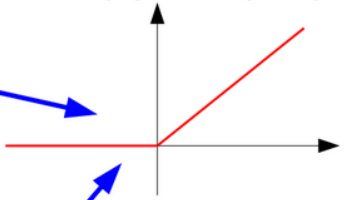
Tutoriel Deep Learning: applications signal

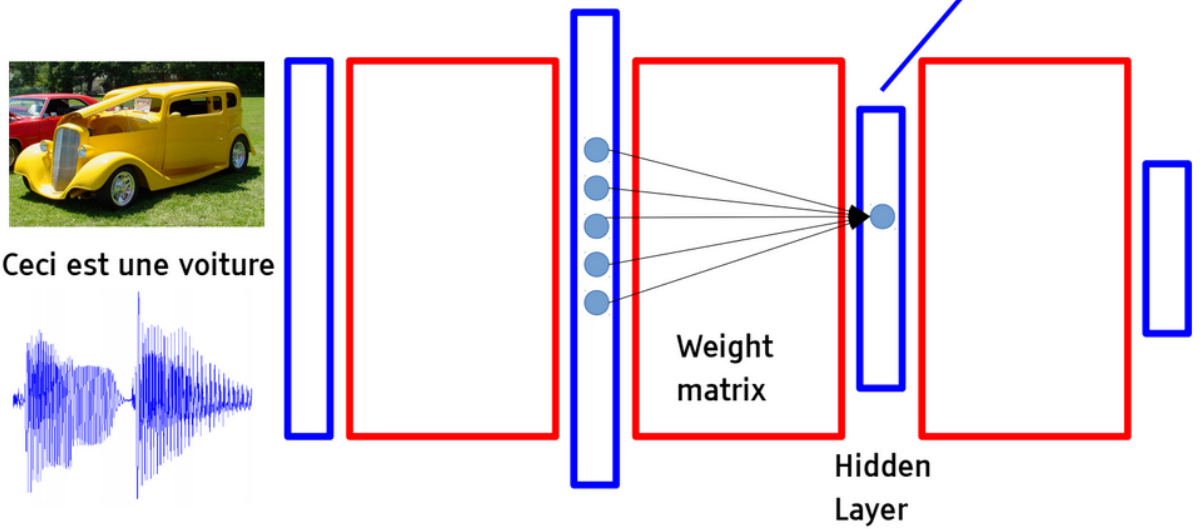
Thomas Pellegrini

Université de Toulouse; UPS; IRIT; Toulouse, France
CCT TSI 26 janvier 2017

- Multiple Layers of simple units
- Each units computes a weighted sum of its inputs
- Weighted sum is passed through a non-linear function
- The learning algorithm changes the weights

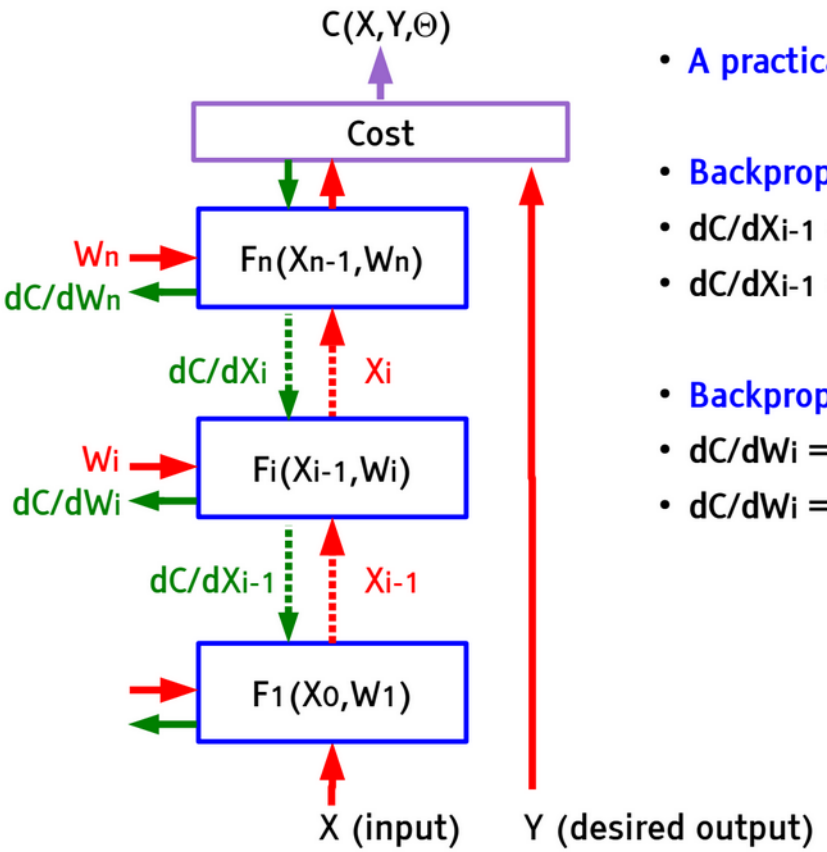
$$ReLU(x) = max(x, 0)$$





[Y. LeCun]

Gradients



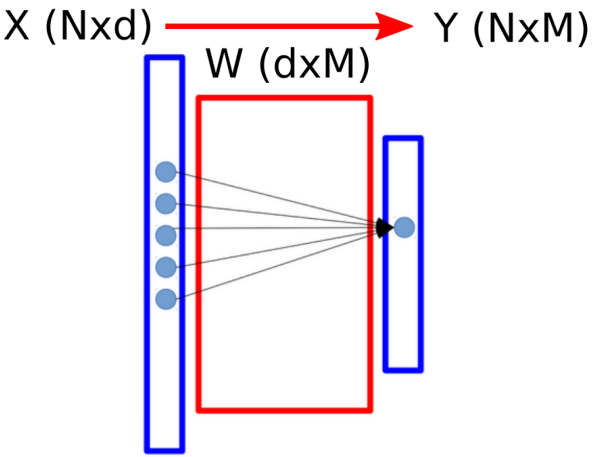
[Y. LeCun]

- **A practical Application of Chain Rule**

- **Backprop for the state gradients:**
- $dC/dX_{i-1} = dC/dX_i \,.\, dX_i/dX_{i-1}$
- $dC/dX_{i-1} = dC/dX_i \,.\, dF_i(X_{i-1},W_i)/dX_{i-1}$

- **Backprop for the weight gradients:**
- $dC/dW_i = dC/dX_i \,.\, dX_i/dW_i$
- $dC/dW_i = dC/dX_i \,.\, dF_i(X_{i-1},W_i)/dW_i$

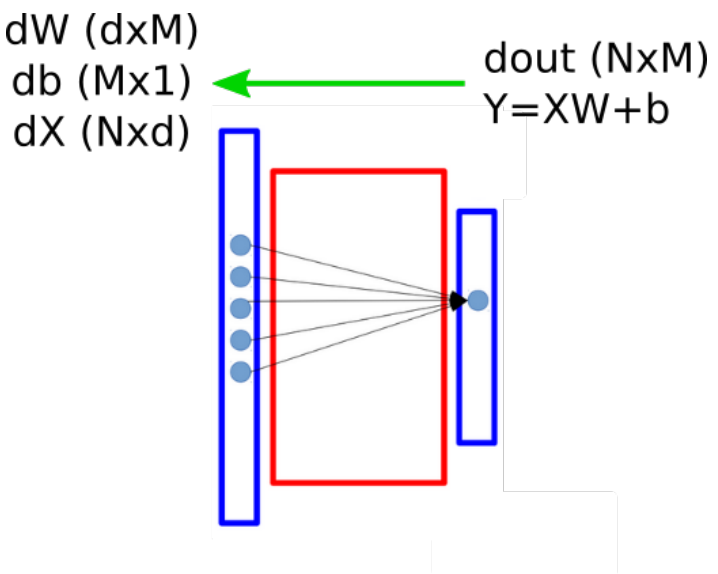
Affine layer: forward



$$Y = X \cdot W + b$$

```
def affine_forward(x, w, b):
    out = np.dot(x, w) + b
    cache = (x, w, b)
    return out, cache
```

Affine layer: backward



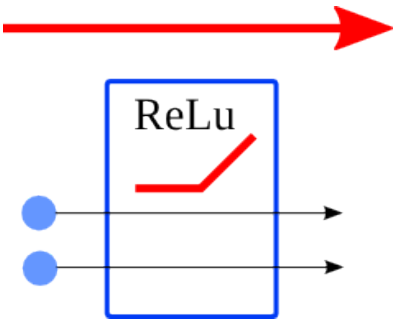
$$dW = X^t \cdot \text{dout}$$

$$db = \sum_{i=1}^{N} \text{dout}^{\text{i}}$$

$$dx = \text{dout} \cdot W^t$$

```
def affine_backward(dout, cache):
  x, w, b = cache
  dx = np.dot(dout, w.T)
  dw = np.dot(x.T, dout)
  db = np.sum(dout, axis=0)
  return dx, dw, db
```

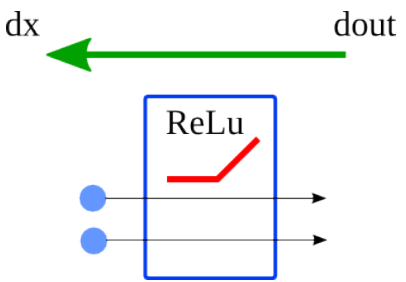
Non-linearity layer: ReLu forward



$$
\begin{aligned}
Y &= \max(0, X) \\
&= X * \mathbb{1}_{\{X>0\}} \\
&= X * [X > 0]
\end{aligned}
$$

```
def relu_forward(x):
  out = np.maximum(np.zeros((x.shape)), x)
  cache = x
  return out, cache
```

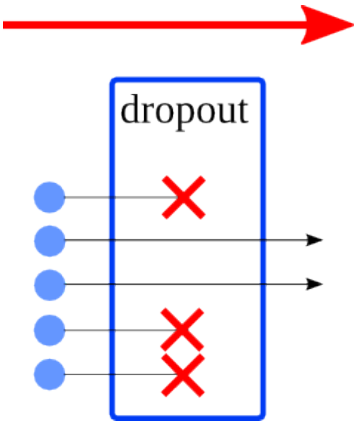
Non-linearity layer: ReLu backward



$$dx = [X > 0] * dout$$

```
def relu_backward(dout, cache):
  x = cache
  dx = dout * ((x>0)*1)
  return dx
```

Dropout layer: forward

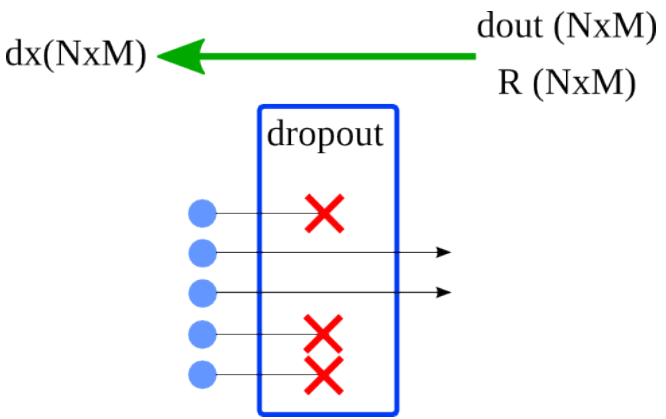


$$r_j \sim \text{bernoulli}(p)$$

$$Y = \mathbf{R} * X$$

```
def dropout_forward(x, p, mode):
  if mode == 'train':
    mask = (np.random.rand(*x.shape) < p) * 1
    out = x * mask
  elif mode == 'test':
    out = x
  cache = (p, mode, mask)
  out = out.astype(x.dtype, copy=False)
  return out, cache
```

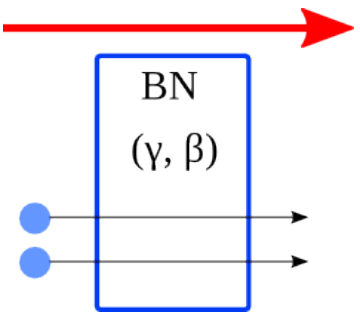
Dropout layer: backward



$$dx = \mathbf{R} * dout$$

```
def dropout_backward(dout, cache):
  p, mode, mask = cache
  if mode == 'train':
    dx = dout * mask
  elif mode == 'test':
    dx = dout
  return dx
```

Batch-normalization layer



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

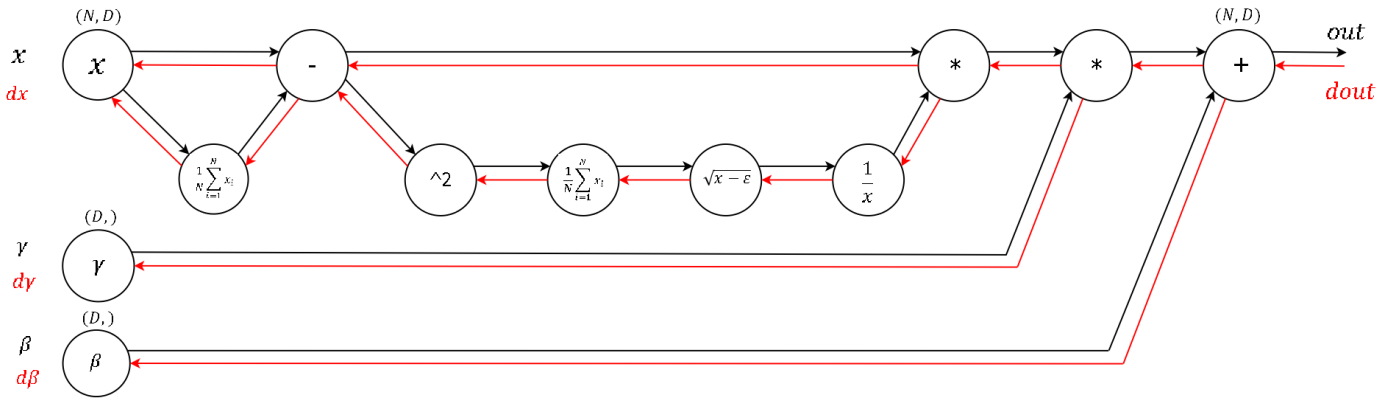
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch-normalization layer



Batch-normalization layer: forward with running mean

```
def batchnorm_forward(x, gamma, beta, bn_param):
  mode = bn_param['mode']
  eps = bn_param.get('eps', 1e-5)
  momentum = bn_param.get('momentum', 0.9)

  N, D = x.shape
  running_mean = bn_param.get('running_mean', np.zeros(D, dtype=x.dtype))
  running_var = bn_param.get('running_var', np.zeros(D, dtype=x.dtype))

  if mode == 'train':
    moy = np.mean(x, axis=0)
    var = np.var(x, axis=0)
    num = x - moy
    den = np.sqrt(var + eps)
    x_hat = num / den
    out = gamma * x_hat + beta
    running_mean = momentum * running_mean + (1. - momentum) * moy
    running_var = momentum * running_var + (1. - momentum) * var
    cache = (x, gamma, beta, eps, moy, var, num, den, x_hat)
  elif mode == 'test':
    x_hat = (x - running_mean)/np.sqrt(running_var + eps)
    out = gamma * x_hat + beta
    cache = (x, gamma, beta)
  bn_param['running_mean'] = running_mean
  bn_param['running_var'] = running_var
  return out, cache
```

Batch-normalization layer: backward with running mean

```
def batchnorm_backward(dout, cache):
  x, gamma, beta, eps, moy, var, num, den, x_hat = cache
  dbeta = np.sum(dout, axis=0)
  dgamma = np.sum(dout*x_hat, axis=0)

  dxhat = gamma * dout
  dnum = dxhat / den
  dden = np.sum(-1.0 * num / (den**2) * dxhat, axis=0)

  dmu = np.sum(-1.0 * dnum, axis=0)
  dvareps = 1.0 / (2 * np.sqrt(var + eps)) * dden

  N, D = x.shape
  dx = 1.0 / N * dmu + 2.0 / N * (x - moy) * dvareps + dnum

  return dx, dgamma, dbeta
```

From scores to probabilities

$$\text{scores: } \mathbf{f} = F_n(X_{n-1}, W_n)$$

Probability associated to a given class k:

$$P(y = k | W, \mathbf{X}) = \frac{\exp(f_k)}{\sum_{j=0}^{C-1} \exp(f_j)} = \mathrm{softmax}(\mathbf{f}, k)$$

```
def softmax(z):
  '''z: a vector or a matrix z of dim C x N '''
  z = z-np.max(z) # to avoid overflow with exp
  exp_z = np.exp(z)
  return exp_z / np.sum(exp_z, axis=0)
```

Categorical cross-entropy loss

$$\mathcal{L}(W) = -\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(W|y^i, \mathbf{x^i})$$

$$\mathcal{L}(W|y^i, \mathbf{x^i}) = -\log(P(y^i|W, \mathbf{x^i}))$$

Only the probability of the correct class is used in $\mathcal{L}$

Categorical cross-entropy loss: gradient

$$
\begin{aligned}
\nabla_{\mathbf{W_k}} \mathcal{L}(W|y^i, \mathbf{x^i}) &= \frac{\partial \mathcal{L}(W|y^i, \mathbf{x^i})}{\partial \mathbf{W_k}} \\
&= -\sum_{j=0}^{C-1} t_j^i \frac{\partial \log(z_j^i)}{\partial \mathbf{W_k}} \qquad \text{with } t_j^i = \mathbb{1}_{\{y^i = j\}} \\
&= -\sum_{j=0}^{C-1} t_j^i \frac{1}{z_j^i} \frac{\partial z_j^i}{\partial \mathbf{W_k}} \\
&= \dots \\
&= -\mathbf{x^i}(t_k^i - z_k^i) \\
&= \begin{cases} \mathbf{x^i}(z_k^i - 1) & \text{if } t_j^i = 1 \text{ (i.e., } y^i = k) \\ \mathbf{x^i} z_k^i & \text{if } t_j^i = 0 \text{ (i.e., } y^i \neq k) \end{cases}
\end{aligned}
$$

Categorical cross-entropy loss

```
def softmax_loss_vectorized(W, X, y, reg):
  """
  Softmax loss function, vectorized version.
  Inputs: same as softmax_loss_naive
  """
  # Initialize the loss and gradient to zero.
  loss = 0.0
  dW = np.zeros_like(W)
  D, N = X.shape
  C, _ = W.shape

  probs = softmax(W.dot(X)) # dim: C, N
  probs = probs.T # dim: N, C
  # compute loss only with probs of the training targets
  loss = np.sum(-np.log(probs[range(N), y]))
  loss /= N
  loss += 0.5 * reg * np.sum(W**2)

  dW = probs # dim: N, C
  dW[range(N), y] -= 1
  dW = np.dot(dW.T, X.T)
  dW /= N
  dW += reg * np.sum(W)

  return loss, dW
```

Our first modern network!

```
def affine_BN_relu_dropout_forward(x, w, b, gamma,\
  beta, bn_param, p, mode):

  network, fc_cache = affine_forward(x, w, b)
  network, bn_cache = batchnorm_forward(network, gamma,
    beta, bn_param)

  network, relu_cache = relu_forward(network)
  network, dp_cache = dropout_forward(network, p, mode)

  cache = (fc_cache, bn_cache, relu_cache, dp_cache)

  return network, cache

def affine_BN_relu_dropout_backward(...):
  ...
```

Our first modern network! Easier with a toolbox...

```
from lasagne.layers import InputLayer, DenseLayer,
  NonlinearityLayer, BatchNormLayer, DropoutLayer
from lasagne.nonlinearities import softmax

net = {}
net['input'] = InputLayer((None, 3, 32, 32))
net['aff'] = DenseLayer(net['input'], \
                num_units=1000, nonlinearity=None)
net['bn'] = BatchNormLayer(net['aff'])
net['relu'] = NonlinearityLayer(net['bn'])
net['dp'] = DropoutLayer(net['relu'])
net['prob'] = NonlinearityLayer(net['dp'], softmax)
```

Questions

- Which features are typically used as input?
- How to choose and design a model architecture?
- How to get a sense of what a model did learn?
- What is salient in the input that makes a model take a decision?

Examples in speech and singing birds

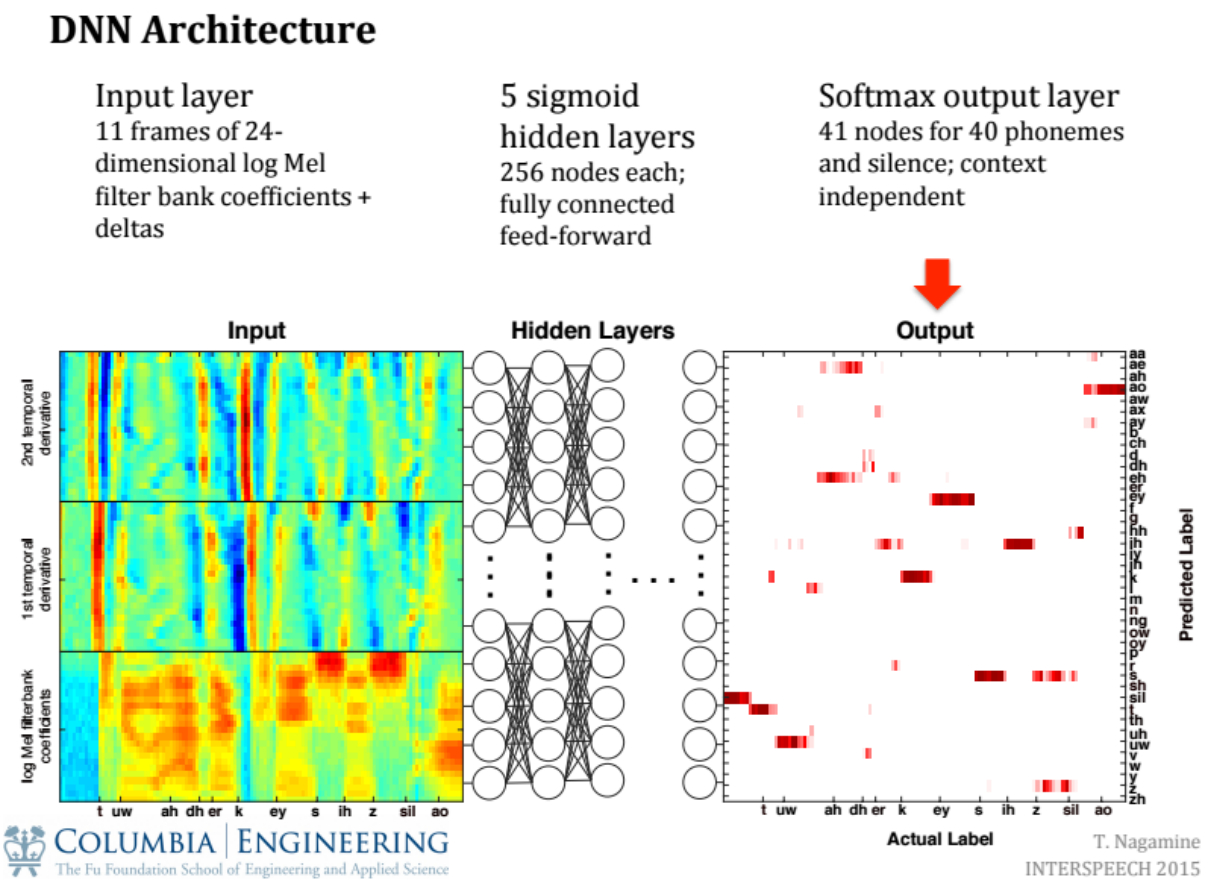
What features are typically used as input?

In audio applications: (log Mel) filter-bank coefficients most used!

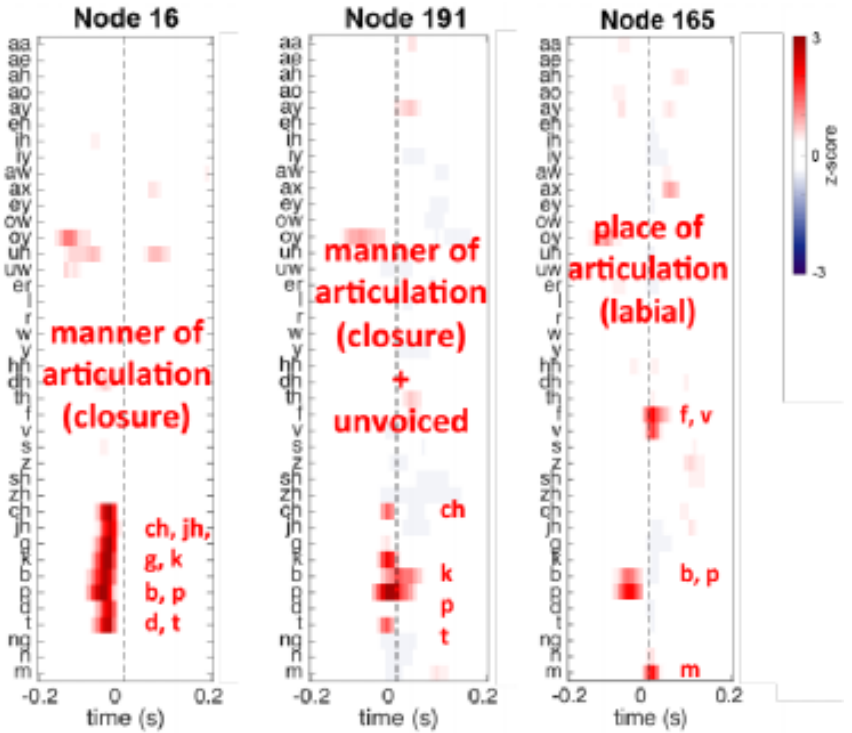
Others:

- Raw signal
- FFT coefficients (module)
- MFCCs usually outperformed by F-BANK coefficients

Phone recognition: DNN

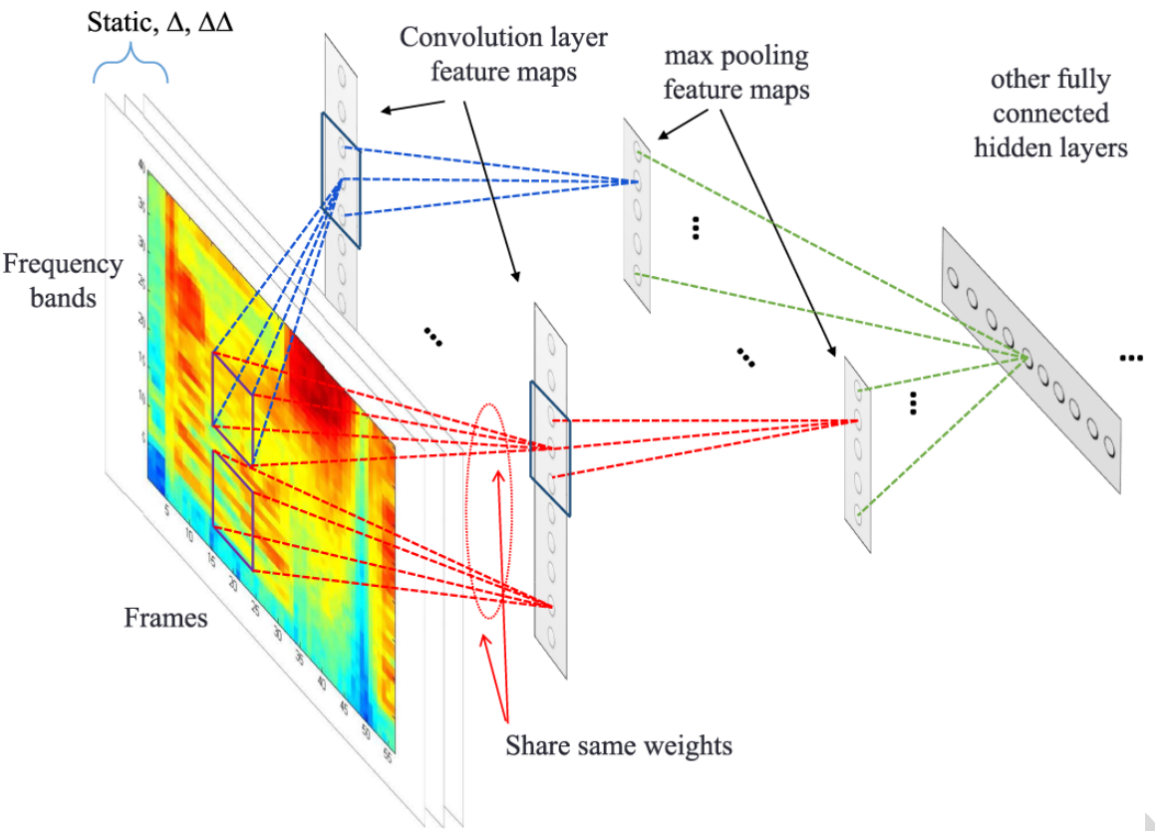


[Nagamine et al., IS 2015; Slide by T. Nagamine]



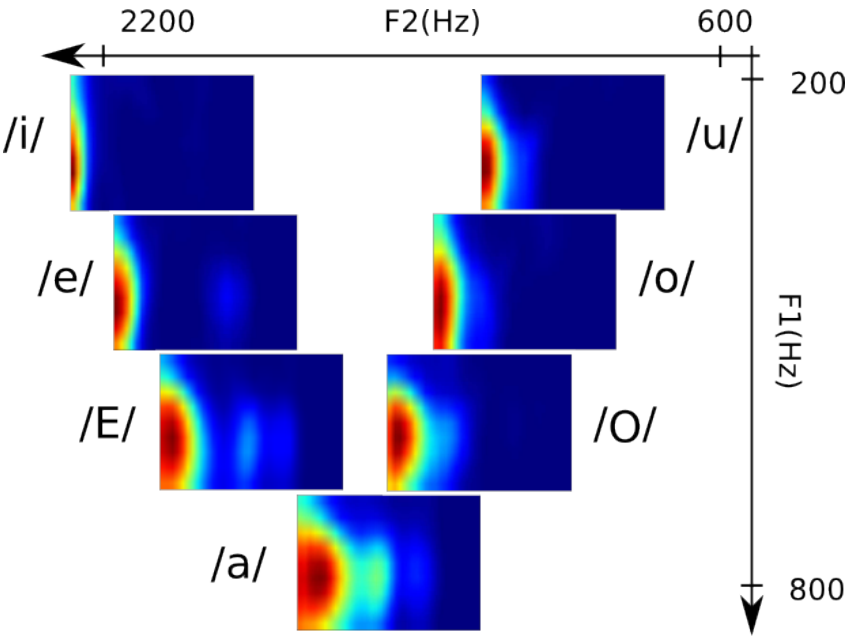
[Nagamine et al., IS 2015; Slide by T. Nagamine]

Phone recognition: CNN

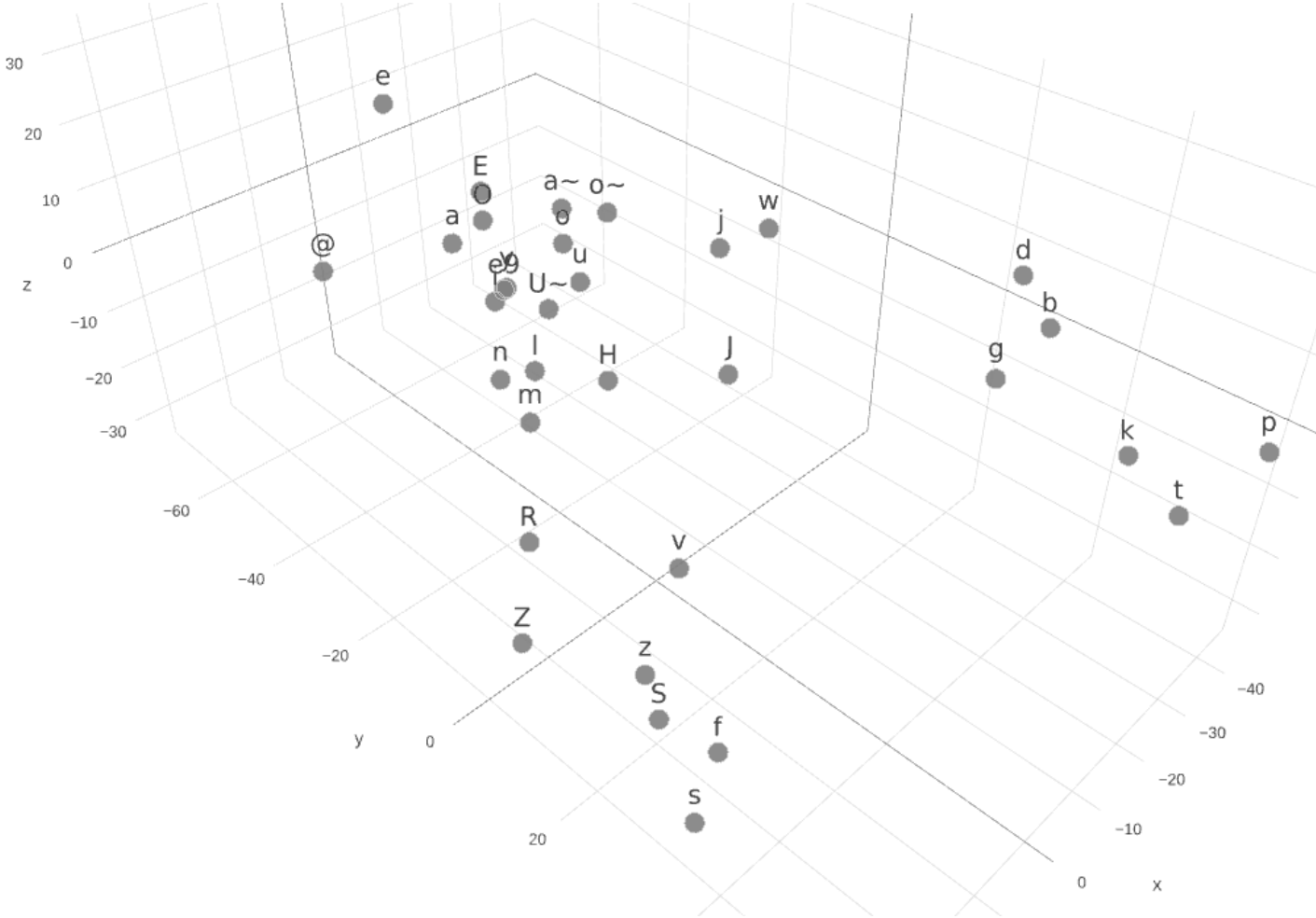


[Abdel-Hamid et al., TASLP 2014]

Convolution maps

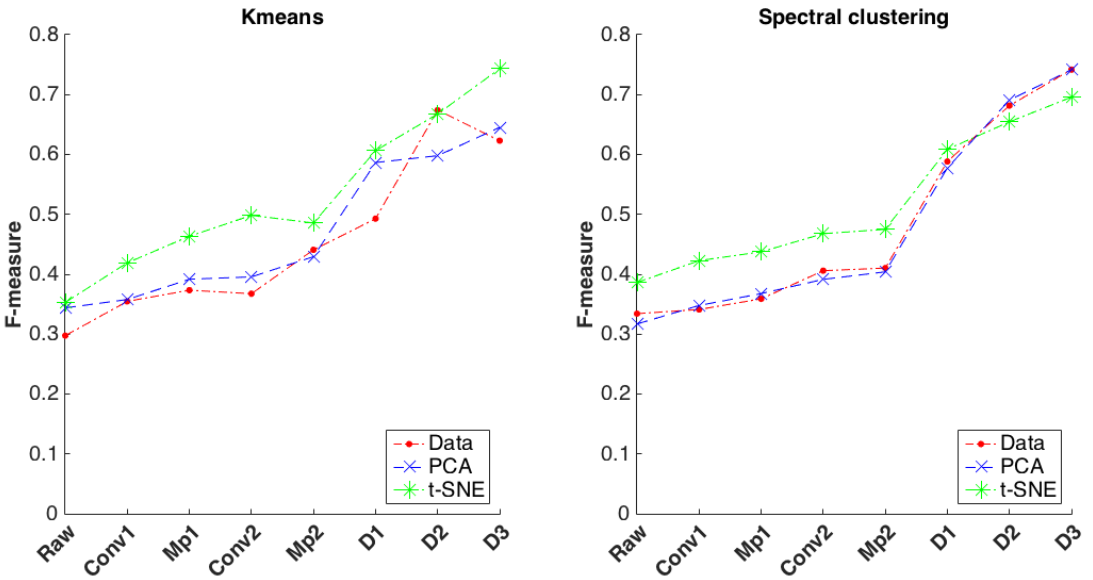


[Pellegrini & Mouysset, IS 2016]



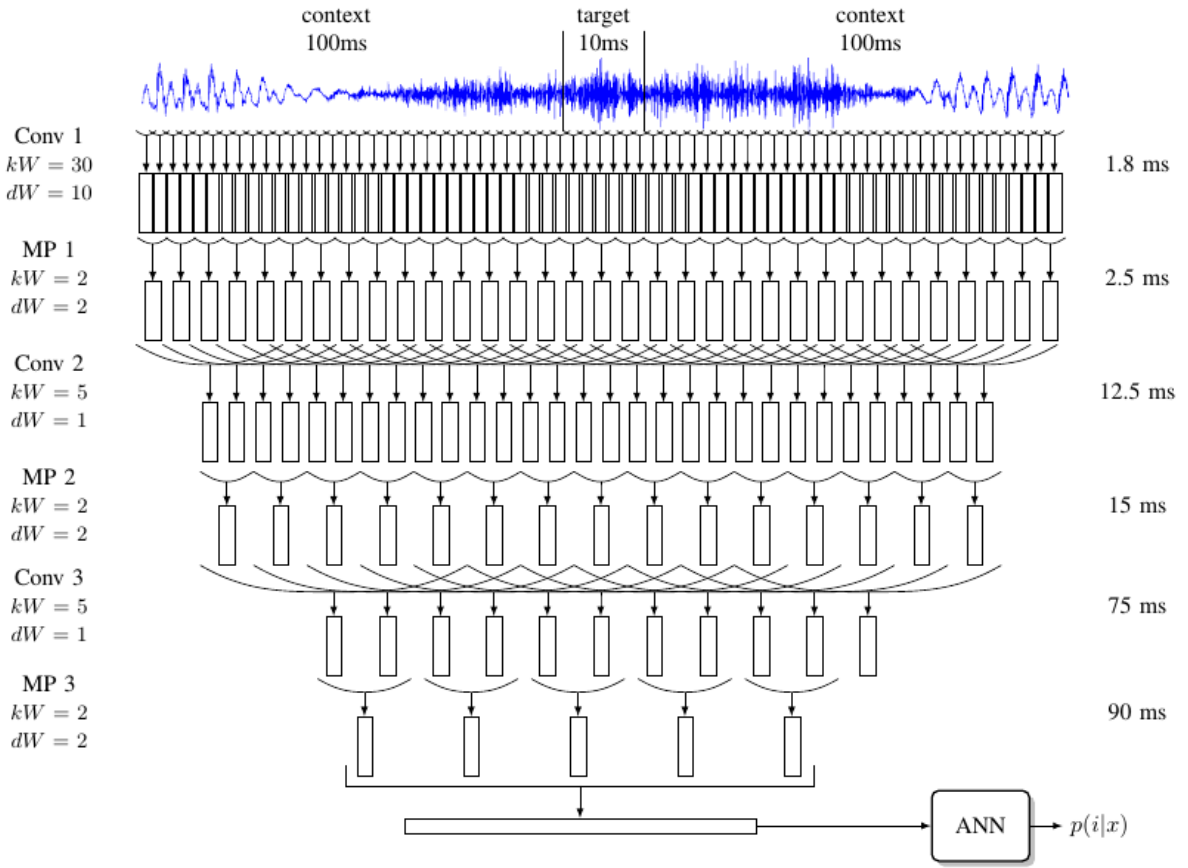
[Pellegrini & Mouysset, IS 2016]

Convolution maps



[Pellegrini & Mouysset, IS 2016]

Phone recognition: CNN with raw speech

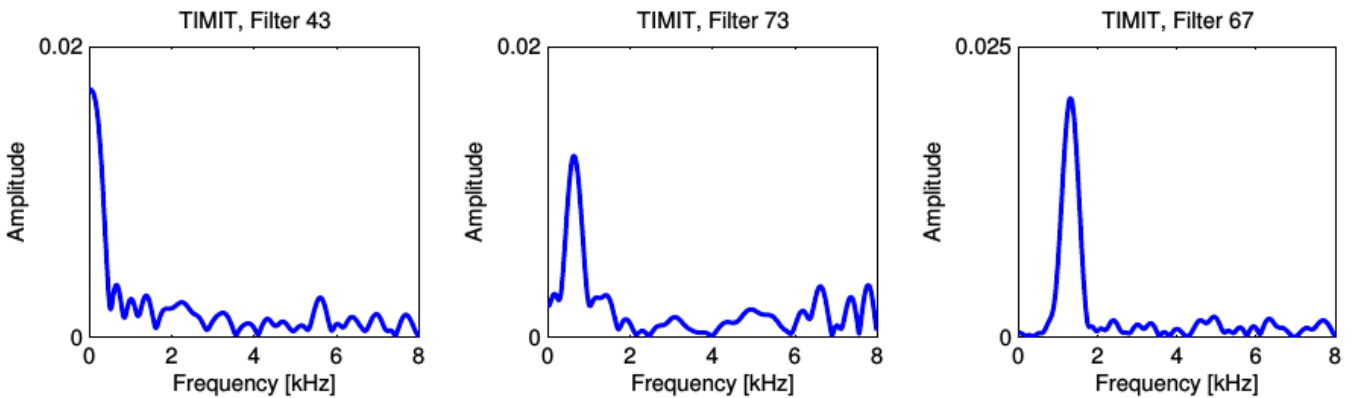


[Magimai-Doss et al., IS 2013 ; Slide by M. Magimai-Doss]

Phone recognition: CNN with raw speech

The first convolution layer can be seen as a bank of matching filters.

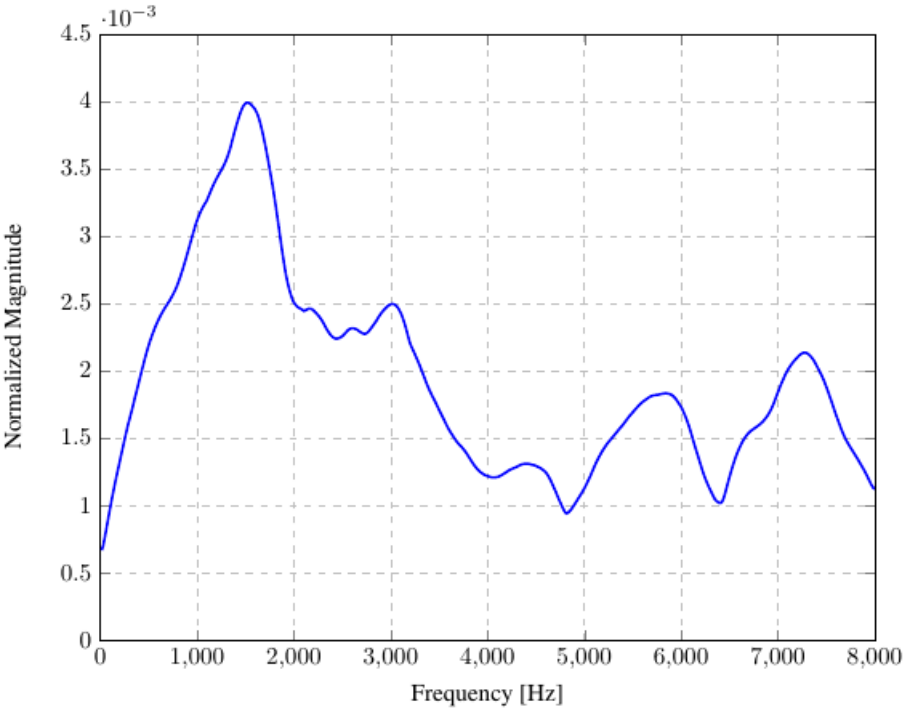
- Standard Mel filterbanks: 20-30 ms window
- Filters learned by the CNN: 2-4 ms window, 0.6 ms
- Found empirically, based on performance on the validation set.
- These filters respond to different frequency bands:



[Magimai-Doss et al., IS 2013 ; Slide by M. Magimai-Doss]

Phone recognition: CNN with raw speech

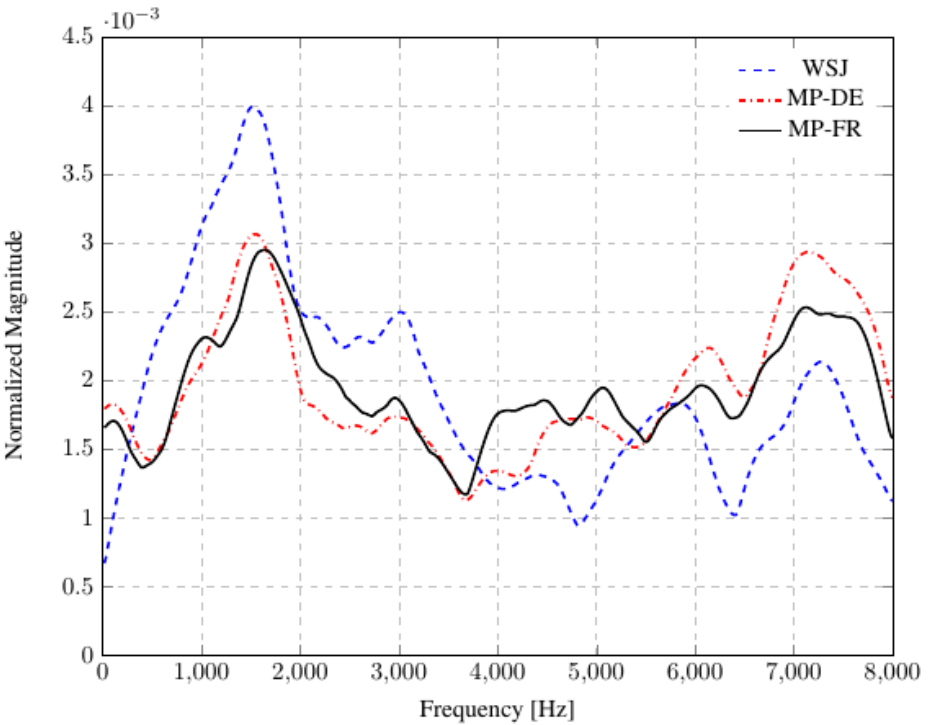
- Filter cumulative response: sum of all filters magnitude spectrum.



[Magimai-Doss et al., IS 2013 ; Slide by M. Magimai-Doss]

Phone recognition: CNN with raw speech

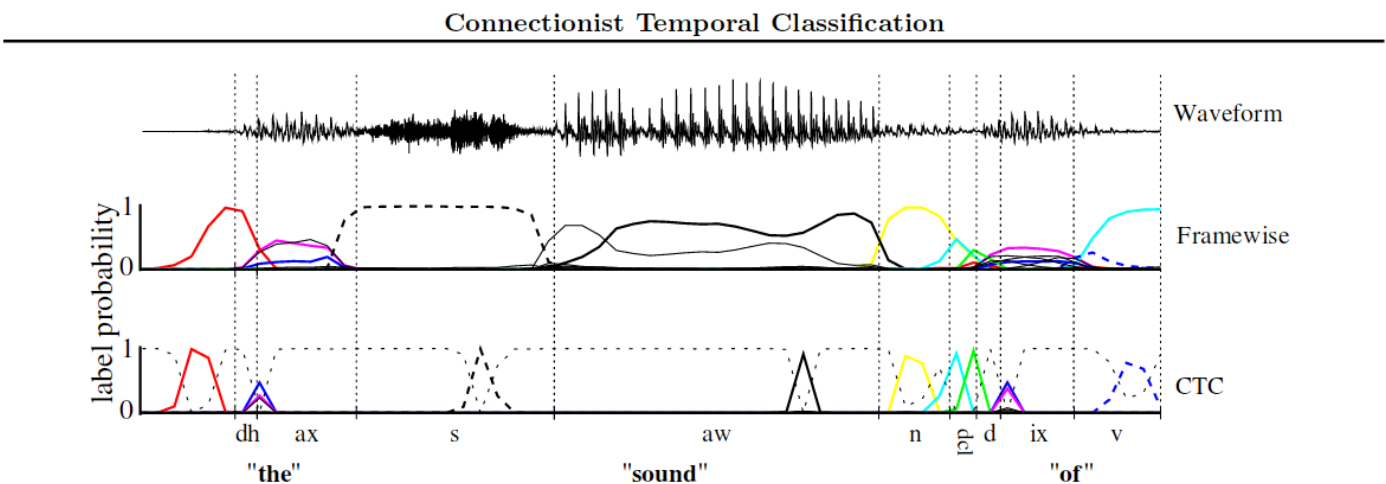
- Filter cumulative response: sum of all filters magnitude spectrum.



[Magimai-Doss et al., IS 2013 ; Slide by M. Magimai-Doss]

Handling time series

- Frame with context: decision at frame-level
- Pre-segmented sequences: TDNN, RNN, LSTM
- Sequences with no previous segmentation : Connectionist Temporal Classification loss [Graves, ICML 2006]



Recent convNets architectures

- Standard convNets

$$x_i = F_i(x_{i-1})$$

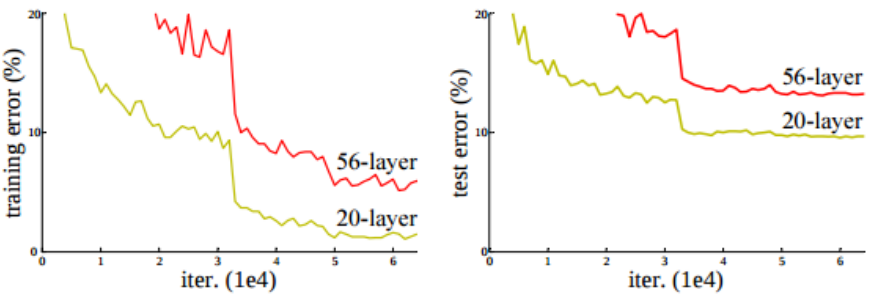


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

[He *et al*, CVPR 2016]

Recent convNets architectures

- Standard convNets [LeCun, 1995]

$$x_i = F_i(x_{i-1})$$

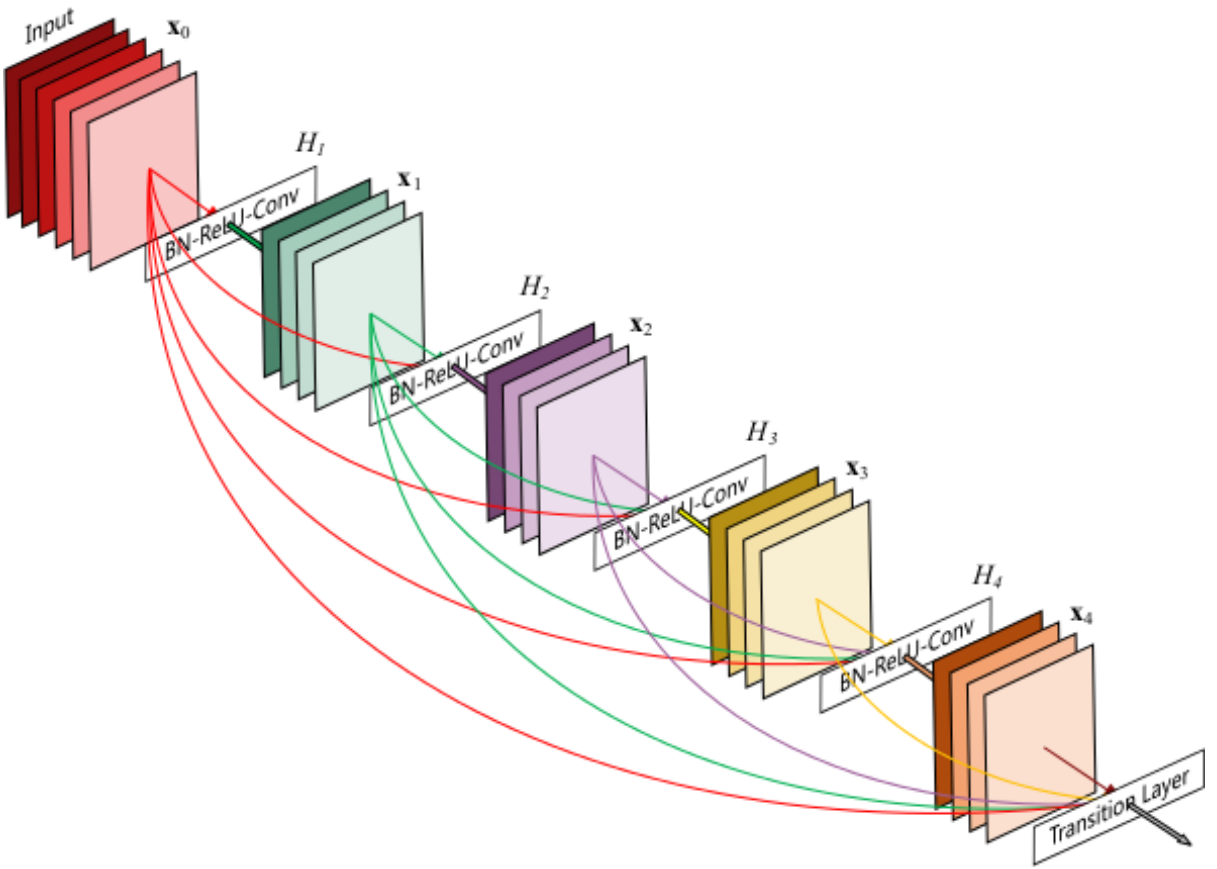
- Residual convNets [He *et al*, CVPR 2016]

$$x_i = F_i(x_{i-1}) + x_{i-1}$$

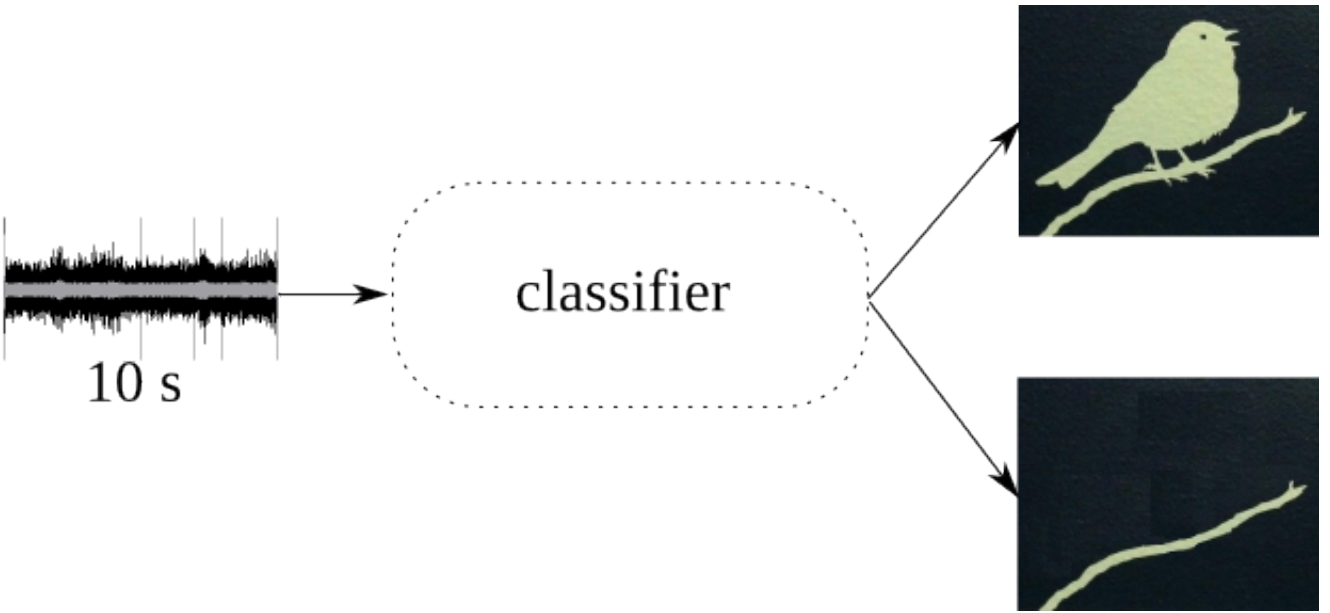
- Densely connected convNets [Huang *et al*, 2016]

$$x_i = F_i([x_0, x_1, \ldots, x_{i-1}])$$

DenseNets: dense blocks



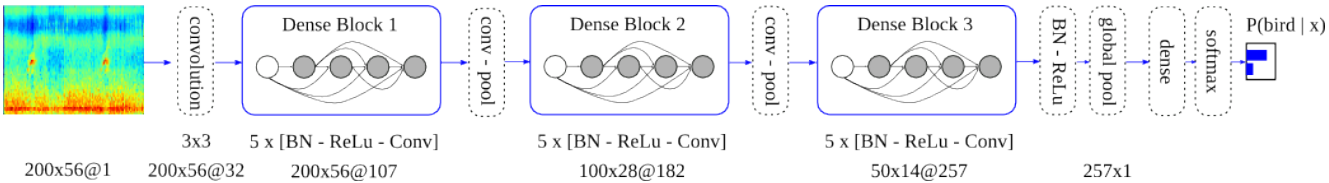
Bird Audio Detection challenge 2017



Bird Audio Detection challenge 2017

	Train	Valid	Test
Freefield1010	6,152	384	1,154
Warblr	6,800	500	700
Merged	14,806	884	0
Tchernobyl	_	_	8,620

Proposed solution: denseNets



- 74 layers
- 328K parameters
- Tchernobyl ROC (AUC) score: 88.79%

- Code densenet + saliency:

  https://github.com/topel/

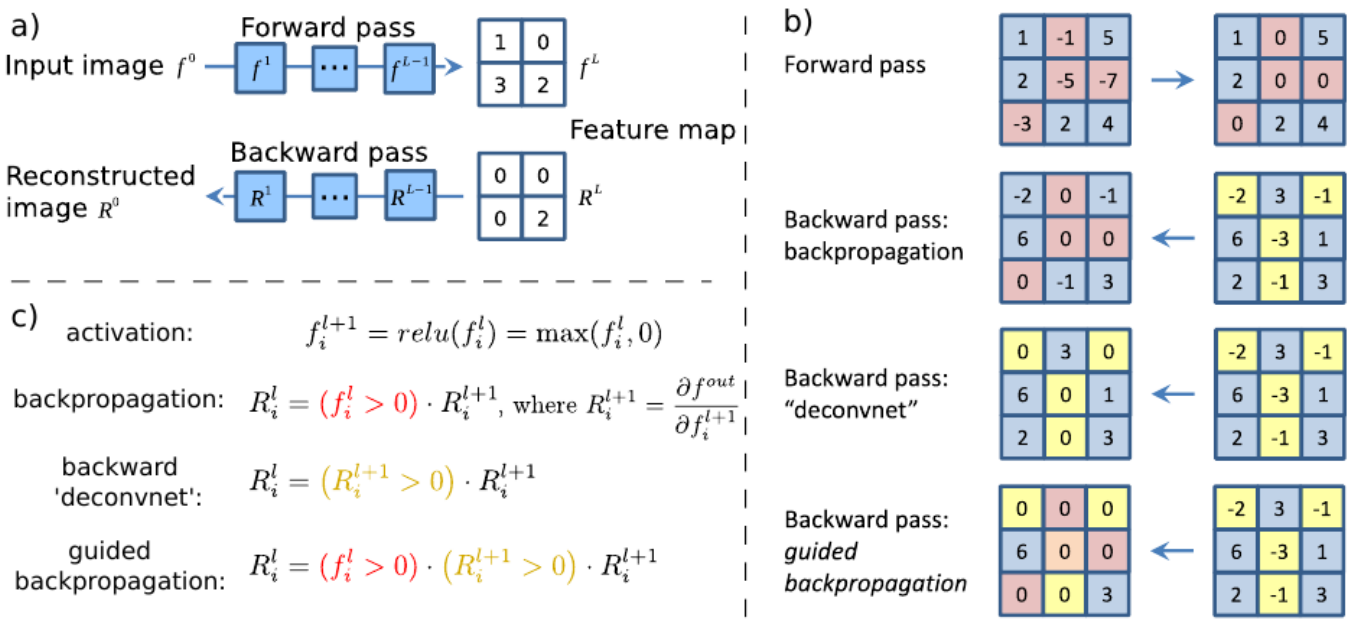
- Audio + saliency map examples:

  https://goo.gl/chxOPD

How to get a sense of what a model did learn?

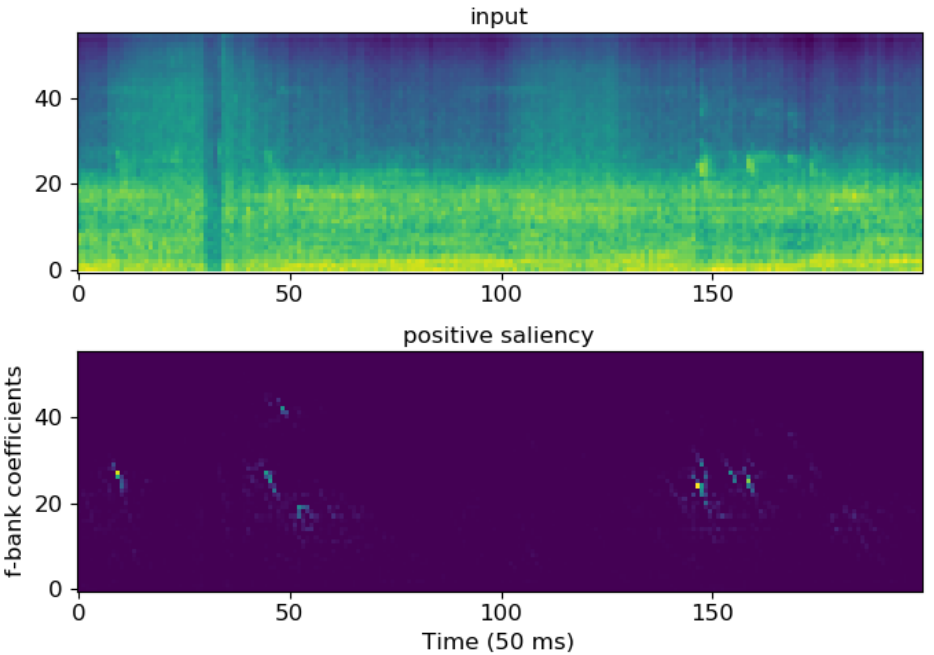
- Analysis of the weights (plotting), activation maps
- Saliency maps: which input elements (e.g., which pixels in case of an input image) need to be changed the least to affect the prediction the most?

Deconvolution methods



[Springenberg et al, ICLR 2015]

0070e5b1-110e-41f2-a9a5, P(bird): 0.966



References

Abdel-Hamid, TASLP 2014	Abdel-Hamid, Ossama, et al. ”Convolutional neural networks for speech recognition.” IEEE/ACM Transactions on audio, speech, and language processing 22.10 (2014): 1533-1545.
Graves, ICML 2006	Graves, Alex, et al. ”Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks.” Proceedings of the 23rd international conference on Machine learning. ACM, 2006.
He, CVPR 2016	He, Kaiming, et al. ”Deep residual learning for image recognition.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
Huang, 2016	Huang, Gao, et al. ”Densely connected convolutional networks.” arXiv preprint arXiv:1608.06993 (2016).
LeCun, 1995	LeCun, Yann, and Yoshua Bengio. ”Convolutional networks for images, speech, and time series.” The handbook of brain theory and neural networks 3361.10 (1995): 1995.
Nagamine, IS 2015	Nagamine, Tasha, Michael L. Seltzer, and Nima Mesgarani. ”Exploring how deep neural networks form phonemic categories.” INTERSPEECH 2015.
Palaz, IS 2013	Palaz, Dimitri, Ronan Collobert, and Mathew Magimai Doss. ”Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks.” arXiv preprint arXiv:1304.1018 (2013).
Pellegrini, IS 2016	Pellegrini, Thomas, and Sandrine Mouysset. ”Inferring phonemic classes from CNN activation maps using clustering techniques.” INTERSPEECH 2016 (2016): 1290-1294.
Springenberg, ICLR 2015	Springenberg, Jost Tobias, et al. ”Striving for simplicity: The all convolutional net.” arXiv preprint arXiv:1412.6806 (2014).