

Introduction to Deep Learning

CNN for Image Classification

Image Classification

- Image classification is the task of assigning a label or class to an entire image



Image Classification



- Images are expected to have only one class for each image

Image Classification



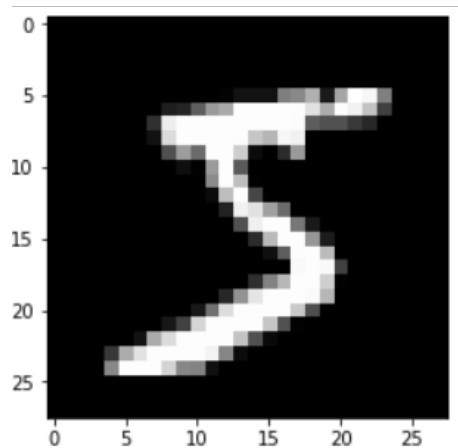
- Image classification models take an image as input and return a prediction about which class the image belongs to

Image Classification







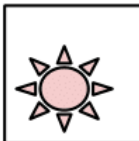

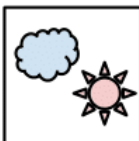


- “Dog vs. cat” classification is one problem of the so-called ***binary classification of images***

Image Classification



- Having a grey scale image with size $28 * 28$ presenting a number from 0-9
- Predict which number the image is presenting ?
- → Problem of ***multi-class classification of images***

Image Classification

Multi-Class		Multi-Label	
C = 3	Samples	Samples	
  	  	  	
	Labels (t)	Labels (t)	
	[0 0 1] [1 0 0] [0 1 0]	[1 0 1] [0 1 0] [1 1 1]	

Multi-class vs. Multi-label classification

Image Datasets

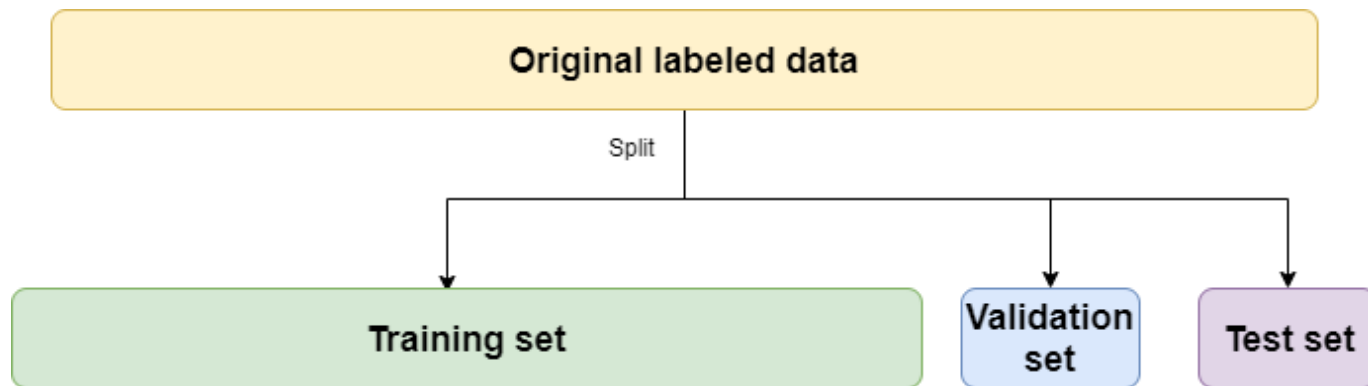
You can visit this website to get information of the dataset: <https://paperswithcode.com/datasets>

Use in this lecture:

- “Dogs vs. Cats” dataset downloaded from Kaggle
<https://www.kaggle.com/c/dogs-vs-cats/data>
- “MNIST Dataset of handwritten digits” downloaded from Kaggle
<https://www.kaggle.com/datasets/hojjatk/mnist-dataset>

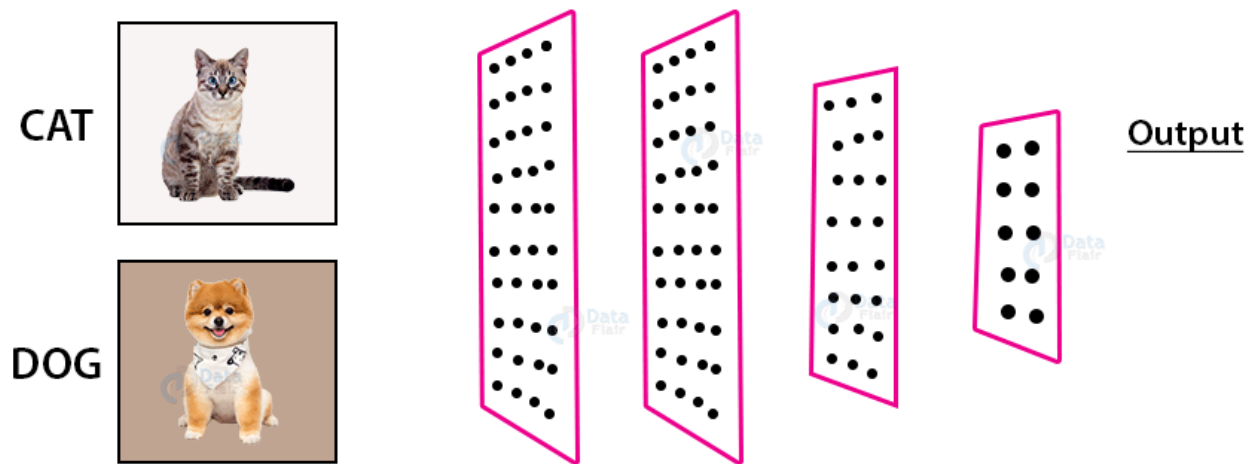
Dataset Preparation

- Standardize images prior to the model requirement
- Standardize directories for training set, validation set and test set



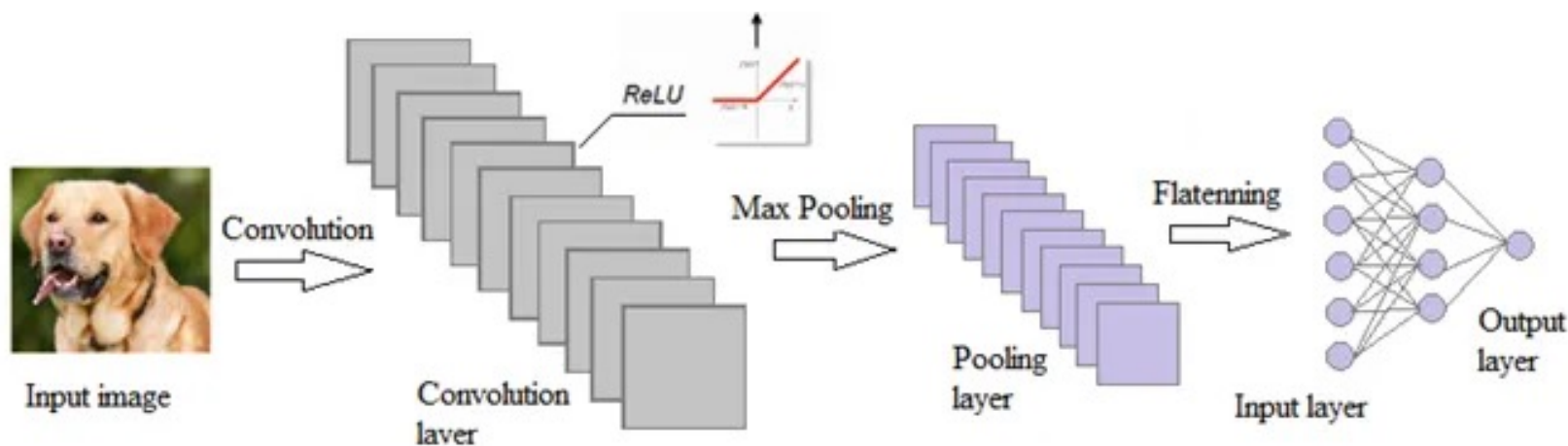
CNN model for Image Classification

- Input data are images → choose CNN model
- General progress: input image → Convolutional layer (Conv) + Pooling Layer (Pool) → Fully Connected Layer (FC) → Output



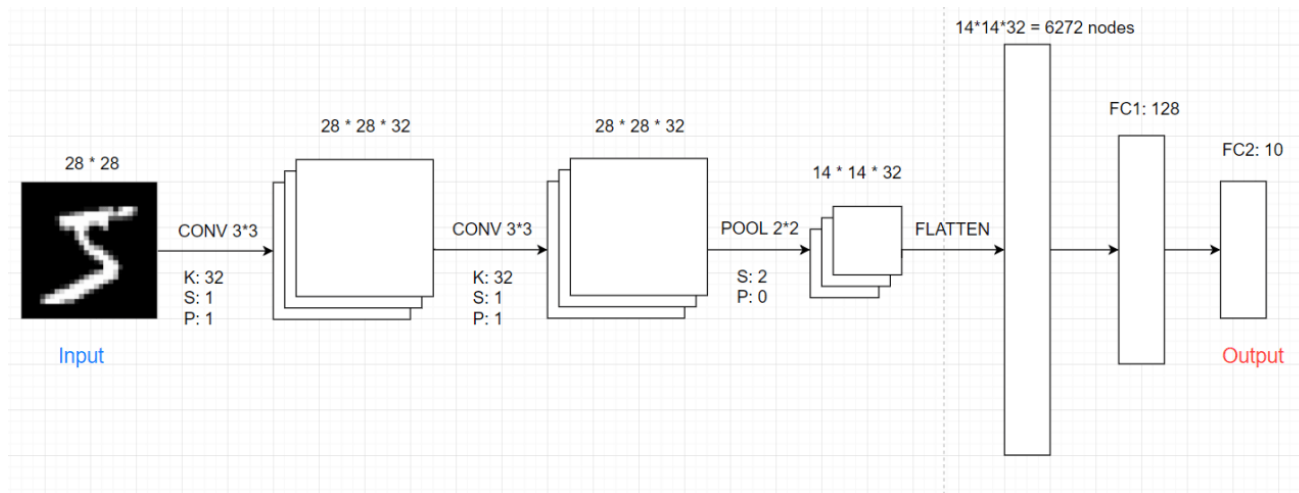
CNN model for Image Classification

- Input data are images → choose CNN model
- General progress: input image → Convolutional layer (Conv) + Pooling Layer (Pool) → Fully Connected Layer (FC) → Output



CNN model for Image Classification

- Input data are images → choose CNN model
- General progress: input image → Convolutional layer (Conv) + Pooling Layer (Pool) → Fully Connected Layer (FC) → Output

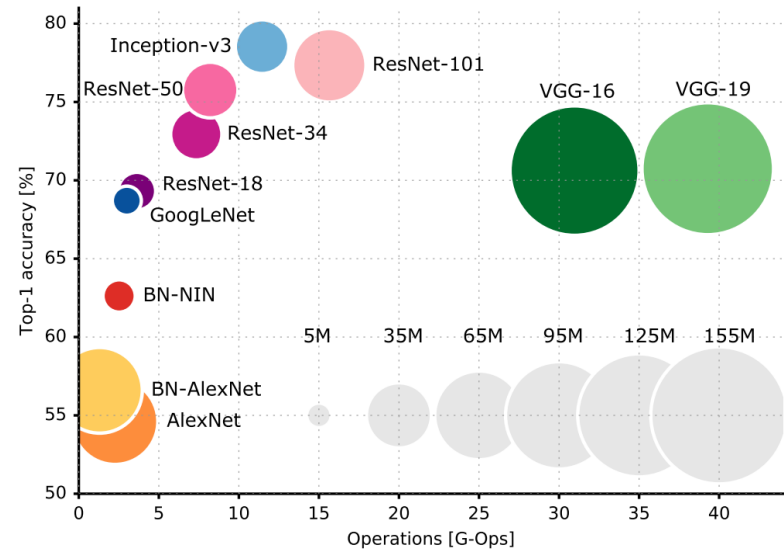
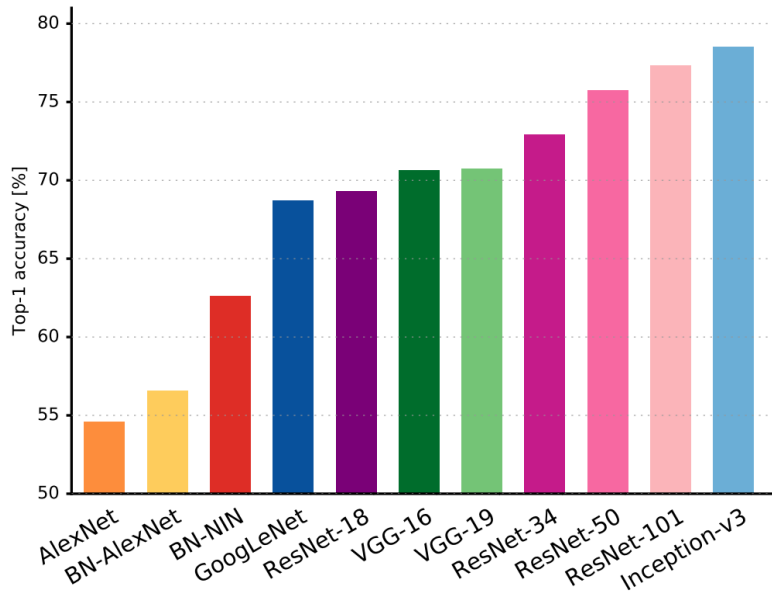


CNN model for Image Classification

- You can build a CNN network by yourself to perform image classification
- You can also use the existing CNN architecture like VGG, ResNet, etc. to perform image classification
- Or you can modify the existing CNN architecture (VGG, ResNet, etc.) to perform image classification

CNN model for Image Classification

Top 1-accuracy, performance and size on the ImageNet dataset



See: <https://paperswithcode.com/sota/image-classification-on-imagenet> for more information

Canziani, Paszke, and Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications." (May 2016).

CNN model for Image Classification

Method	# Params	Extra Data	ImageNet		ImageNet-Real [6]
			Top-1	Top-5	Precision@1
ResNet-50 [24]	26M	—	76.0	93.0	82.94
ResNet-152 [24]	60M	—	77.8	93.8	84.79
DenseNet-264 [28]	34M	—	77.9	93.9	—
Inception-v3 [62]	24M	—	78.8	94.4	83.58
Xception [11]	23M	—	79.0	94.5	—
Inception-v4 [61]	48M	—	80.0	95.0	—
Inception-resnet-v2 [61]	56M	—	80.1	95.1	—
ResNeXt-101 [78]	84M	—	80.9	95.6	85.18
PolyNet [87]	92M	—	81.3	95.8	—
SENet [27]	146M	—	82.7	96.2	—
NASNet-A [90]	89M	—	82.7	96.2	82.56
AmoebaNet-A [52]	87M	—	82.8	96.1	—
PNASNet [39]	86M	—	82.9	96.2	—
AmoebaNet-C + AutoAugment [12]	155M	—	83.5	96.5	—
GPipe [29]	557M	—	84.3	97.0	—
EfficientNet-B7 [63]	66M	—	85.0	97.2	—
EfficientNet-B7 + FixRes [70]	66M	—	85.3	97.4	—
EfficientNet-L2 [63]	480M	—	85.5	97.5	—
ResNet-50 Billion-scale SSL [79]	26M	3.5B labeled Instagram	81.2	96.0	—
ResNeXt-101 Billion-scale SSL [79]	193M	3.5B labeled Instagram	84.8	—	—
ResNeXt-101 WSL [42]	829M	3.5B labeled Instagram	85.4	97.6	88.19
FixRes ResNeXt-101 WSL [69]	829M	3.5B labeled Instagram	86.4	98.0	89.73
Big Transfer (BiT-L) [33]	928M	300M labeled JFT	87.5	98.5	90.54
Noisy Student (EfficientNet-L2) [77]	480M	300M unlabeled JFT	88.4	98.7	90.55
Noisy Student + FixRes [70]	480M	300M unlabeled JFT	88.5	98.7	—
Vision Transformer (ViT-H) [14]	632M	300M labeled JFT	88.55	—	90.72
EfficientNet-L2-NoisyStudent + SAM [16]	480M	300M unlabeled JFT	88.6	98.6	—
Meta Pseudo Labels (EfficientNet-B6-Wide)	390M	300M unlabeled JFT	90.0	98.7	91.12
Meta Pseudo Labels (EfficientNet-L2)	480M	300M unlabeled JFT	90.2	98.8	91.02

Meta Pseudo Labels, Hieu Pham et al. (Jan 2021).

Activation Function

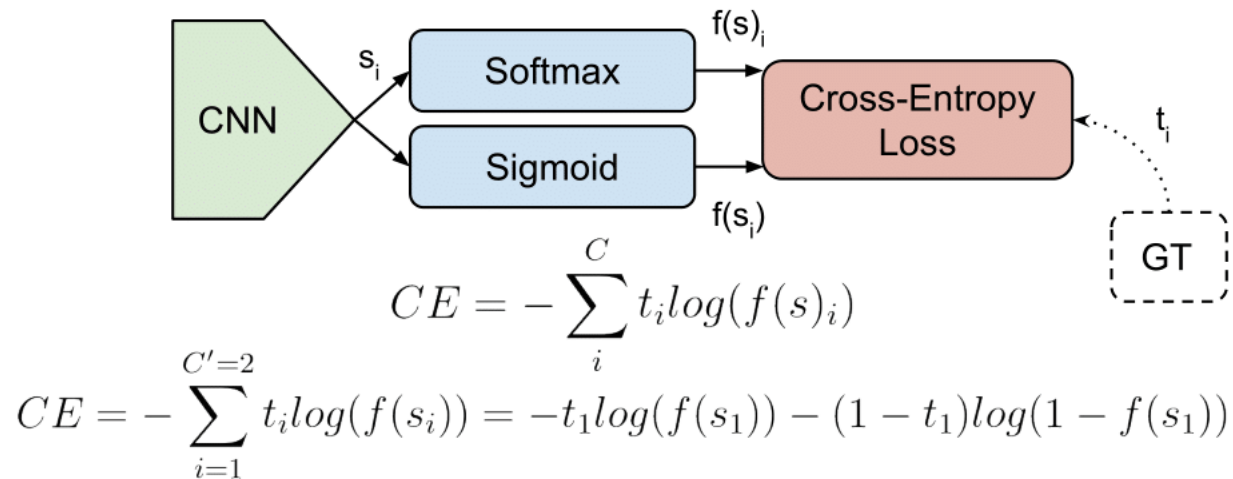
- Binary classification:
 - Activation function at output layer (with one node) is sigmoid function

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

- Multi-class classification:
 - Activation function at output layer (with > 1 nodes) is softmax function

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Loss function



- Cross-entropy loss is used as default loss function for both binary and multi-class classification

Cross-entropy loss

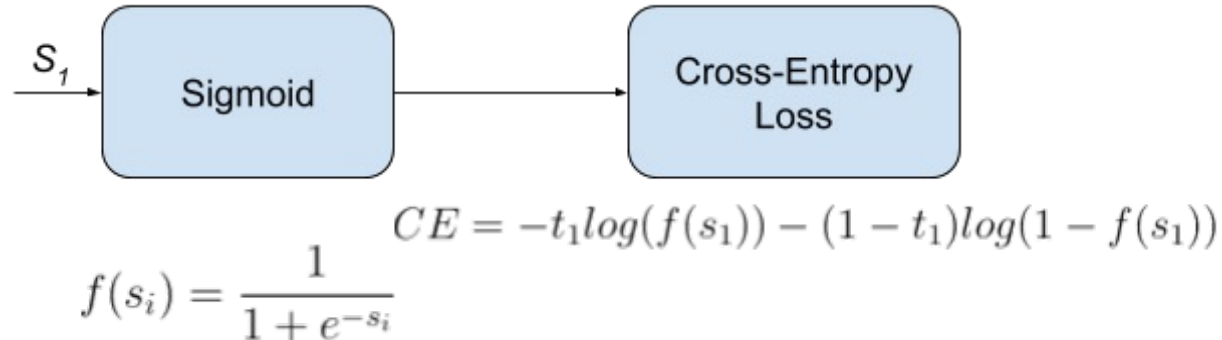
- Formula of cross-entropy loss:

$$CE = - \sum_i^C t_i \log(s_i)$$

- Where t_i , s_i is the groundtruth and the CNN score for each class i in C

Binary cross-entropy loss

- Binary cross-entropy loss:



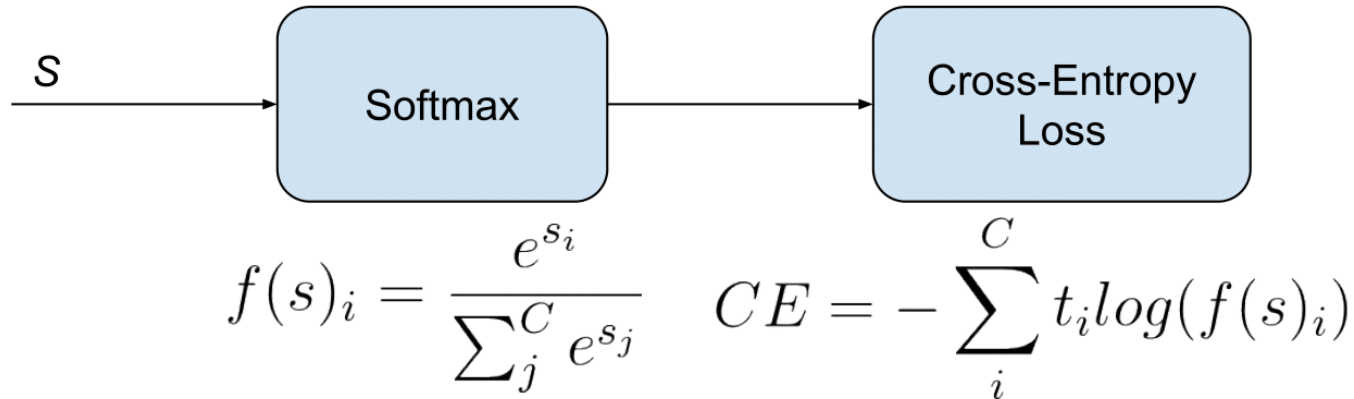
- The loss can be expressed as:

$$CE = \begin{cases} -\log(f(s_1)) & \text{if } t_1 = 1 \\ -\log(1 - f(s_1)) & \text{if } t_1 = 0 \end{cases}$$

Where $t_1 = 1$ means that the class $C_1 = C_i$ is the positive class

Categorical cross-entropy loss

- Categorical cross-entropy loss:



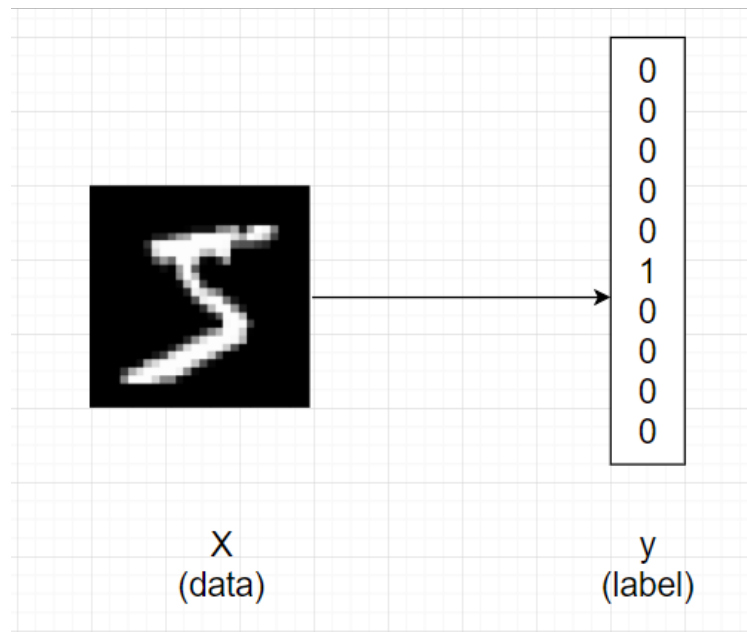
- In multi-class classification, the labels are one-hot, so only the positive class C_p keeps its term in the loss: Only 1 element of the output vector is not zero

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

Where S_p is the CNN score for the positive class

Categorical cross-entropy loss

- Example for the problem of multi-class classification of handwritten digits
- One-hot encoding: transform data label as number i to the vector v of size 10×1 where $v_{i+1} = 1$ and others = 0



Categorical cross-entropy loss

0	a_1
0	a_2
0	a_3
0	a_4
0	a_5
1	a_6
0	a_7
0	a_8
0	a_9
0	a_{10}
y	\hat{y}
Actual value	Predicted value

- Our expectation is a_6 close to 1 and others close to 0

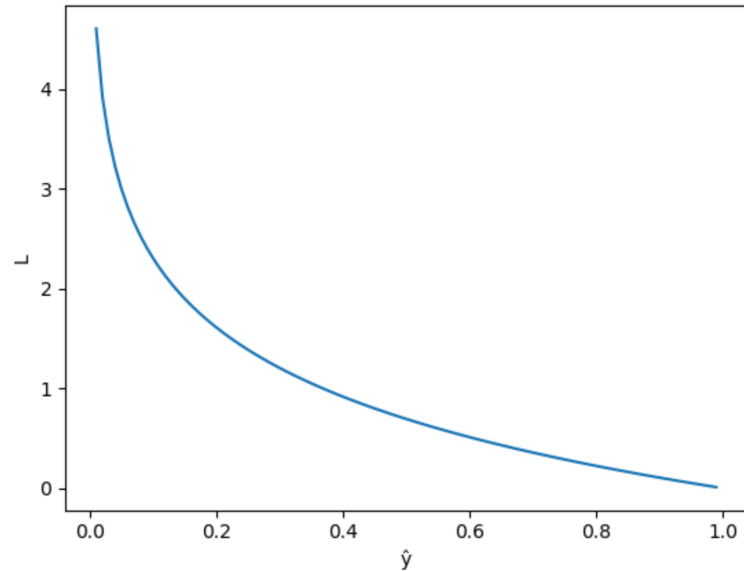
Loss Function

$$L = - \sum_{i=1}^{10} y_i * \log(\hat{y}_i)$$

- With $i = 5$: $L = -\log(\hat{y}_6)$

Loss Function

$$L = -\log(\hat{y}_6)$$



- Loss function L becomes smaller when the predicted value is closer to the actual value, vice versa
- Our problem becomes “finding minimum value of L ”

Pre-trained CNN models

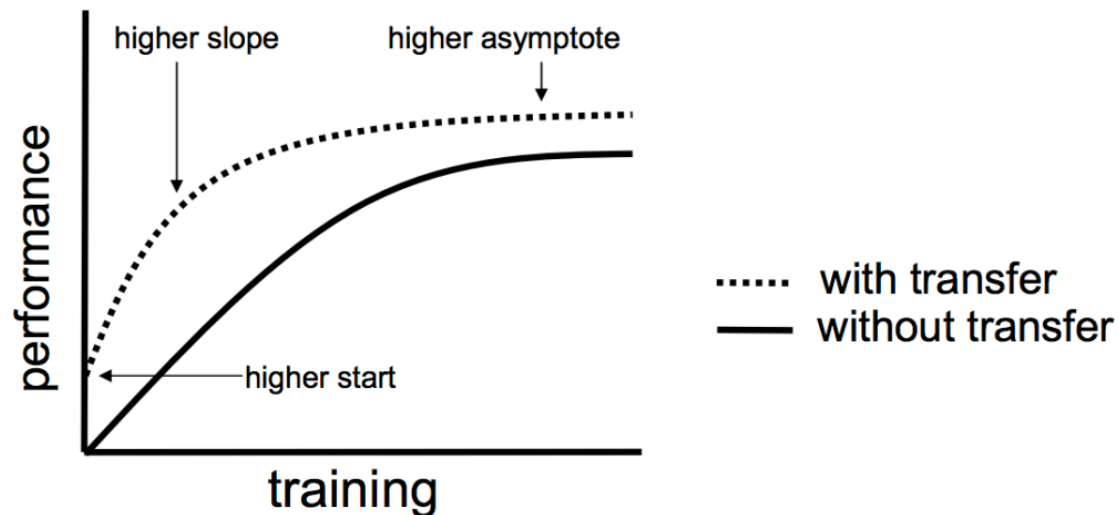
Pre-trained models:

- Training a model on big and general datasets such as ImageNet, VGGFace2 from scratch takes days or weeks
- Many models were trained on ImageNet/VGGFace2 and their weights are publicly available

Pre-trained CNN models

Transfer learning:

- Use pre-trained weights, remove last layers to compute representations of images
- Train a classification model from these features on a new classification task
- The network is used as a generic feature extractor



Pre-trained CNN models

Fine-tuning: retraining the (some) parameters of the network given enough data

- Truncate the last layer(s) of the pre-trained network
- Freeze the remaining layer's weights
- Add a (linear) classifier on top and train it for a few epochs
- Then fine-tune the whole network or the few deepest layers
- Use a smaller learning rate when fine tuning

Example

- **Multi-class classification problem:** Classify 17 types of flowers, in which each type has about 80 images



Bluebell



Buttercup

.....



ColtsFoot

Example

- **Multi-class classification problem:** Classify 17 types of flowers, in which each type has about 80 images



Bluebell



Buttercup



ColtsFoot



Cowslip



Crocus



Daffodil



Daisy



Dandelion



Fritillary



Iris



LilyValley



Pansy



Snowdrop



Sunflower



Tigerlily



Tulip



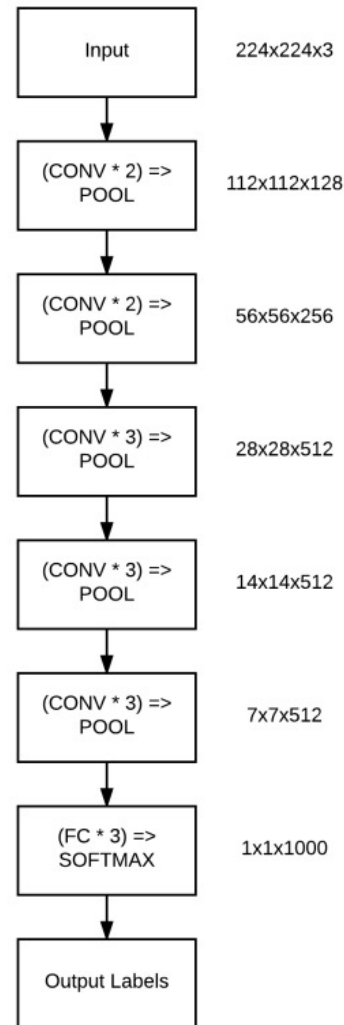
Windflower

Solution: Transfer Learning

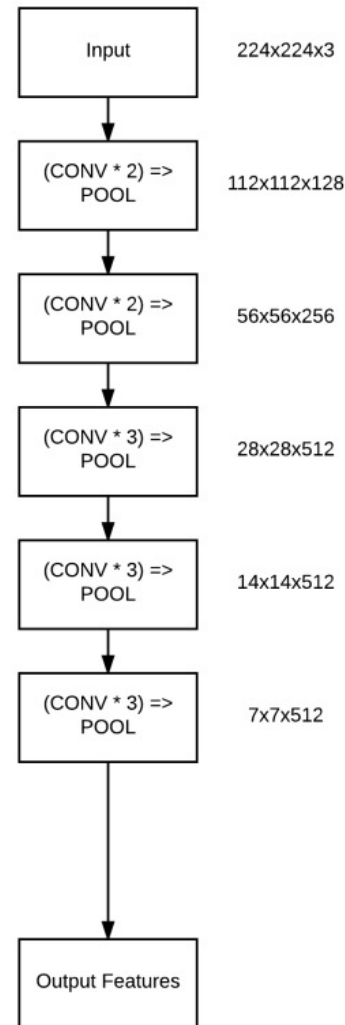
- Use pre-trained model VGG16 on ImageNet dataset, which contains 1,2 million images of 1000 classes
- This pre-trained model is already supported in Keras

Transfer Learning : Feature extractor

Original VGG16

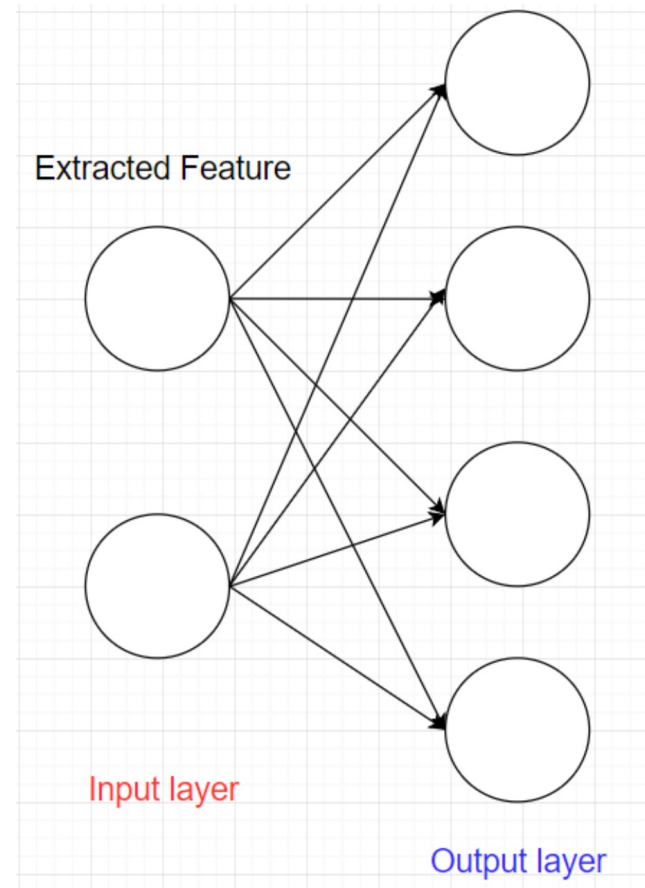


VGG16 with the removal of fully connected layers



Transfer Learning : Feature extractor

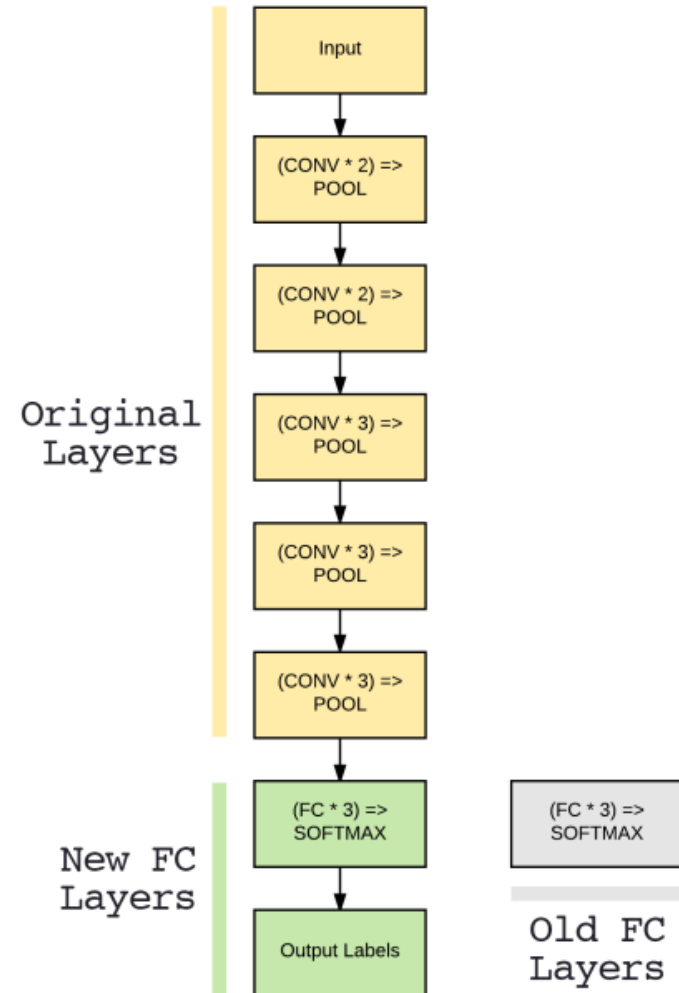
- Output features are used as input of linear classifiers such as linear SVM



of nodes in output layer = # of classes to classify

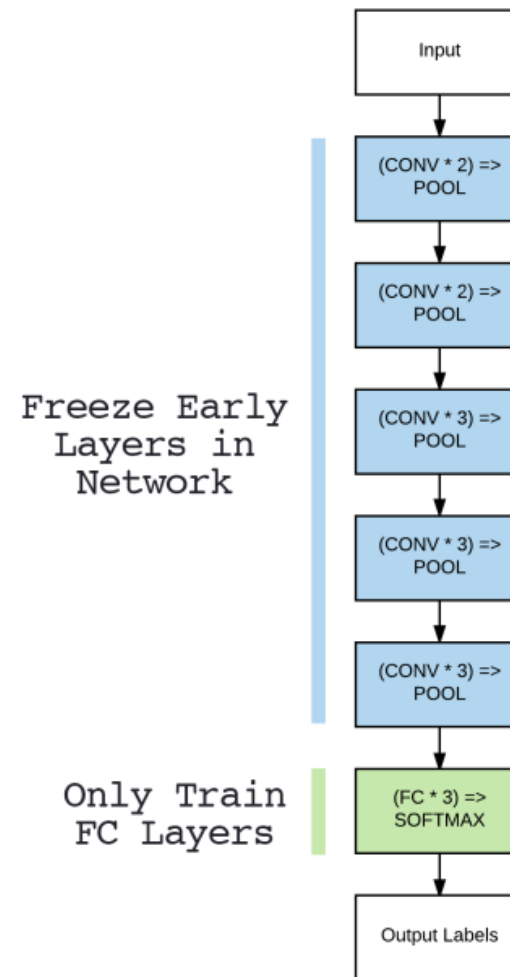
Transfer Learning : Fine Tuning

- ConvNet of VGG16 are kept, FCs of VGG16 are removed
- New FC layers are added to the network



Transfer Learning : Fine Tuning

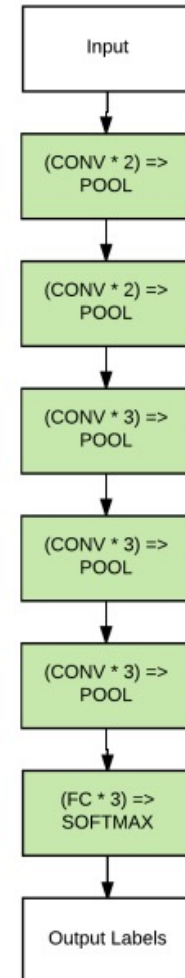
- First stage of training: freeze pre-trained layers, only train newly added layers



Transfer Learning : Fine Tuning

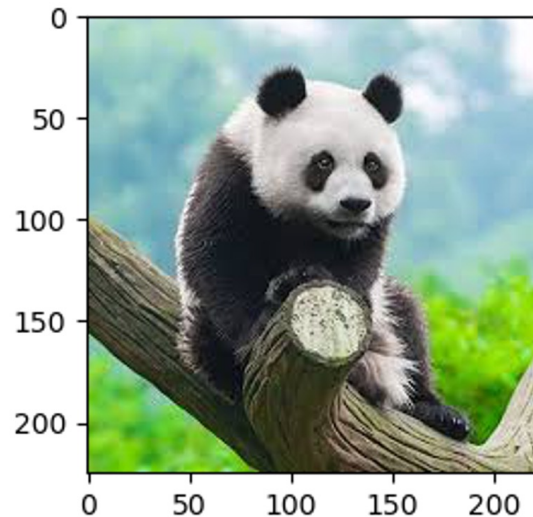
- Second stage of training: unfreeze pre-trained layers, train the whole network

Unfreeze Early
Layers & Train
All

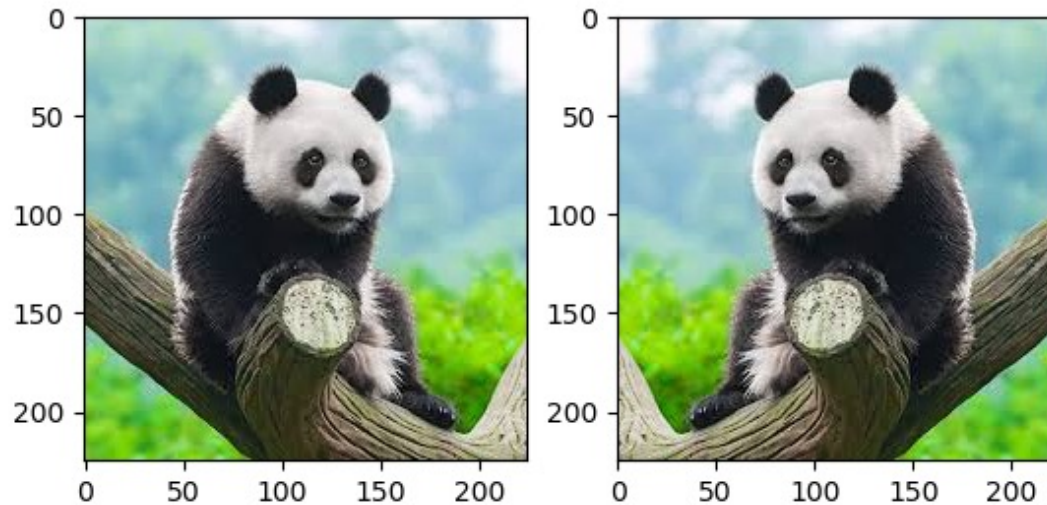


Data augmentation

- Data augmentation is a technique to generate more training data from our dataset

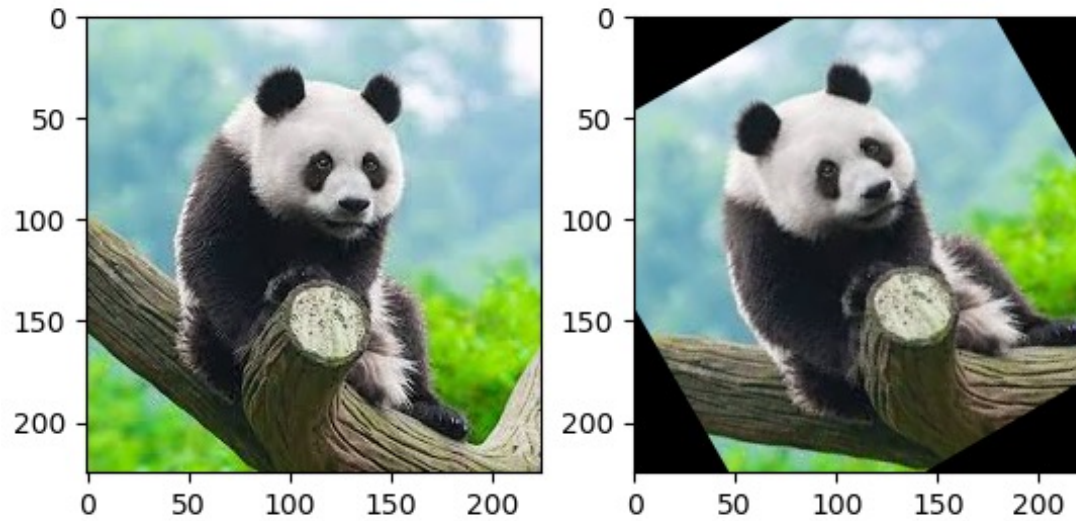


Data Augmentation



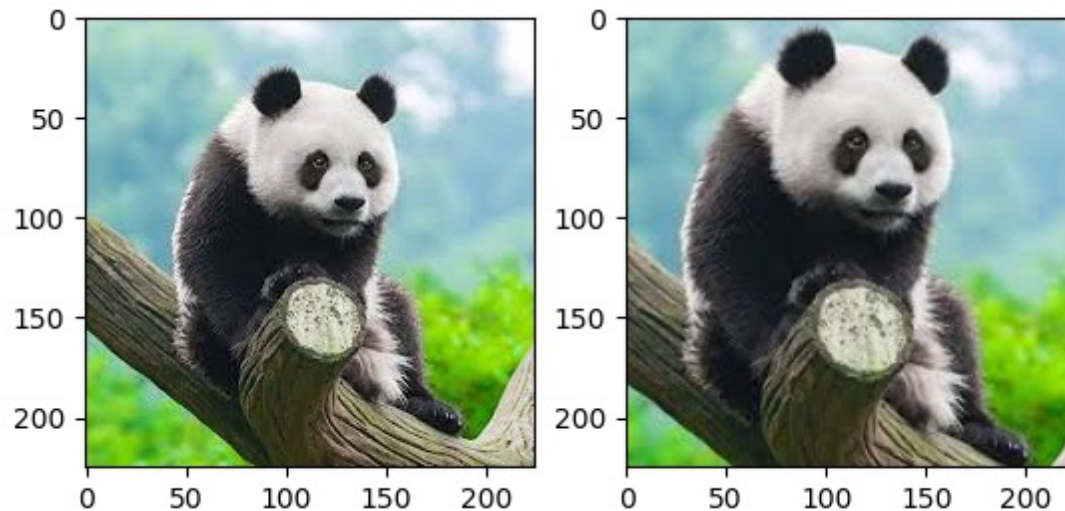
- Flip image horizontally

Data Augmentation



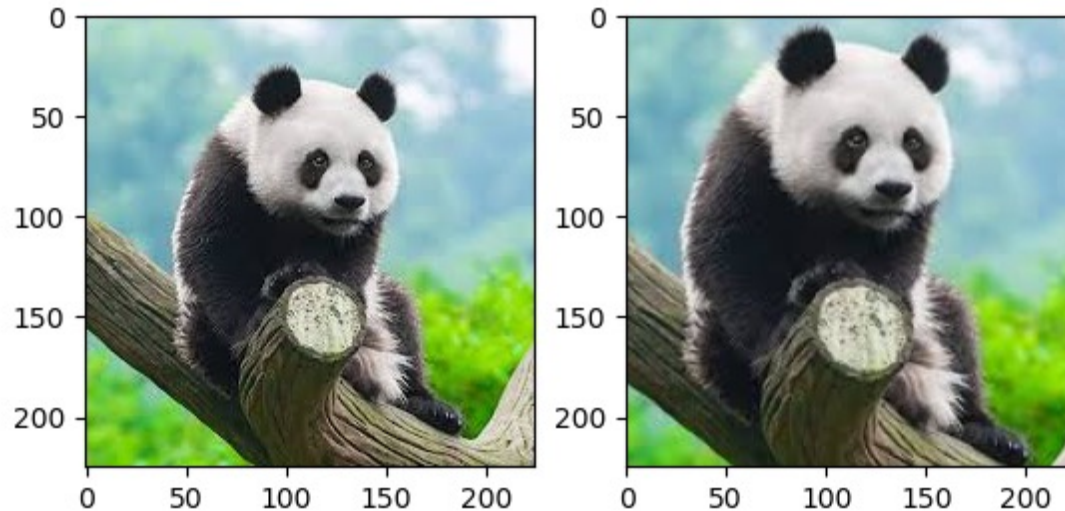
- Rotate image 30 degrees
- Problem of black regions

Data Augmentation



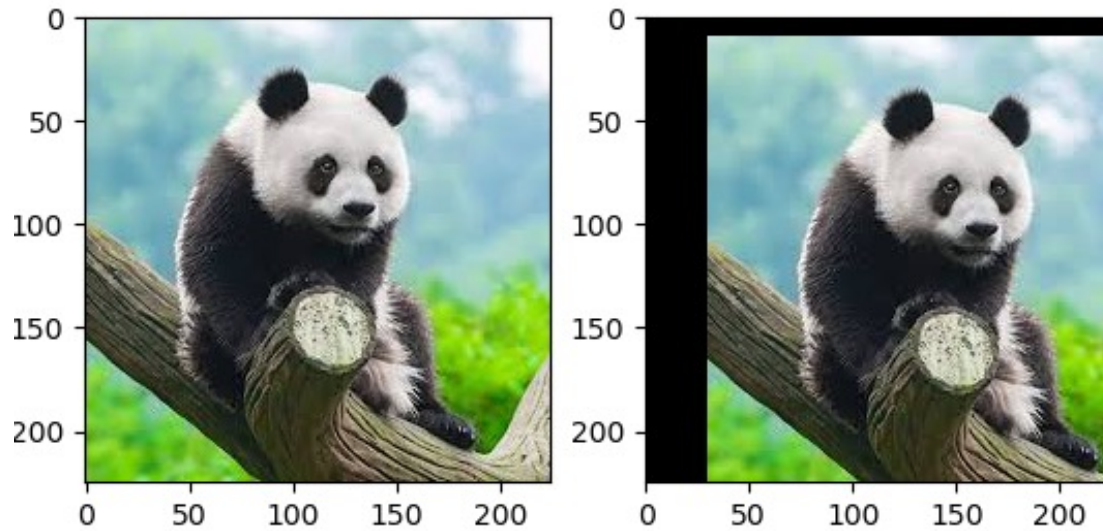
- Scale the image to bigger size

Data Augmentation



- Crop an image region and resize to the same size with the original image

Data Augmentation



- Translate image 30px via x-axis, 10px via y-axis
- Problem of black regions

Data Augmentation with Keras

```
from keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    channel_shift_range=9,
    fill_mode='nearest'
)

train_flow = image_gen.flow_from_directory(train_folder)
model.fit_generator(train_flow, train_flow.n)
```

