MobileNets

- For mobile and embedded devices
- Tradeoff between latency and accuracy
- Use depth-wise separable convolutions. Separate convolutions into:
  - Depth-wise (channel) convolutions, followed by
  - Point-wise 1x1 convolutions
- Further simplification parameters:
  - Width multiplier (fewer channels)
  - Resolution multiplier (smaller resolution)

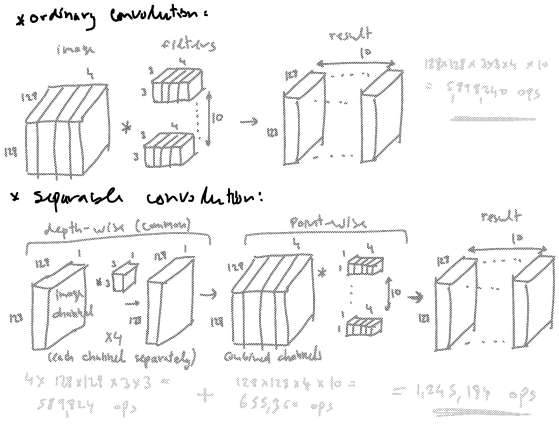
The advantages of depthwise separable convolutions are:

1. Reduced computation: By applying a single filter to each input channel, the depthwise convolution reduces the number of computations required in comparison to traditional convolutional layers.
2. Reduced parameters: The pointwise convolution combines the intermediate feature maps with a 1x1 filter, which greatly reduces the number of parameters in the model.

MobileNet also uses other techniques to further reduce the model size and improve performance, such as:

1. Linear bottlenecks: To reduce the number of input channels to the depthwise convolution, MobileNet applies a linear bottleneck layer that compresses the input feature maps. This helps to reduce the number of parameters in the model.
2. Width multiplier: MobileNet uses a width multiplier to reduce the number of channels in each layer. This allows the model to be scaled up or down depending on the computational resources available.

MobileNets



Input size: 128 x 128 x 4, Filter size: 3 x 3 x 4, Output size: 128 x 128 (same as input width and height)
For each output position: Number of multiplications: 3 x 3 x 4 = 36
Total number of output positions: 128 x 128 = 16,384
Therefore, the total number of multiplications is:Total Multiplications
=16,38436=589,824 ----Total Multiplications=16,38436=589,824

The first part until 128x128x3x3x4 is same number of operations even though we use the same 3x3 values for all 4 slices.....now at this point we just do addition of the values across the slices in traditional method. but here we do 1x1 convolution ....which results in the multiplication operations 128128x4x1 for 1 filter....when we say 10 filters, we donot repeat the 3x3 filter 10 times , we only do this 1x1 conv 10 times.....therefore the first part remains constant for whatever the number of filters --(128x128x3x3x4)... the part that changes is 128x128x4x1.....10 times mean.....128x128x4x1x10.....we then add the number of operations as (128x128x3x3x4) + (128x128x4x1x10) = 1245184...

Depthwise step: Applies a 3x3 filter to each input channel independently.
Pointwise step: Uses 1x1 convolutions to combine channel information.

This is why the number of multiplication operations is smaller in separable conv than traditional conv

Potential Disadvantages:
Limited cross-channel learning: The 3x3 filters in the depthwise step don't directly learn cross-channel spatial patterns.
Reduced parameter flexibility: Fewer parameters to learn complex feature interactions.

Trade-offs:
Computational efficiency: Significantly fewer multiplications.
Parameter reduction: Can help prevent overfitting in some cases.

Mitigating Factors:
Increased depth: Networks using separable convolutions often have more layers, potentially compensating for per-layer limitations.
Empirical performance: Despite theoretical limitations, separable convolutions have shown good performance in practice, especially in mobile and embedded applications

Review image and then continue

MobileNets

* width multiplier:
  - Reduce # of input and output channels to each layer by a factor of α.

input channels: $M \rightarrow \alpha M$

output channels: $N \rightarrow \alpha N$

$\alpha \in [0,1]$ eg. 0.5

- New computational cost:

$$\underbrace{D_k \times D_k \times \alpha M \times D_F \times D_F}_{\text{reduction by factor } \alpha} + \underbrace{\alpha M \times \alpha N \times D_f \times D_f}_{\text{reduction by factor } \alpha^2}$$

MobileNets

* Resolution multiplier:
  - reduce input resolution by a factor $\beta \in [0,1]$

input resolution: $D_f \times D_f \rightarrow \beta D_f \times \beta D_f$

output resolution: $D_f \times D_f \rightarrow \beta D_f \times \beta D_f$

- New computational cost:

$$\underbrace{D_k \times D_k \times \alpha M \times \beta D_F \times \beta D_F + \alpha M \times \alpha N \times \beta D_f \times \beta D_f}_{\text{reduction by factor } \beta^2}$$

MobileNets

* Experimental evaluation

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

we see reduction in parameters and operations

# of operations

No.of parameters doesnt change because we are only touching the spatial resolution, so the filter remains the same

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

MobileNet = depth-wise separable conv.

accuracy

MobileNets

* Experimental evaluation (contd.)

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

skipped

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

MobileNets

```
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.layers import Input, GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model

# Define the input shape of the images
input_shape = (224, 224, 3)

# Load the pre-trained MobileNet model (without the top layers)
base_model = MobileNet(input_shape=input_shape, include_top=False,
weights='imagenet')

# Add a global average pooling layer to reduce the spatial dimensions of the
output
x = GlobalAveragePooling2D()(base_model.output)

# Add a dense layer with softmax activation for classification
x = Dense(1000, activation='softmax')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=x)

# Print the summary of the model
model.summary()
```

Summary of Keras backbone networks

Backbone (Keras)	Strengths	Common Applications
ResNet (ResNet50, ResNet101, ResNet152)	Strong for deep networks, residual connections for better gradient flow	Classification, object detection, segmentation (e.g., Mask R-CNN)
VGG (VGG16, VGG19)	Simple architecture, easy to use for transfer learning, deep features	Classification, transfer learning, feature extraction
InceptionV3	Multi-scale feature extraction, efficient factorized convolutions	Classification, object detection, transfer learning
Xception	Efficient, uses depthwise separable convolutions for better efficiency	Classification, segmentation, object detection
MobileNetV1/V2	Lightweight, highly efficient, designed for mobile devices	Mobile applications, real-time image processing, classification

Summary of Keras backbone networks

EfficientNet (B0-B7)	Compound scaling for efficient and accurate models, excellent performance-to-complexity ratio	Classification, object detection, segmentation
DenseNet (DenseNet121, DenseNet169, DenseNet201)	Feature reuse through dense connectivity, highly parameter-efficient	Classification, transfer learning, medical imaging
NASNet	Automatically discovered architecture, strong performance on various tasks	Classification, object detection, transfer learning
InceptionResNetV2	Combines residual connections with multi-scale Inception modules	Classification, object detection, segmentation
MobileNetV3	Lightweight and efficient, better performance with less complexity than MobileNetV2	Mobile applications, embedded systems, real-time classification
ResNeXt (ResNeXt50, ResNeXt101)	Grouped convolutions for improved efficiency and performance	Classification, object detection, segmentation

Comparisons

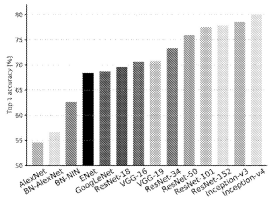


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share the same hue, for example ResNet are all variations of pink.

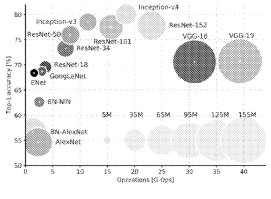


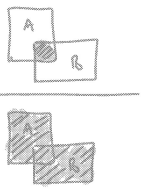
Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same y-axis, and the grey dots highlight the centre of the blobs.

Refer image

Detection evaluation

* IoU / Jaccard index (similarity)

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - A \cap B}$$



$IoU \in [0, 1]$ ← same; ↙ mutually exclusive

IOU - Intersection over union

Here, we talk about only localization and not classification

* Jaccard distance

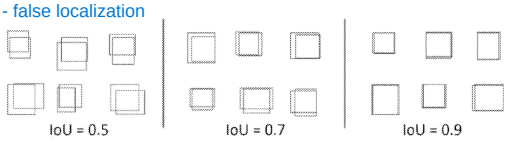
$$d_J(A, B) = 1 - IoU(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

mostly IOU is in CV, Jacard distance in some apps only

Detection evaluation

- **A threshold could be used to determine correct detection**

lower IoU - false localization



- **In addition to evaluating localization, we also need to evaluate classification. That is, evaluate the class that is attached to each detection.**
- **Include a background class for proper evaluation**

Detection evaluation

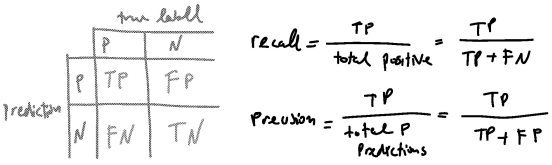
class evaluation

- **Mean average precision (MAP)**
- **Commonly used to evaluate object detection**
- **Whereas IOU only takes into account localization accuracy, MAP measures classification accuracy of detected objects (determined at different IOU threshold values)**

like considering IOU - 0.7 to be something acceptable, we evaluate the classification by comparing with the ground truth

Detection evaluation

- **Precision and recall:**



- **There's a trade off between precision and recall:**
  - **A higher confidence threshold leads to higher precision and lower recall**
  - **A lower confidence threshold leads to lower precision and higher recall**

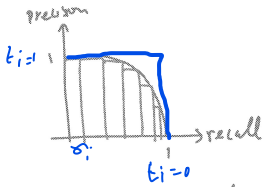
the confidence threshold we talk about here is from the probabality output that we get .....if prob of being a dog is 0.8 and being a cat is 0.7.....say, we think that 0.8 is good enough to say there is a dog, we set a confidence threshold value of 0.8 , then it would be considered dog.....

Detection evaluation

* ROC curve: Receiver Operating Characteristic curve

- Record precision and recall at different confidence levels.

perfect classifier in blue



r_i recall at threshold t_i

p_i precision at threshold t_i

AUC = area under curve

AUC $\in [0, 1]$ bigger is better

AP = average precision (computes AUC)

$$AP = \sum_{\substack{\text{confidence} \\ \text{thresholds } t_i}} (r_{i+1} - r_i)\, p_{i+1}$$

we can just use accuracy to identify if it is a classifier, but we do not know what prob threshold was used to classify.

so, we use precision and recall which hels to understand the accuracy over diff threshold levels of prob for clasification

Detection evaluation

- **Average precision interpretation:**

$$AP = \sum_{\substack{\text{confidence} \\ \text{thresholds } t_i}} (r_{i+1} - r_i)\, p_{i+1}$$

- **Average precision (AP) can be viewed as a weighted sum of precision values at different confidence level thresholds. The weight coefficients are defined by the increase in recall values**
- **With AP we no longer need to select a threshold to determine a single set of precision and recall values**
- **Mean average precision (mAP):**
  - **Compute average precision (AP) for each class, and average the AP for all classes (i.e. ROC curve for each class)**
  - **In detection problems, include a class of background (no object)**

we wanted to eliminate the dependency on any threshold, so we average it for the metric to be fair

Detection evaluation

- **IOU threshold:**
  - **In the context of detection, to compute AP we first have to select on IOU threshold to determine there was a detection**
  - **We can then compute AP as a measure of classification correctness**
- **The selection of IOU threshold is somewhat ambiguous and will affect AP results:**
  - **With a lower IOU threshold standard we will have more detections but with worse localization and possibly worse accuracy**
  - **With a higher IOU threshold standard we will have fewer detections but with better localization and possibly more accuracy**
- **Coco mAP:**
  - **Calculate mAP at different IOU threshold values and average them**

$AP\ @\ [.5 : .05 : .95]$ — 10 threshold values

$$mAP = \frac{mAP_{0.5} + mAP_{0.55} + \cdots + mAP_{0.95}}{10}$$

Object localization

- To localize an object we need to specify a bounding box:

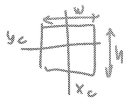
$(x_{min}, y_{min}, x_{max}, y_{max})$

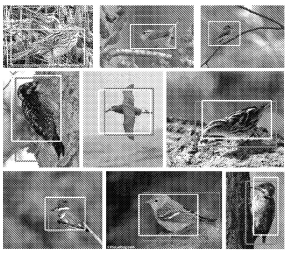
- Because we process the image at different scales, absolute coordinates cannot be used in so we use relative location:

$x_{min}, x_{max}, y_{min}, y_{max} \in [0,1]$

- Alternatively, use:

(x_c, y_c, w, h)





if we use the coordinates values directly for the bounding box, then if we resize the i/p imag , it would change the location....we have to mark the box relative to the image size

Refer image

Object localization

- Training:
  - For each example mark bounding box and label class
  - Include background (no object) boxes
  - One hot encode class labels

marked 1 depending which class is true and all others 0

$y = [p_c, x_c, y_c, w, h, c_1, \dots, c_K]$

p_c: 1 if object box, 0 if not object box; x_c, y_c, w, h: bounding box; $c_1, \dots, c_K$: one-hot encoded class

- Classification:

$\hat{y} = [p_c, x_c, y_c, w, h, c_1, \dots, c_K]$

p_c: object probability; x_c, y_c, w, h: bounding box; $c_1, \dots, c_K$: class probabilities