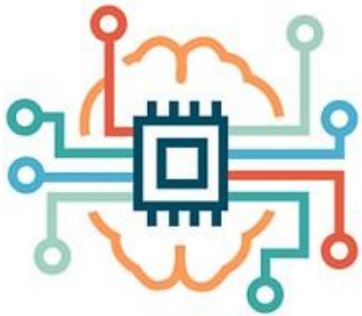
COMP1804
Applied Machine Learning

Lecture 8: Convolutional Neural Network, Part 2

Dr. Dimitrios Kollias



Outline

- A bit of History: DNNs did not dominate
- ImageNet Dataset & Challenge
- AlexNet & VGG Networks
- Other State-of-the-Art DNNs
- Transfer Learning and Fine-Tuning
- Image Augmentation Techniques
- Performance Measures
- Loss Functions

A bit of History: DNNs did not dominate (1)

Although CNNs were well known, they did not immediately dominate the field.
They were often surpassed by other machine learning methods.

Typical computer vision pipelines consisted of manually engineering feature extraction pipelines.
Rather than *learn the features*, the features were *crafted*.
Most of the progress came from having clever ideas for features,
and the learning algorithm was often relegated to an afterthought.

A bit of History: DNNs did not dominate (2)

Thus, rather than training *end-to-end* (pixel to classification) systems, classical pipelines looked more like this:

- Obtain an interesting dataset. In early days, these datasets required expensive sensors (at the time, 1 megapixel images were state-of-the-art).

- Preprocess the dataset with hand-crafted features based on some knowledge of optics, geometry, other analytic tools.

- Feed the data through a standard set of feature extractors such as the SIFT (scale-invariant feature transform), the SURF (speeded up robust features)

- Dump the resulting representations into your favourite classifier and train it.

A bit of History: DNNs did not dominate (3)

- **Problem with this pipeline:**

feature extraction cannot be tweaked according to the classes and images.
So if the chosen feature lacks the representation required to distinguish the categories, the accuracy of the classification model suffers a lot, irrespective of the type of classification strategy employed.

- Common theme:

pick multiple feature extractors and club them inventively to get a better feature.
But this involves **too many heuristics as well as manual labor to tweak parameters according to the domain to reach a decent level of accuracy**.

A bit of History: DNNs did not dominate (4)

- **Another problem:**

it is completely different from how we humans learn to recognize things.
Just after birth, a child is incapable of perceiving his surroundings, but as he progresses and processes data, he learns to identify things.
This is the philosophy behind deep learning, wherein no hard-coded feature extractor is built in.
It combines the extraction and classification modules into one integrated system and it learns to extract, by discriminating representations from the images and classify them based on supervised data.

A bit of History: DNNs did not dominate (5)

- Deep models with many layers → large amounts of data to significantly outperform traditional methods

- However: limited storage capacity of computers, relative expense of sensors and comparatively tighter research budgets in the 1990s, most research relied on tiny datasets, many of which contained only hundreds or (a few) thousands of images captured in unnatural settings with low resolution.

ImageNet Dataset (1)

- In 2009, the ImageNet dataset was released, which is a large annotated photographs' dataset for computer vision research

- Goal: resource for promoting research and development of improved methods for computer vision

- There are about 14 million images in the dataset, about 21 thousand groups/classes, about 1.2 million images with bounding box annotations (e.g. boxes around identified objects in the images)

- The photographs were annotated by humans using crowdsourcing platforms such as Amazon's Mechanical Turk

ImageNet Dataset (2)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- annual competition held between 2010 and 2017
- goal: both promote the development of better computer vision techniques and to benchmark the state of the art
- challenge tasks use subsets (approximately 1.2 million images) of the ImageNet dataset for:

i) "*image classification*": assigning a class label to each image based on the main object in the photograph (among 1,000 object classes)

ii) "*object detection & localisation*": localizing the objects within each photograph / image classification + draw a bounding box around each object present

AlexNet: Winner of ImageNet

In 2012, AlexNet, an 8-layer CNN, won the ImageNet Challenge by a phenomenally large margin (error 15.3%; the runner up was not a deep learning method: error 26.2%; prior competitors' error was 25.7% and 28.2%).

This network showed, for the first time, that the features obtained by learning can transcend manually-designed features, breaking the previous paradigm in computer vision.

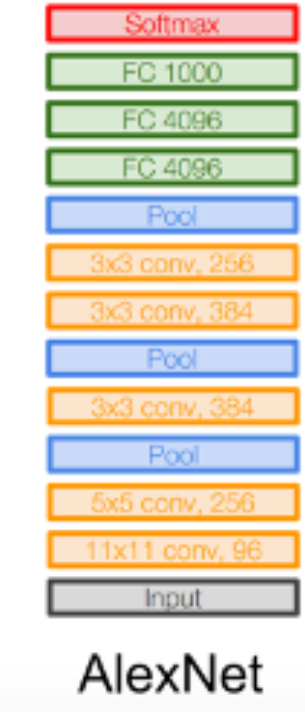
AlexNet: Architecture (1)

- AlexNet was named after Alex Krizhevsky, the first author of the paper that introduced it

- It consists of eight layers: five convolutional layers, two fully-connected hidden layers, and one fully-connected output layer

- AlexNet used the ReLU instead of the sigmoid as its activation function

- Successful attempt to learn hierarchical representations of visual data

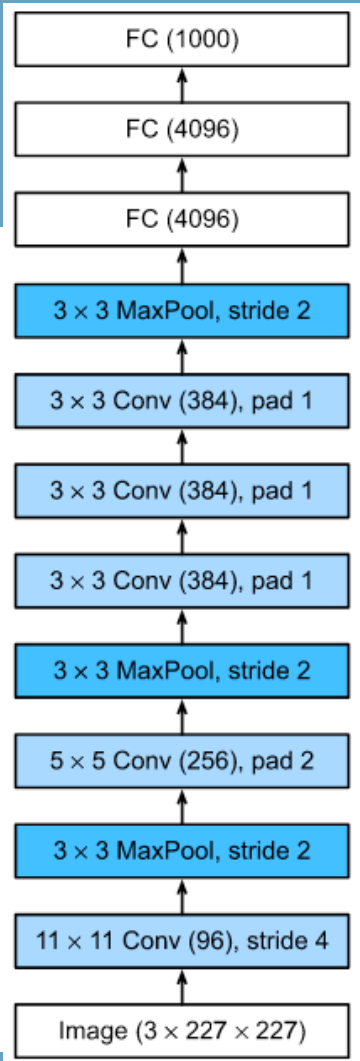


AlexNet: Architecture (2)

- In first layer, the filter is 11×11.
Since most images in ImageNet are more than 10 times higher & wider than the MNIST ones, objects in ImageNet tend to occupy more pixels → a larger filter is needed to capture the object

- The filter size in the second layer is reduced to 5×5, followed by 3×3

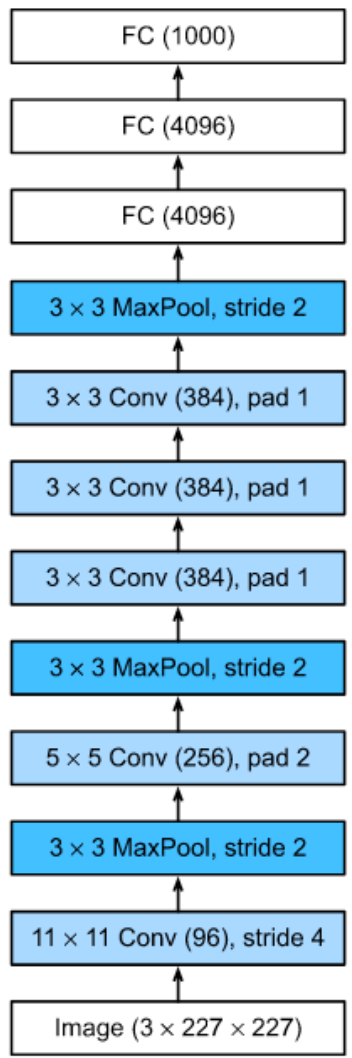
- After the first, second, and fifth conv layers, the network adds max pooling layers with a window shape of 3×3 and a stride of 2



AlexNet: Architecture (3)

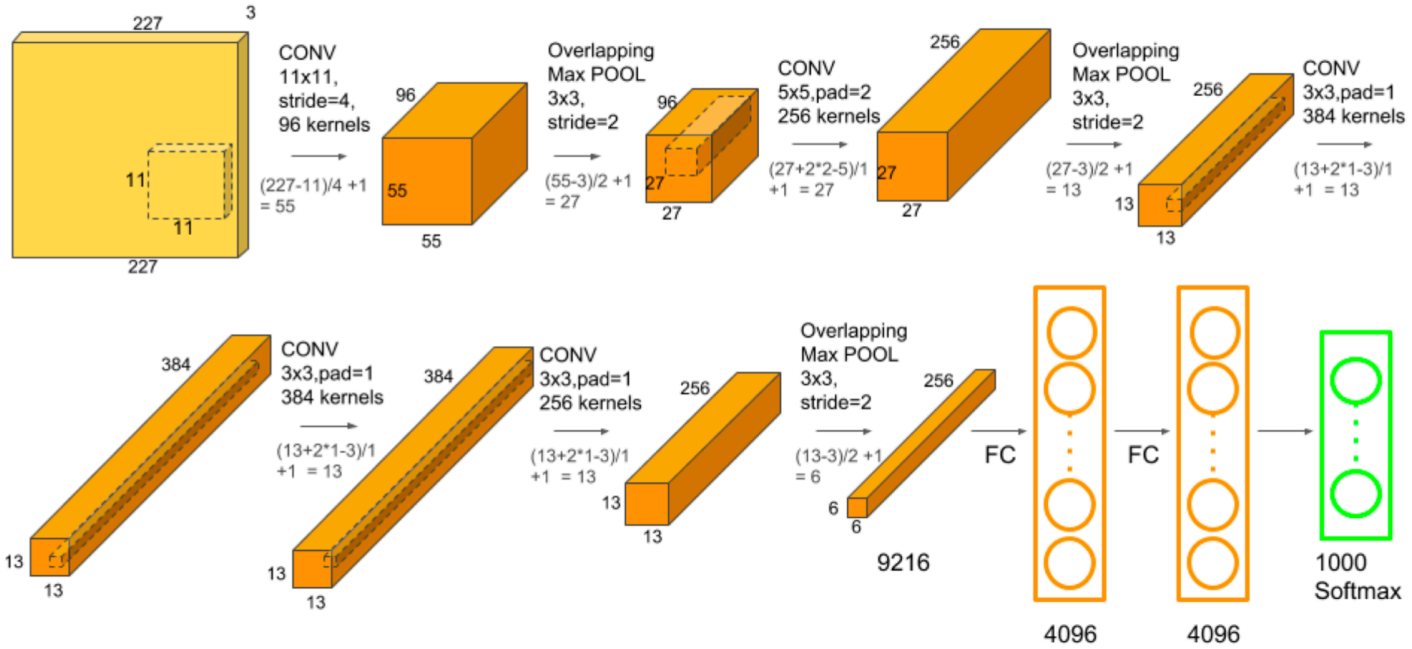
- AlexNet changed the sigmoid activation function to a simpler ReLU activation function

- After the last conv layer there are two fully-connected (fc) layers with 4096 outputs. These two huge fc layers produce model parameters of nearly 1 GB.



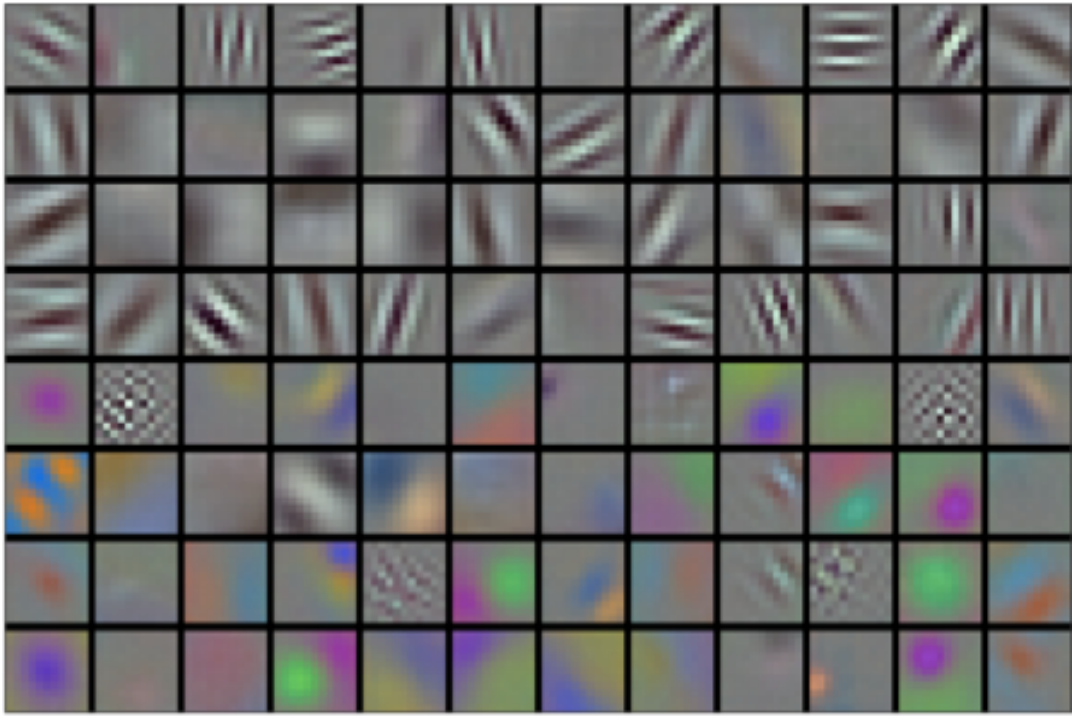
	Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters
1	3* 227 * 227					
2	Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944
3	96 * 55 * 55					
4	Max Pooling	3 * 3		2		
5	96 * 27 * 27					
6	Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656
7	256 * 27 * 27					
8	Max Pooling	3 * 3		2		
9	256 * 13 * 13					
10	Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120
11	384 * 13 * 13					
12	Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488
13	384 * 13 * 13					
14	Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992
15	256 * 13 * 13					
16	Max Pooling	3 * 3		2		
17	256 * 6 * 6					
18	FC6 + Relu					256 * 6 * 6 * 4096=37748736
19	4096					
20	FC7 + Relu					4096 * 4096=16777216
21	4096					
22	FC8 + Relu					4096 * 1000=4096000
23	1000 classes					
24	Overall					62369152=62.3 million
25	Conv VS FC					Conv:3.7million (6%) , FC: 58.6 million (94%)

AlexNet: Architecture (4)



AlexNet: Features (1)

In the lowest layers of the network, the model learned feature extractors that resembled some traditional filters.



AlexNet: Features (2)

Higher layers in the network might build upon these representations to represent larger structures, like eyes, noses, blades of grass, and so on.

Even higher layers might represent whole objects like people, airplanes, dogs, or frisbees.

Ultimately, the final hidden state learns a compact representation of the image that summarizes its contents such that data belonging to different categories are separated easily.

VGG Net (1)

While AlexNet offered empirical evidence that deep CNNs can achieve good results, it did not provide a general template to guide subsequent researchers in designing new networks.

The idea of using blocks first emerged from the Visual Geometry Group (VGG) at Oxford University, in their eponymously-named *VGG* network.

VGG Net (2)

- was the first runner-up in image classification and the winner in object localisation at the ILSVRC 2014

- achieved error 7.3% (vs 15.3% of AlexNet)

- has 16 or 19 layers and is deeper than AlexNet (8 layers)

- however, VGG consists of 138 million parameters (AlexNet consists of 61 million parameters)

VGG Net:
Why is it Important

It is the first year that there are deep learning models obtaining the error rate under 10%.

The most important is that there are many other models built on top of VGGNet or based on the 3×3 conv idea of VGGNet for other purposes or other domains.

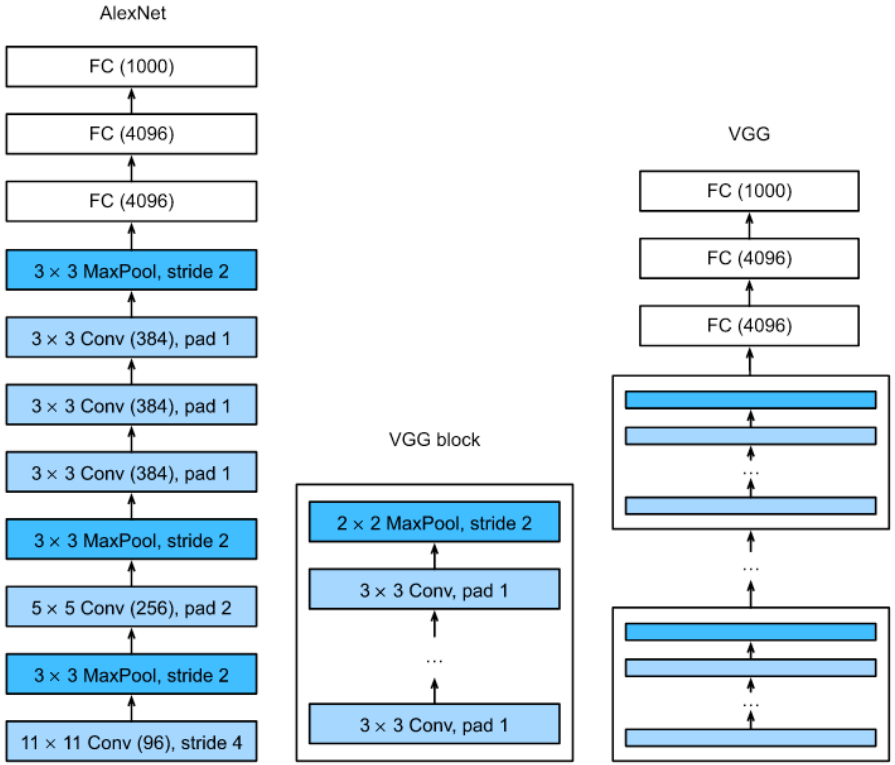
VGG Net:
VGG Block (1)

Basic building block of classic CNNs is a sequence of:
(i) a conv layer with padding to maintain the resolution
(ii) a nonlinearity such as a ReLU
(iii) a pooling layer such as a max pooling

One VGG block consists of a sequence of conv layers, followed by a max pooling layer for spatial downsampling. Convolutions with 3×3 kernels with padding of 1 (keeping height and width) and 2×2 max pooling with stride of 2 (halving the resolution after each block) are used

Like AlexNet, the VGG Network can be partitioned into two parts:
the first consisting mostly of conv and pooling layers;
the second of fully-connected ones

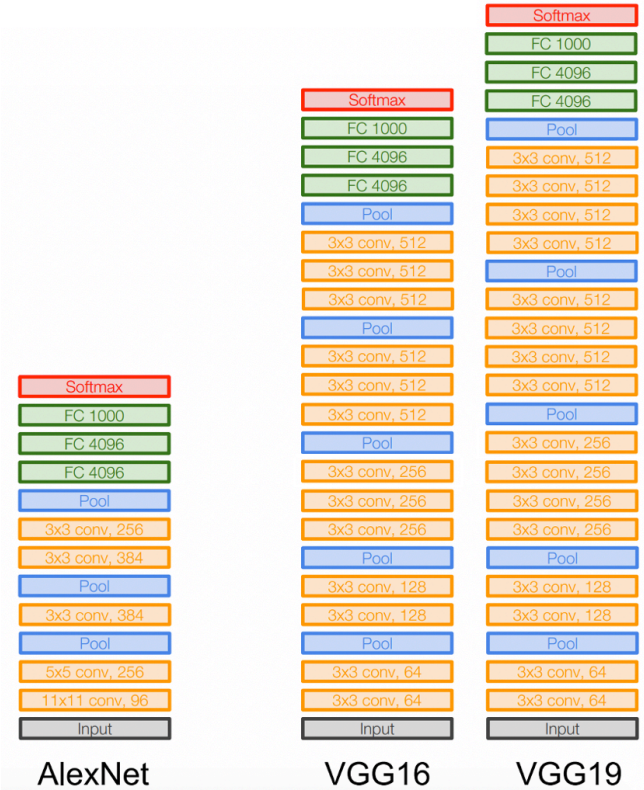
VGG Net:
VGG Block (2)



The conv part of the network connects several VGG blocks.

The fully-connected part of the VGG network is identical to that covered in AlexNet.

VGG Net: Architecture



VGG Net:
Comparison to AlexNet (1)

Smaller filters, Deeper networks, Uniform architecture:
all convolutional layers are the same:

- 8 layers (AlexNet) -> 16 - 19 layers (VGG16Net)
Only 3x3 conv stride 1, pad 1 and 2x2 max pool stride 2

- VGG: very small receptive fields (3x3 with a stride of 1);
AlexNet: large receptive fields (11x11 with a stride of 4)
3x3 kernels help in retaining finer level properties of the image
Fewer parameters (27 times the number of channels instead of AlexNet's 49 times the number of channels)

VGG Net: Comparison to AlexNet (2)

Fewer parameters:

Suppose there is only 1 filter per layer, 1 layer at input, and exclude the bias:
1 layer of 11×11 filter, number of parameters = 11×11=121
5 layer of 3×3 filter, number of parameters = 3×3×5=45

1 layer of 7×7 filter, number of parameters = 7×7=49
3 layers of 3×3 filters, number of parameters = 3×3×3=27

With fewer parameters to be learnt, it is better for faster convergence, and reduced overfitting problem.

VGG Net: Comparison to AlexNet (3)

With a given receptive field, multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increases the depth of the network which enables it to learn more complex features, and that at a lower cost.

Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer
For instance for input 9x9:

- If we stack thee 3x3 conv layers:
  the first layer will have output feature map of size: 7x7 (= [9 – 3 + 2*0] / 1 + 1)
  the second layer will have output feature map of size: 5x5 (= [7 – 3 + 2*0] / 1 + 1)
  the third layer will have output feature map of size: 3x3 (= [5 – 3 + 2*0] / 1 + 1)

- If we stack one 7x7 conv layer, the output feature map will have size:
  3x3 (= [9 – 7 + 2*0 / 1 + 1])

VGG Net:
Number of Parameters

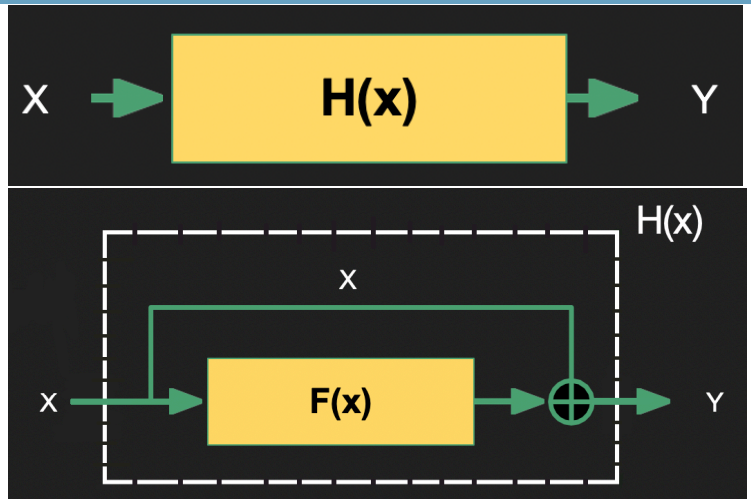
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 2)	8194

Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0

VGG Net

It has been shown that the features of the first or second fully connected layer of VGG, generalize well to other tasks

Other State-of-the-Art DNNs: ResNet & Residual Learning



H(x) is the true mapping function we want to learn

New representation F(x)

$$F(x) := H(x) - x$$

Residual Learning

If F(x) = 0 → identity mapping; if that is a solution the network will be able to find it

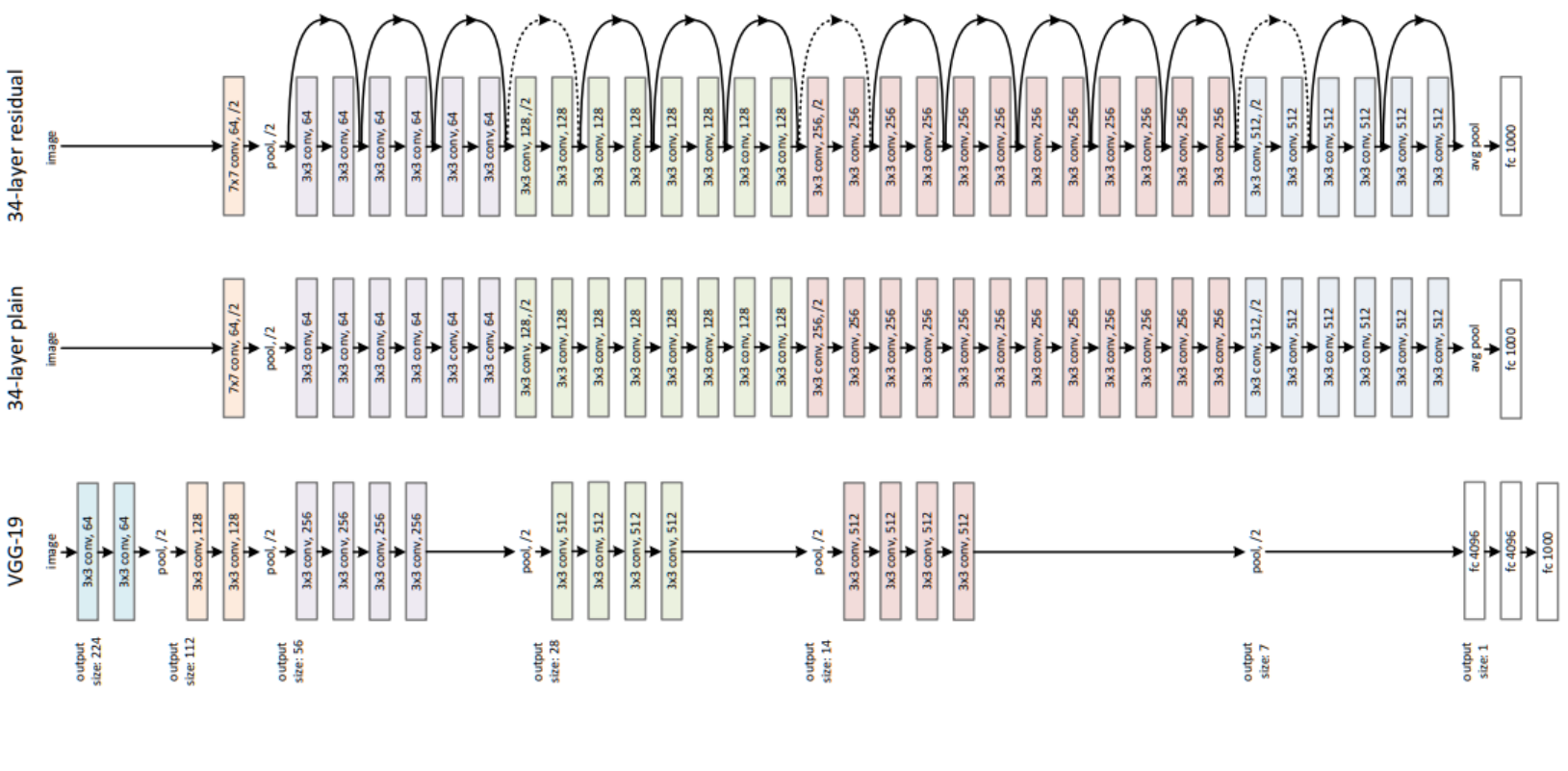
Other State-of-the-Art DNNs: Comparison of ResNet to VGG (1)

Difference:
No pooling layers; downsampling is performed by conv layers with stride = 2.

ResNet-34 has:

- fewer filters
- lower complexity than VGG:
  19.6 billion FLOPs (multiply-adds) in VGG19
  3.6 billion FLOPs in ResNet-34

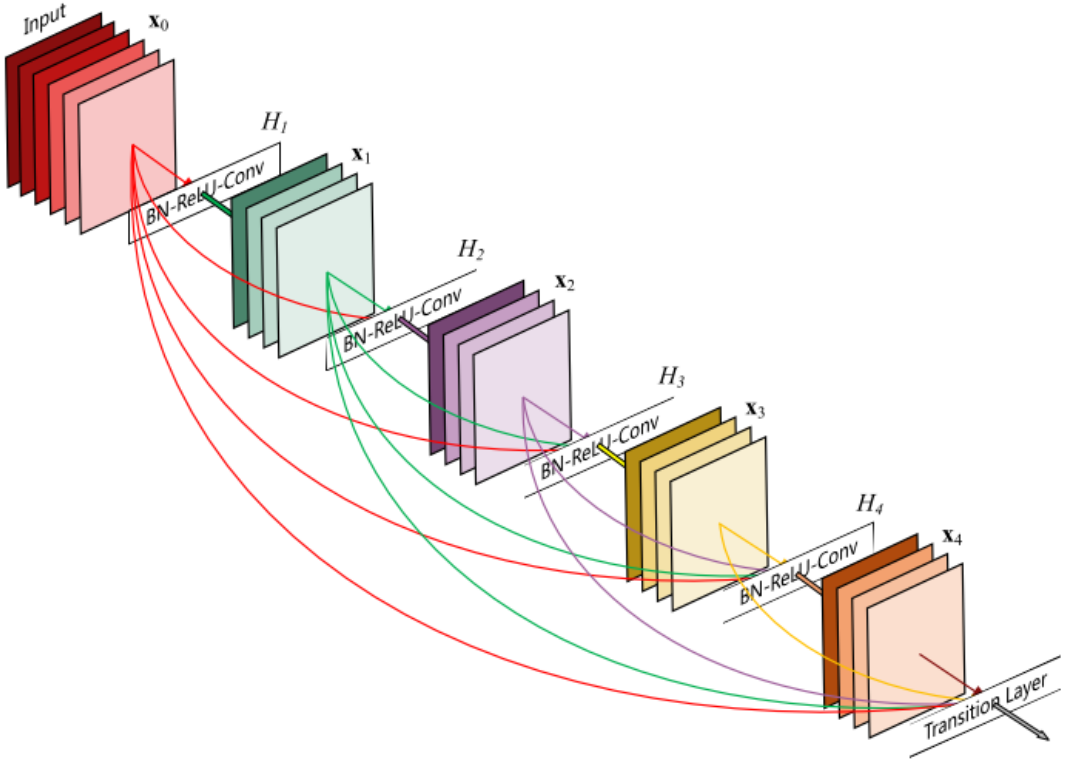
Other State-of-the-Art DNNs: Comparison of ResNet to VGG (2)



Other State-of-the-Art DNNs: Densely Connected CNN: DenseNet (1)

DenseNet-121/169/201:
It connects all layers directly with each other.
In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer.

Other State-of-the-Art DNNs: Densely Connected CNN: DenseNet (2)



Other State-of-the-Art DNNs

- ResNeXt
- NasNet
- EfficientNet

Transfer Learning (1)

If a dataset is small, the developed network may overfit.

- An obvious solution is to collect more data.
However, collecting and labelling data can consume a lot of time and money.

- Another solution is to apply transfer learning to migrate the knowledge learned from the source dataset to the target dataset.
For example, although the images in ImageNet are mostly unrelated to chairs, models trained on this dataset can extract more general image features that can help identify edges, textures, shapes, and object composition.
These similar features may be equally effective for recognizing a chair.

Transfer Learning (2)

What is Transfer Learning?

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. It refers to the situation where what has been learned in one setting is exploited to improve generalization in another setting

Transfer learning is an optimization that allows rapid progress or improved performance when modelling the second task.

Transfer Learning (3)

Nevertheless, transfer learning is popular in deep learning given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained.
Transfer learning only works in deep learning if the model features learned from the first task are general.

Fine Tuning (1)

A common technique in transfer learning: fine tuning;
it consists of the following four steps:

1. Pre-train a neural network model, i.e., the source model, on a source dataset (e.g., the ImageNet dataset).
2. Create a new neural network model, i.e., the target model.

This replicates all model designs and their parameters on the source model, except the output layer.

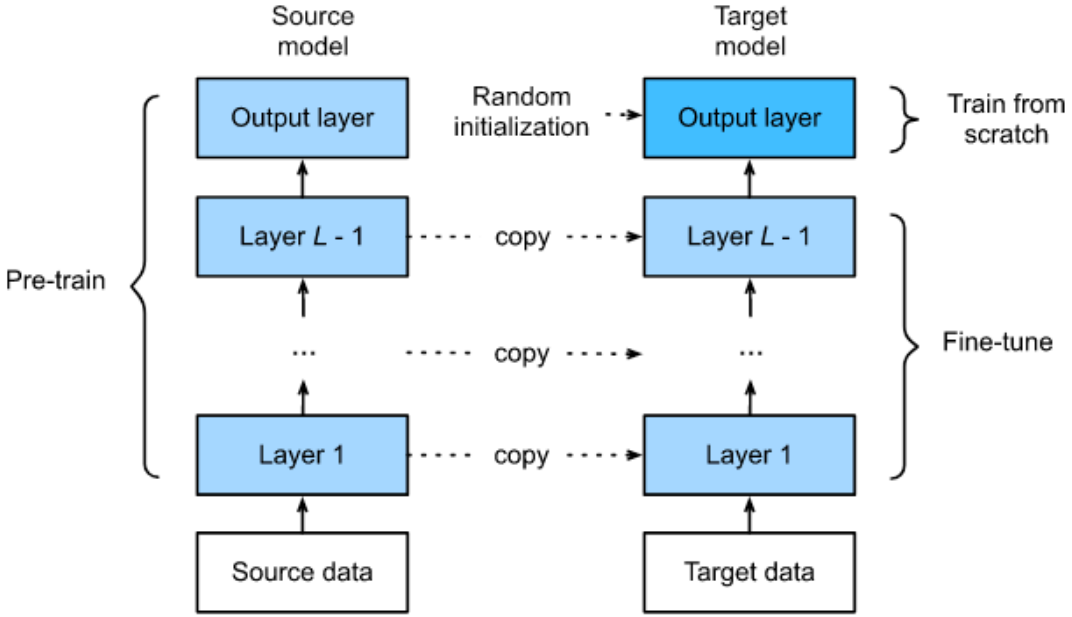
We assume that these model parameters contain the knowledge learned from the source dataset and this knowledge will be equally applicable to the target dataset. We also assume that the output layer of the source model is closely related to the labels of the source dataset and is therefore not used in the target model.

Fine Tuning (2)

3. Add an output layer whose output size is the number of target dataset categories to the target model, and randomly initialize the model parameters of this layer.

4. Train the target model on a target dataset, such as a chair dataset. We will train the output layer from scratch, while the parameters of all remaining layers are fine-tuned based on the parameters of the source model.

Fine Tuning (3)



Generally, fine tuning parameters use a smaller learning rate, while training the output layer from scratch can use a larger learning rate.

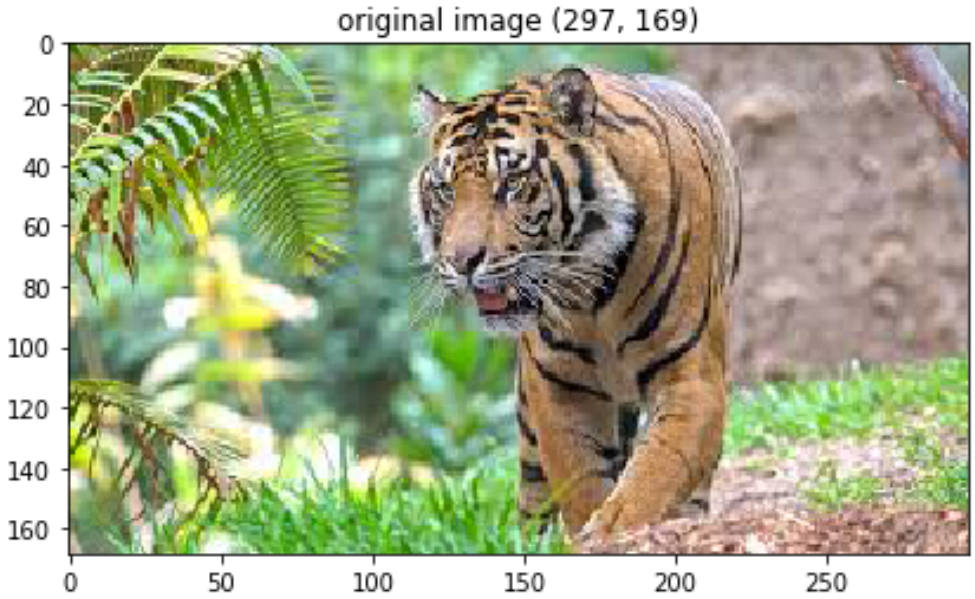
Image Augmentation Techniques

Data augmentation is the technique of increasing the size of data used for training a model.
For reliable predictions, the deep learning models often require a lot of training data, which is not always available.
Therefore, the existing data is augmented in order to make a better generalized model.
Although data augmentation can be applied in various domains, it's commonly used in computer vision.

Some of the most common data augmentation techniques used for images are:

Image Augmentation Techniques: Scaling

Scaling is a linear transformation that enlarges or shrinks objects by a scale factor that is the same in all directions.



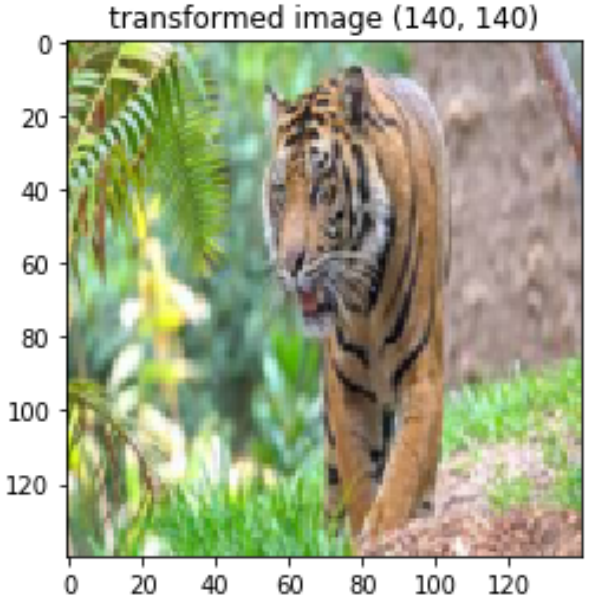
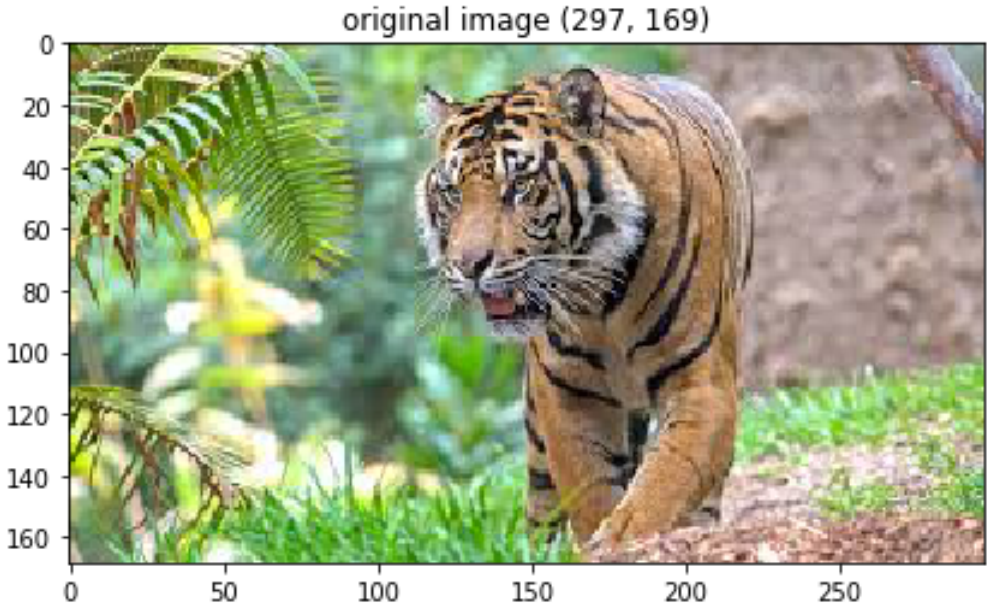


Image Augmentation Techniques: Cropping

In cropping, a portion of the image is selected
e.g. in the given example the center cropped image is returned



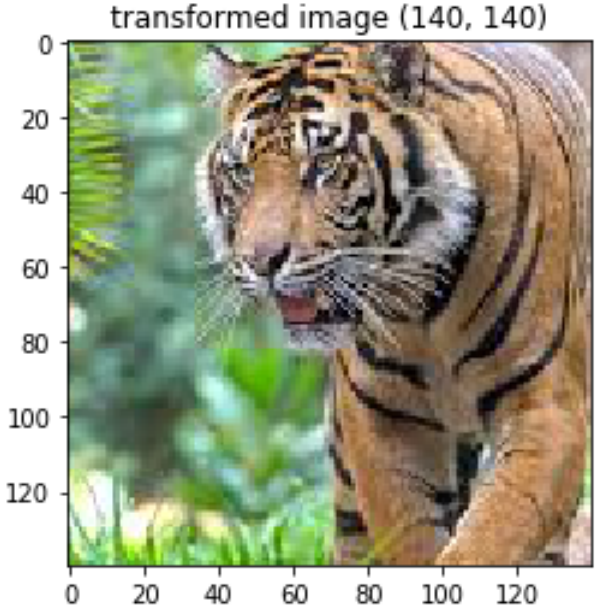
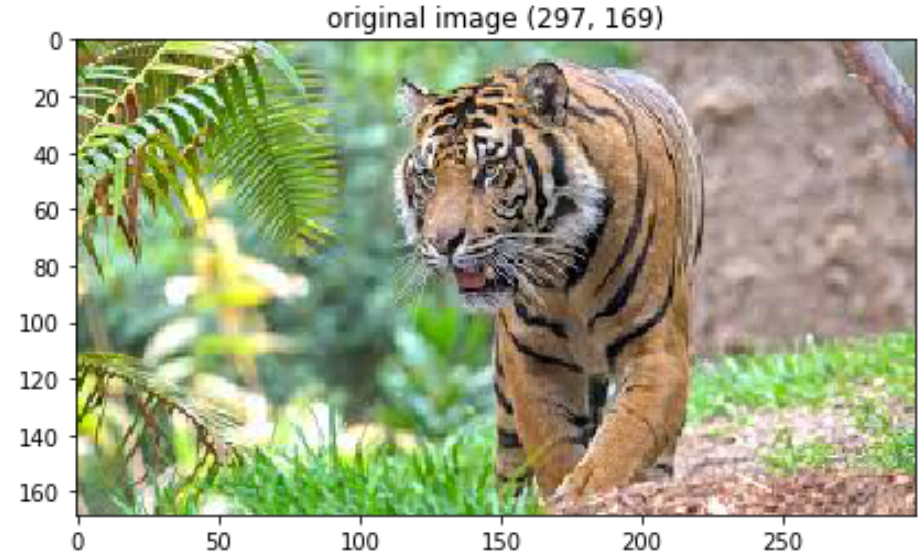


Image Augmentation Techniques: Padding

In padding, the image is padded with a given value on all sides.



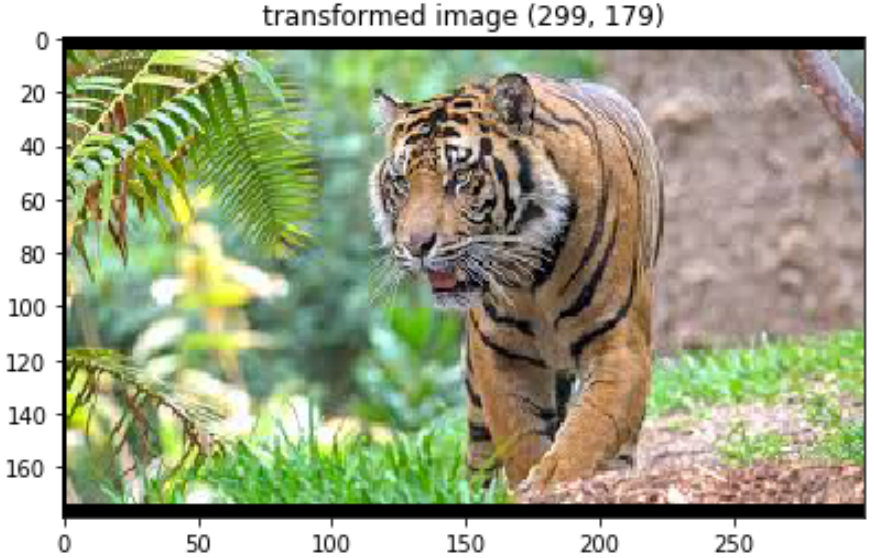
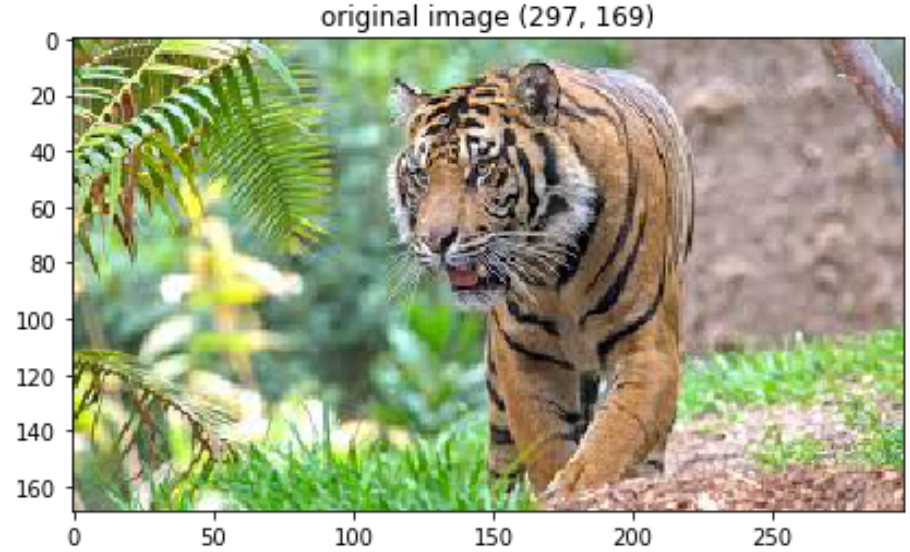


Image Augmentation Techniques: Rotation

Rotation is a circular transformation around a point or an axis.



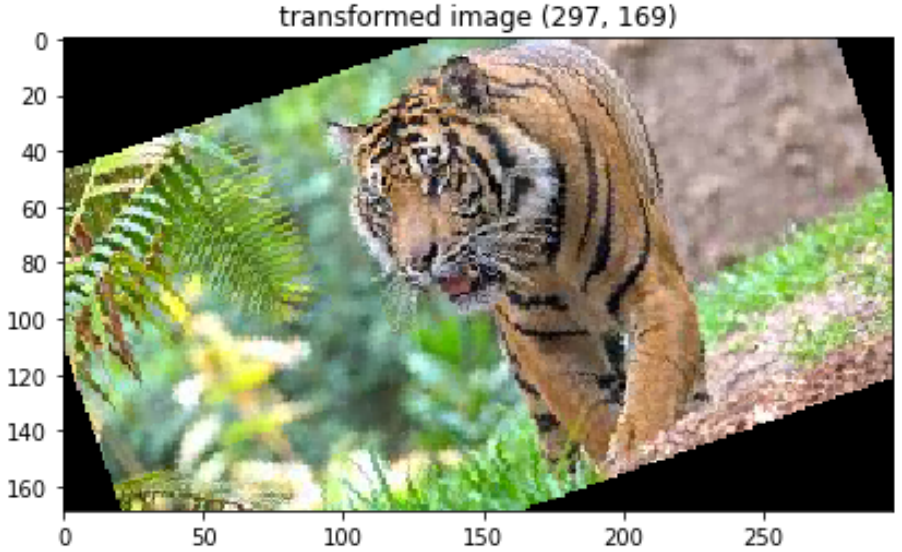
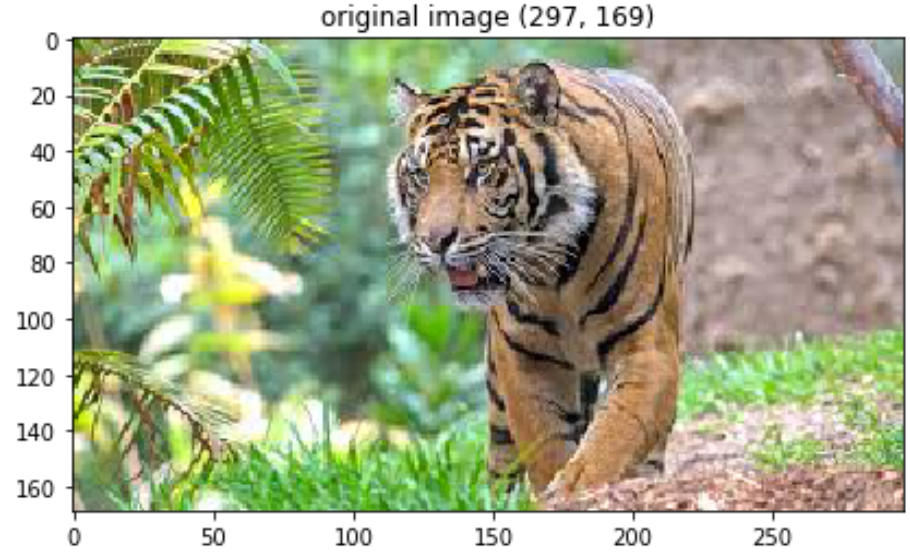


Image Augmentation Techniques: Translation

A translation is a function that moves every point with a constant distance in a specified direction.
In translation, the image is moved either along the x-axis or y-axis.



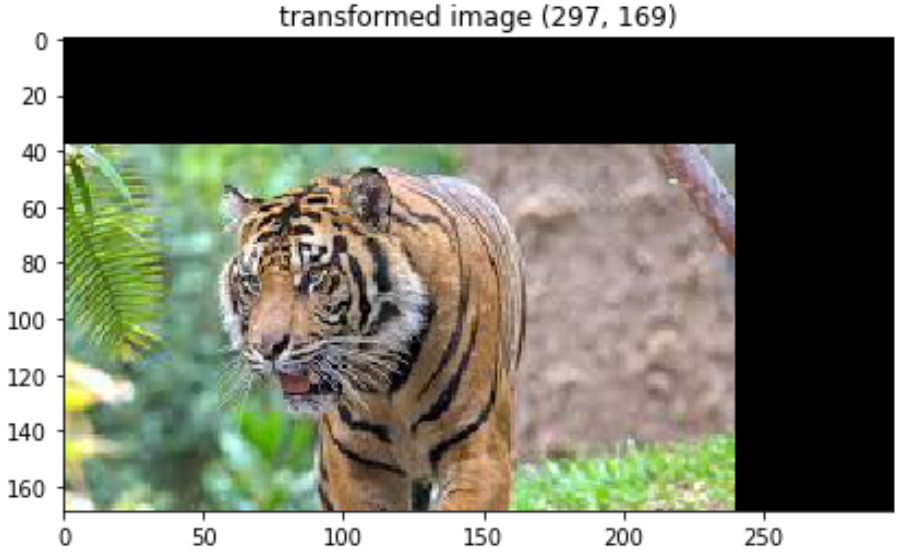


Image Augmentation Techniques: Color Augmentation (1)

Color augmentation or color jittering deals with altering the color properties of an image by changing its pixel values:

Brightness

One way to augment is to change the brightness of the image.

The resultant image becomes darker or lighter compared to the original one.

Contrast

The contrast is defined as the degree of separation between the darkest and brightest areas of an image.

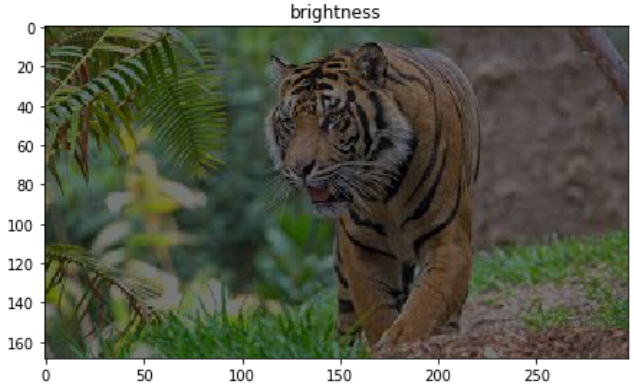
Saturation

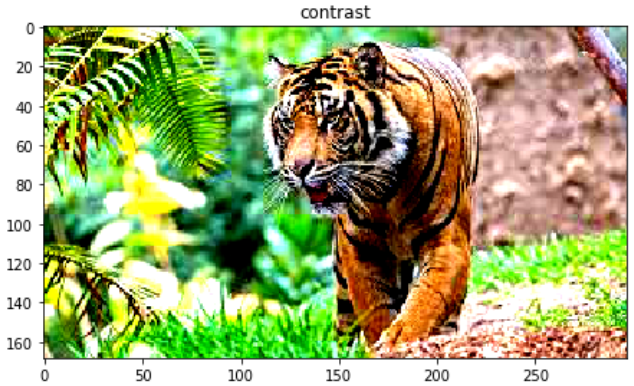
Saturation is the separation between colors of an image.

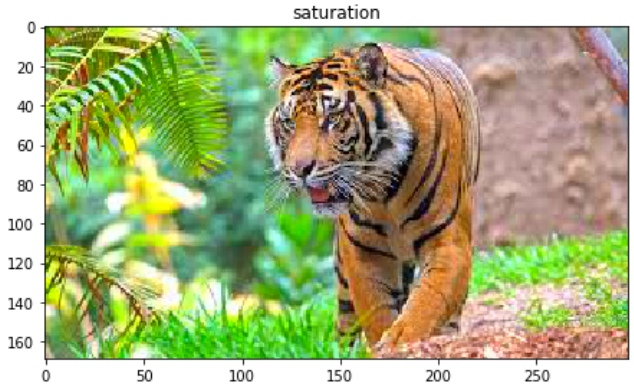
Hue

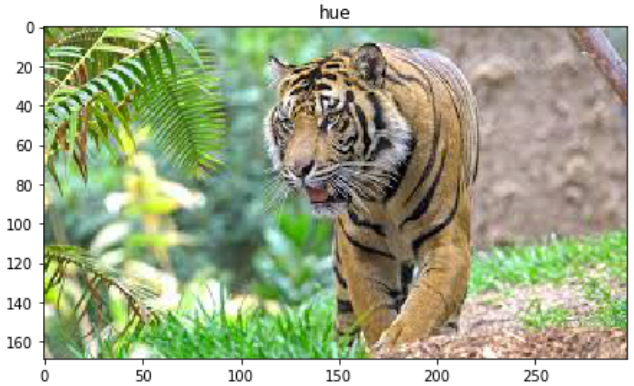
Hue can be described as the shade of the colors in an image.

Image Augmentation Techniques: Color Augmentation









Some Performance Measures: Classification Accuracy

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Accuracy: takes a value in the range [0, 1].
It works well only if there are equal number of samples belonging to each class.
For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get **98% training accuracy** by simply predicting every training sample belonging to class A.

When the same model is tested on a test set with 80% samples of class A and 20% samples of class B, then the **test accuracy would drop down to 80%.**

Some Performance Measures: Confusion Matrix (1)

Confusion matrix is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

Let's go through this with an example.

Let's assume we are building a binary classification to classify cat images from non-cat images. And let's assume our test set has 1100 images (1000 non-cat images, and 100 cat images), with the below confusion matrix.

		Actual Class	
		Cat	Non-Cat
Predicted Class	Cat	90	60
	Non-Cat	10	940

Some Performance Measures: Confusion Matrix (2)

There are 4 important terms :

True Positives : The cases in which we predicted Cat and the actual output was also Cat.

True Negatives : The cases in which we predicted Not-Cat and the actual output was Not-Cat.

False Positives : The cases in which we predicted Cat and the actual output was Not-Cat.

False Negatives : The cases in which we predicted Not-cat and the actual output was Cat.

Some Performance Measures: Precision & Recall

Precision : When the model predicts positive, how often is it correct?

$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

Recall : It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive).

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

Some Performance Measures: F1 Score

F1 Score is the harmonic mean between precision and recall.

The range for F1 Score is [0, 1].

It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

The greater the F1 Score, the better is the performance of our model.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Loss Functions

In the context of an optimization algorithm, the function used to evaluate a candidate solution (i.e. a set of weights) is referred to as the objective function.

We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively.

Typically, with neural networks, we seek to minimize the error.

As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply "*loss*."

The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model.

In simpler terms:

The loss function evaluates how bad the model fits the data.

Usually, the loss function on the dataset is the sum of pointwise loss function
(i.e., is the average of the loss of each datapoint)

Loss Functions:
Binary Cross Entropy

For a binary classification problem (dog exists or doesn't exist in an image):

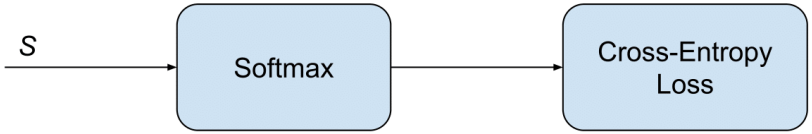
$$CE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1)$$

t_1 is the label (can be either 0 or 1) and s_1 is the output of the network (a probability value in the range [0, 1]; the output layer of the network has a sigmoid activation function)

Notice that when actual label is 1 (t_1 = 1), second half of function disappears whereas in case actual label is 0 (t_1 = 0) first half is dropped off. In short, we are just multiplying the log of the actual predicted probability for the ground truth class.
An important aspect of this is that cross entropy loss penalizes heavily the predictions that are *confident but wrong*.

Loss Functions: Categorical Cross Entropy

Also called **Softmax Loss**. It is used for multi-class classification (mutually exclusive classes).



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

In that case, the labels are one-hot (e.g. [1, 0, 0]), so only the positive class (p) keeps its term in the loss. There is only one element of the label vector t which is not zero $t_i=t_p$.
So discarding the elements of the summation which are zero due to labels, we can write:

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

where s_p is the output score for the positive class.

Loss Functions: Mean Squared Error (MSE)

Called **Mean Square Error/Quadratic Loss/L2 Loss** and is used in Regression Tasks.

As the name suggests, *Mean square error* is measured as the average of squared difference between predictions and actual observations.

It's only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions.

Plus MSE has nice mathematical properties which makes it easier to calculate gradients.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Loss Functions: Mean Absolute Error (MAE)

Called **Mean Absolute Error/L1 Loss** and is used in Regression Tasks.

It is measured as the average of sum of absolute differences between predictions and actual observations.

Like MSE, this as well measures the magnitude of error without considering their direction.

Unlike MSE, MAE needs more complicated tools such as linear programming to compute the gradients.

Plus MAE is more robust to outliers since it does not make use of square.

$$MAE = \frac{\sum_{i=1}^{n} \left| y_i - \hat{y}_i \right|}{n}$$

Thank you!