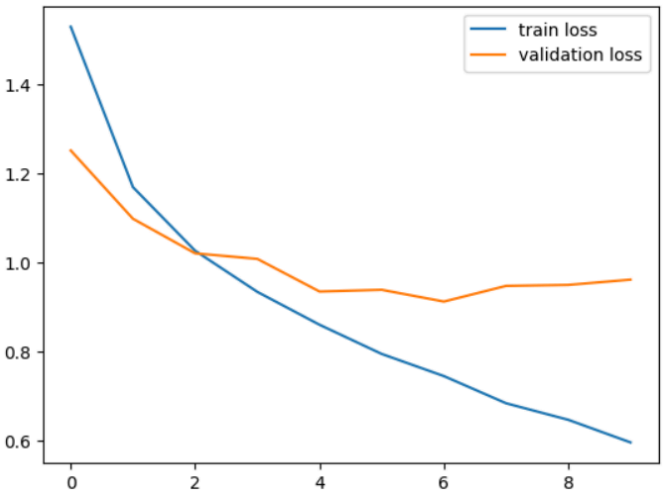


Figure 1: Training and validation loss

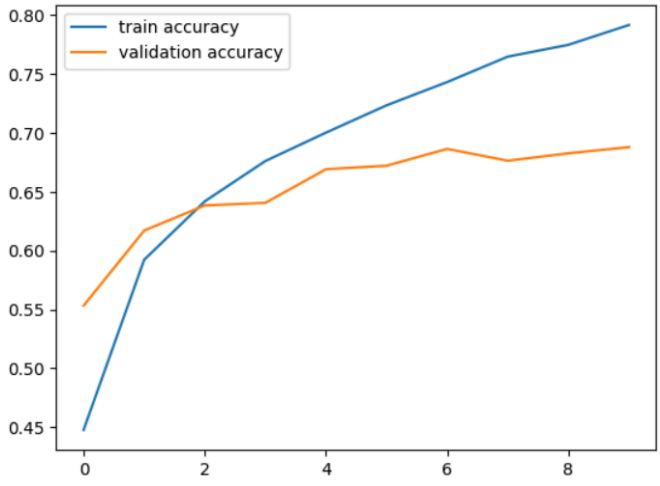


Figure 2: Training and validation accuracy

2. ***ResNet50: 93.73%***

ResNet works very well on datasets with sufficient complexity due to its depth and ability to avoid vanishing gradients. It also improves object detection speed. Overall it shows a high accuracy due to this.

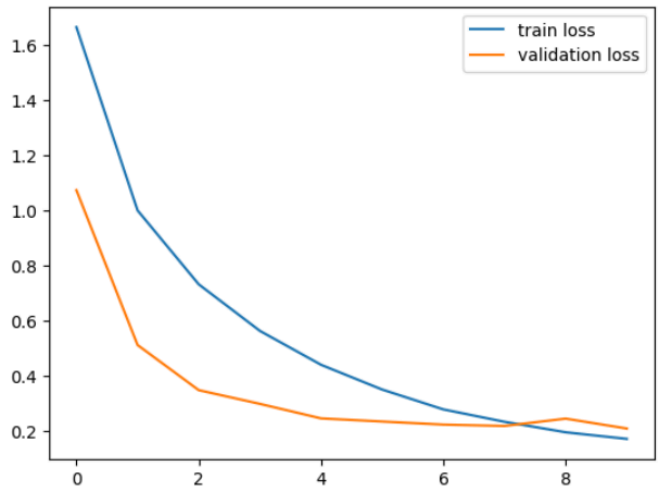


Figure 3: Training and validation loss

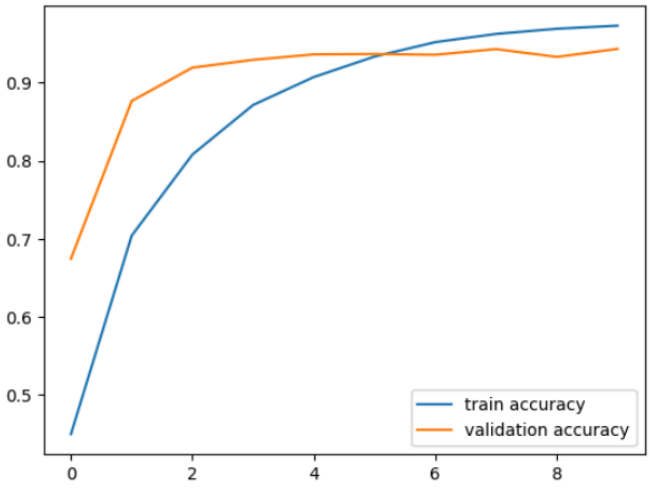


Figure 4: Training and validation accuracy

3. ***ResNet101:94.58%***

   This is even deeper than ResNet50 and captures even more detailed hierarchical feature, allowed it to have the highest accuracy.

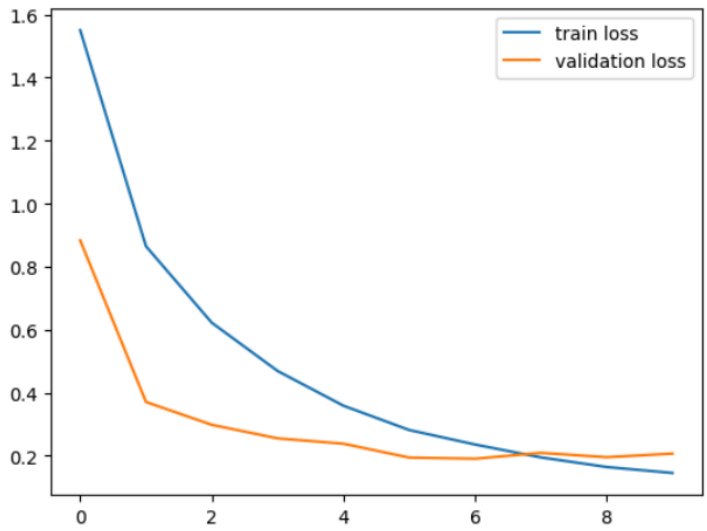


Figure 5: Training and validation loss

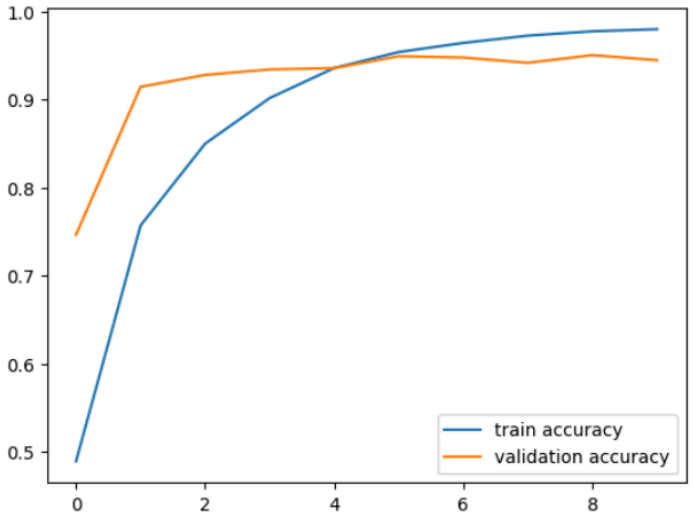


Figure 6: Training and validation accuracy

4. ***MobileNet: 79.42%***

   MobileNet focuses on more lightweight and efficient computation with limited computational resources rather than high accuracy. Due to this its accuracy is not as high but it has high speed and resource efficiency.

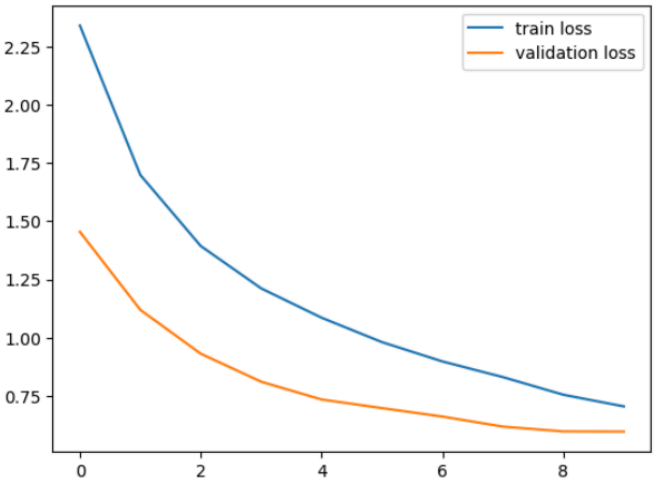


Figure 7: Training and validation loss

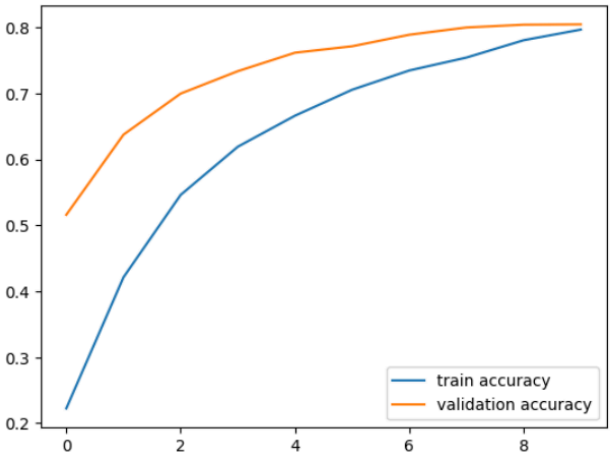


Figure 8: Training and validation accuracy

5. ***MobileNetV2: 90.46%***

MobileNetV2 has better accuracy and efficiency due to its improved architectural innovations. It includes shortcut connections between bottlenecks, enhancing gradient flow and making training more effective.

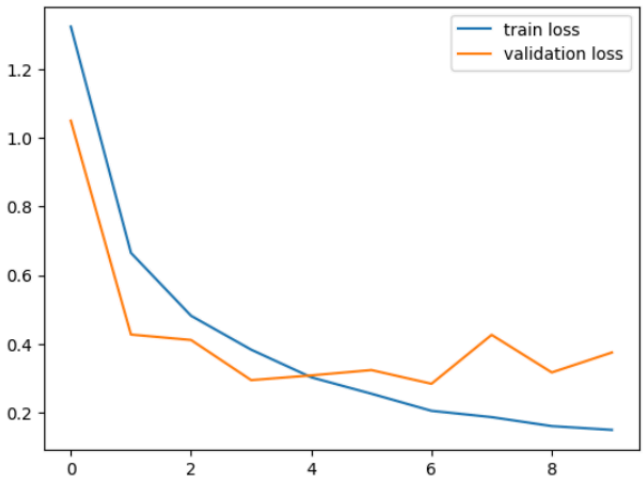


Figure 9: Training and validation loss

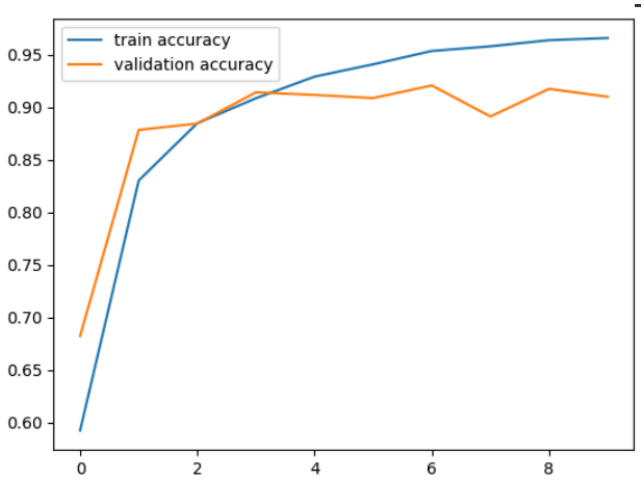


Figure 10: Training and validation accuracy

6. ***MobileNetV3Small: 79.18%***

MobileNetV3Small is designed for highly resource limited environments which limits its accuracy. It is not able to capture as many complex patterns compared to MobileNetV2 due to this.

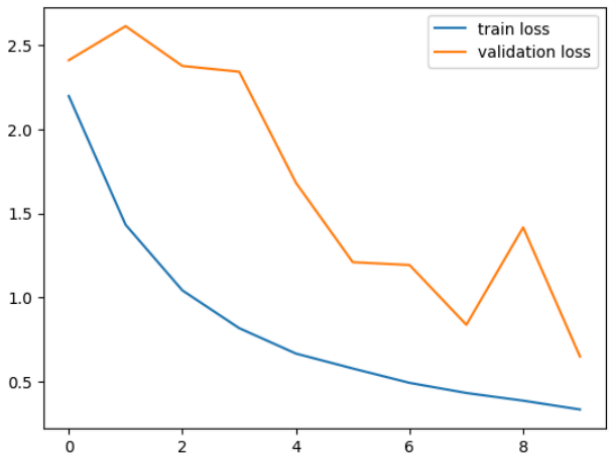


Figure 11: Training and validation loss

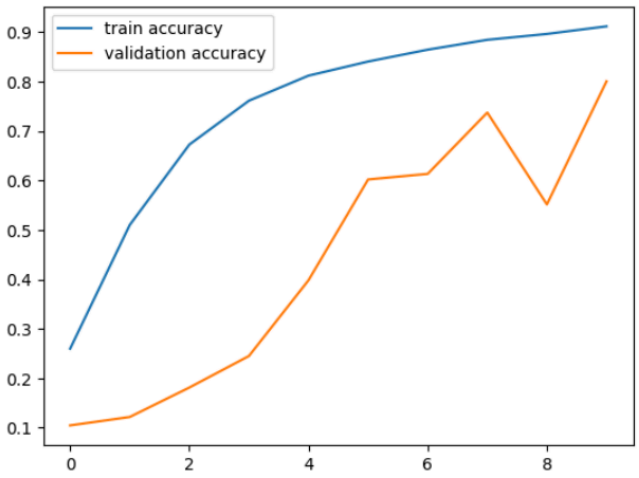


Figure 12: Training and validation accuracy

7. ***MobileNetV3Large: 81.63%***

MobilNetV3Large has slightly more capacity than MobileNetV3Small, leading to slightly better performance. MobileNetV3 is faster and more accurate than MobileNetV2 on classification tasks, but this is not necessarily true for our task which is object detection.

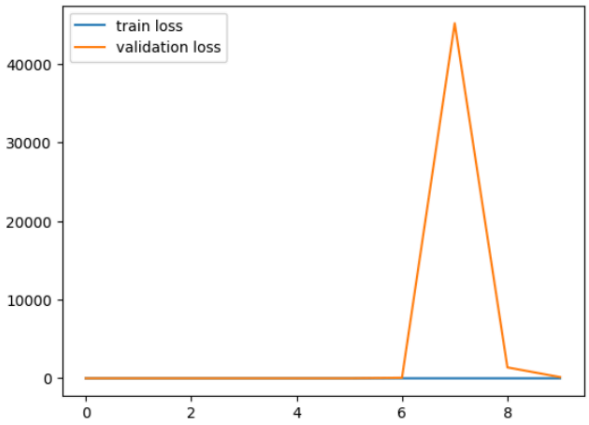


Figure 13: Training and validation loss

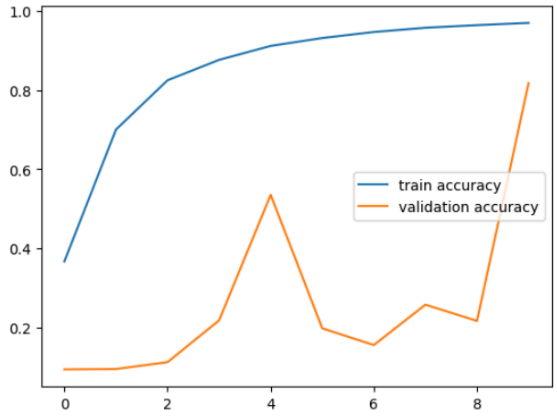


Figure 14: Training and validation accuracy

8. ***VGG16: 58.46%***

VGG16 has a simple uniform architecture but has a very large number of parameters, making it prone to overfitting. It lacks residual connections or depthwise separable convolutions, limiting its effectiveness. This explains its extremely low accuracy.

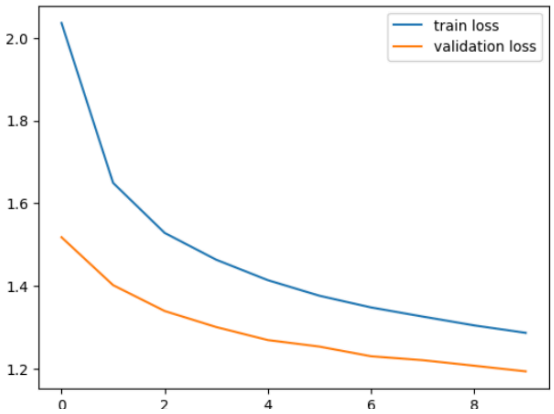


Figure 15: Training and validation loss

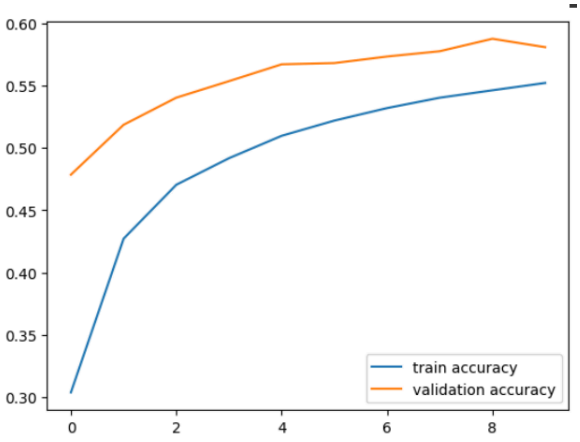


Figure 16: Training and validation accuracy

9. *V6619: 56.95%*

This is similar to VGG16 but with additional layers, which may worsen the overfitting problem without improving any generalization.

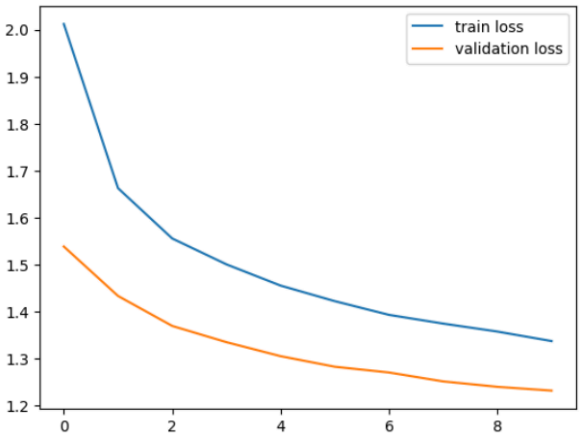


Figure 17: Training and Validation loss

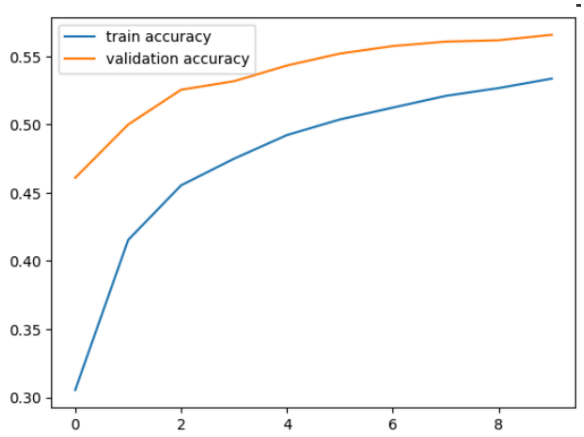


Figure 18: Training and Validation Accuracy

10. *InceptionNetV3: 92.54%*

InceptionNet uses mixed convolutions of different kernel sizes allowing it to capture input data through multiple convolutional layers (with different sizes) in parallel. Hence it has an ability to handle complex datasets which is why it has high accuracy just slightly below ResNet101.

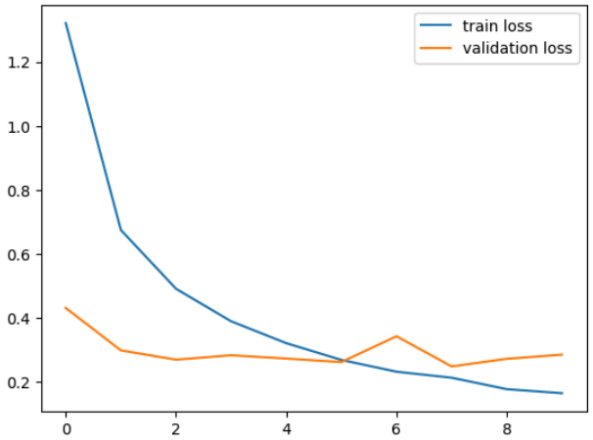


Figure 19: Training and Validation Loss

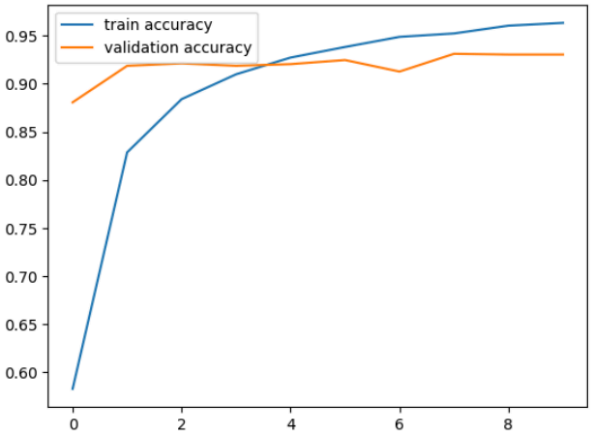


Figure 20: Training and Validation Accuracy

7) Conclusion

In conclusion, we have calculated the results of the effectiveness of transfer learning in object detection by using pretrained CNNs on the CIFAR-10 dataset. Amongst all the models, ResNet101 and ResNet50 performed with the highest accuracy of 94.58% and 93.78%. This shows how good the Residual Network Architectures are at enabling deep architectures and efficient feature learning, as well as their ability to generalise well to smaller datasets.

VGG16 and VGG19 are strong CNN models for image classification but struggle with object detection due to limitations in architecture and optimisation. The same can be said for MobileNet and MobileNetV3.

MobileNetV2 and InceptionNetV3 showed very strong performance. It is also important to note that under limited resource constraints MobileNetV2 comes ahead in terms of resource efficiency and accuracy. So selecting the right model depends on the task's and system's computational constraints and the level of accuracy needed.

Overall this comparative analysis shows the importance of model selection in transfer learning for object detection by providing insights into accuracy. A note to be made is an accurate log of training time for each model has not been included in this study due to Google Colab's free tier being operated on shared GPU resources i.e. performance is affected depending on the number of users accessing the GPU at a time. Regardless, the study provides insights into the accuracy of different models. Future work can be done to fine-tune the architectures or to explore larger datasets.

8) Works Cited

Jain, Dr. Madhur, et al. "Comparison of VGG-16, VGG-19, and ResNet-101 CNN Models for the purpose of Suspicious Activity Detection." *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, vol. ISSN : 2456-3307, Volume 9, no. Issue 1, January-February-2023, pp.121-130, https://ijsrcseit.com/CSEIT239012.

L, Navaneetha Krishnan. *Transfer Learning for Object Detection*, 15 October 2023, https://medium.com/@nkrishnan.laksh/transfer-learning-for-object-detection-a499b7bdd12b.