Escuela Técnica Superior de
Ingeniería Informática

Aprendizaje Profundo

MobileNetV2

Inverted Residuals and Linear Bottleneck

Profesores

Miguel A. Gutierrez Naranjo

David Solis Martin

Alumno

Germán Lorenz Vieta

Índice

Introducción

Las redes neuronales han revolucionado distintas áreas de la inteligencia artificial permitiendo una precisión sobrehumana en tareas desafiantes de reconocimiento de imágenes.
Usualmente este impulso para mejorar la precisión requiere más poder de cómputo que muchos dispositivos móviles no poseen en la actualidad.

La arquitectura que se presenta está diseñada específicamente para entornos móviles y recursos limitados el cual disminuye significativamente la cantidad de operaciones y memoria necesaria manteniendo la precisión.

La principal contribución del artículo es el módulo **Inverted Residual con Linear Bottleneck** el cual toma una representación comprimida de baja dimensión, la expande y se filtra con una convolución ligera en profundidad para proyectarla en una representación de baja dimensión.

Trabajo relacionado

Existe trabajo relacionado en el área en la búsqueda de una arquitectura que encuentre balance entre precisión y rendimiento por distintos equipos que han mejorado los diseños iniciales tales como AlexNet [5], VGGNet[6] , GoogleNet[7] y ResNet[8]. También existe trabajo en la explotación de hiperparametros [9, 10, 11], poda de redes [12, 13, 14, 15, 16, 17] y aprendizaje por conectividad [18, 19] o cambiando la conectividad ShuffleNet [20] o introduciendo la escasez [21].

El trabajo de MobileNetV2 está basado en MobileNetV1 manteniendo su simplicidad mejorando su precisión en tareas de clasificación y detección de múltiples imágenes

Discusión e intuición de bloques

Depthwise Separable Convolution

La idea básica es reemplazar un operador convolucional completo por uno que divide la convolución en dos capas separadas. [22, 23]

- La primera capa se denomina DepthWise Convolution y realiza y filtrado ligero mediante la aplicación de un filtro convolucional por canal de entrada.

- La segunda capa es una convolución 1x1 llamada Pointwise Convolution que es responsable de crear nuevas funciones mediante combinaciones lineales de los canales de entrada.

La Convolución Regular toma un tensor de entrada h_i x w_i x d_i en L_i y aplica un kernel convolucional k e $R^{k \times k \times di \times dj}$ para producir un tesnor de salida h_i x w_i x d_j en L_j con costo computacional $h_i * w_i * d_i * d_j * k * k$.

La Depthwise Separable Convolution funcionan casi tan bien como las regulares o estandar pero cuestan $h_i * w_i * d_i (k^2 + d_j)$ lo cual es sustancialmente beneficioso en el uso de recursos computacionales y de asignación de memoria.

La reducción de cálculo es de casi un factor k^2 "aprox" $(k^2 * d_j / (k^2 + d_j)$ mientras que MobileNetV2 con kernel = 3 usa Depthwise Separable Convolution 3 x 3 por lo que el costo computacional es 8 a 9 veces más pequeño que la estándar.

Linear Bottlenecks

Considere que una red profunda consta de n capas las cuales tienen un tensor de activación de dimension h_i x w_i x d_i que trataremos como h_i x w_i "pixeles" con d_i dimensiones. Entonces para un conjunto de entradas dado por imágenes decimos que el conjunto de activaciones de capa L_i, forma una "variedad de intereses".
Este hecho se puede explotar reduciendo la dimensionalidad de una capa para reducir el espacio operativo tal como se explora en MobilenetV1 [22] para equilibrar el cálculo y la precisión a través de un parámetro multiplicador de ancho llamado factor de expansión t. Sin embargo esta intuición se rompe en redes profundas ya que tienen transformaciones no lineales por coordenadas como ReLU.

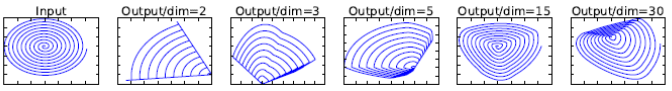

Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

Por ejemplo ReLU aplicado a una línea en un espacio de 1 dimensión produce un "rayo" según denominan los autores donde en un espacio R^n generalmente da como resultado una curva lineal por partes con n articulaciones. En la *Figura 1* se observa cómo afecta ReLU en las transformaciones fuertemente.
De esta forma los autores dan a comprender que la reducción de caracteristicas afecta las **variables de interés** de la siguiente forma:

- Si los canales disminuyen al aplicar ReLU existe alta probabilidad de pérdida de información
- Si los canales aumentan al aplicar ReLU existe la probabilidad de perder menos información

Por otro lado cuando ReLU colapsa el canal se pierde información pero si tenemos muchos canales y múltiples activaciones esa información podrá conservarse en otros canales (por la extra dimensionalidad), esto significa que:

- Por intuición, si trabajamos con un subespacio de interés nosotros queremos que las **variables de interés** no se pierdan y por ello se utiliza un factor de expansión que compensa la reducción de dimensionalidad.

Entonces, hay dos propiedades que los autores indican como importantes para que la **variedad de interés** se encuentre en un subespacio de baja dimensión del espacio de activación de dimensión superior las cuales son:

- Si la **variedad de interés** sigue siendo un volumen distinto de cero después de transformación ReLU, corresponde a una transformación lineal.
- ReLU es capaz de conservar información completa solo si la variedad de entrada se encuentra en un subespacio de baja dimensión del espacio de entrada.

Estas ideas nos dan a entender que si la **variedad de interés** es de baja dimensión podemos capturarla con un ***Linear Bottlenecks*** en la capa convolucional. En la etapa experimental el uso de estas capas es crucial ya que evita la destrucción de demasiada información por no linealidad [24].

En la *Figura 2* se grafica la evolución de los distintos bloques Bottleneck.

- El bloque **Regular** sencillamente mezcla las capas al aplicar un kernel
- El bloque **Separable** separa cada capa y añade el factor de expansión t
- El bloque **Separable con linear Bottleneck** agregar el Pointwise para comprimir información y encontrar esa "variedad de interes"
- El bloque **Bottleneck con capa de expansión** difiere en que su input es un Pointwise y su output también
    - PointWise es quien hace las combinaciones lineales de los canales de entrada

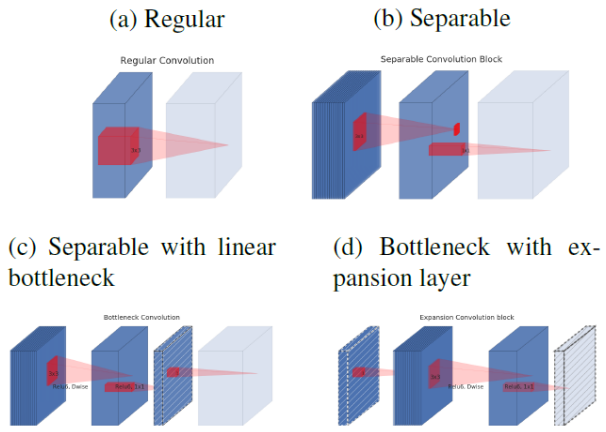

Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

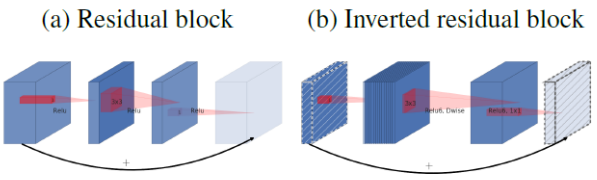

Figure 3: The difference between residual block [8, 30] and inverted residual. Diagonally hatched layers do not use non-linearities. We use thickness of each block to indicate its relative number of channels. Note how classical residuals connects the layers with high number of channels, whereas the inverted residuals connect the bottlenecks. Best viewed in color.

Inverted Residuals

Los bottleneck blocks parecen similares a los bloques residuales donde cada bloque contiene una entrada seguida de varios bottleneck blocks seguidos de una expansión pero inspirados por la intuición de que los bottleneck blocks contienen toda la información y la capa de expansión actúa como un detalle que acompaña a una

transformación no lineal del tensor, usamos atajos entre los bottleneck dependiendo del stride del deepwise.

El motivo de utilizarlos es similar al de las conexiones residuales clásicas [8, 25] ya que queremos mejorar la capacidad de un degradado para propagarse a través de capas expansivas y el diseño invertido es más eficiente en memoria inspirados en ResNet[8]. En la *Figura 3* se observa la diferencia entre bloques

Para un bloque de tamaño (h x w) y factor de expansión t, un kernel de tamaño k con d' canales de entrada y d'' canales de salida, el numero total de multiplicidad requerida es h * w * d' * t(d'+k^2 + d''). En la *Tabla 3* se comparan los tamaños necesarios para cada resolución entre MobileNet V1, V2 y ShuffleNet.

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

Table 3: The max number of channels/memory (in Kb) that needs to be materialized at each spatial resolution for different architectures. We assume 16-bit floats for activations. For ShuffleNet, we use $2x, g = 3$ that matches the performance of MobileNetV1 and MobileNetV2. For the first layer of MobileNetV2 and ShuffleNet we can employ the trick described in Section 5 to reduce memory requirement. Even though ShuffleNet employs bottlenecks elsewhere, the non-bottleneck tensors still need to be materialized due to the presence of shortcuts between the non-bottleneck tensors.

Interpretación del flujo de la información

La arquitectura nueva proporciona una separación natural entre los dominios de entrada y salida de los bloques de construcción (capas bottleneck) y las capas de transformación (funciones no lineales que convierten la entrada en una salida dependiendo de las circunstancias de ejecución).
La primera puede verse como la **capacidad** de la red y la segunda como la **expresividad.** Esto contrasta con los bloques convolucionales tradicionales que entrelazan ambas caracteristicas como funciones en la capa de salida.

Arquitectura

El componente básico es una **bottleneck residual block** descrito en *Tabla 1*.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

La arquitectura de MobileNetV2 contiene la capa inicial de convolución completa con 32 canales, seguidos de capas **bottleneck** descritos en *Tabla 2*.

Usa ReLU6 como no-linealidad con kernel de 3 x 3 ya que es el estándar en redes modernas y una normalización con dropout durante el entrenamiento.

Excepto la primera capa se usa una tasa de expansión t constante en la red los demas bottleneck usan un t = 6. Los mejores resultados se obtuvieron con este factor de expansión aplicado al tamaño del tensor de entrada. Dependiendo del stride de la **deep wise convolutio**n dentro de la bottleneck se activará o no el bloque **inverted residual**.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

Hiperparametros

La arquitectura se adaptó a diferentes puntos de rendimiento mediante el uso de la resolución de la imagen de entrada y el multiplicador de ancho como hiperparamétros ajustables según precisión y rendimiento.
La red principal tiene un ancho de 1 con 224 x 224 y costo computacional de 300 millones de multiplicaciones (MAdds) con 3,4 millones de parámetros. Para resoluciones de entre 96 a 224 y ancho de multiplicación de entre 0,35 a 1,4. El costo computacional oscila entre 7 a 585 millones de MAdds mientras que el tamaño varía entre 1,7 a 6,9 millones de parámetros.

Implementación

Inferencia eficiente en memoria

Al momento de redacción del artículo los autores comentan que las inverted residual bottleneck permiten una implementación especialmente eficiente en memoria para dispositivos móviles. Esta implementación presente en Tensorflow [31] o Caffe [32] crea un hipergrafo acíclico dirigido que consta de aristas que representan operaciones y nodos que representan tensores. El cálculo está programado para minimizar el numero total de tensores en memoria. En la *Figura 5* se puede observar la comparativa entre Performance y MAdds de los modelos ShuffleNet, NAS, MobileNetV1 y MobileNetV2

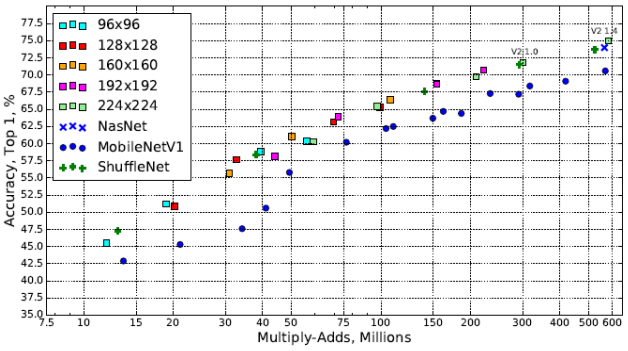

Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.

Pruebas

Clasificación de ImageNet

En el artículo se usó para entrenar Tensorflow con RMSPropOptimizer con both decay y momentum en 0,9. Se usó batch normalization después de cada capa y wight decay en 0.00004 con ImageNet[1].
Se usó para MobileNetV1 tasa de aprendizaje en 0,045 y learning rate decay en 0,98 por epoch.
Se comparó MobileNetV2 con V1 y ShuffleNet obteniendo una mejor clasificación usando menos parámetros y un tiempo de respuesta completamente superior..

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

Detección de objetos

De la misma manera los autores describen cómo se evaluó y comparó el rendimiento de MobileNetV2 y MobileNetV1 como extractores de funciones [33] para la detección de objetos con SSD [34] en el conjunto de datos COCO [2]. También se comparó con YOLOv2 [35] y SSD Original (con red VGG-16) [6] expuestos en *Tabla* 6.

En el artículo se utilizó SSDLite que es una modificación de SSD en la cual se reemplazan las convoluciones estándar por separables ya que este diseño está alineado con el diseño general de MobileNet y se lo considera más eficiente desde lo computacional.
En la Tabla 5 se puede ver la reducción drástica en MAdds.

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

Para MobileNetV1 se configuró según [33], para MobileNetV2 la primera capa de SSDLite se hicieron modificaciones en la capa de expansión de la capa 15 (con stride de salida en la 16). La segunda y el resto de capas SSDLite se adjuntan encima de la última capa (con un stride de salida en la 32).
Los entrenamientos en el artículo fueron bajo la API de Tensorflow [38]

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Table 6: Performance comparison of MobileNetV2 + SSDLite and other realtime detectors on the COCO dataset object detection task. MobileNetV2 + SSDLite achieves competitive accuracy with significantly fewer parameters and smaller computational complexity. All models are trained on `trainval35k` and evaluated on `test-dev`. SSD/YOLOv2 numbers are from [35]. The running time is reported for the large core of the Google Pixel 1 phone, using an internal version of the TF-Lite engine.

Los resultados muestran que MobileNetV2 es más eficiente y 10 veces más pequeño mientras que aún así supera a YOLOv2 en COCO.

Segmentación Semántica

Se compara MobileNetV1 y MobileNetV2 utilizándolos como extractores de funciones con DeepLabv3 [39] para la tarea de segmentación semántica móvil. DeepLabv3 adopta una atrous convolution [29, 30 ,36, 37].

Para la segmentación semántica, utilizaron *output_stride* =16 u 8 para mapas de características más densos. Realizaron los experimentos en el conjunto de datos PASCAL VOC 2012 [3], con imágenes anotadas adicionales de [28] y métrica de evaluación mIOU.
Los resultados se resumen en la Tabla 7.

Network	OS	ASPP	MF	mIOU	Params	MAdds
MNet V1	16	✓		75.29	11.15M	14.25B
	8	✓	✓	78.56	11.15M	941.9B
MNet V2*	16	✓		75.70	4.52M	5.8B
	8	✓	✓	78.42	4.52M	387B
MNet V2*	16			**75.32**	**2.11M**	**2.75B**
	8		✓	77.33	2.11M	152.6B
ResNet-101	16	✓		80.49	58.16M	81.0B
	8	✓	✓	82.70	58.16M	4870.6B

Table 7: MobileNet + DeepLabv3 inference strategy on the PASCAL VOC 2012 *validation* set. **MNet V2***: Second last feature map is used for DeepLabv3 heads, which includes (1) Atrous Spatial Pyramid Pooling (**ASPP**) module, and (2) 1×1 convolution as well as image-pooling feature. **OS**: *output_stride* that controls the output resolution of the segmentation map. **MF**: Multi-scale and left-right flipped inputs during test. All of the models have been pretrained on COCO. The potential candidate for on-device applications is shown in bold face. PASCAL images have dimension 512×512 and atrous convolution allows us to control output feature resolution without increasing the number of parameters.

Según el artículo se observa que es más eficiente construir con DeepLabv3 encima del penúltimo mapa de características de MobileNetV2 que en el mapa de características de la última capa original, ya que el penúltimo mapa de características contiene 320 canales en lugar de 1280 ,y al hacerlo, lograron un rendimiento similar, requiriendo 2.5 veces menos operaciones que las contrapartes de MobileNetV1

Al final de la *Tabla 7*, identificaron un candidato potencial para aplicaciones mobile que alcanza *75.32% mIOU* y solo requiere *2.75B MAdds.*

Pruebas personales

Para la exposición me inspire en el código fuente del usuario saunack en github [26] el cual implementa MobileNetV2 basándose estrictamente en el artículo original con SSD como demostración de capacidad en object detection con 25 épocas usando el set de datos MNIST.

Conclusiones

- En el artículo se describe una arquitectura de red simple que permite crear una familia de modelos para dispositivos móviles altamente eficiente.

- La arquitectura propuesta permite una inferencia muy eficiente en memoria confiando en operaciones estándar en todos los marcos neuronales.

- Para el conjunto ImageNet la arquitectura mejoró el estado de arte en cuanto a rendimiento.

- Para la detección de objetos la red supera a los mejores detectores en tiempo real en COCO tanto en precisión como en complejidad del modelo al momento de su publicación. La combinación con SSDLite brinda una capacidad superior a YOLOv2.

- Por el lado teórico el bloque convolucional propuesto tiene una propiedad única que separa la **expresividad** de la red de su **capacidad.**

Bibliografía

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. Int. J. Comput. Vision, 115(3):211–252, December 2015.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll´ar, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In ECCV, 2014.

[3] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserma. The pascal visual object classes challenge a retrospective. IJCV, 2014.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Bartlett et al. [27], pages 1106–1114.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015, pages 1–9. IEEE Computer Society, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13:281–305, 2012.

[10] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Bartlett et al. [27], pages 2960–2968.

[11] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In Francis R. Bach and David M. Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, volume 37 of JMLR Workshop and Conference Proceedings, pages 2171–2180. JMLR.org, 2015.

[12] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992], pages 164–171. Morgan Kaufmann,1992.

[13] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989], pages 598– 605. Morgan Kaufmann, 1989.

[14] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. En Corinna Cortes

[15] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J. Dally. DSD: regularizing deep neural networks with dense-sparse-dense training flow. CoRR, abs/1607.04381, 2016.

[16] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pages 1379–1387, 2016.

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. CoRR, abs/1608.08710, 2016.

[18] Karim Ahmed and Lorenzo Torresani. Connectivity learning in multi-branch networks. CoRR, abs/1709.09582, 2017.

[19] Tom Veniat and Ludovic Denoyer. Learning time efficient deep architectures with budgeted super networks. CoRR, abs/1706.00046, 2017.

[20] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. CoRR, abs/1707.01083, 2017.

[21] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. CoRR, abs/1702.06257, 2017.

[22] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.

[23] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[24] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. CoRR, abs/1610.02915, 2016.

[25] Saining Xie, Ross B. Girshick, Piotr Doll´ar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. CoRR, abs/1611.05431, 2016.

[26] MobileNetv2-SSD notebook. Disponible en https://github.com/saunack/MobileNetv2-SSD

[27] Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors. Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, 2012.

[28] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In ICCV, 2011.

[29] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In Wavelets: Time-Frequency Methods and Phase Space, pages 289–297. 1989.

[30] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann Le-Cun. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv:1312.6229, 2013.

[31] Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man´e, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[32]Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

[33] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In CVPR, 2017.

[34] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In ECCV, 2016.

[35] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242, 2016.

[36] George Papandreou, Iasonas Kokkinos, and Pierre-Andre Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In CVPR, 2015.

[37] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. TPAMI, 2017.

[38] Jonathan Huang, Vivek Rathod, Derek Chow, Chen Sun, and Menglong Zhu. Tensorflow object detection api, 2017.

[39] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. CoRR, abs/1706.05587, 2017.