FACIAL EMOTION RECOGNITION USING SUPERVISED AND SEMI-SUPERVISED LEARNING

AGENDA

- INTRODUCTION
- DATASET
- APPROACH & RESULTS
- CONCLUSION
- LIFE IN TOSHIBA AND JAPAN

INTRODUCTION

- Facial Emotion Recognition Classification (Kaggle challenge 2013)*

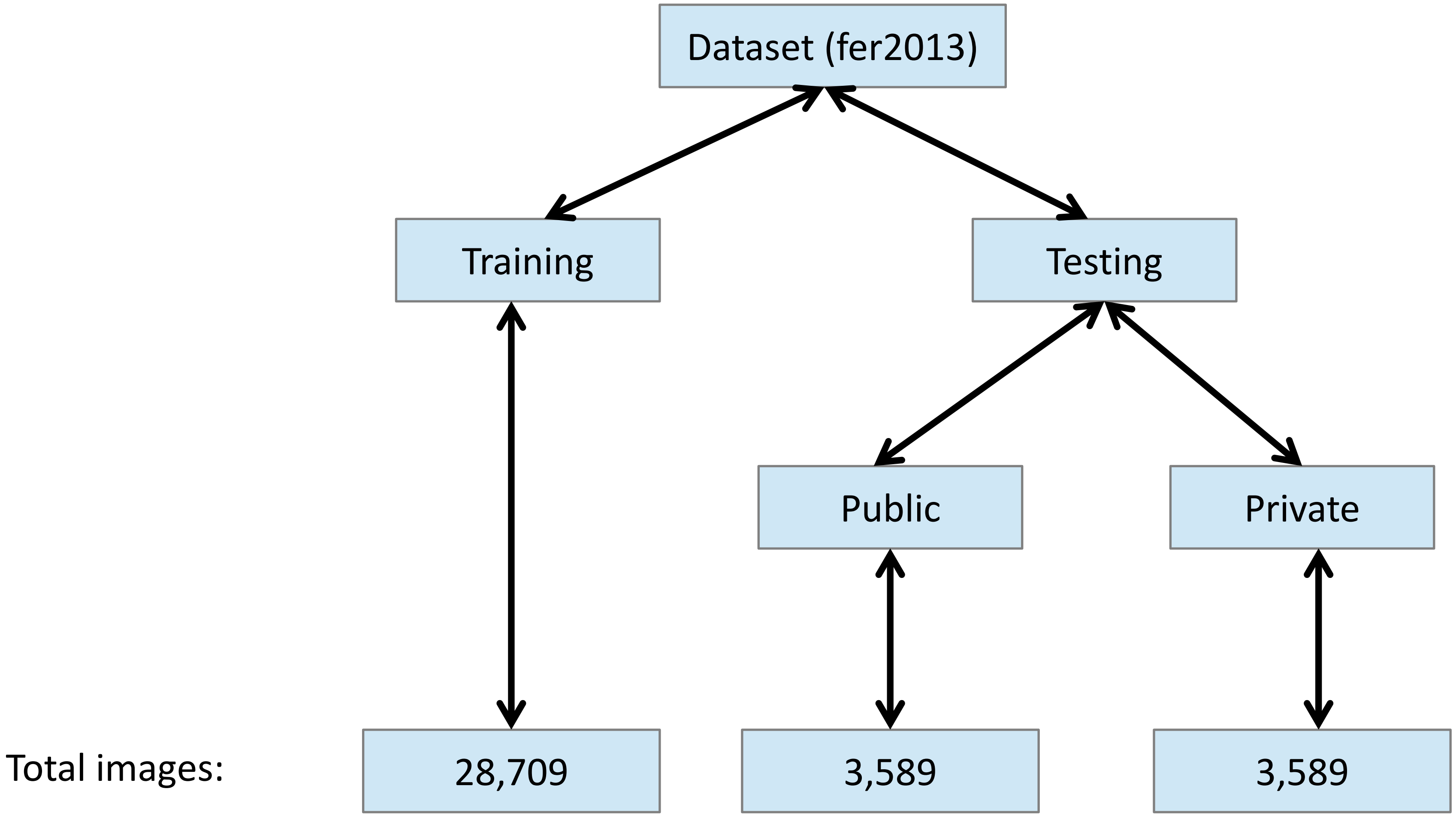
Input image

Output category of emotion

0=Angry
1=Disgust
2=Fear
3=Happy
4=Sad
5=Surprise
6=Neutral

*https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

Dataset

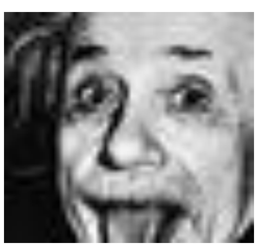


*https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

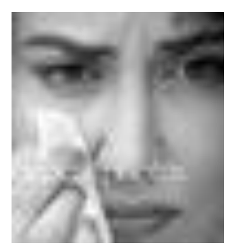
Image specification

- Each image is 48 x 48 pixels – 2304 pixel values.
- Images are grayscale in nature – pixel value ranges from 0 to 255.
- Face has been automatically registered – occupies the same amount of space in each image.
- Number of images per class: [3995 436 4097 7215 4830 3171 4965]

Good and clear images



Bad images with thumbnails written across



APPROACH & RESULTS

Problem of multi-class (our case: 7) categorization.
Many possible approaches:

- k-nearest neighbors (kNN)
- Support Vector Machine (SVM)
- Hierarchical classification (Decision trees)
- **Neural networks – Convolutional Neural Networks (CNN)**

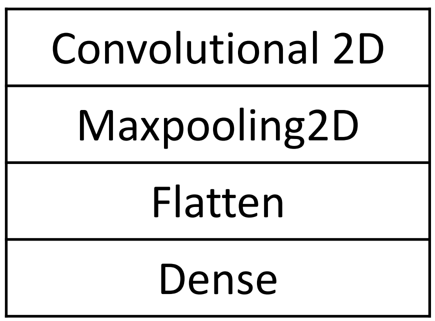
Divided into two categories:

1. Fully supervised learning – Training the neural network with all the training images (100%) in the dataset.

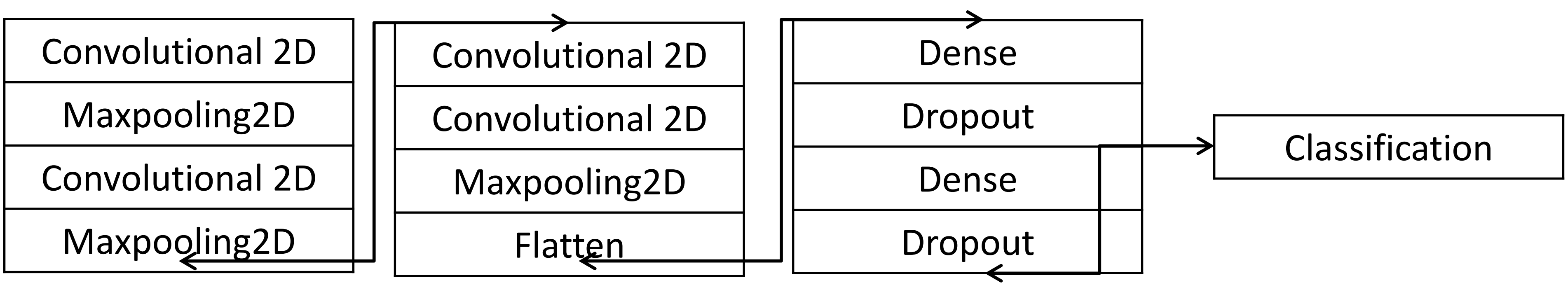
2. Semi-supervised learning – Training the neural network with only a part of the training images (10%, 20%) and unlabeled data.

Fully supervised learning

- Start simple! - With a one layer CNN architecture → accuracy achieved **36%**



- Literature* suggested to have at least **three CNN layers** for
- multi-class categorization problem



*https://arxiv.org/pdf/1701.08816.pdf

Architectural changes and its impact

CNN architectural change	Accuracy / Impact on accuracy (%)
Simple 1 layer CNN	36%
4 conv2D, 3 maxpooling, 2 dense	56%
4 conv2D, 2 maxpooling, 2 dense	56.7% / increased by ~0.5%
4 conv2D, 1 maxpooling, 1 average pooling, 2 dense	58.9% / increased by ~2-2.5%
Adding kernel regularizers with l2 regularization (0.001) in the dense layer	**59.7%** / increased by ~1.0%
Included strides in the maxpooling layer	58.6% / decreased by ~1.0%
Replace normal dropout with **Gaussian dropout**	**59.7%** / almost remains the same
Resizing the image (64 x 64)	**59.7%** / almost remains the same
Changing the size and number of filters in Conv2D	**59.7%** / almost remains the same

Architectural changes were inspired from the literature* of multi-class categorization.

Main impacts:

- Removing an extra Maxpooling2D layer.
- Replacing the second Maxpooling2D
- layer with Average pooling layer.
- *l2* kernel regularizers.

*https://arxiv.org/pdf/1701.08816.pdf

Final cnn architecture for fully supervised learning

Layer	Hyperparameters
Convolutional	*32* filters, *3 x 3*, L2 regularization (*0.001*), activation: prelu, zeropadding (*2, 2*)
Convolutional	*64* filters, *3 x 3*, activation: prelu, zeropadding (*1, 1*)
Pooling	Maxpool (*5 x 5)*
Convolutional	*128* filters, *3 x 3*, activation: prelu, zeropadding (*1, 1*)
Convolutional	*128* filters, *3 x 3*, activation: prelu, zeropadding (*1, 1*)
Pooling	Average pool (3 *x 3)*
Dense	1024
Dropout	0.25
Dense	1024
Dropout	0.25
Softmax	Fully connected

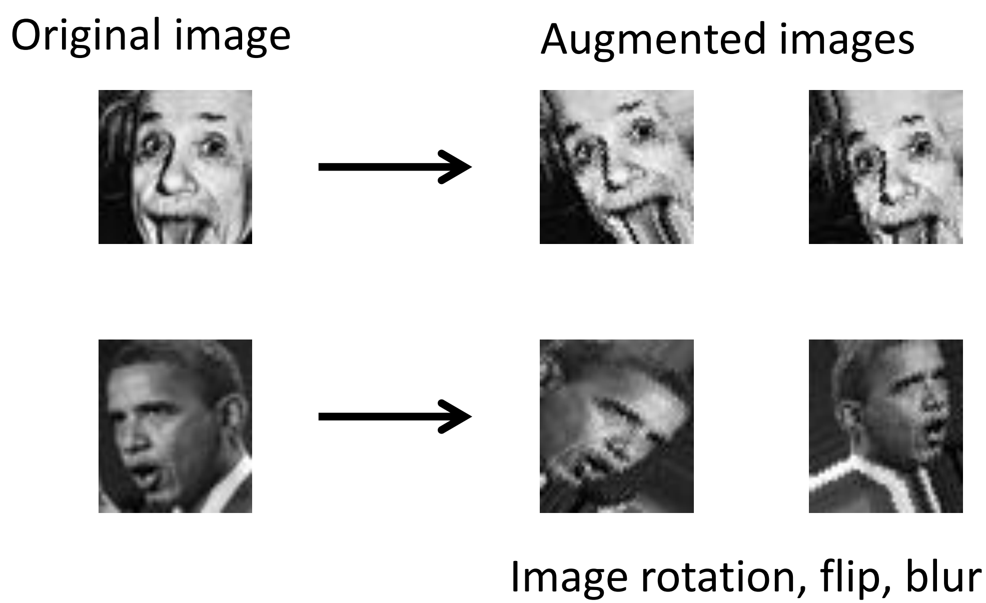
<u>Total parameters:</u>
4,829,319
<u>Trainable parameters:</u>
4,829,255
<u>Non-trainable parameters:</u>
64

CNN model summary

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 46, 46, 32)        320
batch_normalization_1 (Batch (None, 46, 46, 32)        128
p_re_lu_1 (PReLU)            (None, 46, 46, 32)        67712
zero_padding2d_1 (ZeroPaddin (None, 50, 50, 32)        0
conv2d_2 (Conv2D)            (None, 48, 48, 64)        18496
p_re_lu_2 (PReLU)            (None, 48, 48, 64)        147456
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 64)          0
zero_padding2d_2 (ZeroPaddin (None, 13, 13, 64)        0
conv2d_3 (Conv2D)            (None, 11, 11, 128)       73856
p_re_lu_3 (PReLU)            (None, 11, 11, 128)       15488
zero_padding2d_3 (ZeroPaddin (None, 15, 15, 128)       0
conv2d_4 (Conv2D)            (None, 13, 13, 128)       147584
p_re_lu_4 (PReLU)            (None, 13, 13, 128)       21632
zero_padding2d_4 (ZeroPaddin (None, 17, 17, 128)       0
average_pooling2d_1 (Average (None, 5, 5, 128)         0
flatten_1 (Flatten)          (None, 3200)              0
dense_1 (Dense)              (None, 1024)              3277824
p_re_lu_5 (PReLU)            (None, 1024)              1024
dropout_1 (Dropout)          (None, 1024)              0
dense_2 (Dense)              (None, 1024)              1049600
p_re_lu_6 (PReLU)            (None, 1024)              1024
dropout_2 (Dropout)          (None, 1024)              0
dense_3 (Dense)              (None, 7)                 7175
activation_1 (Activation)    (None, 7)                 0
=================================================================
Total params: 4,829,319
Trainable params: 4,829,255
Non-trainable params: 64
```

Data augmentation on the training images

- Instead of training the network with the original images in the training dataset, manipulate the images and then train the network.
- *ImageDataGenerator* in *keras* has this functionality.



Impact on accuracy due to synthetic images

Parameters	**Accuracy**
rotation_range=20, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.1, zoom_range=0.1, horizontal_flip=True	**63.14%**
rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True	60.62%
rotation_range=90, width_shift_range=0.4, height_shift_range=0.4, shear_range=0.4, zoom_range=0.4, horizontal_flip=True	48.55%

- Highest accuracy achieved on kaggle competition: 70% ; Approach used > CNNs with
- **Support Vector Machine** & Transformation to images generating more data.
- Two recent papers claim to achieve accuracy around 75%:
- 1. CNN-SIFT algorithm*
- 2. Illumination correction, landmark facial features first#

*https://arxiv.org/pdf/1608.02833.pdf
#https://arxiv.org/pdf/1612.02903.pdf

Confusion matrix

Class	0	1	2	3	4	5	6	**Total**
0	197	4	44	58	90	8	89	**490**
1	17	8	10	9	9	1	1	**55**
2	47	2	153	46	145	59	76	**528**
3	13	0	14	765	33	12	42	**879**
4	41	0	43	63	298	5	144	**594**
5	10	0	63	45	20	258	20	**416**
6	17	1	29	66	107	12	394	**626**
Total	**342**	**15**	**356**	**1052**	**702**	**355**	**766**	3588

Semi-supervised learning

- Selecting the ratio of labeled / unlabeled data.

Percentage (labeled/unlabeled)	Accuracy (%) of the built CNN architecture
(100/0)	59
(50/50)	53
(30/70)	49
(20/80)	**45**
(10/90)	**40**

- Selecting the architecture for semi-supervised learning with
- Neural Networks:
- 1. Entropy minimization*
- 2. Autoencoder#
- 3. **Ladder Network**$
- 3. Mean Teacher^β

* https://papers.nips.cc/paper/2740-semi-supervised-learning-by-entropy-minimization
\# http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf
$ https://arxiv.org/abs/1507.02672
^β https://arxiv.org/abs/1703.01780

Ladder network

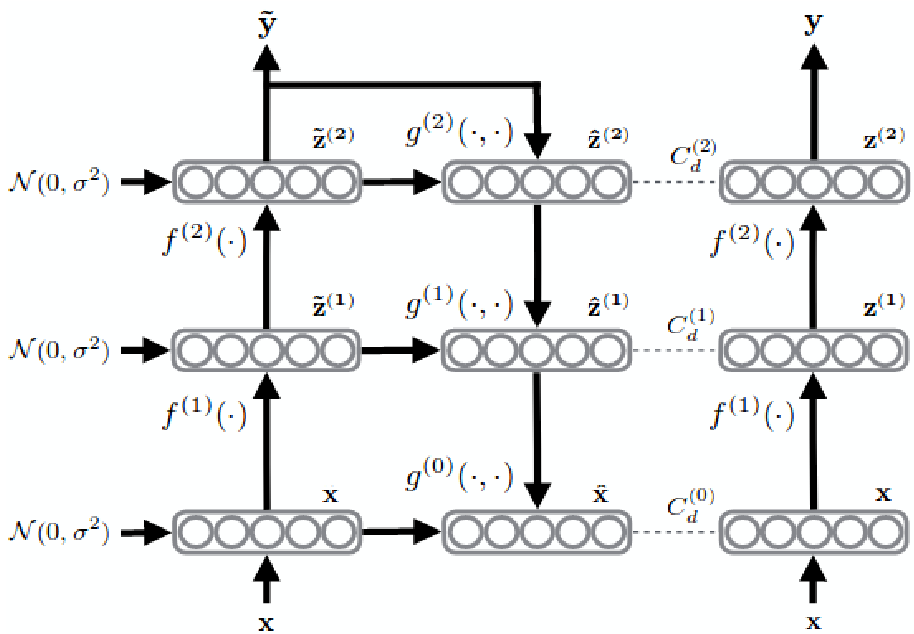


Image source: Main paper (Ladder Network)

<u>Main parts:</u>

1. Encoder – clean and noisy
2. Decoder
3. Denoising function

<u>Loss function:</u>

1. **Categorical cross entropy** -
Noisy Y and labeled data.
2. **Mean squared error** – Between the outputs of the decoder and clean encoder at each layer.

<u>Implementation:</u>

- Encoder - **Pre-trained** CNN architecture designed for supervised
- learning with only 10% labeled data.
- Decoder - Basically an inverse of an encoder - **Deconvolutional** and **upsampling**.
- Denoising function - Function which **reconstructs** a noisy signal from encoder to a
- clean signal which will be compared with the clean encoder.

Breaking the ladder network architecture

Algorithm 1 Calculation of the output $\mathbf{y}$ and cost function C of the Ladder network

Require: $\mathbf{x}(n)$
 # Corrupted encoder and classifier
 $\tilde{\mathbf{h}}^{(0)} \leftarrow \tilde{\mathbf{z}}^{(0)} \leftarrow \mathbf{x}(n) + \texttt{noise}$
 for l = 1 **to** L **do**
  $\tilde{\mathbf{z}}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{W}^{(l)}\tilde{\mathbf{h}}^{(l-1)}) + \texttt{noise}$
  $\tilde{\mathbf{h}}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\tilde{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}))$
 end for
 $P(\tilde{\mathbf{y}} \mid \mathbf{x}) \leftarrow \tilde{\mathbf{h}}^{(L)}$
 # Clean encoder (for denoising targets)
 $\mathbf{h}^{(0)} \leftarrow \mathbf{z}^{(0)} \leftarrow \mathbf{x}(n)$
 for l = 1 **to** L **do**
  $\mathbf{z}^{(l)}_{\mathrm{pre}} \leftarrow \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$
  $\boldsymbol{\mu}^{(l)} \leftarrow \texttt{batchmean}(\mathbf{z}^{(l)}_{\mathrm{pre}})$
  $\boldsymbol{\sigma}^{(l)} \leftarrow \texttt{batchstd}(\mathbf{z}^{(l)}_{\mathrm{pre}})$
  $\mathbf{z}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{z}^{(l)}_{\mathrm{pre}})$
  $\mathbf{h}^{(l)} \leftarrow \texttt{activation}(\boldsymbol{\gamma}^{(l)} \odot (\mathbf{z}^{(l)} + \boldsymbol{\beta}^{(l)}))$
 end for

 # Final classification:
 $P(\mathbf{y} \mid \mathbf{x}) \leftarrow \mathbf{h}^{(L)}$
 # Decoder and denoising
 for l = L **to** 0 **do**
  if l = L **then**
   $\mathbf{u}^{(L)} \leftarrow \texttt{batchnorm}(\tilde{\mathbf{h}}^{(L)})$
  else
   $\mathbf{u}^{(l)} \leftarrow \texttt{batchnorm}(\mathbf{V}^{(l+1)}\hat{\mathbf{z}}^{(l+1)})$
  end if
  $\forall i : \hat{z}_i^{(l)} \leftarrow g(\tilde{z}_i^{(l)}, u_i^{(l)})$ # Eq. (2)
  $\forall i : \hat{z}_{i,\mathrm{BN}}^{(l)} \leftarrow \frac{\hat{z}_i^{(l)} - \mu_i^{(l)}}{\sigma_i^{(l)}}$
 end for
 # Cost function C for training:
 $C \leftarrow 0$
 if $t(n)$ **then**
  $C \leftarrow -\log P(\tilde{\mathbf{y}} = t(n) \mid \mathbf{x}(n))$
 end if
 $C \leftarrow C + \sum_{l=0}^{L} \lambda_l \left\| \mathbf{z}^{(l)} - \hat{\mathbf{z}}^{(l)}_{\mathrm{BN}} \right\|^2$ # Eq. (3)

Image source: Main paper (Ladder Network)

Noisy encoder and clean encoder:

$x_{\_n}, z^{(1)}{}_{\_n}, \ldots z^{(L)}{}_{\_n}, y_{\_n}$ = $\mathrm{Encoder}_{noisy}$
$x, z^{(1)}, \ldots z^{(L)}, y$ = $\mathrm{Encoder}_{clean}$
$x^{\wedge}, z^{\wedge(1)}, \ldots z^{\wedge(L)}$ = Decoder

Encoder [additional parameters before activation]:
Batchnormalization()
Mean and **scaling** ()

Noisy encoder [additional parameters to the clean en
Gaussian noise (σ = 0.2) added to each layer.

Loss function:
1. Supervised learning - Categorical cross entropy ($y_{\_n}$, Y)
2. Unsupervised learning – Mean squared error ($z^{(L)}$, $z^{\wedge(L)}$)

Denoising function:
g($z^{\wedge(L+1)}$, $z^{(L)}{}_{\_n}$) -> element wise multiplication, passed to the below layer

Decoder: Output from dense layer of the noisy encod
Deconvolution()
Upsampling()

Implementation of the ladder network architecture

<u>Noisy encoder and clean encoder:</u>

$x_{\_n}, z^{(1)}{}_{\_n}, \ldots z^{(L)}{}_{\_n}, y_{\_n}$ = $Encoder_{noisy}$
$x, z^{(1)}, \ldots z^{(L)}, y$ = $Encoder_{clean}$
$x^{\wedge}, z^{\wedge(1)}, \ldots z^{\wedge(L)}$ = Decoder

<u>Decoder:</u> Output from dense layer of the noisy encoder.
Deconvolution()
Upsampling()

<u>Encoder [additional parameters before activation]</u>
Batchnormalization()
Mean and **scaling** ()

<u>Noisy encoder [additional parameters to the clean encoder]:</u>
Gaussian noise (σ = 0.2) added to each layer.

Trained the CNN architecture previously built with 10% labe

Connected the output from the noisy encoder using **Lambd**

<u>Denoising function:</u>
$g(z^{\wedge(L+1)}, z^{(L)}{}_{\_n})$ -> element wise multiplication, passed to the below layer

Element wise multiplication – Used **merge** layer.
Lambda encapsulation for multiplying scalar with tensor

<u>Loss function:</u>
1. Supervised learning - Categorical cross entropy ($y_{\_n}$,Y)
2. Unsupervised learning – Mean squared error ($z^{(L)}, z^{\wedge(L)}$)

ae = Model(inp_img, outputs = [ns_enc_op, de_op_1, de_op_2

ae.compile(opt, loss=['cat','mse','mse'..], loss_weights=[alpha,b

ae.fit(X, [Y, cl_enc_1, cl_enc_2,... cl_enc_4], batch_size...)

Used **Callbacks** for dividing the training data into labeled and u

Preliminary Result

- Testing with **10%** labeled data:
- Baseline accuracy of supervised learning – 40%
- With semi-supervised learning – no denoising function yet – 43.5%
- Future work:
- > Test with different variations of the Gaussian noise (σ = 0.1, 0.3, 0.5).
- > Test with variations in architecture. Only one layer denoising function.
- > Test with different denoising functions.
- > Testing with different ratio of labeled to unlabeled data.

CONCLUSION

- Dataset needs to be studied properly:
- > **Manipulating** the dataset features which may suit the CNN architecture.
- > Illumination and occlusion inclusion.
- > Class **balance** in the dataset.
- Supervised learning can achieve **exceptional accuracy**, however, **labeled data** for many categorization tasks is <u>not</u> readily available. At the same time, it does <u>not</u> reflect how humans learn.
- Semi-supervised learning:
- > Can **improve** the accuracy compared to only supervised learning with a very **small amount** of labeled data.
- > Diverse applications in many fields where there is a scarcity of labeled data.
- > Combine CNN with some **clustering** technique for predicting the inherent structure in the dataset.
- > More intuitive and related to how humans learn.
- Using *keras:*
- > Suitable for simple model implementation and initial start-up.
- > Modifications in layer outputs, going deep in the network, changing the architecture, combining two different models – better switch to Tensorflow, PyTorch.