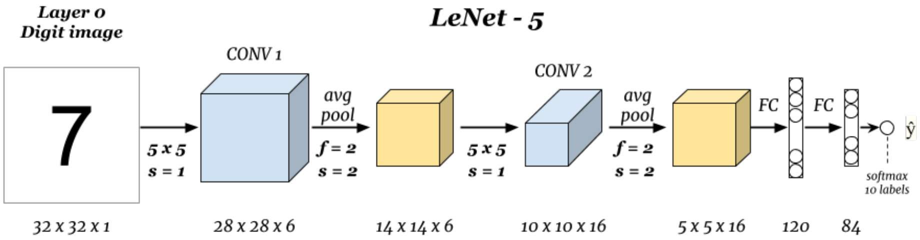
Learning with CNNs

Tianxiang (Adam) Gao

February 13, 2025

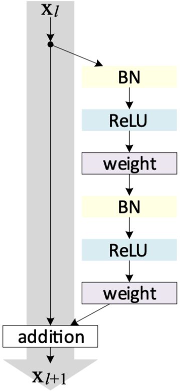
Outline

Recap: Convolutional Neural Networks (CNNs)



- **Convolution operation:** It slides a small **filter** over the input image, performing a **locally linear transformation** to produce a feature map that detects patterns.
- **Padding and stride:** Methods for controlling feature map size, preserving spatial dimensions, and improving *computational efficiency*.
- **Convolution over volumes and with multiple filters:** The input can be multi-channel, and so as the output with the use of multiple filters.
- **Weight Sharing and Sparsity:** Neurons in CNNs **share** weights across locations, with each output relying on a **small, localized** input region.
- **Hierarchical Feature Detection:** Early layers capture basic features (like edges), which later layers combine into higher-level features.

Recap: Stabilizing CNN Training



- **Pooling**: Reduces the spatial dimensions of feature maps by downsampling, typically using max or average pooling.
- **Batch Normalization**: Normalizes pre-activation values at hidden layers, mitigating internal covariate shift and accelerating training.
- **Skip Connections**: Create shortcuts by directly adding input to output, stabilizing information flow in deep neural networks.
- **Classic CNNs**: Spatial dimensions shrink while the number of channels increases as depth grows. Overparameterized and deeper CNNs are generally preferred.

Recap: Semantic Segmentation and U-Net

Input

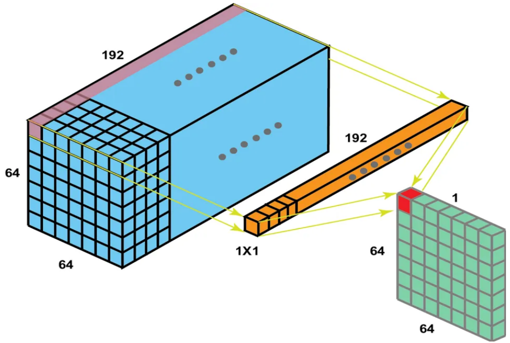
Semantic Labels

- Assigns a class label to **each pixel** in the image
- The output is an **image** with the same spatial dimensions as the input
- The encoder (typically a CNN) extracts **hierarchical** feature representations
- The decoder uses **transposed convolution** (deconvolution) to restore spatial resolution
- Skip connections **combines** low-level spatial details with high-level semantic features

Outline

1×1 Convolutions

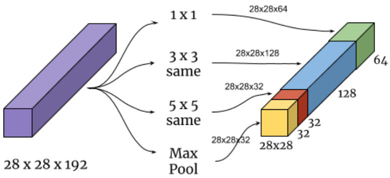
$$\underbrace{\begin{bmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{bmatrix}}_{\textbf{input image } 3 \times 3} * \underbrace{\begin{bmatrix} 2 \end{bmatrix}}_{\textbf{filter } 1 \times 1} = \underbrace{\begin{bmatrix} 6 & 0 & 2 \\ 2 & 10 & 16 \\ 4 & 14 & 4 \end{bmatrix}}_{\textbf{feature map } 3 \times 3}$$

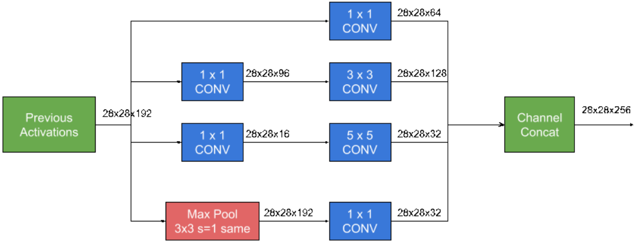


- Recall that each channel in the input data serves as an **indicator map** for certain patterns detected by the filer.
- Each pixel, with multiple channels, represents a **set** of indicators that capture the distribution of **multiple patterns** within a local region of the original image.
- Applying a 1×1 convolution enables **cross-channel interaction** to detect complex patterns that integrate multiple underlying features.
- This approach is useful for combining lower-level features into **higher-level** representations.

Lin et al. *"Network in Network"*, ICLR 2014.

Inception Networks





- Inception networks allow the model to freely select the best filter sizes within a layer, rather than manually choosing them, by providing multiple filter options.
- This approach can be computationally expensive (e.g., a 5x5 filter requires about 120 million operations).

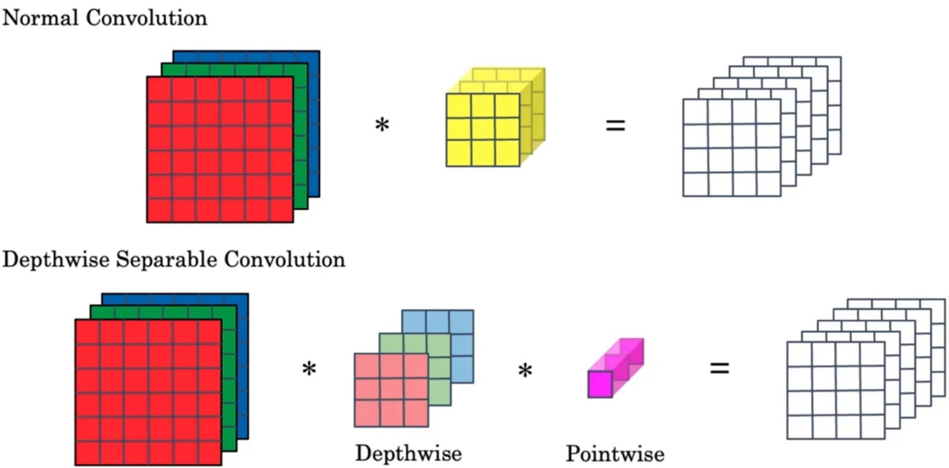
$$(28 \times 28 \times 32) \times (5 \times 5 \times 192) \approx 120 \text{ million}$$

- Applying 1×1 convolutions reduces computation by approximately 90%.

$$(28 \times 28 \times 16) \times (1 \times 1 \times 192) + (28 \times 28 \times 32) \times (5 \times 5 \times 16) \approx 2 \text{ million} + 10 \text{ million}$$

Szegedy et al. *"Going deeper with convolutions"*, CVPR 2014.
Image is adapted from https://indoml.com/.

MobileNets

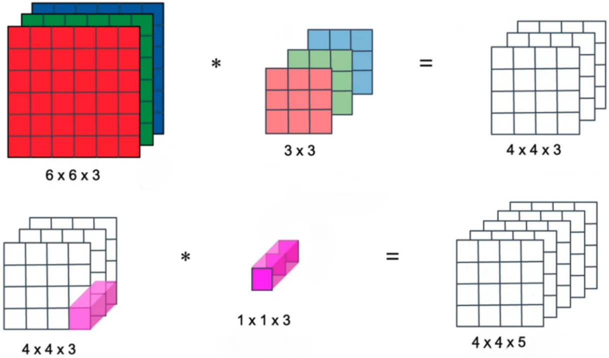


The computational cost of a normal convolution is:

$$\underbrace{2,160}_{\textbf{Computational cost}} = \underbrace{3 \times 3 \times 3}_{\textbf{\# params}} \times \underbrace{4 \times 4}_{\textbf{\# locations}} \times \underbrace{5}_{\textbf{\# filters}}$$

Howard, Andrew G. "*Mobilenets: Efficient convolutional neural networks for mobile vision applications.*" arXiv 2017.

Depthwise Separable Convolution

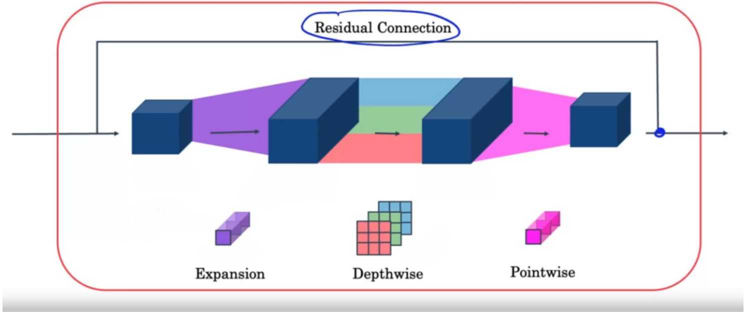


- Each channel has an **independent** filter for the convolution operation.
- The result is then processed using a pointwise or 1×1 convolution.
- Computational cost ratio: $\frac{1}{C_{out}} + \frac{1}{f^2}$

$$672 = \underbrace{3 \times 3}_{\text{\# params}} \times \underbrace{4 \times 4}_{\text{\# locations}} \times \underbrace{3}_{\text{\# filters}} + \underbrace{3}_{\text{\# params}} \times \underbrace{4 \times 4}_{\text{\# locations}} \times \underbrace{5}_{\text{\# filters}}$$

Images adapted from *CS230 Deep Learning* at Stanford.

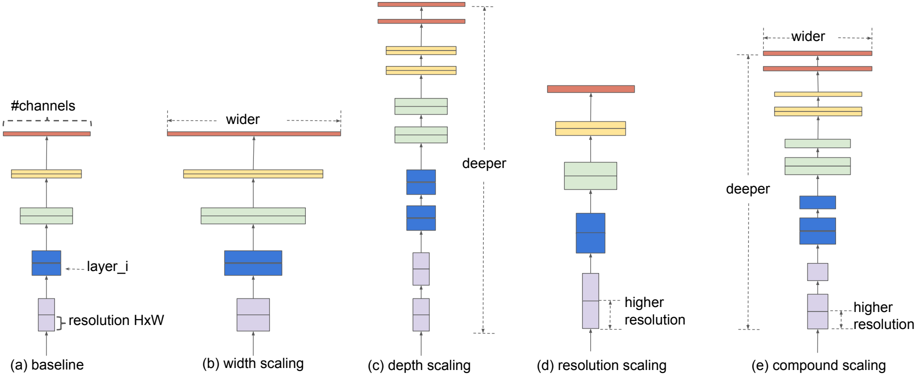
MobileNetV2: Inverted Residual Blocks



- The **expansion convolution** allows CNNs to learn richer functions by increasing the number of channels before depthwise convolution
- When deploying on resource-constrained devices, a **pointwise convolution** is used to project back to a smaller number of channels, reducing memory usage.
- VGG-16 $\approx$ 528 MB, ResNet-152 $\approx$ 230 MB, Inception $\approx$ 85 MB, but MobileNet $\approx$ **5-16 MB**.

Sandler, Mark, et al. *"Mobilenetv2: Inverted residuals and linear bottlenecks."* CVPR 2018.

EfficientNet: Compound Scaling



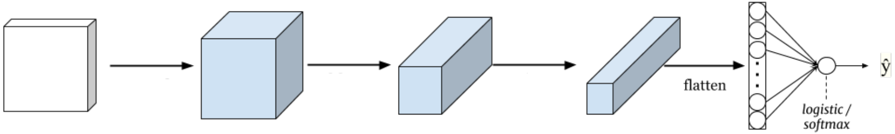
- **Depth** (d): A deeper CNN can capture **richer hierarchical features** and more complex patterns.
- **Width** (w): Wider networks can **capture more fine-grained features** and **easier to train**.
- **Resolution** (r): Higher resolution inputs help CNNs **capture more fine-grained spatial details**.
- **Compound Scaling**: EfficientNet scales all three dimensions **together**: $d = \alpha^{\phi}$, $w = \beta^{2\phi}$, $r = \gamma^{2\phi}$ such that $\alpha\beta^2\gamma^2 \approx 2$.

Tan & Le, *"EfficientNet: Rethinking Model Scaling for CNNs"*, ICML 2019.

Outline

Transfer Learning

- **Definition**: Transfer learning leverages a **pre-trained** model on a large dataset to solve a new, related task on a smaller dataset.
- **Motivation**: Training deep networks from scratch demands significant data and computational resources. The initial layers in CNNs capture fundamental features that are transferable across various tasks, allowing the higher layers to focus on learning task-specific features.



- **Typical Workflow**:
  1. **Pre-training**: A model is trained on a large dataset (e.g., ImageNet).
  2. **Freezing Layers**: Early layers are frozen to retain their learned features.
  3. **Fine-tuning**: With a lower learning rate, higher layers are retrained on the target dataset, allowing for adaptation to new, task-specific features.
- Frozen early layers can be considered as **fixed** feature extractors, allowing activations to be precomputed, which reduces computation and speeds up training.
- For **larger** target datasets, freeze fewer layers to enable more flexibility in learning higher-level task-specific features.

Data Augmentation

- **Definition**: Data augmentation involves generating new training samples by applying various transformations to existing data, helping improve model generalization.
- **Motivation**: Increases the diversity of the training dataset, which can reduce overfitting and improve the model's performance on unseen data.
- **Flips, rotations, and scaling:**

- **Random cropping**:

- **Color Adjustments:** Changes to colors or brightness

Mixup

- **Mixup** creates new training samples by blending pairs of images and their corresponding **labels**:

$$\tilde{\boldsymbol{x}} = (1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2, \quad \text{and} \quad \tilde{\boldsymbol{y}} = (1-\lambda)\boldsymbol{y}_1 + \lambda\boldsymbol{y}_2,$$

  where $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ are original training samples, and λ is the mixing factor drawn from a Beta distribution.
- This technique helps the model generalize by learning smoother transitions between classes.

Zhang, H., et al. *"Mixup: Beyond Empirical Risk Minimization."* ICLR 2018.

Outline

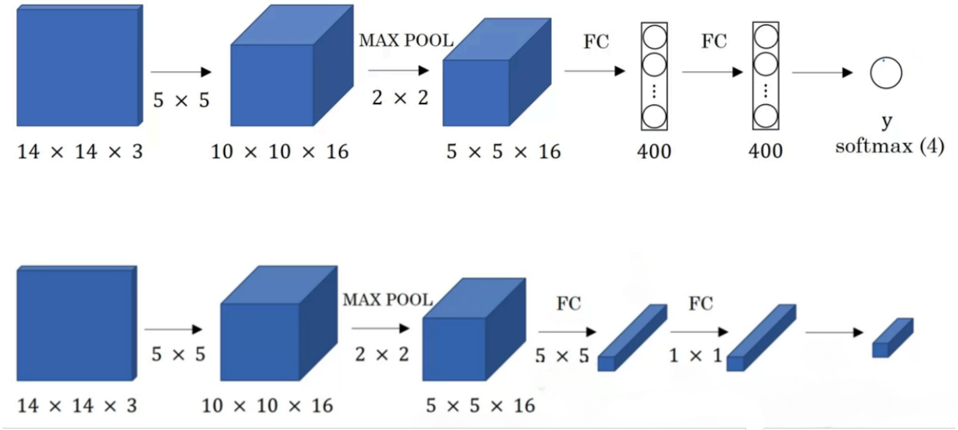
Sliding Window Detection

- Use a **pre-trained classifier** to identify cars in images.
- **Scan** the entire image with a sliding window, classifying each cropped section.

- **Problem**: This approach is computationally intensive due to the number of windows.

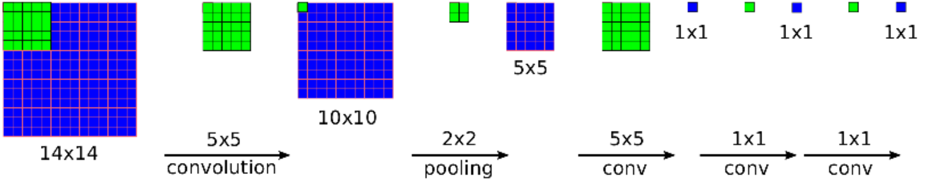
Convolutional Layers as Fully Connected Layers

Fully connected layers can be implemented by convolutional layers:

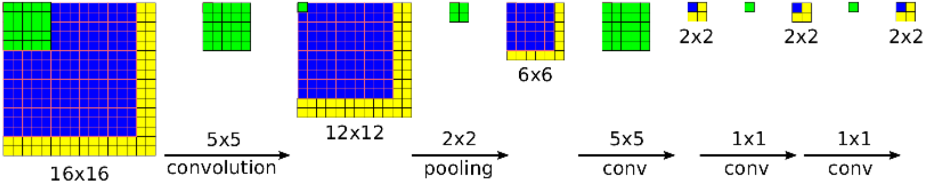


Convolution Implementation of Sliding Windows

ConvNets:



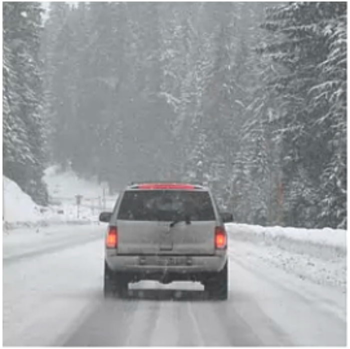
Detection:



- During **training**, a ConvNet produces only a **single** spatial output
- When applied at **test time** over a **larger** image, it produces a spatial output **map**.
- However, with a fixed sliding window size, the bounding boxes may lack precision.

Sermanet, P., et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." ICLR 2014

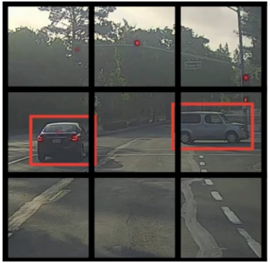
Classification with Object Localization



- **Input**: An image containing a **single object**
- **Output**: Class probability and bounding box $\boldsymbol{y} = (p_c, b_x, b_y, b_w, b_h)$
- $p_c \in [0, 1]$: Object presence confidence score; (b_x, b_y): object center; b_w, b_h: width and height
- Use **squared loss** for bounding box predictions
- **Multi-classification**: the output becomes $\boldsymbol{y} = (p_c, b_x, b_y, b_w, b_h, c_1, c_2, \cdots, c_N)$

YOLO Algorithm

YOLO Algorithm Steps: Divide image into $S \times S$ **grid cells**



- Each cell predicts **class probabilities** and **bounding boxes**

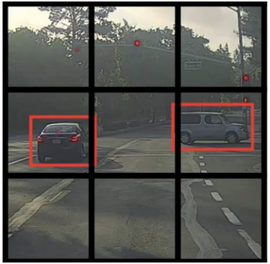
$$y_s = (p_c, b_x, b_y, b_w, b_h, c_1, \cdots, c_N), \quad \forall s \in [S]$$

- In each cell, object center (b_x, b_y) uses local coordinates in $[0,1] \times [0,1]$
- b_w and b_h can be greater than 1 (relative to image size)

Redmon, J., et al. "You only look once: Unified, real-time object detection." CVPR 2016.

YOLO Algorithm: IoU and Non-Maximum Suppression (NMS)

Problem: Multiple bounding boxes may be predicted for the same object.



- Choose the bounding box with the highest confidence score p_c as the best detection.
- Remove all other highly overlapping boxes with **Intersection over Union (IoU)** > 0.5 compared to the best box.

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- Keep low-overlap boxes as they likely correspond to different objects (e.g., other cars).

Redmon, J., et al. "You Only Look Once: Unified, Real-Time Object Detection." CVPR 2016.

Outline

Verification vs. Recognition

Verification

- **Input**: An image
- **Output**: Confirms if the image matches the claimed person

Recognition

- **Database**: Contains K known identities
- **Input**: An image
- **Output**: Returns the person's ID if in the database; otherwise, not recognized

One-Shot Learning and Similarity Function

Problem: Traditional deep learning models require large amounts of labeled data, but face recognition needs to identify a person with only a few examples.

One-Shot Learning

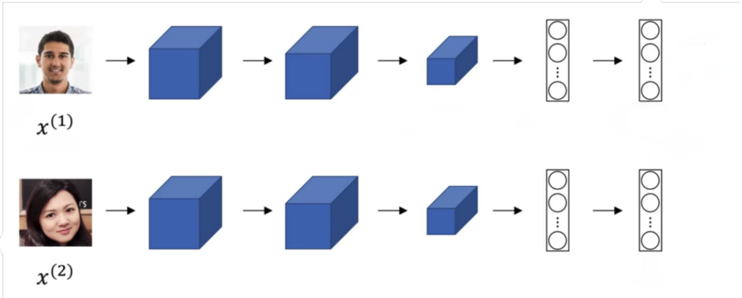
- Learns to recognize a person from **a single example** rather than relying on large datasets.
- Challenges with new identities that were not seen during training.
- Solution: Instead of classification, use a similarity function to compare images

Learning a Similarity Function

- Define a function $d(\boldsymbol{x}_1, \boldsymbol{x}_2)$ that measures the **difference** between two images
- If $d(\boldsymbol{x}_1, \boldsymbol{x}_2) \leq$ threshold, classify them as the **same** person and return the identity; otherwise, reject.

Siamese Network

Definition: A Siamese network is a neural architecture consisting of **two identical** sub-networks that share weights and parameters.



- The ConvNet $\boldsymbol{f_\theta}(\boldsymbol{x})$ represents the **embedding** (feature representation) of the input image $\boldsymbol{x}$.
- The similarity between two images is measured using the distance between their **embeddings**:

$$d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \|\boldsymbol{f_\theta}(\boldsymbol{x}^{(i)}) - \boldsymbol{f_\theta}(\boldsymbol{x}^{(j)})\|$$

- The model learns parameters $\boldsymbol{\theta}$ such that:
  - If $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ belong to the same person, $d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is small.
  - If they belong to different people, $d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is large.

Taigman, Yaniv, et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification." CVPR 2014.

Triplet Loss

Anchor

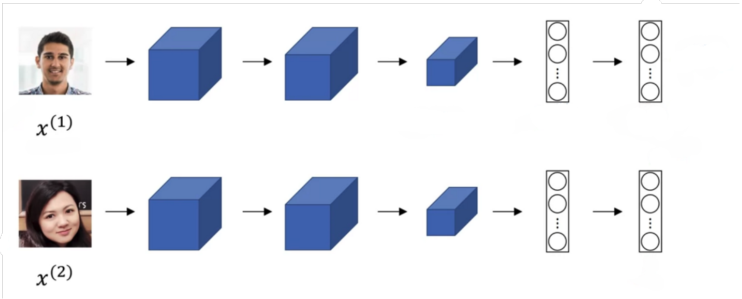
Positive

Negative

- Training uses image triplets: an **anchor** $\boldsymbol{a}$, a **positive** $\boldsymbol{p}$, and a **negative** $\boldsymbol{n}$.
- Objective:
$$d(\boldsymbol{a}, \boldsymbol{p}) + \alpha \leq d(\boldsymbol{a}, \boldsymbol{n})$$
where $\alpha > 0$ is the **margin**, preventing trivial solutions.
- Triplet loss:
$$\ell(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n}) = [d(\boldsymbol{a}, \boldsymbol{p}) + \alpha - d(\boldsymbol{a}, \boldsymbol{n})]_+ ,$$
where $[x]_+ = \max\{x, 0\}$.
$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n})} \ell(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n})$$
- Requires multiple images per identity for effective learning.

Schroff, Florian, et al. "FaceNet: A Unified Embedding for Face Recognition and Clustering." CVPR 2015.

Face Recognition via Binary Classification



- The ConvNet $\boldsymbol{f_\theta}(\boldsymbol{x})$ extracts a feature **embedding** from the input image $\boldsymbol{x}$.
- The similarity between two images is measured as the distance between their embeddings:

$$d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \|\boldsymbol{f_\theta}(\boldsymbol{x}^{(i)}) - \boldsymbol{f_\theta}(\boldsymbol{x}^{(j)})\|$$

- The distance $d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is passed through a **logistic activation** $\sigma(\cdot)$ to classify whether they belong to the same person:

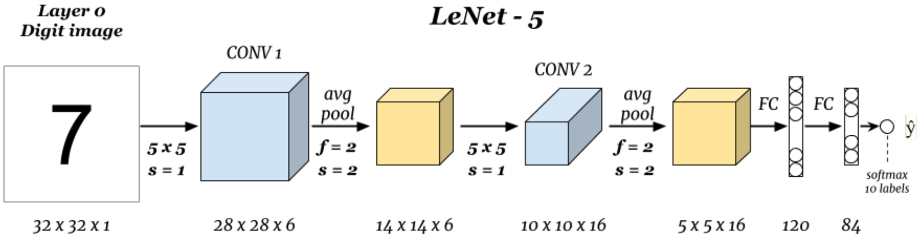
$$\hat{y} = \sigma(d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}))$$

Taigman, Yaniv, et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification." CVPR 2014.

Outline

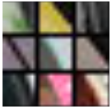
1. Classic CNNs: Inception, MobileNets
2. Practical Advice for Using CNNs
3. Object Detection
4. Face Recognition
5. Neural Style Transfer

Understanding CNNs Through Visualization



How to Visualize CNN Filters:

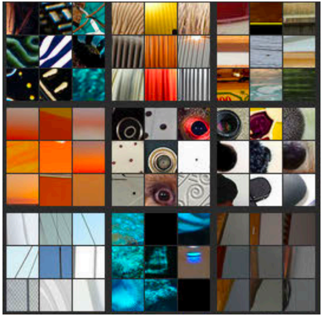
- Process many input images through a **well-trained** CNN.
- Select a **specific convolutional filter** in a layer.
- Find **9 different images** where this filter had the **highest activations** in the feature maps.
- Use DeconvNet to **reconstruct** the input regions responsible for these activations.
- This produces **9 reconstructed patches**, revealing what pattern this filter detects.



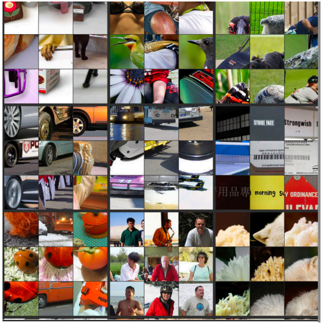
Zeiler, Matthew D., and Rob Fergus. "Visualizing and Understanding Convolutional Networks." ECCV 2014.

Visualizing Deep Layers

Layer 1



Layer 2



Layer 3

Neural Style Transfer

Content

Style

Transfered

- Randomly initialize an initial image $\boldsymbol{g}$
- Optimize $\boldsymbol{g}$ using gradient descent to minimize the total loss:

$$\mathcal{L}(\boldsymbol{g}) = \alpha \mathcal{L}_{\text{content}}(\boldsymbol{g}, \boldsymbol{c}) + \beta \mathcal{L}_{\text{style}}(\boldsymbol{g}, \boldsymbol{s}),$$

where

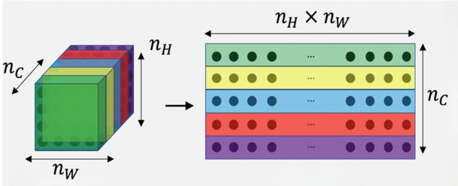
$$\mathcal{L}_{\text{content}}(\boldsymbol{g}, \boldsymbol{c}) = \sum_{\ell} \|\boldsymbol{a}^{\ell}(\boldsymbol{g}) - \boldsymbol{a}^{\ell}(\boldsymbol{c})\|^2$$

Here, $\boldsymbol{a}^{\ell}$ represents the activations of layer ℓ in a pre-trained CNN, *e.g.*, VGG.

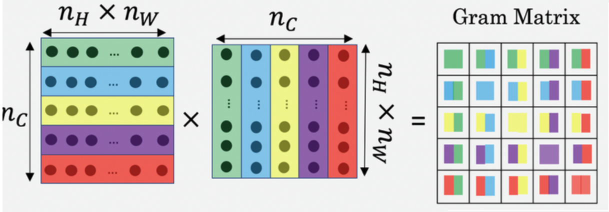
Gatys, Leon A., et al. "Image style transfer using convolutional neural networks." CVPR 2016.

Style Matrix

- The **style** is defined as how correlated the activations are across different **channels**.



- The style matrix $\boldsymbol{G}^\ell \in \mathbb{R}^{n_c \times n_c}$ is defined by: $\boldsymbol{G}^\ell_{k\bar{k}} := \left\langle \boldsymbol{a}^k, \boldsymbol{a}^{\bar{k}} \right\rangle, \forall k, \bar{k} \in [n_c]$.



- Then the style cost is

$$\mathcal{L}_{\text{style}}(\boldsymbol{g}, \boldsymbol{s}) = \sum_\ell \frac{1}{n \times n \times n_c} \|\boldsymbol{G}^\ell(\boldsymbol{g}) - \boldsymbol{G}^\ell(\boldsymbol{s})\|^2$$