Neural Networks & Deep Learning

Car Type Classification: Project documentation

Classifying Cars images using CNN architecture

كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

Team Information :

- Idea number : 7
- Discussion Time : 10:40

	ID	Full Name [In Arabic]	Attendance [Handwritten Signature]	Final Grade
1	20210111	أحمد محمد مقبول عبد الهادي		
2	20210121	أحمد مصطفى سيد أحمد		
3	20210597	عمر أيمن فاروق أحمد		
4	20210281	حسام أحمد صلاح نورالدين		
5	20210023	أحمد إسماعيل سالم		
6	20210098	أحمد محمد بسيوني		

1) Upload to GitHub: (2 grades)
2) Preprocessing: (5 grades)
3) ResNet Implementation: (4 grades)
4) Xception Finetuning: (4 grades)
5) DenseNet Finetuning: (4 grades)
6) Results for your models (accuracy with visualization, loss curve with visualization, confusion matrix with visualization, recall, precision, f-score, ROC, AUC graph) (6 grades)
7) Your documentation includes an explanation of the three architectures with step-by-step details and graphs. It also incorporates references used, most importantly the papers that introduced these architectures. (3 grades)
8) Your Documentation includes also a comparison of the three models is conducted based on their results first, the pros and cons of each architecture are highlighted, and explaining the advantages of a specific architecture for the given task and dataset. (2 grades)

About the project

- https://github.com/AhmedMakboul/Car-Classification
- You can find the project code in this repository.

Our task is to use CNN-based architecture to classify the images of the cars to classes.

Dataset

- This dataset consists of various types of cars. The dataset is organized into 2 folders (train, test) and contains subfolders for each car category. There are 4,165 images (JPG) and **7 classes of cars**.
- Classes of the data
  - Audi
  - Hyundai Creta
  - Mahindra Scorpio
  - Rolls Royce
  - Swift
  - Tata Safari
  - Toyota Innova



Architecture

We leverage some of the most renowned and widely used architectures in convolutional neural networks (CNNs), such as ResNet, Xception, and DenseNet. Each of these architectures represents a significant milestone in the evolution of deep learning, reflecting the cumulative expertise and innovative contributions of researchers. These designs have substantially enhanced the ability of machines to process and understand visual data, pushing the boundaries of computer vision applications.

ResNet

ResNet (Residual Network) is a groundbreaking deep learning architecture introduced by Microsoft Research in 2015, **primarily designed to address the problem of vanishing gradients in very deep neural networks**.

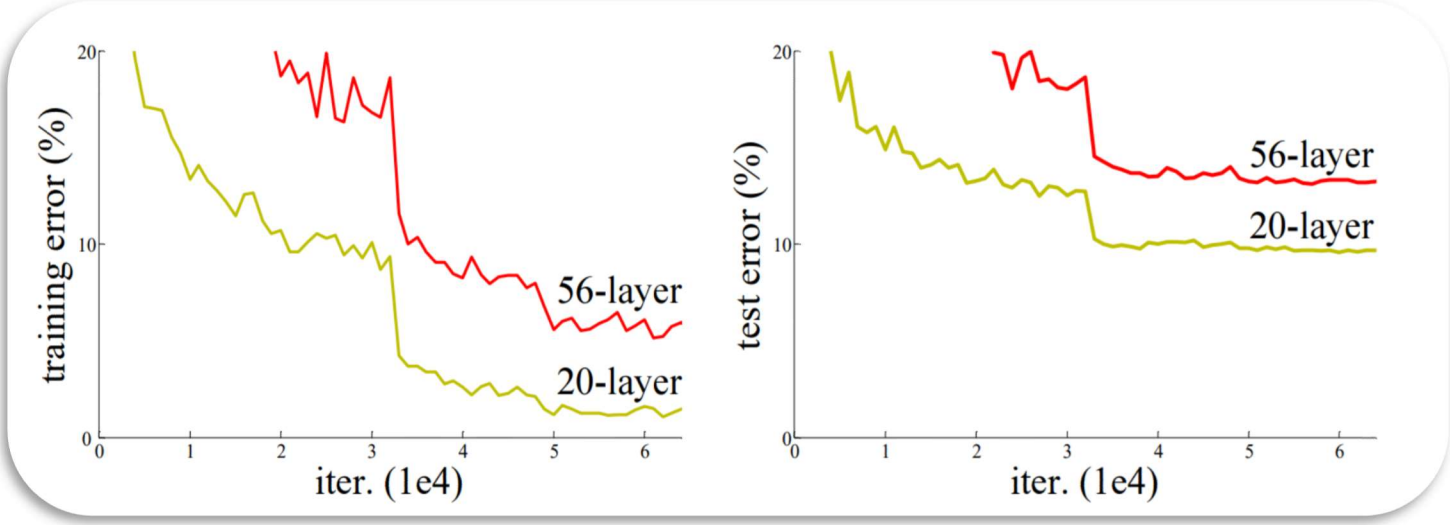
It introduced the concept of *residual learning*, **where shortcut connections allow the model to skip layers and pass information directly to deeper layers**. This innovation enables the training of extremely deep networks, such as ResNet-50, ResNet-101, and ResNet-152, **without significant degradation in performance**. ResNet's success has made it a cornerstone in computer vision, excelling in tasks like image classification, object detection, and segmentation.

Residual Networks, is a type of a Convolutional Neural Network (CNN) architecture that was introduced to address the **challenges** of training **very deep neural networks**

Why we need Deeper neural networks?

- **Feature Abstraction (hierarchical features**)
- **Expressiveness (more complex non-linear)**
- **End-to-End classifiers (no need for manual features)**

In theory deeper networks **should at least give us the same or higher performance** than the less layers network.

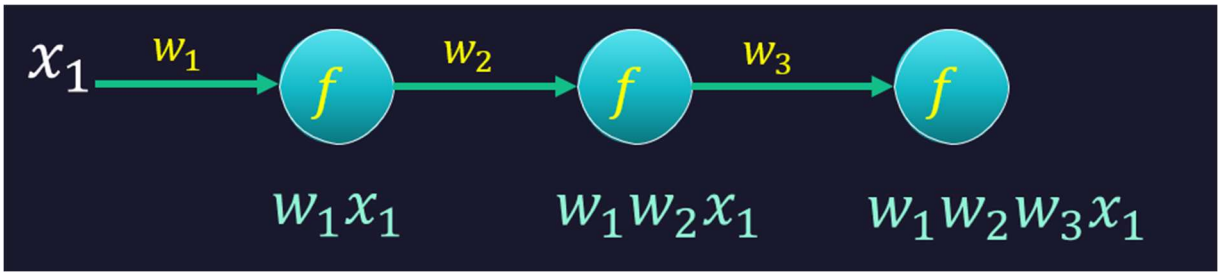


Why that is not True?

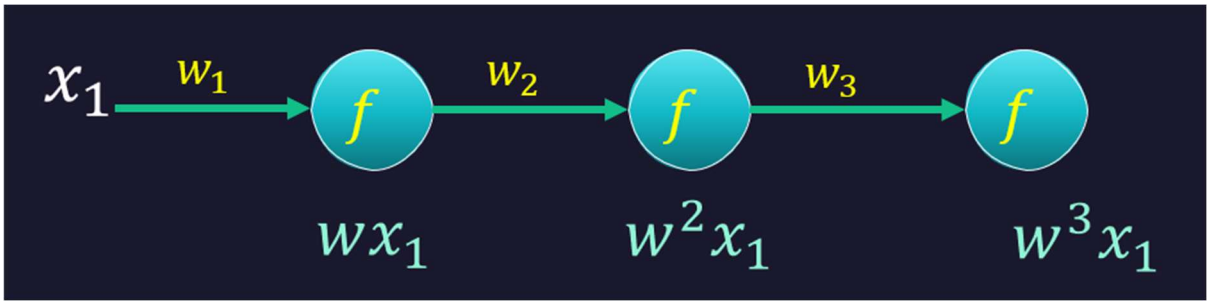
- Due to some challenges with depth
  - **vanishing/exploding gradients**
    - **This issue arises during the training of deep networks when the gradients become very small (vanishing) or very large (exploding), making it difficult for the network to learn effectively**
  - **Overfitting**
  - **Computational Intensity**
  - **Memory Constraints**
    - **More parameters means more memory is needed and computations**

Vanishing/exploding gradients

- Suppose Deep neural network (DNN) with a lot of layers
- Each layer multiplies the input X with a weight w and sends the result through an activation function
- We will ignore the bias for simplicity
- The activation function just passes the input without any non-linear transformations



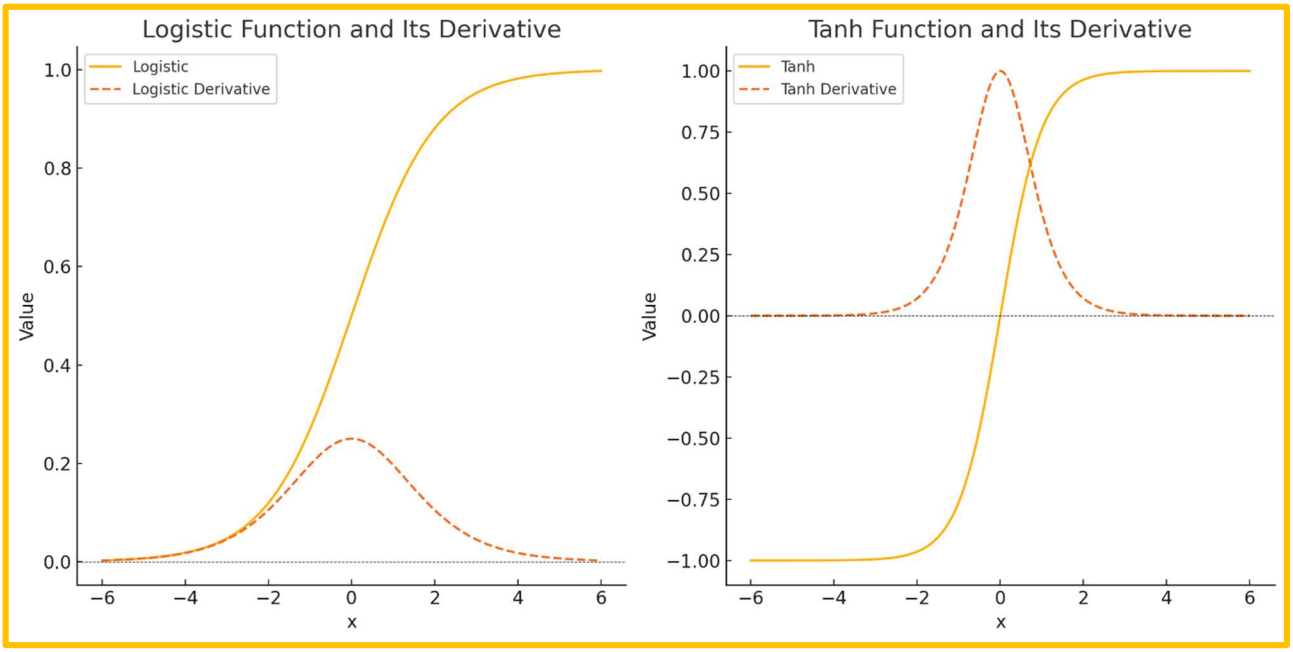
- Suppose all the weights are **EQUAL**



- **By the time you reach the third layer, you take the weight to the power of 3 $w^3 = 0.6^3 = 0.216$**
- **Imagine a network with 15 layers $w^{15} = 0.6^{15} = 0.0047$ [Vanishing]**
- **As you can see, the weight is now vanishingly small, and it further shrinks exponentially with every additional layer**
- **If your weight is larger than 1 let's say 1.6**
- **At the third layer $w^3 = 1.6^3 = 4.016$**
- **At the 15th layer $w^{15} = 1.6^{15} = 1152.92$ [Exploding]**

How to alleviate from Vanishing/ Exploding gradients?

- **Avoid some activation functions like the logistic and tanh, to avoid the saturation and decreasing the derivatives, for example the logistic derivative maximum value is about 0.25.**



- **Batch Normalization**
  - **Normalize activations within a layer, stabilizing training and reducing internal covariate shift(*change in the distribution of the input*).**
- **Skip Connections (ResNets)**
- **Long Short-Term Memory Networks (LSTMs) and Gated Recurrent Units (GRUs)**

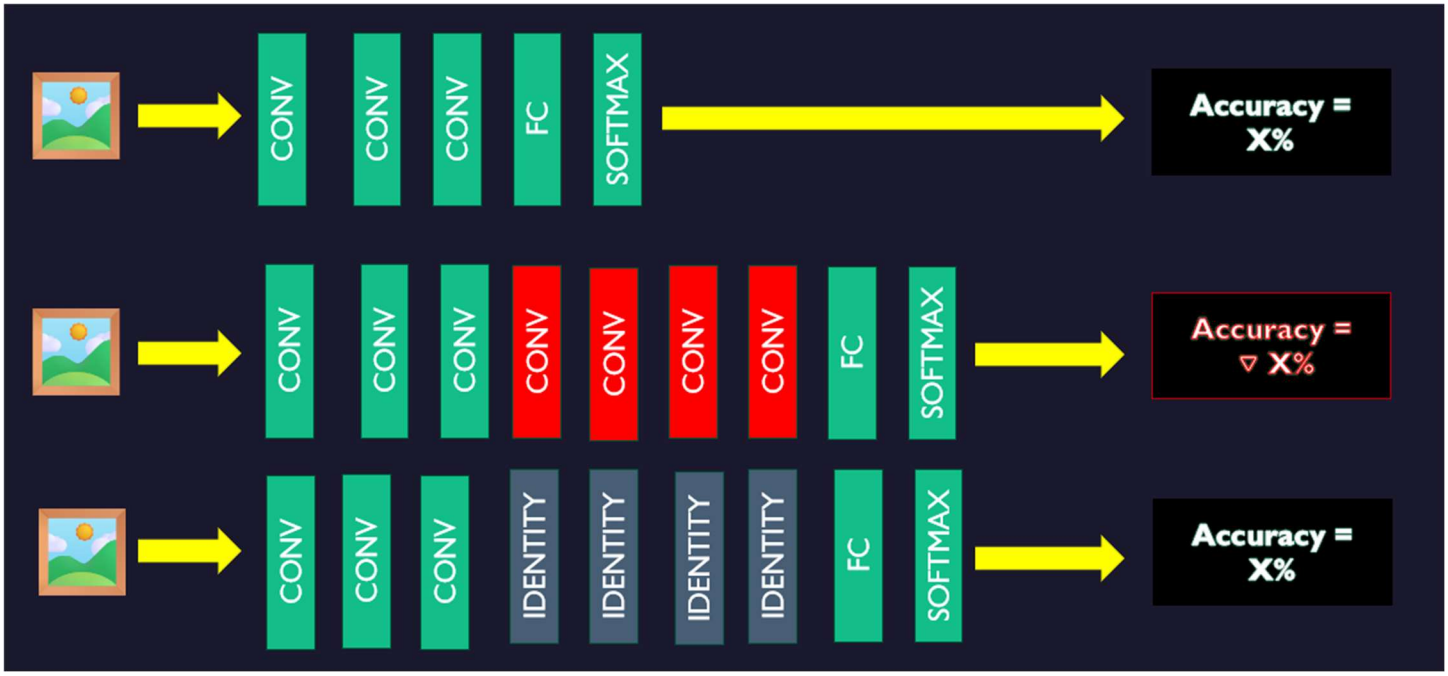
Convolution 1×1

- **Pointwise convolution**
- **Unlike traditional convolutions that apply filters across spatial dimensions (width and height), a 1x1 convolution operates solely on the channel dimension of the input volume.**
- **Doesn't change the spatial dimensions (width and height) but can increase or decrease the depth by applying k filters of the 1×1 filters**
- **1x1 convolutions seem linear due to their small receptive field, they are typically followed by non-linear activation functions such as ReLU, which introduces non-linearity to the network and allows for more expressive power.**
- **Feature Interactions: 1x1 convolutions can also be used to capture interactions between different channels of the input volume**

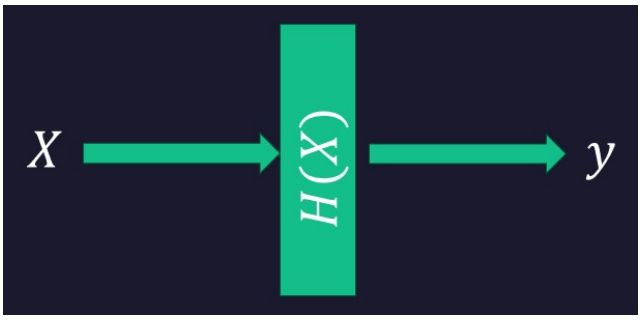
- By applying filters across channels, the network can learn to combine features from different channels, potentially capturing more complex patterns and relationships within the data.(learn from colors channels 🎨)
- Bottleneck Layers: In architectures like ResNet, 1x1 convolutions are used as bottleneck layers, where they reduce the number of channels before applying larger convolutions, thereby improving computational efficiency.

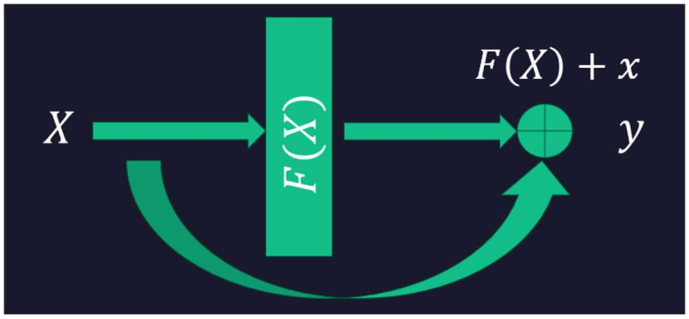
Degradation problem that ResNets try to solve

- It's not overfitting.
- The degradation problem occurs when increasing the depth of a neural network leads to a drop in training accuracy, not just testing accuracy.
- This means that even the network's ability to fit the training data deteriorates as the depth increases.
- This happens because deeper networks can suffer from optimization difficulties, such as vanishing gradients and poor weight initialization, making it hard for the model to learn effectively.
- Adding more layers does not always improve performance because these layers may end up redundant or even harmful.
- ResNet's Solution: ResNet introduces skip connections (identity mappings), allowing gradients to flow directly through the network and bypass unnecessary layers
- . This mitigates the issue, ensuring that deeper networks perform at least as well as their shallower counterparts, improving training accuracy.

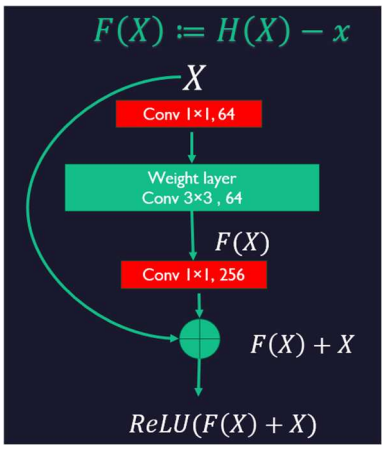


- $H(X)$ is the true mapping function we want to learn
- It's hard to make $H(X)$ learn the identity mapping
- What lacks $F(X)$ to reach the desired $H(X)$
  - $F(X) \coloneqq H(X) - x$
  - $H(X) \coloneqq F(X) + x$

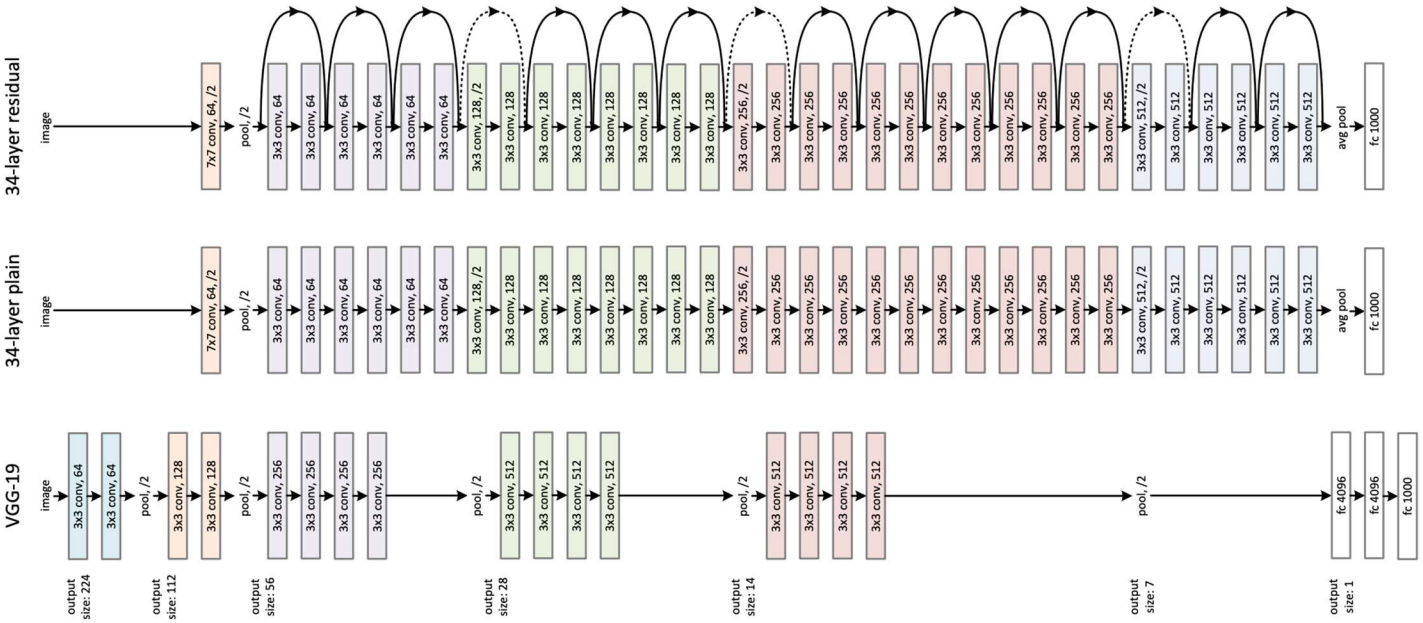




- The formulation of $F(x) + x$ can be realized by feedforward neural networks with "shortcut connections"
- Identity shortcut connections add neither extra parameter nor computational complexity.
- The purpose of the shortcut connection is to allow the network to learn the residual function, which captures the difference between the input and output of the block.
- The combined output of the main path and the shortcut connection is passed through an activation function, typically ReLU
- The activation function introduces non-linearity to the residual block, allowing the network to learn complex mappings.
- Residual block with conv 1×1
- For each residual function F, we use a stack of 3 layers instead of 2
- The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions
- leaving the 3×3 layer a bottleneck with smaller input/output dimensions

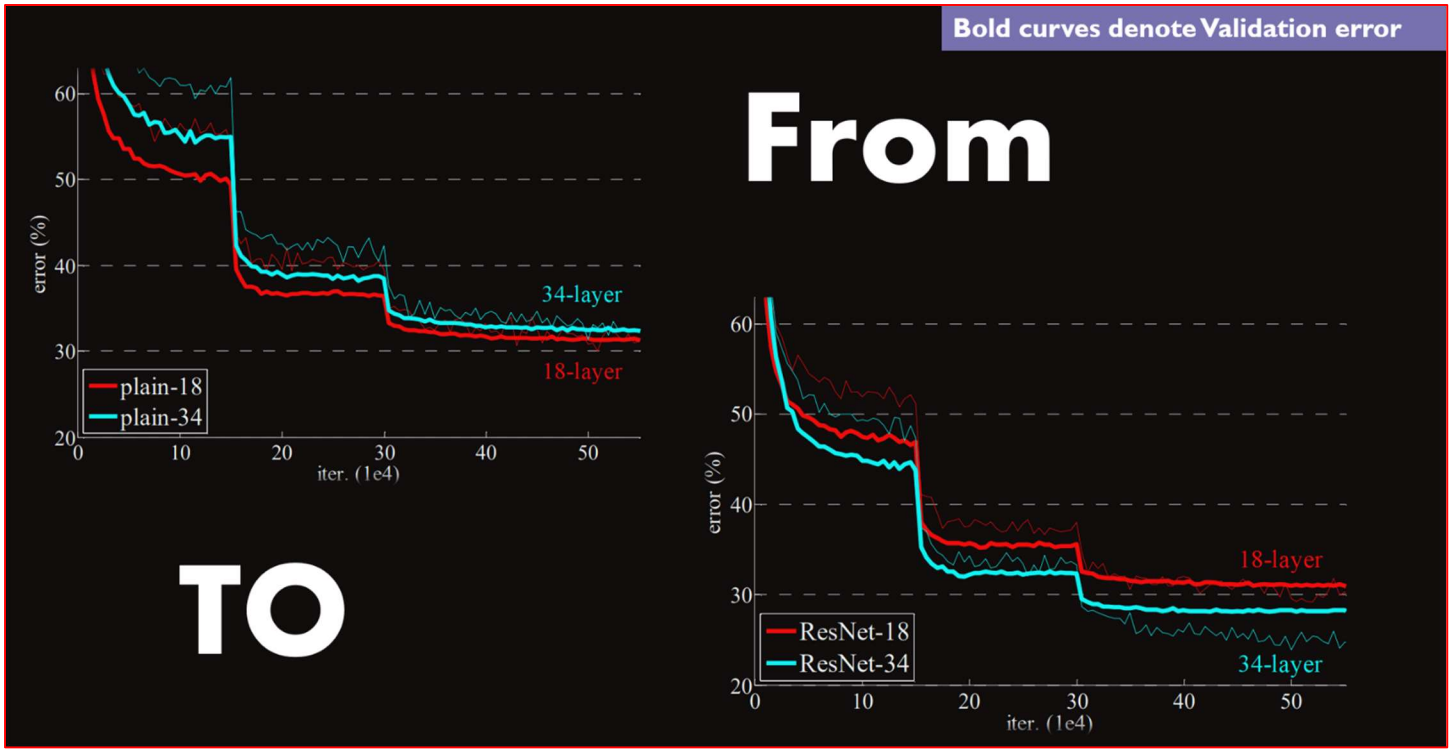


- The use of Bottleneck Resnet was motivated by their ability to reduce computational cost
- Non-bottleneck Resnet they would achieve a high accuracy but with higher cost in the more depth architectures
- Despite their reduced computational cost, bottleneck ResNets maintain high accuracy even with increased depth.
- This degradation problem is not exclusive to plain (non-residual) networks but is also witnessed in bottleneck ResNets (saturation in accuracy and then degradation)



- 34-Layer ResNet is better than 18-layer ResNet (by 2.8%)
  - lower training error
  - Degradation problem is addressed
- 34-layer ResNet reduced top-1 error by 3.5 percent compared to Plain Net
- 18-layer plain/residual nets are comparably accurate, but…
  - 18-layer Resnet converges faster
  - ResNet eases optimization by providing
    - faster convergences

TOP-1 Error	Plain networks	ResNet
18-layer	27.94%	27.88%
34-layer	28.54%	25.03% ▼



- **Identity Vs. projection Shortcuts**
  - **Identity Shortcuts:**
    - **Skip connection**
    - **the input to the block is directly added to the output of the block without any transformation.**
    - **used when the dimensions of the input and output feature maps are the same**
- **Projection Shortcuts:**
  - **skip connections**
    - **the input to the block is projected to match the dimensions of the output feature maps before being added to the output of the block.**
    - **Using Conv 1×1 to do the projection**
- **Zero Padding Shortcuts:**
  - **zero padding may be used in the shortcut connections to match the dimensions of the input and output feature maps before adding them together. This ensures compatibility for element-wise addition.**
- **The dashed connections is where we choose one of these strategies to solve the dispatching**
- **These options are tested in the paper**
  - **A. Zero Padding Shortcut**
  - **B. Projection Shortcuts for increasing Dimensions**
  - **C. All Shortcuts are projections**

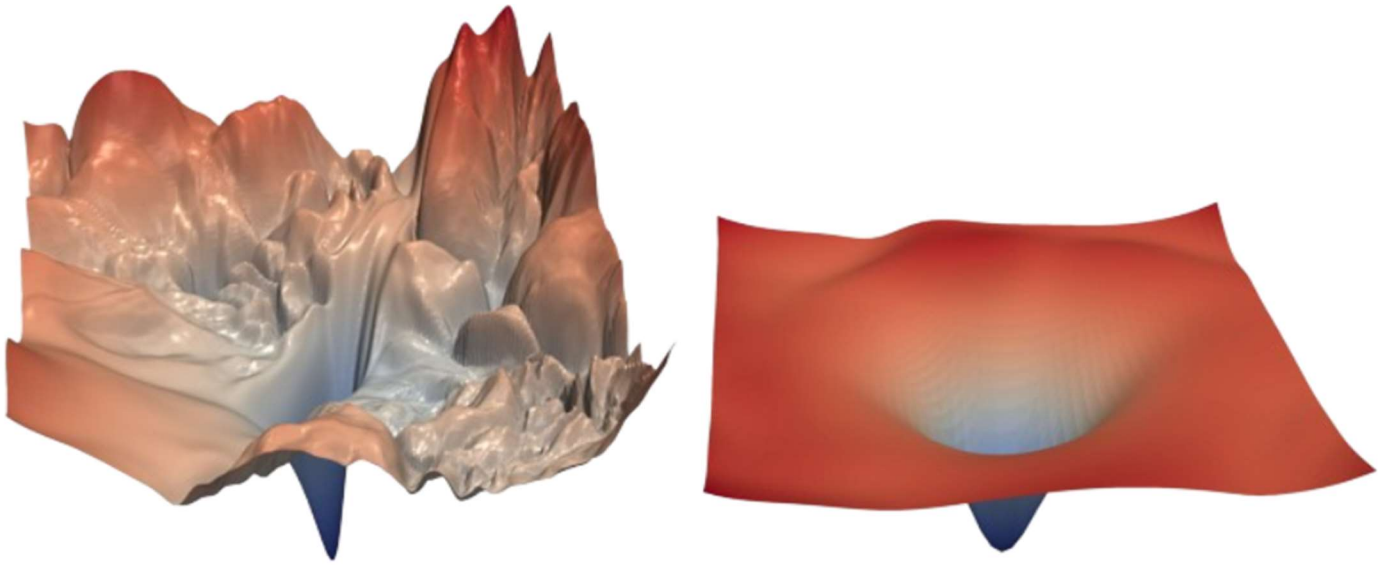
Model	Top-1 error	Top-5 error
Plain-34	28.54%	10.02%
ResNet-34 A	25.03%	7.76%
ResNet-34 B	24.52% ▼	7.46%
ResNet-34 C	24.19% ▼	7.40 %

- **All three options are considerably better than the plain counterpart.**
- **B is slightly better than A. We argue that this is because the zero-padded dimensions in A indeed have no residual learning**
- **C is marginally better than B, and we attribute this to the extra parameters introduced by many (thirteen) projection shortcuts**
- **The paper prefeed to use B to reduce number of parameters**

Model	Top-1 error
VGG	24.4%
ResNet-34 B	21.84% ▼
ResNet-34 C	21.53% ▼
ResNet-101	19.87 ▼
ResNet-152	19.38 ▼

VGG-19 vs ResNet

- **VGG-19 do 19.6 Billion Float Operations (FLOP)**
- **Plain network with 34 layer 3.6 Billion FLOPs**
- **ResNet-34 layer 3.6 Billion FLOPs**
- **ResNet Solve the problem of degradation**
- **caused by vanishing/exploding gradients by the Construction “ensemble of many shallow networks”**
- **ResNet can be deep to reach 50/100/152 layers**



- **On the left : the loss without skip connections**
- **On the right : the loss with skip connections**
- **This is one reason why ResNet is faster to converge.**

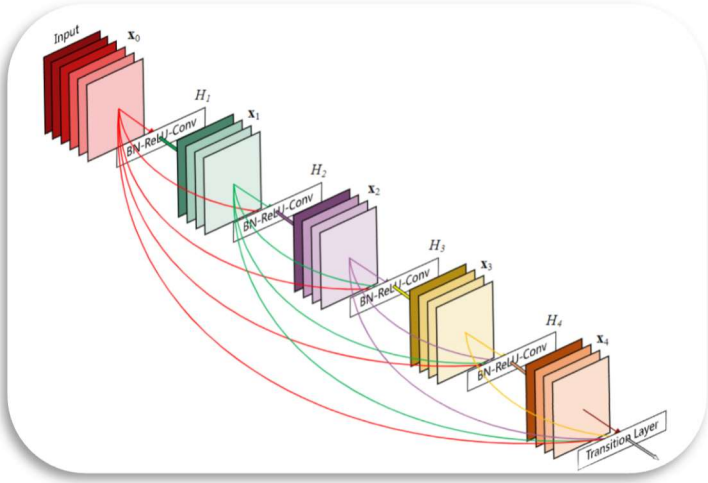
DenseNet: skip connections via concatenation

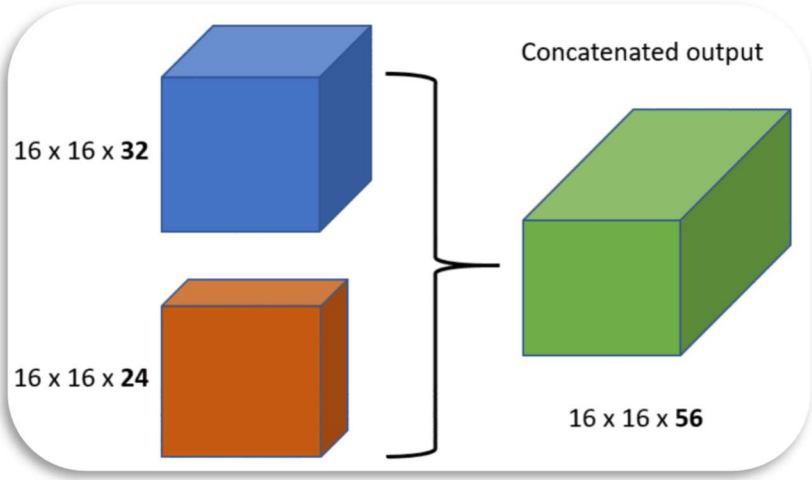
DenseNet (Densely Connected Convolutional Network) is a deep learning architecture introduced to address inefficiencies in feature propagation and parameter redundancy in convolutional neural networks. Unlike traditional architectures, DenseNet connects each layer to every other layer in a feed-forward fashion.

This innovative approach ensures that features learned in earlier layers are reused across all subsequent layers, promoting better gradient flow during training and reducing the risk of vanishing gradients.

By using fewer parameters compared to other networks of similar depth, DenseNet achieves remarkable efficiency in computation and memory while delivering state-of-the-art performance on tasks like image classification, object detection, and segmentation. Its compact and efficient design makes it a cornerstone for modern computer vision applications.

- Instead of element wise addition we can use concatenation that would result in DenseNet a variation of ResNet





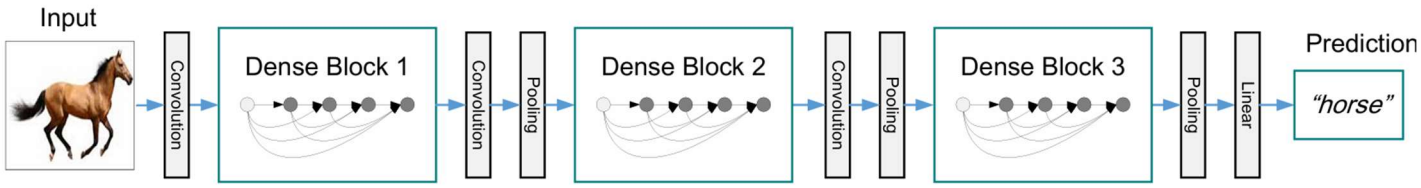


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and chang

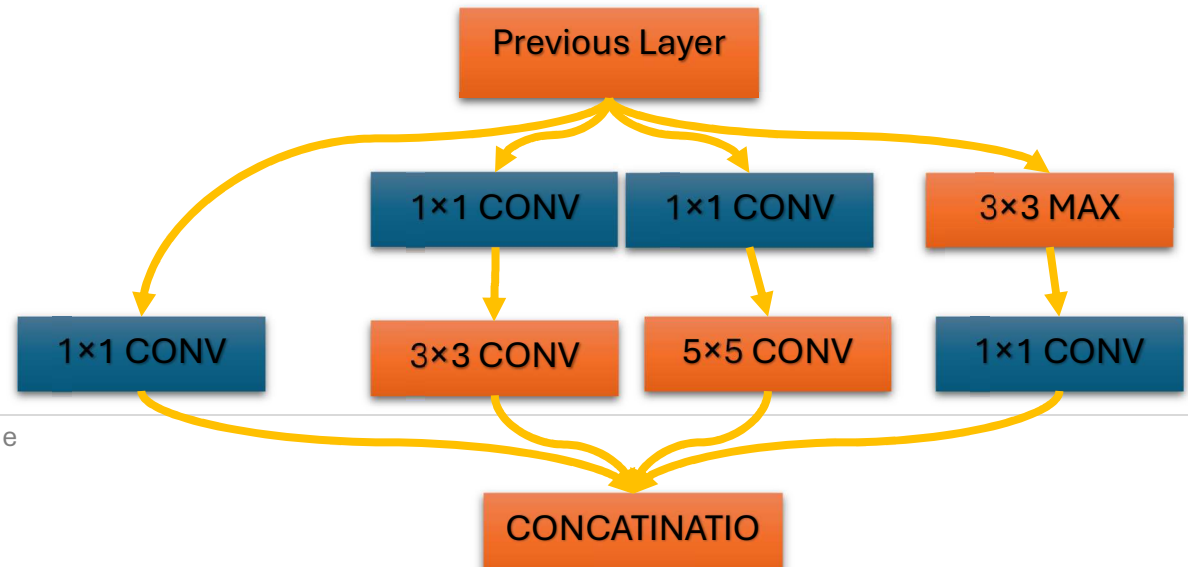
Inception

Inception networks, introduced by Google in the Inception v1 (or GoogLeNet) model, are a class of deep convolutional neural networks designed to improve the efficiency and accuracy of image recognition tasks. They achieve this by employing a novel "Inception module," which processes input data at multiple scales simultaneously.

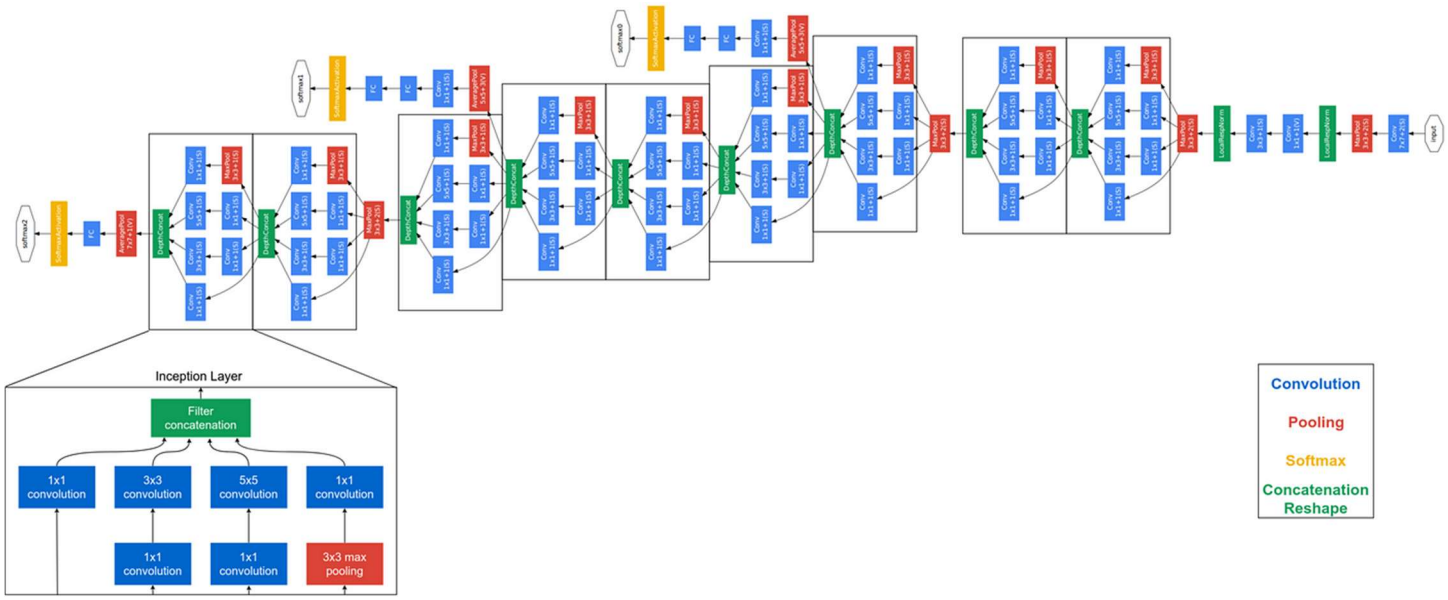
This module combines filters of different sizes (e.g., 1x1, 3x3, and 5x5) and a pooling operation within the same layer, capturing fine-grained details and broader contextual information without drastically increasing computational cost.

The Inception architecture is notable for its depth, width, and careful balance between computational efficiency and model performance, making it highly effective in tasks like image classification, object detection, and other computer vision applications. Over time, the Inception network has evolved into more advanced versions, such as Inception v2, v3, and v4, incorporating optimizations like batch normalization and factorized convolutions to enhance training stability and accuracy.

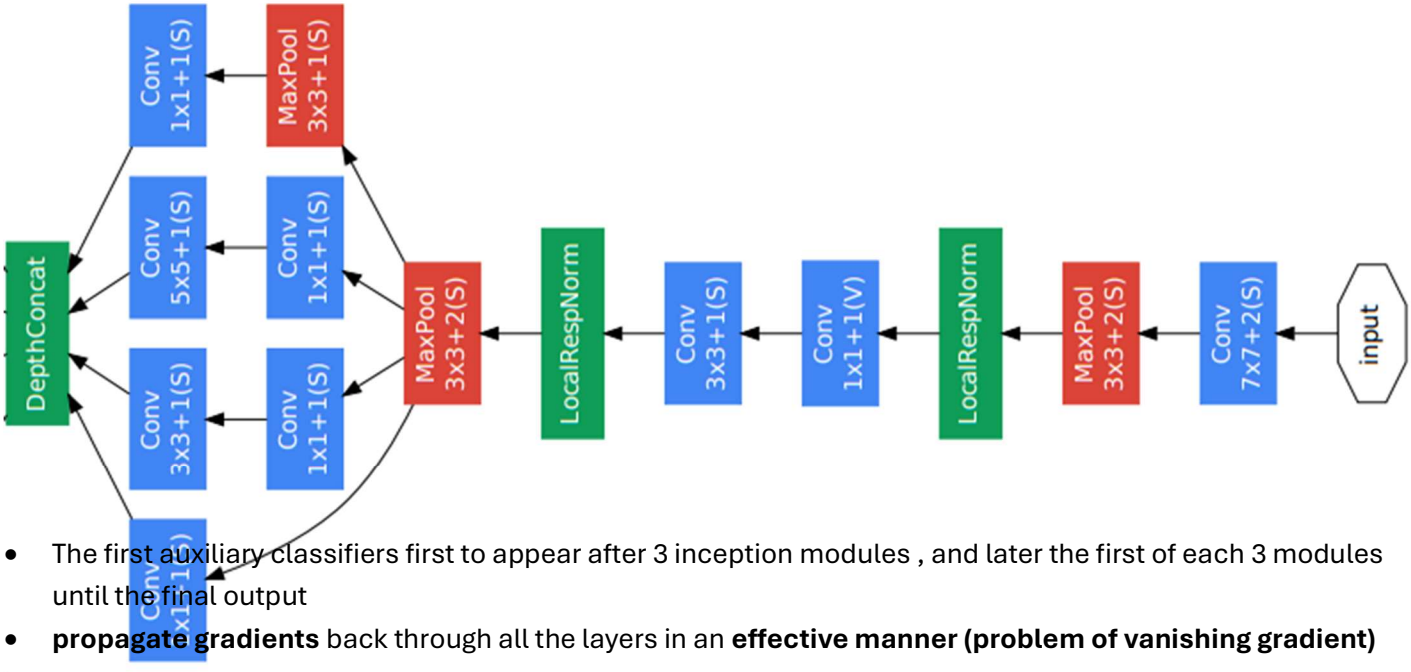
Inception module



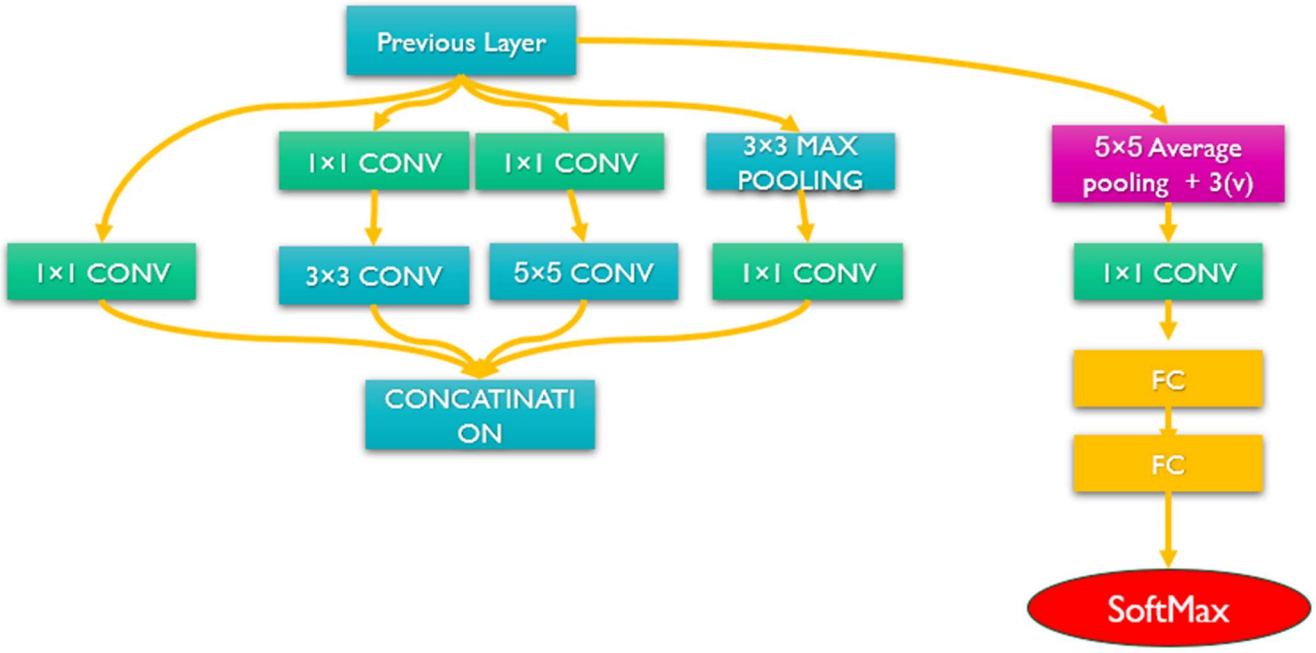
- **Intuition: visual information should be *processed at various scales* and *then aggregated* so that the next stage can abstract features from the different scales simultaneously.**
- **GoogleNet: Trained using DistBelif which is a framework for training DNN avoid using GPUs and use parallel computing with clusters of commodity machines**



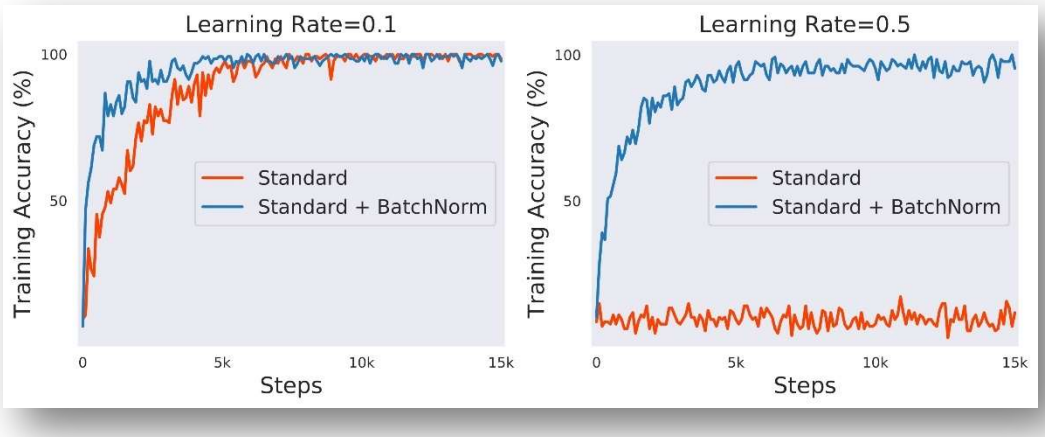
- start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion.



- The first auxiliary classifiers first to appear after 3 inception modules , and later the first of each 3 modules until the final output
- **propagate gradients** back through all the layers in an **effective manner (problem of vanishing gradient)**

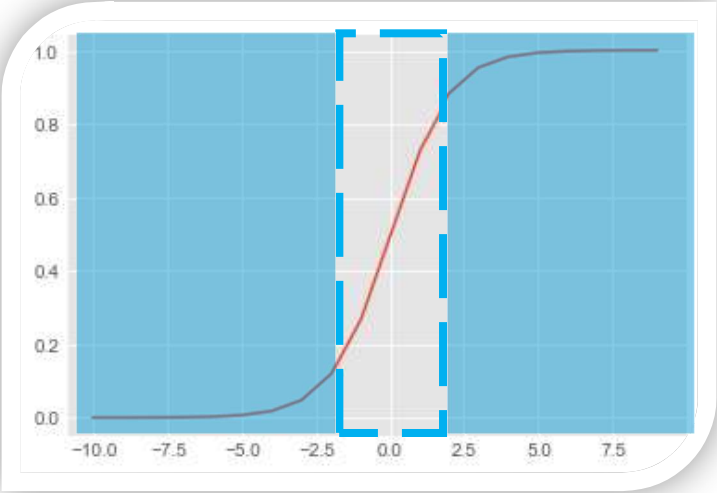


- The distribution of **each input's layer changes during training**, as the parameters of the previous layers change.
- This **slows down the training** by requiring lower learning rates and careful parameter initialization
- This is called **Internal Covariate Shift**
- We need to **fix the distribution** of the layer inputs x as the training progress to reduce the **Covariate shift**
- Using Batch Normalization
- Batch Normalization would allow us to use higher learning rates which mean faster training



Batch Normalization

- Input : $Values\ of\ x\ over\ a\ mini-batch: B = \{x_1 \dots m\}$;
- Step1 (mean) : $\mu_B = \frac{1}{m}\sum x_i$
- Step2 (Variance) : $\sigma_B^2 = \frac{1}{m}\sum(x_i - \mu_B)^2$
- Step3 (Normalize) : $\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
- ϵ Constant to prevent numbers from becoming to small causing vanishing gradient problem
- Step4 (scale & shift) : $y_i = \boldsymbol{\gamma}\widehat{x}_i + \boldsymbol{\beta}$
- Output : $\{y_i = BN(x_i)\}$
- **Gamma(scale) and Beta(shift) would be a learnable parameter that the gradient propagation can decide how to change to reduce the loss**
- normalizing each input of a layer ***may change what the layer can represent***
- Normalize the input of sigmoid would constrain them to the linear regime of the logistic
- We need to make sure that the transformation inserted in the network can represent the **identity transform, that is why we use Gamma and Beta.**



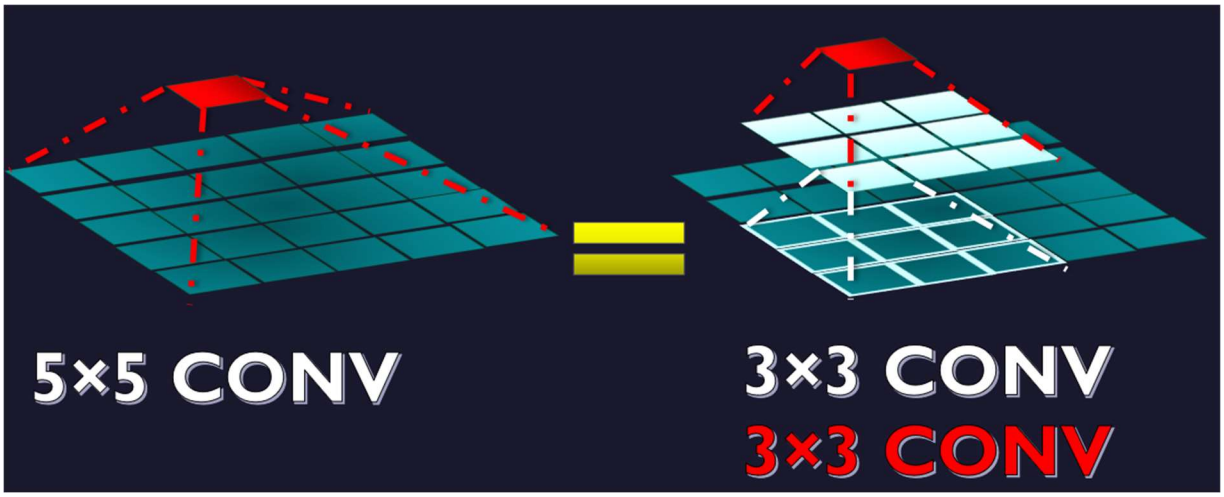
Convolution Factorization

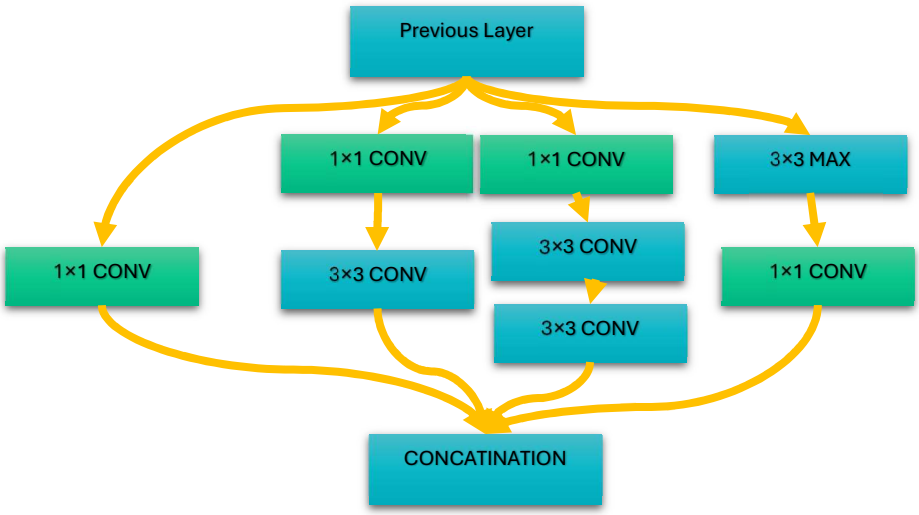
Convolution factorization is a technique used to reduce the computational complexity of convolutional operations in deep neural networks. It involves decomposing a standard convolutional filter into smaller, more efficient operations.

One common method is to factorize a large kernel (e.g., a 3x3 or 5x5 filter) into a series of smaller kernels, such as a combination of 1x3 and 3x1 convolutions. This reduces the number of parameters and computations required, leading to faster processing and less memory usage, without significantly compromising the model's performance.

Convolution factorization is widely used in architectures like Xception and MobileNet to optimize efficiency, especially in resource-constrained environments.

- 5×5 conv is (5×5)/(3×3) = 2.78 times more computationally expensive than a 3×3 convolution
- We want to replace the 5×5 CONV by a multi-layer network with less parameters with same input size and output depth.





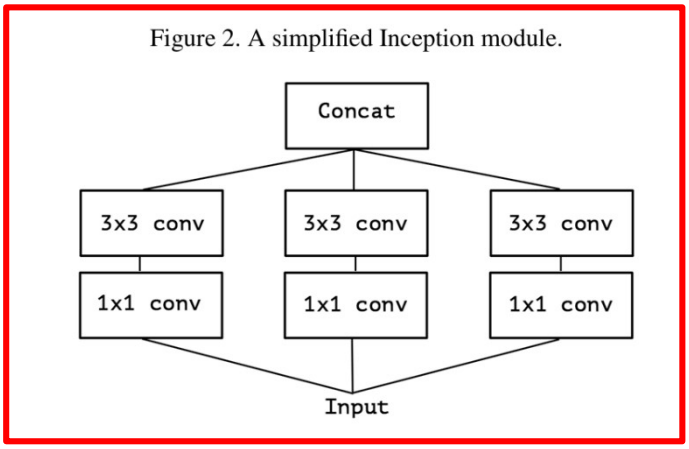
Xception

Xception, short for "Extreme Inception," is a deep convolutional neural network architecture introduced by François Chollet in 2017. It builds upon the Inception model but introduces a key innovation by replacing the standard convolutional layers with *depthwise separable convolutions*.

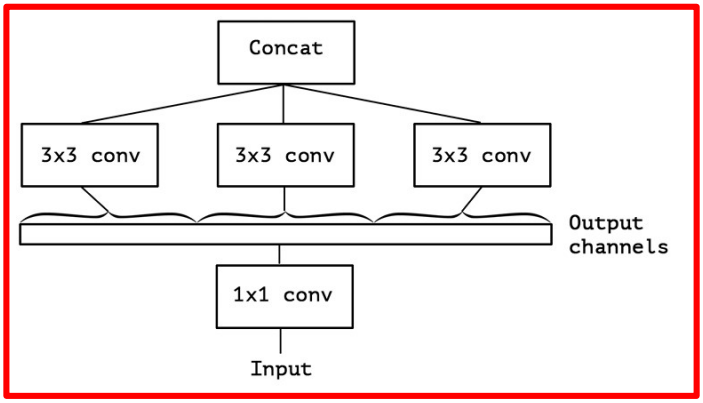
This approach splits the convolution operation into two parts: a depthwise convolution, which applies a single filter to each input channel, and a pointwise convolution, which combines the outputs across channels. This separation reduces the number of parameters and computational complexity while maintaining high performance. Xception achieves state-of-the-art results on various image recognition benchmarks, and its architecture is particularly well-suited for tasks requiring high efficiency and accuracy.

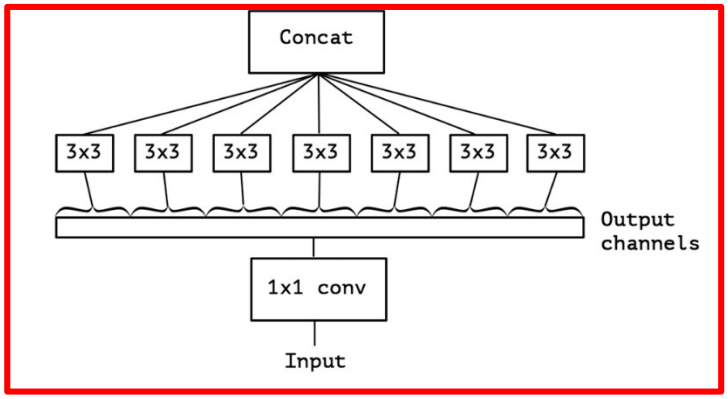
Its design has influenced many subsequent models, particularly in the field of transfer learning, where its lightweight yet powerful structure provides an effective balance between performance and computational cost.

- **Simplified inception module**



- **Extreme version of the inception module, with one spatial convolution per output channel of the 1x1 convolution.**





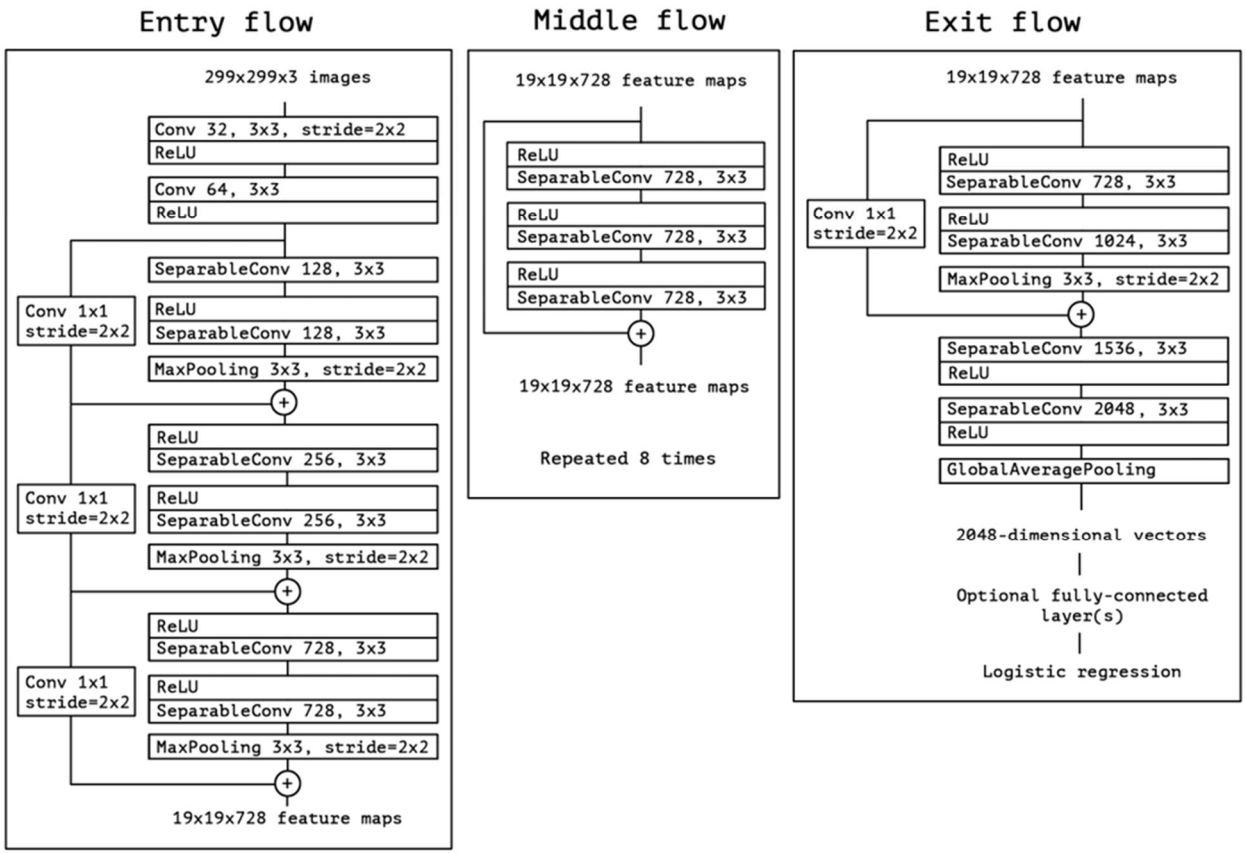
- **An "extreme" version of an Inception module, based on this stronger hypothesis, would first use a 1x1 convolution to map cross-channel correlations, and would then separately map the spatial correlations of every output channel.**
- **A depthwise separable convolution**
  - **Depthwise conv : spatial convolution performed independently over each channel of an input**
  - **Followed by pointwise convolution (1×1) , projecting the channels output by the depthwise convolution onto a new channel space.**

Architecture

Xception is a deep convolutional neural network consisting of several components, with a particular focus on depthwise separable convolutions. The architecture can be broken down into three main sections:

- **Entry Flow**
- **Middle Flow**
- **Exit Flow**

Figure 5. The Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization [7] (not included in the diagram). All SeparableConvolution layers use a depth multiplier of 1 (no depth expansion).



Entry flow

This is the initial stage of the network, responsible for extracting low-level features from the input image.

- **Initial Convolution Layer: The model starts with a standard 3x3 convolution with a stride of 2. This is followed by batch normalization and a ReLU activation function, which helps stabilize training and speed up convergence.**
- **Depthwise Separable Convolution Layers: The first major innovation in Xception occurs here. The entry flow consists of several layers of *depthwise separable convolutions*. These layers replace traditional convolutions, which use a single filter to convolve across all channels, with two smaller operations:**
    - **Depthwise Convolution: Each input channel is convolved separately with its own filter.**
    - **Pointwise Convolution (1x1 convolution): The output from the depthwise convolution is then passed through a 1x1 convolution to combine information across channels.**

This factorization reduces the number of parameters and computational cost while still allowing the model to capture rich feature representations.

- **Activation and Pooling: After each depthwise separable convolution, the network applies batch normalization followed by a ReLU activation. Pooling layers are applied intermittently to downsample the feature maps.**

Middle flow

The middle flow is where the model captures more complex patterns and features from the input data. This stage consists of several residual blocks based on depthwise separable convolutions.

- **Residual Blocks: Each block in the middle flow uses depthwise separable convolutions for both the convolutional operations and the 1x1 pointwise convolutions. These blocks are designed to learn increasingly abstract features at higher levels of complexity.**
    - **Depthwise Separable Convolutions: Similar to the entry flow, each block in the middle flow uses depthwise separable convolutions (depthwise + pointwise convolutions).**
    - **Residual Connection: A skip connection (also known as a residual connection) is applied, where the input to the block is added directly to the output of the block. This helps prevent the vanishing gradient problem and makes the network easier to train.**

The number of blocks in the middle flow typically ranges from 8 to 12, depending on the version of the model. These blocks increase the representational power of the network while maintaining efficiency.

Exit flow

The exit flow consists of layers that transition the feature maps into a classification output or any other final task, like regression.

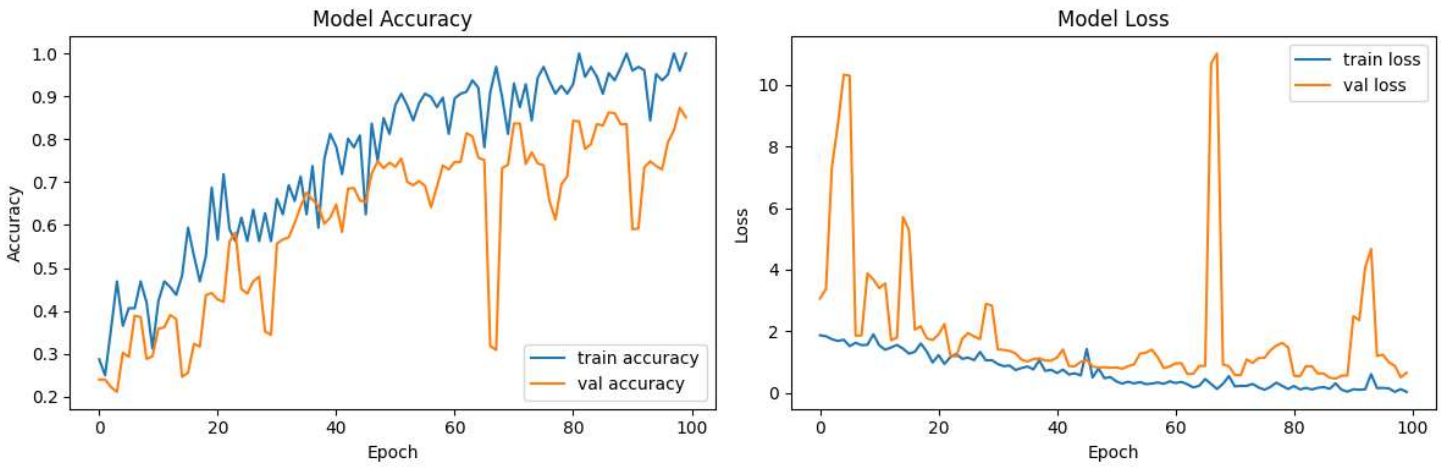
- **Depthwise Separable Convolutions: The exit flow continues with depthwise separable convolutions, but the network applies fewer of these layers, which helps reduce the overall computational load.**
- **Global Average Pooling (GAP): Instead of using fully connected layers, which would increase the number of parameters, Xception uses global average pooling. GAP reduces each feature map to a single value by averaging over the spatial dimensions (height and width). This operation helps reduce the number of parameters and the risk of overfitting, and is particularly useful for large datasets.**
- **Fully Connected Layer: Finally, the output from GAP is passed through a fully connected (dense) layer, which is typically followed by a softmax activation function for classification tasks.**

Performance on our task

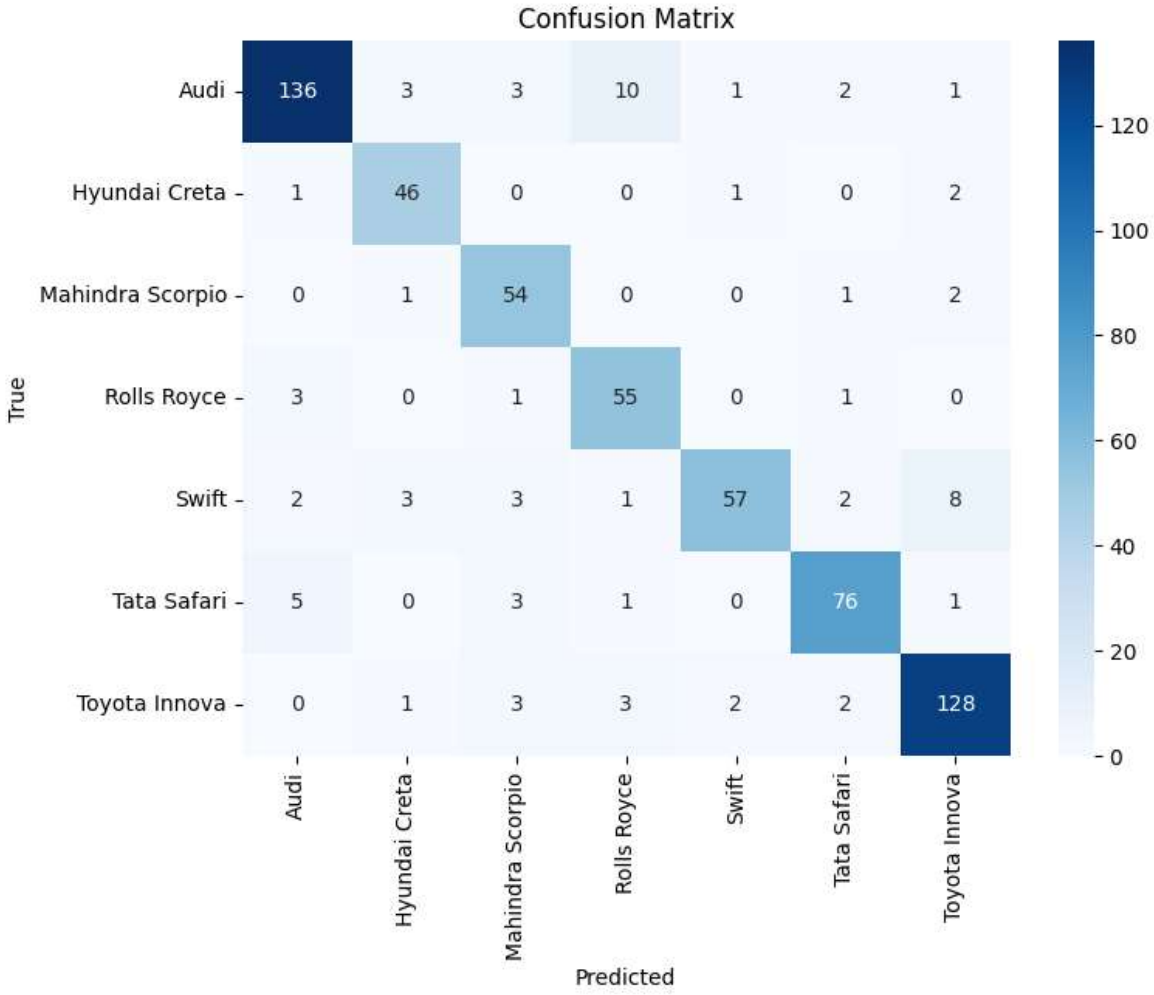
ResNet-34 performance on the task

	precision	recall	f1-score	support
Audi	0.93	0.87	0.90	156
Hyundai Creta	0.85	0.92	0.88	50
Mahindra Scorpio	0.81	0.93	0.86	58
Rolls Royce	0.79	0.92	0.85	60
Swift	0.93	0.75	0.83	76
Tata Safari	0.90	0.88	0.89	86
Toyota Innova	0.90	0.92	0.91	139
accuracy			0.88	625
macro avg	0.87	0.88	0.88	625
weighted avg	0.89	0.88	0.88	625

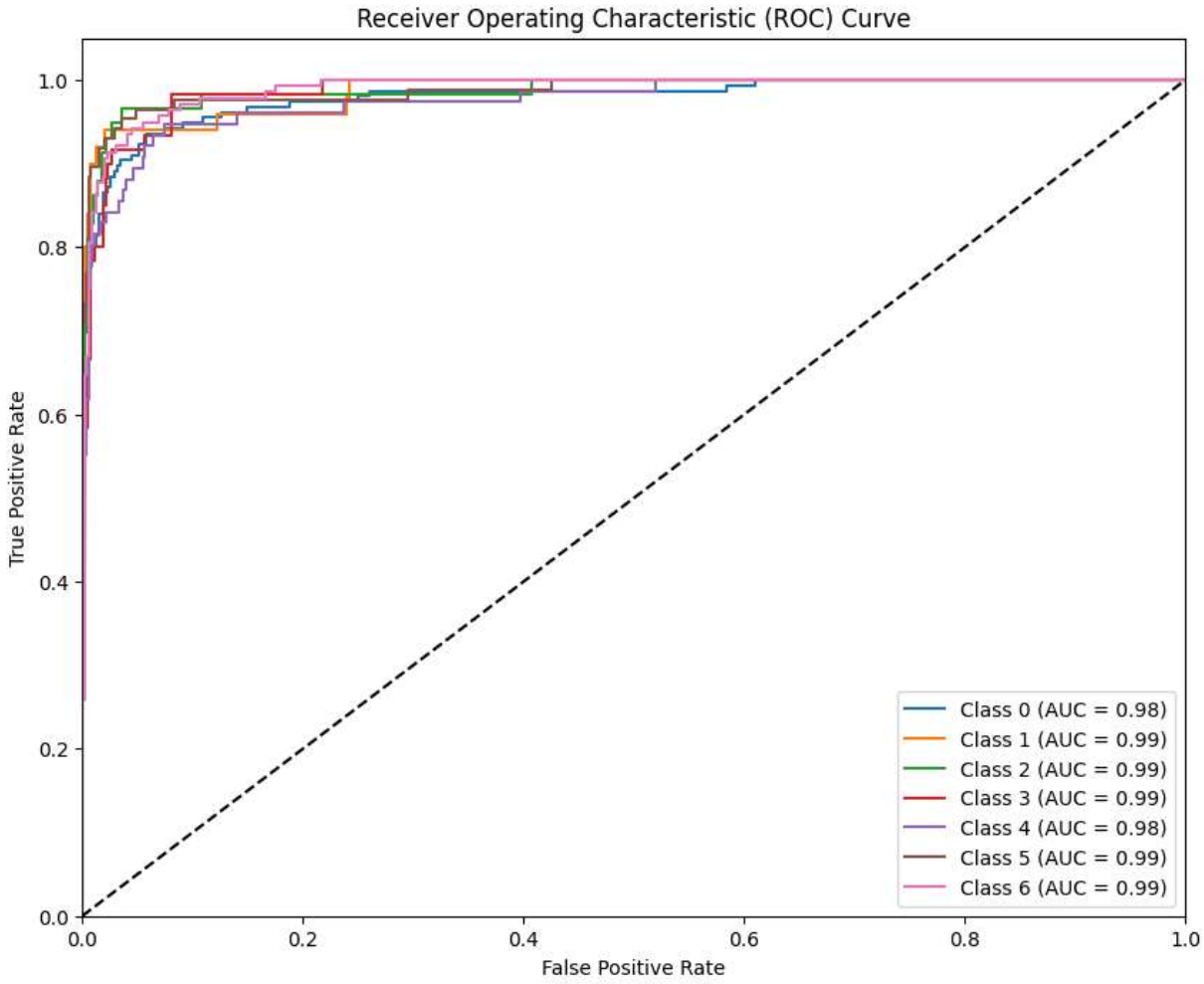
- **Classification report of test evaluation, that includes Recall, precision and f1-score of each class, and the average of all classes, and also we calculated the accuracy of all classes.**



- **The graph shows the accuracy of the model and the loss for both training set and validation set.**
- **Train is higher than the validation accuracy but with acceptable margin that show there is no overfitting.**

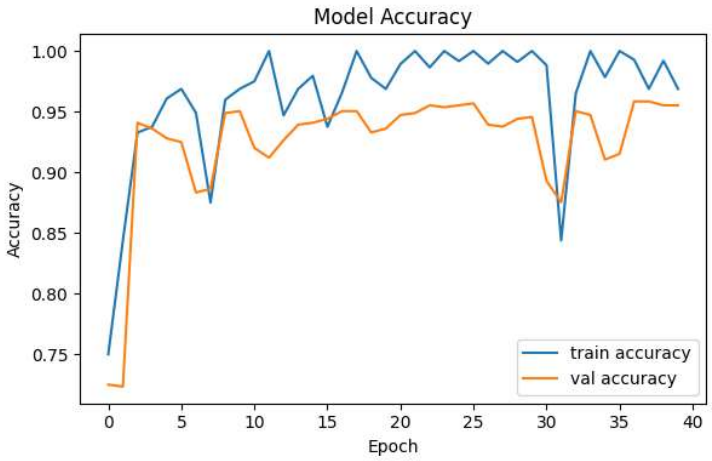


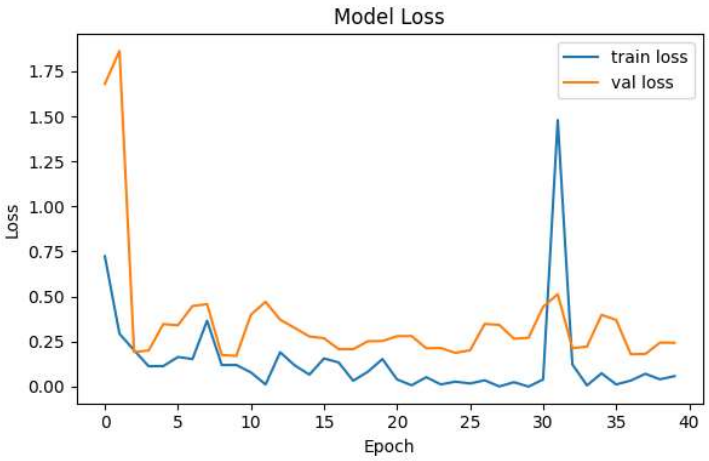
- **Confusion matrix of test performance.**

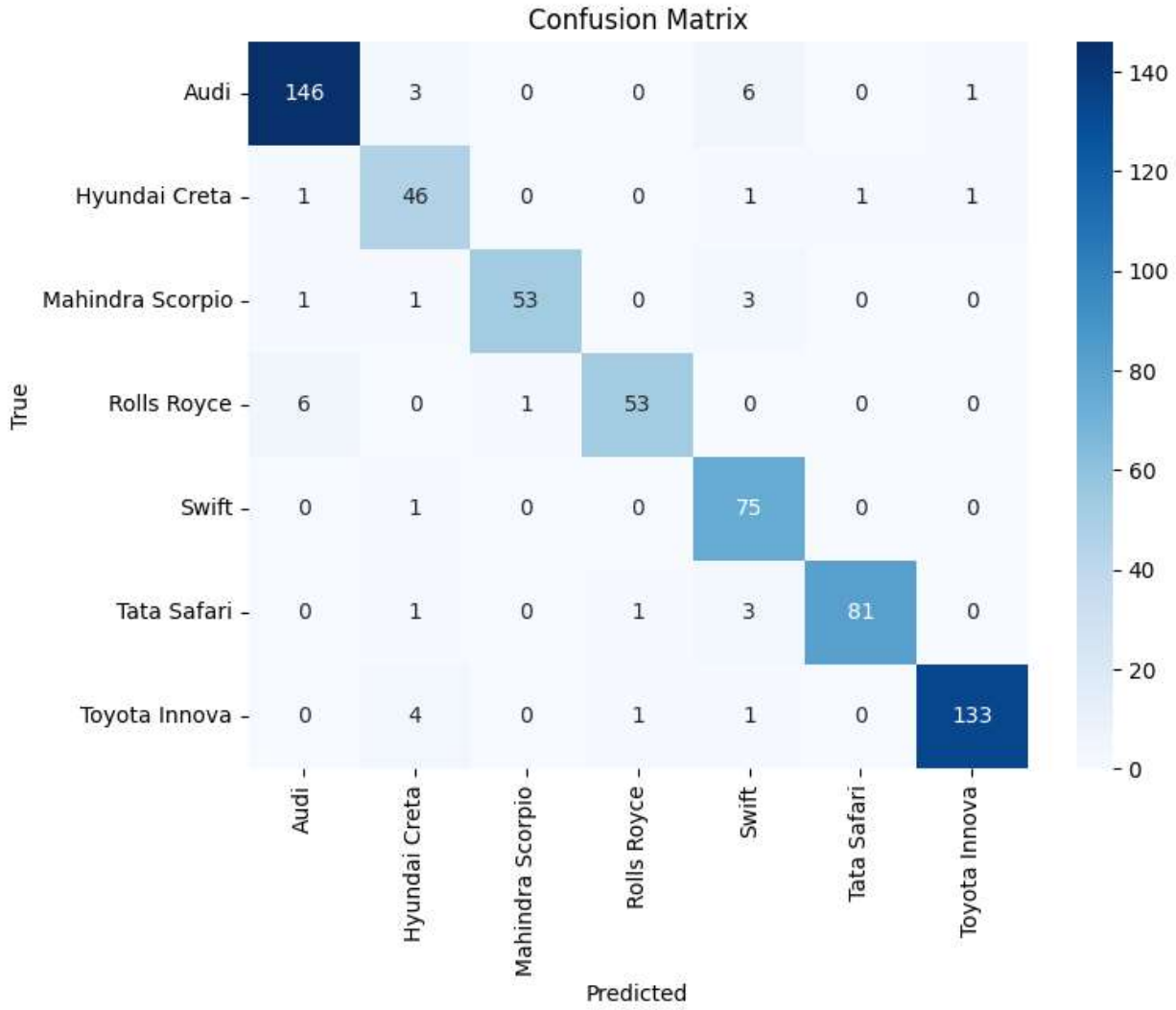


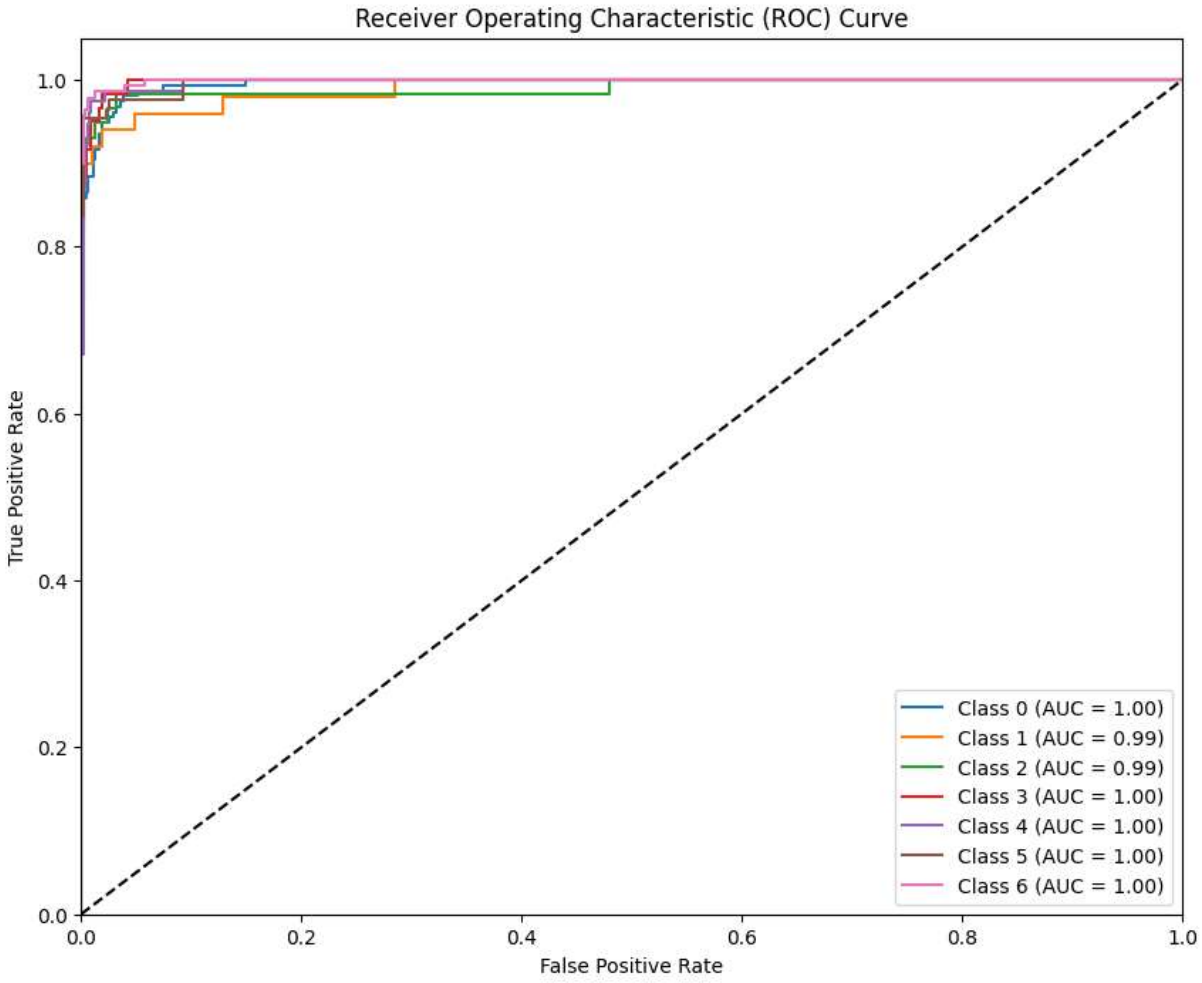
DenseNet performance on the task

	precision	recall	f1-score	support
Audi	0.95	0.94	0.94	156
Hyundai Creta	0.82	0.92	0.87	50
Mahindra Scorpio	0.98	0.91	0.95	58
Rolls Royce	0.96	0.88	0.92	60
Swift	0.84	0.99	0.91	76
Tata Safari	0.99	0.94	0.96	86
Toyota Innova	0.99	0.96	0.97	139
accuracy			0.94	625
macro avg	0.93	0.93	0.93	625
weighted avg	0.94	0.94	0.94	625


Model Accuracy
Accuracy
Epoch
train accuracy
val accuracy

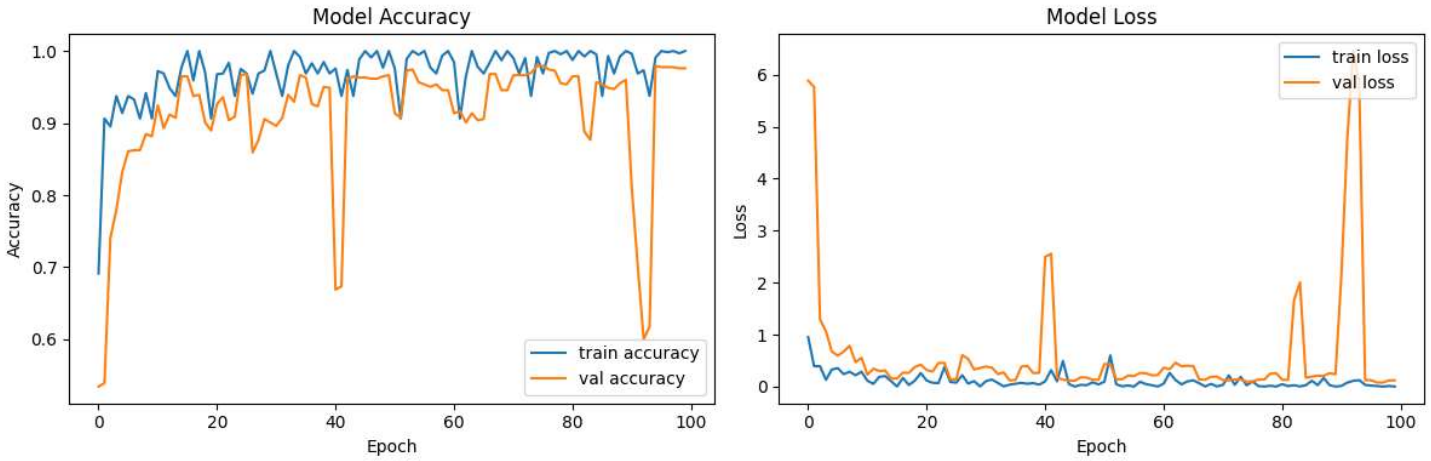

Model Loss
Loss
Epoch
train loss
val loss

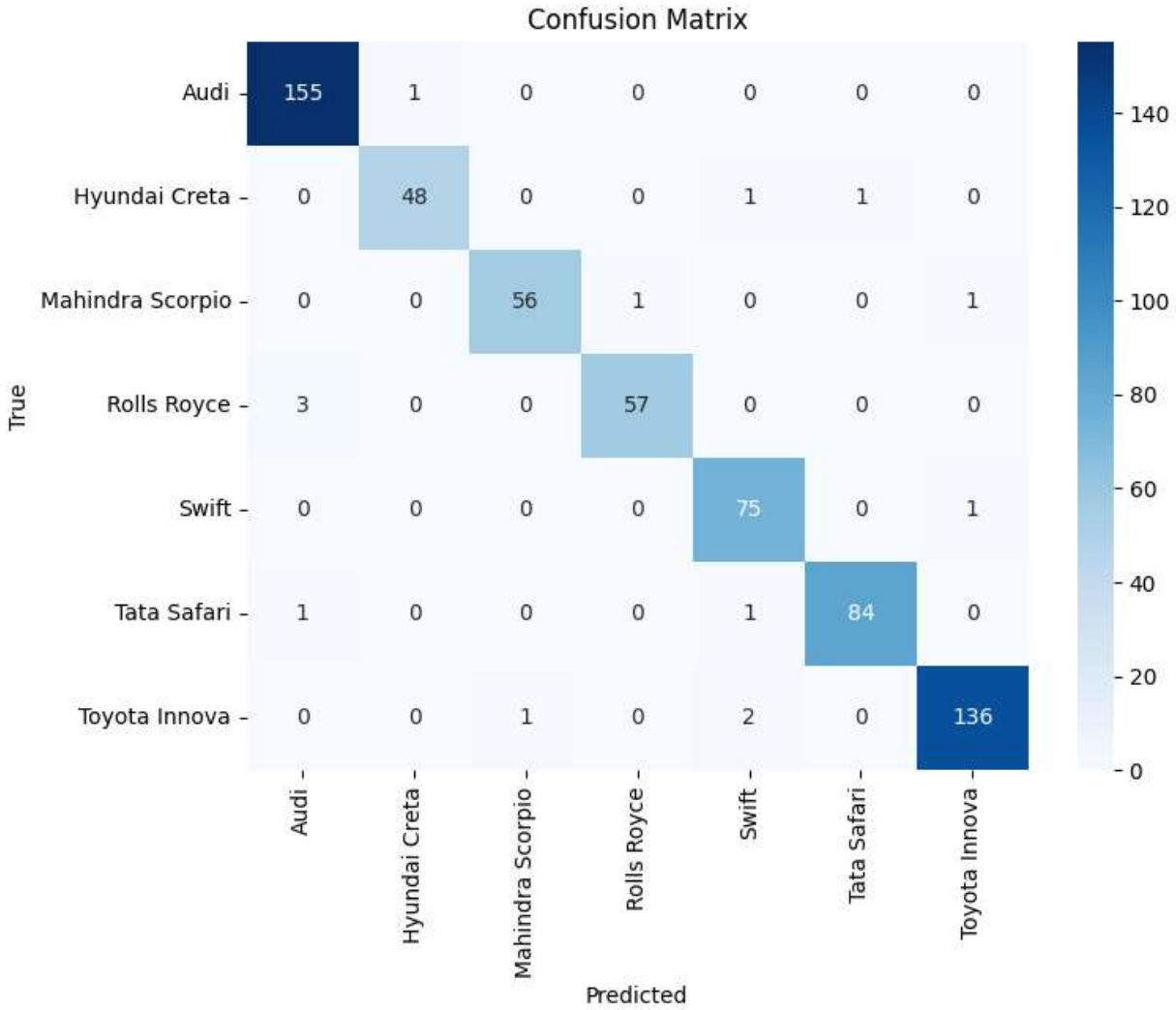

Confusion Matrix
True
Predicted
Audi
Hyundai Creta
Mahindra Scorpio
Rolls Royce
Swift
Tata Safari
Toyota Innova

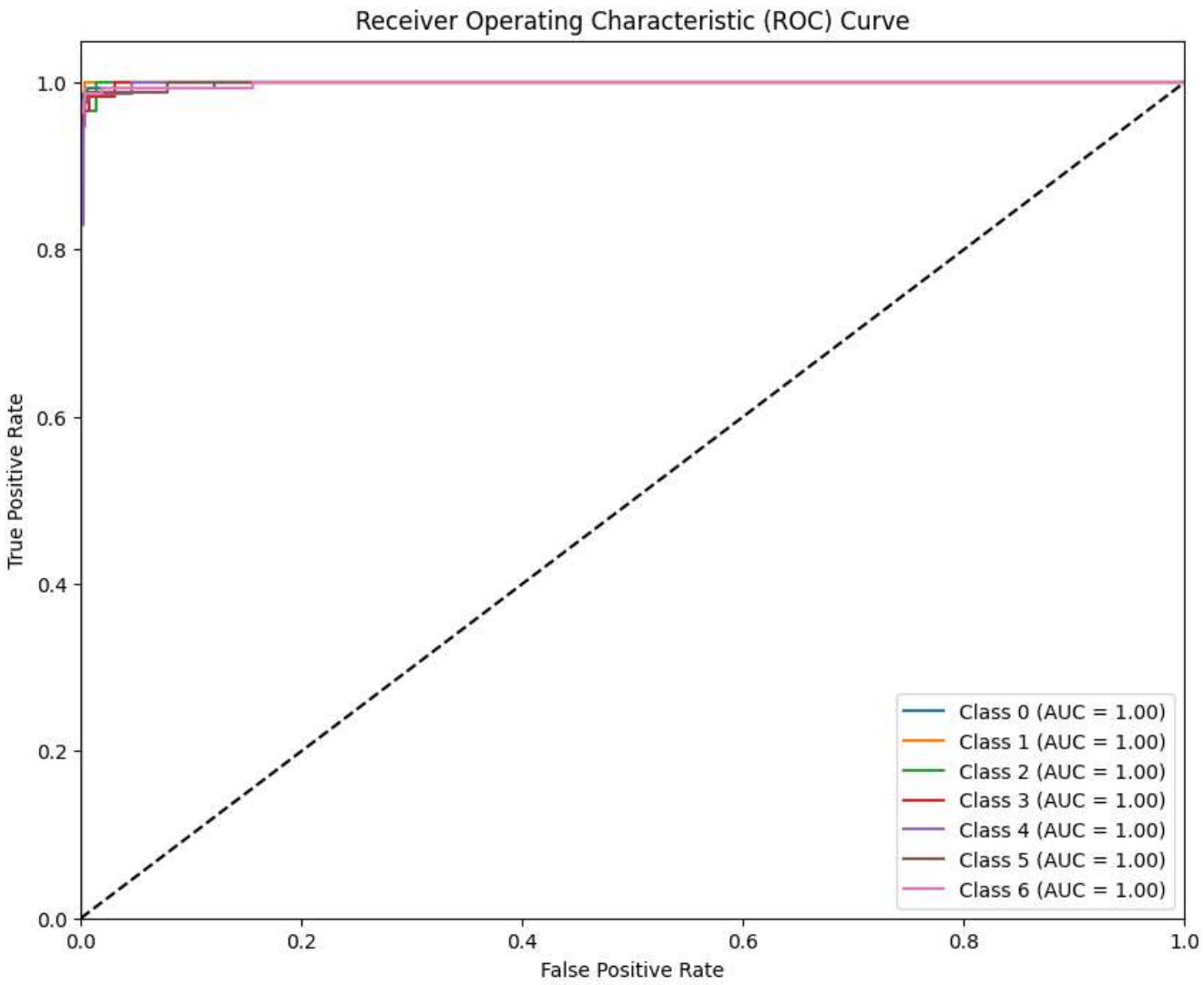


Xception performance on the task

	precision	recall	f1-score	support
Audi	0.97	0.99	0.98	156
Hyundai Creta	0.98	0.96	0.97	50
Mahindra Scorpio	0.98	0.97	0.97	58
Rolls Royce	0.98	0.95	0.97	60
Swift	0.95	0.99	0.97	76
Tata Safari	0.99	0.98	0.98	86
Toyota Innova	0.99	0.98	0.98	139
accuracy			0.98	625
macro avg	0.98	0.97	0.98	625
weighted avg	0.98	0.98	0.98	625


Model Accuracy
Accuracy
Epoch
train accuracy
val accuracy
Model Loss
Loss
Epoch
train loss
val loss


Confusion Matrix
True
Predicted
Audi
Hyundai Creta
Mahindra Scorpio
Rolls Royce
Swift
Tata Safari
Toyota Innova


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 0 (AUC = 1.00)
Class 1 (AUC = 1.00)
Class 2 (AUC = 1.00)
Class 3 (AUC = 1.00)
Class 4 (AUC = 1.00)
Class 5 (AUC = 1.00)
Class 6 (AUC = 1.00)

Summary of Performance

	Average Recall	Average precision	f1-score
ResNet from scratch	0.88	0.89	0.88
DenseNet	0.94	0.94	0.93
Xception	0.98	0.98	0.98

References

Paper “Deep Residual Learning for Image Recognition” (2015)

Paper “Going Deeper with Convolutions” (2014)

Paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" (2015)

Paper “Rethinking the Inception Architecture for Computer Vision” (2015)

Paper “Xception: Deep Learning with Depthwise Separable Convolutions” (2017)

Paper "Densely Connected Convolutional Networks" (2018)

Paper “Visualizing the Loss Landscape of Neural Nets” (2018)