Xception: Deep Learning with Depthwise Separable Convolutions

Francois Chollet, Google, Inc.

Jordan Kevin Buwa Mbouobda

African Master's in Machine Intelligence

August 28, 2025

Overview

Introduction

Introduction and Motivation

Convolutional layers in CNNs typically learn spatial and cross-channel correlations **jointly**, leading to:

- High computational cost
- Large number of parameters

Inception modules improved efficiency by:

- Using 1×1 convolutions for channel mixing
- Followed by 3×3 or 5×5 convolutions for spatial filtering

Xception takes this idea to the extreme:

- **Depthwise convolution:** spatial filtering per channel
- **Pointwise convolution:** mixes channel-wise information
- *Fully decouples spatial and cross-channel learning*

Result: A simpler, more efficient architecture that outperforms Inception V3 with similar parameter count.

AIMS African Institute for Mathematical Sciences SENEGAL

Inception model

Inception vs. Depthwise Separable Conv

- Inception: Partial separation of concerns.
- Depthwise Separable Convolution:
  1. Depthwise convolution: spatial filtering, per channel.
  2. Pointwise (1x1) convolution: cross-channel mixing.
- Xception = Extreme version of Inception.

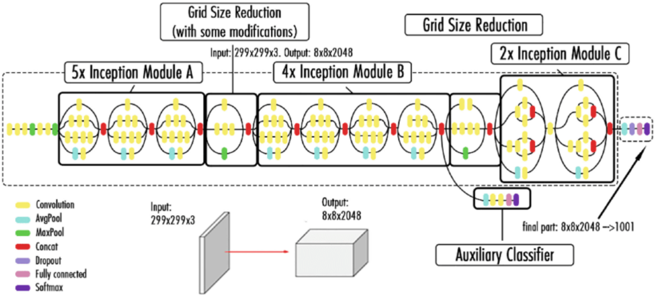


Figure: Inception V3 (source)

Architecture overview

Entry Flow

- Processes raw input and reduces spatial dimensions while increasing depth.
- Two standard `Conv2D` layers:
  - Followed by `ReLU` and BatchNorm
- Three modules, each containing:
  - Two or three `SeparableConv2D` layers
  - Residual shortcut (with 1×1 Conv if needed)
  - `MaxPooling2D` for downsampling

Middle Flow

- Core feature extractor, repeated 8 times
- Each module:
  - Three `SeparableConv2D` layers
  - Each followed by `ReLU` and BatchNorm
  - Ends with a residual connection
- **No change** in feature map size or depth
- Acts as a deep tower for high-level feature extraction

Exit Flow

- Final transformation before classification
- Two `SeparableConv2D` layers + BatchNorm + `ReLU`
- Residual connection (may include 1×1 Conv)
- Final `SeparableConv2D`, followed by:
  - Global Average Pooling
  - Fully-connected or logistic regression layer
- **For ImageNet:** Softmax over 1000 classes

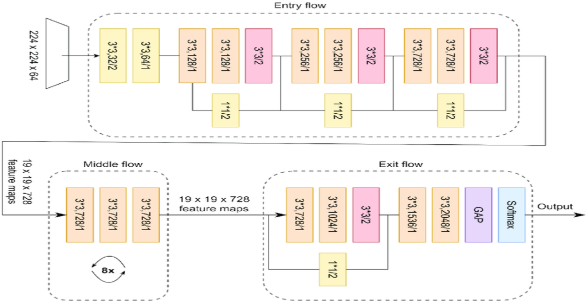


Figure: Xception Architecture (source)

Separable convolution

Depthwise Separable Convolution

- Depthwise: One filter per input channel.
- Pointwise: 1x1 convolution combines channels.
- Much fewer parameters than regular Conv2D.

  Standard Conv: $D_k \cdot D_k \cdot M \cdot N$ Separable Conv: $D_k \cdot D_k \cdot M + M \cdot N$

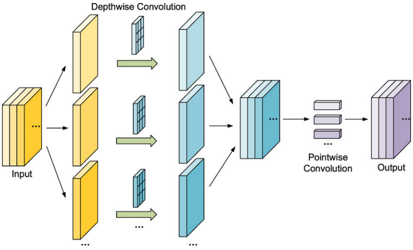


Figure: Depthwise separable convolution (source)

Non-linearity Effects

- ReLU/ELU between depthwise and pointwise layers were tested.
- Result: Omitting the non-linearity improves performance.
- Explanation: Non-linearity may harm shallow (1-channel) intermediate spaces.

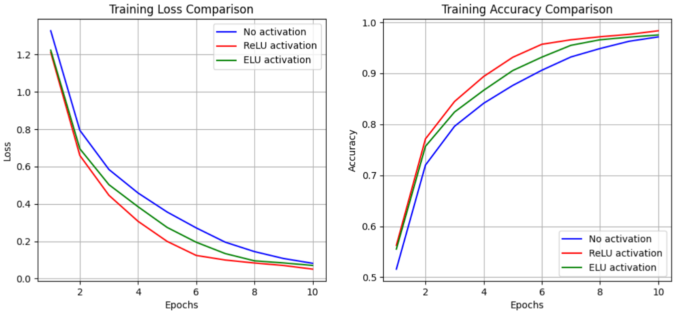


Figure: Effect of non-linearity

Training and evaluation

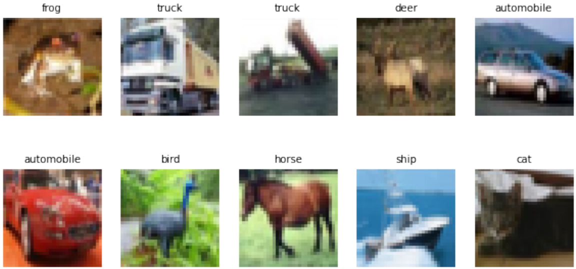
Dataset

Datasets: CIFAR-10 Overview

- **Dataset:** 60,000 color images of size $32 \times 32 \times 3$ (RGB).
- **Classes:** 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
- **Split:** 45,000 training images, 5,000 validation images and 10,000 test images.
- **Use:** Standard benchmark for image classification, data augmentation, and deep learning methods.

Training setup:

- Adam, GradScaler for dynamic gradient scaling



AIMS African Institute for Mathematical Sciences SENEGAL

CIFAR-10 Preprocessing Overview

- **Training set:**
  - Resize to 320×320, then **random crop** 299×299
  - **Random horizontal flip** (p=0.5) for augmentation
  - Convert to tensor, scale to [0, 1]
  - Normalize using ImageNet stats:
    $\mu = (0.485, 0.456, 0.406), \quad \sigma = (0.229, 0.224, 0.225)$
- **Validation / Test sets:**
  - Resize to 299×299 and **center crop**
  - Tensor conversion + same normalization
- **Split:** 45k train, 5k val, 10k test
- **Batch size:** 128 (DataLoader with shuffle for train, fixed order for val/test)

Comparison with Inception

Performance Comparison

- **Model Size:** Inception V3 is slightly larger with 24.3M parameters compared to Xception's 20.8M, which is about 17% more.
- **Accuracy:** Inception V3 achieves an accuracy of 85.44%, outperforming Xception's 84.87%. This is a gain of $+0.57\%$ with a 17% increase in parameters.
- **Efficiency (Accuracy per million parameters):**
  - Inception V3 $\rightarrow \frac{85.44}{24.3} \approx 3.52\%$ per million parameters
  - Xception $\rightarrow \frac{84.87}{20.8} \approx 4.08\%$ per million parameters

  Xception is more parameter-efficient, but Inception V3 provides higher absolute performance.

Model/performances	**Parameters**	**Accuracy**	**Running time (s)**
Inception V3	$24,371,444$	85.44%	1902.98
Xception	$20,825,402$	84.87%	3116.90

Table: Performance Comparison

The difference in running time simply comes from the fact that Inception model was loaded using pytorch API.

Training curves

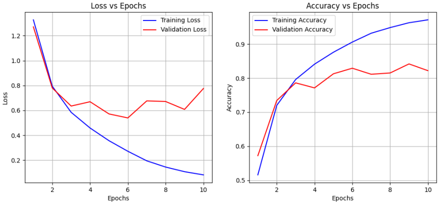


Figure: Training Xception

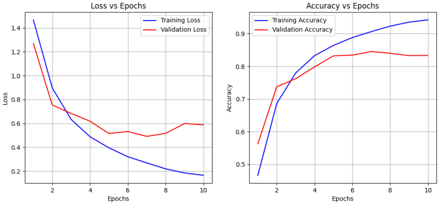


Figure: Training Inception V3

Training curves

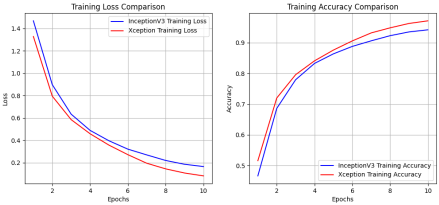


Figure: Inception V3 vs Xception

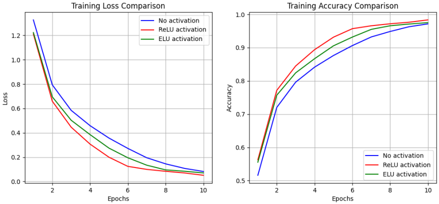


Figure: Xception with different activations

Conclusion

Conclusion

- Xception is a simple, scalable CNN architecture.
- Replaces Inception modules with depthwise separable convolutions.
- Outperforms Inception V3 on both ImageNet and JFT at equal parameter cost.
- Easy to implement and tune with modern frameworks.

The implementation can be found here.
The full article can be found here.

Djeredjef!