CENG 501
Deep Learning
Week 11

Sinan Kalkan
Types of Convolution:
Unshared convolution

• In some cases, sharing the weights do not make sense
  – When?

• Different parts of the input might require different types of features

• In such a case, we just have a network with local connectivity

• E.g., a face.
  – Features are not repeated across the space.
Types of Convolution:

Dilated (Atrous) Convolution

Purpose: Increase effective receptive field size without increasing parameters.

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

https://github.com/vdumoulin/conv_arithmetic
Types of Convolution: Transposed Convolution

Purpose: Increasing layer width+height (upsampling).

The size of the output:

- Regular convolution: \( O = \frac{W - F + 2 \times P}{S} + 1 \)
- Transpose convolution: \( W = (O - 1) \times S + F - 2 \times P \)

Figure: https://d2l.ai/chapter_computer-vision/transposed-conv.html
Types of Convolution: Transposed Convolution

https://github.com/vdumoulin/conv_arithmetic
Types of Convolution: 3D Convolution

Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: 1x1 Convolution

Purpose: Reduce number of channels.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: **Separable Convolution**

**Purpose:** Reduce number of parameters and multiplications.

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{bmatrix}
= \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix}
-1 & 0 & 1 \\
\end{bmatrix}
\]

![Diagram of separable convolution](https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215)
Types of Convolution:
Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:

Group Convolution

Purpose: Reduce number of parameters and multiplications.

Normal Convolution

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:

Deformable Convolution

Purpose: Flexible receptive field.

Figure 1: Illustration of the sampling locations in $3 \times 3$ standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

Figure 2: Illustration of $3 \times 3$ deformable convolution.

Types of Convolution:

Deformable Convolution


Bilinear interpolation for $x(p)$.

$$\mathcal{R} = \{(-1, -1), (-1, 0), \ldots, (0, 1), (1, 1)\}$$

defines a $3 \times 3$ kernel with dilation 1.

For each location $p_0$ on the output feature map $y$, we have

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n), \quad (1)$$

where $p_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta p_n | n = 1, \ldots, N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n + \Delta p_n). \quad (2)$$

Now, the sampling is on the irregular and offset locations $p_n + \Delta p_n$. As the offset $\Delta p_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$x(p) = \sum_{q} G(q, p) \cdot x(q), \quad (3)$$

where $p$ denotes an arbitrary (fractional) location ($p = p_0 + p_n + \Delta p_n$ for Eq. (2)), $q$ enumerates all integral spatial locations in the feature map $x$, and $G(\cdot , \cdot)$ is the bilinear interpolation kernel. Note that $G$ is two dimensional. It is separated into two one dimensional kernels as

$$G(q, p) = g(q_x, p_x) \cdot g(q_y, p_y), \quad (4)$$

where $g(a, b) = \max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(q, p)$ is non-zero only for a few $q$s.

In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta p_n$ is computed as

$$\frac{\partial y(p_0)}{\partial \Delta p_n} = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot \frac{\partial x(p_0 + p_n + \Delta p_n)}{\partial \Delta p_n}$$

$$= \sum_{p_n \in \mathcal{R}} \left[ w(p_n) \cdot \sum_{q} \frac{\partial G(q, p_0 + p_n + \Delta p_n)}{\partial \Delta p_n} x(q) \right], \quad (7)$$

where the term $\frac{\partial G(q, p_0 + p_n + \Delta p_n)}{\partial \Delta p_n}$ can be derived from Eq. (4). Note that the offset $\Delta p_n$ is 2D and we use $\partial \Delta p_n$ to denote $\partial \Delta p_n^x$ and $\partial \Delta p_n^y$ for simplicity.
Pooling

- **Example**
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged
- **Max pooling with F=3, S=2** or **F=2, S=2** are quite common.
  - Pooling with bigger receptive field sizes can be destructive
- **Avg pooling is an obsolete choice.** Max pooling is shown to work better in practice.


Sample diagrams and visual explanations are provided to illustrate the concepts.

Previously on CENG501!
Pooling

- Pooling provides invariance to small translation.

- If you pool over different convolution operators, you can gain invariance to different transformations.

Non-linearity

• Sigmoid
• Tanh
• ReLU and its variants
  – The common choice
  – Faster
  – Easier (in backpropagation etc.)
  – Avoids saturation issues
• ...

Previously on CENG501!
Normalization

https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Previously on CENG501!
Fully-connected layer

- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used
A Blueprint for CNNs

INPUT $\rightarrow$ [[CONV $\rightarrow$ RELU]$^N$$\rightarrow$ POOL?]$^M$$\rightarrow$ [FC $\rightarrow$ RELU]$^K$$\rightarrow$ FC

where the $^*$ indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$). For example, here are some common ConvNet architectures you may see that follow this pattern:

- **INPUT $\rightarrow$ FC**, implements a linear classifier. Here $N = M = K = 0$.
- **INPUT $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ FC**
- **INPUT $\rightarrow$ [CONV $\rightarrow$ RELU $\rightarrow$ POOL]$^2$$\rightarrow$ FC $\rightarrow$ RELU $\rightarrow$ FC**. Here we see that there is a single CONV layer between every POOL layer.
- **INPUT $\rightarrow$ [CONV $\rightarrow$ RELU $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ POOL]$^3$$\rightarrow$ [FC $\rightarrow$ RELU]$^2$$\rightarrow$ FC**. Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

http://cs231n.github.io/convolutional-networks/
Feed-forward through convolution

\[ a^l_i = \sigma(\text{net}^l_i) \]

\[ \text{net}^l_i = \sum_{j=1}^{F} w_j \cdot a^{l-1}_{i+j-1} \]

For example:

\[ \text{net}^1_1 = w_1 a^{l-1}_1 + w_2 a^{l-1}_2 + w_3 a^{l-1}_3 \]
Backpropagation through convolution

Previously on CENG501!

\[
\text{Feedforward:} \quad a_i^l = \sigma(\text{net}_i^l) \\
\text{net}_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j}^{l-1}
\]

Gradient wrt. weights:

\[
\frac{\partial L}{\partial w_k} = \frac{\partial L}{\partial a_1^l} \frac{\partial a_1^l}{\partial w_k} + \frac{\partial L}{\partial a_2^l} \frac{\partial a_2^l}{\partial w_k} + \ldots \\
= \sum_i \frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial w_k} \\
= \sum_i \frac{\partial L}{\partial a_i^l} \frac{\partial a_i^l}{\partial \text{net}_i^l} \frac{\partial \text{net}_i^l}{\partial w_k}
\]
Backpropagation through convolution

Feedforward:
\[ a^l_i = \sigma (\text{net}^l_i) \]
\[ \text{net}^l_i = \sum_{j=1}^{F} w_j \cdot a^{l-1}_{i+j} \]

Gradient wrt. input layer:
\[ \frac{\partial L}{\partial a^{l-1}_{3}} = ? \]
\[ = \frac{\partial L}{\partial a^l_1} \frac{\partial a^l_1}{\partial \text{net}^l_1} \text{net}^l_1 + \frac{\partial L}{\partial a^l_2} \frac{\partial a^l_2}{\partial \text{net}^l_2} \text{net}^l_2 \]
\[ + \frac{\partial L}{\partial a^l_2} \frac{\partial \text{net}^l_2}{\partial a^{l-1}_3} \text{net}^l_2 \]
\[ = \frac{\partial L}{\partial \text{net}^l_1} w_3 + \frac{\partial L}{\partial \text{net}^l_2} w_2 + \frac{\partial L}{\partial \text{net}^l_3} w_1 \]

In general:
\[ \frac{\partial L}{\partial a^{l-1}_{i}} = \sum_{j=1}^{F} \frac{\partial L}{\partial \text{net}^{l}_{i-j+1}} w_j \]

Previously on CENG501!
Feed-forward through pooling

\[ a_i^l = \max\{a_{i+j-1}^l\}^{F}_{j=1} \]

For example:

\[ net_1^l = \max\{a_1^{l-1}, a_2^{l-1}, a_3^{l-1}\} \]
Backpropagation through pooling

Feedforward:

\[ a_i^l = \max\{a_{i+j}^{l-1}\}_{j=1}^F \]

Using derivative of max:

\[
\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial \text{net}_k^l} \frac{\partial \text{net}_k^l}{\partial a_i^{l-1}} = \begin{cases} 
\frac{\partial L}{\partial \text{net}_k^l}, & a_i^{l-1} \text{ is max} \\
0, & \text{otherwise}
\end{cases}
\]

This requires that we save the index of the max activation (sometimes also called the switches) so that gradient “routing” is handled efficiently during backpropagation.
How to initialize the weights?

Option 1: randomly
- E.g. using He initialization (check Week 7 slides)
- This has been shown to work nicely in the literature

Option 2:
- Train/obtain the “filters” elsewhere and use them as the weights
- Unsupervised pre-training using image patches (windows)
- Avoids full feedforward and backward pass, allows the search to start from a better position
- You may even skip training the convolutional layers

Today

• Convolutional Neural Networks
  – Transfer learning
  – Visualizing and understanding CNNs
  – Widely-used CNN architectures
  – CNN Applications

<table>
<thead>
<tr>
<th>Week</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 11 (24 May)</td>
<td>CNNs</td>
</tr>
<tr>
<td>Week 12 (31 May)</td>
<td>RNNs</td>
</tr>
<tr>
<td>Week 13 (7 June)</td>
<td>RNNs</td>
</tr>
<tr>
<td>Week 14 (14 June)</td>
<td>RNNs + Attention</td>
</tr>
<tr>
<td>Week 15 (21 June)</td>
<td>Modern Models</td>
</tr>
</tbody>
</table>

These slides available at: [https://kovan.ceng.metu.edu.tr/~sinan/DL/week_11.pdf](https://kovan.ceng.metu.edu.tr/~sinan/DL/week_11.pdf)
Administrative Issues

- Programming assignment 1
- Take-Home Exam 1

- Office Hour:
  - Every Tuesday, 21:00

- Project paper selection
  - [https://docs.google.com/spreadsheets/d/1tzPHq_Vgu6gCwNyXJHGvqeA6pgU67H0nKYjqiSWfKc/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1tzPHq_Vgu6gCwNyXJHGvqeA6pgU67H0nKYjqiSWfKc/edit?usp=sharing)
  - Deadline: 19th of April
Trade-offs in architecture

• Between filter size and number of layers
  – Keep the layer widths fixed.
  – Deeper networks with smaller filter sizes perform better (if you keep the overall computational complexity fixed)

• Between layer width and number of layers
  – Keep the size of the filters fixed.
  – Increasing depth improves performance

• Between filter size and layer width
  – Keep the number of layers fixed.
  – No significant difference
Memory

Main sources of memory load:

• Activation maps:
  – Training: They need to be kept during training so that backpropagation can be performed
  – Testing: No need to keep the activations of earlier layers

• Parameters:
  – The weights, their gradients and also another copy if momentum is used

• Data:
  – The originals + their augmentations

• If all these don’t fit into memory,
  – Load your data batch by batch from disk
  – Decrease the size of your batches
Memory constraints

- Using smaller RFs with stacking means more memory since you need to store more activation maps.

- In such memory-scarce cases,
  - the first layer may use bigger RFs with S>1
  - information loss from the input volume may be less critical than the following layers.

- E.g., AlexNet uses RFs of 11x11 and S = 4 for the first layer.
TRANSFER LEARNING:
USING TRAINED CNN & FINE-TUNING
Using trained CNN

• Also called transfer learning
  – Rare to design and train a CNN from scratch!

• Take a trained CNN, e.g., AlexNet
  – Use a trained CNN as a feature detector:
    • Remove the last fully-connected layer
    • The activations of the remaining layer are called CNN codes
    • This yields a 4096 dimensional feature vector for AlexNet
    • Now, add a fully-connected layer for your problem and train a linear classifier on your dataset.
  – Alternatively, fine-tune the whole network with your new layer and outputs
    • You may limit updating only to the last layers because earlier layers are generic, and quite dataset independent

• Pre-trained CNNs
Finetuning

1. If the new dataset is **small** and **similar** to the original dataset used to train the CNN:
   – Finetuning the whole network may lead to overfitting
   – Just train the newly added layer

2. If the new dataset is **big** and **similar** to the original dataset:
   – The more, the merrier: go ahead and **train the whole network**

3. If the new dataset is **small** and **different** from the original dataset:
   – Not a good idea to train the whole network
   – However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   – Add your layer to earlier parts of the network

4. If the new dataset is **big** and **different** from the original dataset:
   – We can “finetune” the whole network
   – This amounts to a new training problem by initializing the weights with those of another network
More on finetuning

• You cannot change the architecture of the trained network (e.g., remove layers) arbitrarily

• The sizes of the layers can be varied
  – For convolution & pooling layers, this is straightforward
  – For the fully-connected layers: you can convert the fully-connected layers to convolution layers, which makes it size-independent.

• You should use small learning rates while fine-tuning
See also:


How transferable are features in deep neural networks?

Jason Yosinski,1 Jeff Clune,2 Yoshua Bengio,3 and Hod Lipson4
1 Dept. Computer Science, Cornell University
2 Dept. Computer Science, University of Wyoming
3 Dept. Computer Science & Operations Research, University of Montreal
4 Dept. Mechanical & Aerospace Engineering, Cornell University
VISUALIZING AND UNDERSTANDING CNNS
Many different mechanisms

• Visualize layer activations
• Visualize the weights (i.e., filters)
• Visualize examples that maximally activate a neuron
• Visualize a 2D embedding of the inputs based on their CNN codes
• Occlude parts of the window and see how the prediction is affected
• Data gradients
Visualize activations during training

• Activations are dense at the beginning.
  – They should get sparser during training.
• If some activation maps are all zero for many inputs, dying neuron problem => high learning rate in the case of ReLUs.

Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black) and mostly local.

http://cs231n.github.io/convolutional-networks/
Visualize the weights

- We can directly look at the filters of all layers
- First layer is easier to interpret
- Filters shouldn’t look noisy

http://cs231n.github.io/convolutional-networks/
Visualize the inputs that maximally activate a neuron

- Keep track of which images activate a neuron most

Maximally activating images for some POOL5 (5th pool layer) neurons of an AlexNet. The activation values and the receptive field of the particular neuron are shown in white. (In particular, note that the POOL5 neurons are a function of a relatively large portion of the input image!) It can be seen that some neurons are responsive to upper bodies, text, or specular highlights.
Embed the codes in a lower-dimensional space

- Place images into a 2D space such that images which produce similar CNN codes are placed close.
- You can use, e.g., t-Distributed Stochastic Neighbor Embedding (t-SNE)

![t-SNE embedding of a set of images based on their CNN codes. Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced the associated code, and more related visualizations at different scales refer to t-SNE visualization of CNN codes.](image1)

Figure 1: Illustration of t-SNE on MNIST dataset

Figure: Laurens van der Maaten and Geoffrey Hinton
Occlude parts of the image

- Slide an “occlusion window” over the image
- For each occluded image, determine the class prediction confidence/probability.

Three input images (top). Notice that the occluder region is shown in grey. As we slide the occluder over the image we record the probability of the correct class and then visualize it as a heatmap (shown below each image). For instance, in the left-most image we see that the probability of Pomeranian plummets when the occluder covers the face of the dog, giving us some level of confidence that the dog’s face is primarily responsible for the high classification score. Conversely, zeroing out other parts of the image is seen to have relatively negligible impact.

http://cs231n.github.io/convolutional-networks/
Data gradients

- Generate an image that maximizes the class score.

More formally, let $S_c(I)$ be the score of the class $c$, computed by the classification layer of the ConvNet for an image $I$. We would like to find an $L_2$-regularised image, such that the score $S_c$ is high:

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2,$$

where $\lambda$ is the regularisation parameter. A locally-optimal $I$ can be found by the back-propagation

- Use: Gradient ascent!

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Figure 1: Numerically computed images, illustrating the class appearance models, learnt by a ConvNet, trained on ILSVRC-2013. Note how different aspects of class appearance are captured in a single image. Better viewed in colour.
The gradient with respect to the input is high for pixels which are on the object.

Consider the linear score model for the class $c$:

$$S_c(I) = w_c^T I + b_c$$

(2)

where the image $I$ is represented in the vectorised (one-dimensional) form, and $w_c$ and $b_c$ are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of $w$ defines the importance of the corresponding pixels of $I$ for the class $c$.

In the case of deep ConvNets, the class score $S_c(I)$ is a highly non-linear function of $I$, so the reasoning of the previous paragraph can not be immediately applied. However, given an image $I_0$, we can approximate $S_c(I)$ with a linear function in the neighbourhood of $I_0$ by computing the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b,$$

(3)

where $w$ is the derivative of $S_c$ with respect to the image $I$ at the point (image) $I_0$:

$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}.$$ 

(4)
Alternative to FC: Global Average Pooling

• We have $k$ feature maps: $f_1, \ldots, f_n$.

• Global average pooling is then:

$$F^k = \sum_{x,y} f_k(x, y)$$

• Classification scores are obtained by:

$$S_c = \sum_k w^c_k F_k$$


• Advantages:
  – No parameters, hence significant improvement in terms of overfitting problem.
  – Forces the feature maps to capture confidence maps.
  – It is more suitable to the nature of CNNs.
  – Provides invariance to spatial transformations.
Class Activation Maps

- Weighted combination of the feature maps before GAP:

\[ M(x, y) = \sum_k w_k f_k(x, y) \]

Class Activation Maps

- GradCAM:

\[ \alpha^c_k = \sum_{x, y} \frac{\partial S_c}{\partial f_k} \]

\[ M^c(x, y) = ReLU \left( \sum_k \alpha^c_k f_k(x, y) \right) \]


Figure: https://pypi.org/project/grad-cam/
Feature inversion

• Learns to reconstruct an image from its representation

This section introduces our method to compute an approximate inverse of an image representation. This is formulated as the problem of finding an image whose representation best matches the one given \[ [34] \]. Formally, given a representation function \( \Phi : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{d} \) and a representation \( \Phi_0 = \Phi(x_0) \) to be inverted, reconstruction finds the image \( x \in \mathbb{R}^{H \times W \times C} \) that minimizes the objective:

\[
x^* = \arg\min_{x \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(x), \Phi_0) + \lambda \mathcal{R}(x)
\]

where the loss \( \ell \) compares the image representation \( \Phi(x) \) to the target one \( \Phi_0 \) and \( \mathcal{R} : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R} \) is a regulariser capturing a natural image prior.

• Regularization term here is the key factor, e.g. a combination of the two terms:

\[
\mathcal{R}_\alpha(x) = \|x\|_\alpha, \quad \mathcal{R}_{\nu^\beta}(x) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\beta/2}
\]

Figure 1. What is encoded by a CNN? The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN\[13\] (before the softmax is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen.
Feature inversion with perceptual losses

Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.
Visualization distill.pub

https://distill.pub/2017/feature-visualization/

https://distill.pub/2018/building-blocks/
Fooling ConvNets

- Given an image $I$ labeled as $l_1$, find minimum "$r$" (noise) such that $I + r$ is classified as a different label, $l_2$.
- I.e., minimize:
  \[
  \arg \min_r \text{loss}(I + r, l_2) + c|r|
  \]

```
x
  “panda”
  57.7% confidence

+ .007 ×

sign(∇ₓ J(θ, x, y))
  “nematode”
  8.2% confidence

= x

x + csign(∇ₓ J(θ, x, y))
  “gibbon”
  99.3% confidence
```

---

**EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES**

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow, shlens, szegedy}@google.com

---

Intriguing properties of neural networks

<table>
<thead>
<tr>
<th>Christian Szegedy</th>
<th>Wojciech Zaremba</th>
<th>Ilya Sutskever</th>
<th>Juan Illarraza</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Inc.</td>
<td>New York University</td>
<td>Google Inc.</td>
<td>New York University</td>
</tr>
</tbody>
</table>

---

![Ostrich image]

Spring 2021
Sinan Kalkan

---

56
More on adversarial examples

• How to classify adversarial examples?
  – You need to train your network against them!
  – That is very expensive and training against all kinds of adversarial examples is not possible
  – However, training against adversarial examples increases accuracy on non-adversarial examples as well.

• They are still an unsolved issue in neural networks

• Adversarial examples are problems of any learning method

• See I. Goodfellow for more on adversarial examples:
• “We provide a new understanding of the fundamental nature of adversarially robust classifiers and how they differ from standard models. In particular, we show that there provably exists a trade-off between the standard accuracy of a model and its robustness to adversarial perturbations. We demonstrate an intriguing phenomenon at the root of this tension: a certain dichotomy between “robust” and “non-robust” features. We show that while robustness comes at a price, it also has some surprising benefits. Robust models turn out to have interpretable gradients and feature representations that align unusually well with salient data characteristics. In fact, they yield striking feature interpolations that have thus far been possible to obtain only using generative models such as GANs.”