Pyramidal Deep Models for Computer Vision

Alfredo PETROSINO* and Ihsan ULLAH**

*Computer Vision and Pattern Recognition (CVPR) Lab
University of Naples Parthenope, Department of Science and Technology

** Data Mining and Machine Learning Group (DMMLG)
National University of Ireland Galway, Ireland
• Pyramidal Neurons
• Deep Pyramidal Models
• Deep Learning Libraries
• Practicing with Pyramidal Models
  • Pyramidal CNN
  • EmoP3D
• Question & Answers
Types of Pyramidal Models

- Partial, Reverse, or Non-Strict Pyramids
  - Spatial Pyramid Matching (SPM) [1] used it in pooling layer.
  - Or H. Fan et al. [2] use it in last Conv layer of CNN
  - P. Wang et al. [3] applied it in temporal pyramid pooling

- Strict and actual Pyramids
  - V. Cantoni & A. Petrosino’s Pyramidal Neural Network [6]
  - S. L. Phung & A. Bouzerdoum’s PyraNet [7]
  - B. Fernandes et al. I-PyraNet [8]
  - I. Ullah and A. Petrosino’s [9], and Spyr_CNN [10]

[11] E. D. Nardo et al., Accepted in ECCV Workshop on Brain Driven Computer Vision, 2018
Strict Pyramidal Models

VISMAC 2016

VISMAC 2018
## Strict Pyramidal Models Cont...

<table>
<thead>
<tr>
<th>Limitation in existing models</th>
<th>Strict Pyramidal model advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of general Criteria for designing a network - Increase number of layers and kernels</td>
<td>Decrease the number of kernels in each layer at a constant ratio - Results in specific n-layers</td>
</tr>
<tr>
<td>Ambiguity due to large number of features at final Convolutional layers</td>
<td>Reduced unambiguous features at final Convolutional layer.</td>
</tr>
<tr>
<td>Explosion of parameters</td>
<td>Reduces large number of parameters</td>
</tr>
<tr>
<td>No New Weighting Scheme</td>
<td>3DPyraNet introduced a new Position Oriented Weighing scheme</td>
</tr>
</tbody>
</table>
- Types of weighting schemes

(a) A Fully Connected Neural Network
Weights: 5x4 + 4x2 = 28

(b) Shared CNN Kernel
Weights: 3x1 + 3x1 = 6

(c) Locally connected Unshared Kernels in CNN
Weights: 3x4 + 3x2 = 18

(d) Shared position oriented locally connected
Weights: 5x1 + 4x1 = 9
• Weighting scheme in CNN and PyraNet

2D CNN [1,2]

Input Image/Map (5x5)

Weights Kernel (3x3) RF = 3x3

Output Map (3x3)

2D PyraNet [11]

Input Image/Map (5x5)

Weights Matrix (5x5) RF = 3x3

Output Map (3x3)
Each time a new partially shared Kernel calculates the weighted sum.
2D CNN

Same Kernel slid over whole Image/Map

Receptive Field

2D PyraNet

Each time a new partially shared Kernel calculates the weighted sum
2D CNN

Same Kernel slid over whole Image/Map

Receptive Field

2D PyraNet

Output Map

Weights Matrix

Input Image/Map

Each time a new partially shared Kernel calculates the weighted sum
2D CNN

Same Kernel slid over whole Image/Map

Receptive Field

2D PyraNet

Output Map

Weights Matrix

Input Image/Map

Each time a new partially shared Kernel calculates the weighted sum
Each time a new partially shared Kernel calculates the weighted sum.
2D CNN

Same Kernel slid over whole Image/Map

Receptive Field

2D PyraNet

Each time a new partially shared Kernel calculates the weighted sum

Output Map

Weights Matrix

Input Image/Map
2D CNN

Same Kernel slid over whole Image/Map

Receptive Field

2D PyraNet

Output Map

Weights Matrix

Input Image/Map

Each time a new partially shared Kernel calculates the weighted sum
2D CNN

Same Kernel slid over whole Image/Map

2D PyraNet

Each time a new partially shared Kernel calculates the weighted sum

Output Map
Weights Matrix
Input Image/Map
2D CNN

Same Kernel slided over whole Image/Map

2D PyraNet

Output Map

Weights Matrix

Input Image/Map

Receptive Field

Each time a new partially shared Kernel calculates the weighted sum
3D-Weight Scheme in EmoP3D

- 3D weighting scheme

Output feature map

Receptive Field

3D weight Matrix

Input Frames/Maps
Pyramidal Deep Models for Computer Vision: Lab Session (Caffe)
Deep Learning with Caffe

Outline

- Caffe
- Designing a Pyramidal Network for MNIST Dataset
  - Layers Definition Protobuf
  - Solver Protobuf
  - Shell Scripts
- Training & Testing
- Model Zoo
Convolution Architecture For Feature Extraction (Caffe)

- Developed by Berkeley Vision and Learning Center
- Open Framework, models, and examples for different deep learning models
- Initial focus was on vision now used also in different other application areas
- Fast Well tested code
- Tools, recipes, and demos
- Optimized conversion between CPU and GPU
- Website: [http://caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/)
- Github: [https://github.com/BVLC/caffe](https://github.com/BVLC/caffe) (Watch= 2211, Star= 25315, Fork= 15477, Contributors = 270)
- Titan: cuDNN Training: 20.25 secs / 20 iterations (5,120 images), Testing: 66.3 secs / validation set (50,000 images)
- K40: cuDNN Training is 19.2 secs / 20 iterations (5,120 images), Testing is 60.7 secs / validation set (50,000 images)
A Simple Network

DataLayer

W (data, diffs) → Inner Product Layer → X (Data, Diffs) → FC (Data, diffs) → SoftMax_LossLayer

Y (Data, Diffs)
Main Components

- Blob
- Layer
- Net
- Solver
Blobs

- An N-D Array of storing and communicating data
- Hold data, derivatives, and parameters
- Lazily allocate memory
- Shuttle between CPU and GPU
- Format in Convolution Layer:

Data
Number x K Channels x Height x Width
256 x 3 x 227 x 227

Weight (Filters)
N Output x K Input x Height x Width
32 x 3 x 7 x 7

Bias
N Output x K Input x Height x Width
32 x 1 x 1 x 1
Layers

- Main Unit of Computation implemented as layers
  * Data Access
  * Convolution
  * Pooling
  * Activation Functions
  * Loss Functions
  * Drop Out
  * Etc.
Designing a Network Cont...

- **Net**
  - A bundle of layers and blobs that connect them
  - A definition file tells Caffe to create and check the Net
  - Exposes Forward and Backward methods
  - An example file in Caffe:
    - `Lenet_train_test.prototxt`
    - `train_val.prototxt` (for Imagenet)
Available Layers, loss functions and activation functions:

- **Loss Functions:**
  - Linear Regression
  - Euclidean Loss
- **Classification**
  - SoftMaxWithLoss
  - Hinge Loss
- **Attributes/Multiclassification**
  - SigmoidCrossEntropyLoss
- **And NewTask**
  - More Layers (In future)

- **Layer Types:**
  - Convolution
  - Pooling
  - Normalization
  - InnerProduct etc.

- **Activation Functions**
  - Sigmoid
  - Tanh
  - ReLU
  - LReLU
  - PReLU
- **And more**
Example on MNIST Dataset

- MNIST
- A handwritten characters
- Size of 32x32
- Training set = 60000 (6000 per class)
- Test Set = 10000 (1000 per class)
Deep Learning model

- Data Layer for train and test

```python
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train.lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_test.lmdb"
    batch_size: 100
    backend: LMDB
  }
}
```

- Two ‘top’ represents two Outputs
- Phase represents that this layer is for testing or training layer
- Scale input in [0,1] range
Data Layer for train and test

- Conv1 Layer

```python
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler { type: "xavier"
    }
    bias_filler { type: "constant"
    }
  }
}
```

- Learning rate and decay multipliers for the filters
- Learning 20 Filters
- Each filter is 5x5
- Leave 1 pixel for each filtering step
- Initialize weights from Xavier
- Initialize a constant rather than 0
Example on MNIST “definition protobuf” Cont...

- Data Layer for train and test
  - Pool1 + Conv2 Layers

```protobuf
definition protobuf

layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

Selecting max value in a 2x2 neighborhood
Example on MNIST “definition protobuf” Cont...

- Data Layer for train and test
  - Pool2 and Fully Connected (ip1)

```protobuf
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

Having 500 Neurons
Data Layer for train and test

- Relu1 and Fully Connected (ip2)

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```
Example on MNIST Cont...

- Data Layer for train and test
  - Accuracy and SoftMax Loss Layer

```yaml
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}

layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```
Deep Neural Network training

- **Solver**
  - Calls Forward / Backward and updates net parameters
  - Periodically evaluates model on the test network(s)
  - Snapshots model and solver state

- **Solver types available**
  - SGD, AdaDelta, AdaGrad, Nestrov, RMSprop, Adam
Example for MNIST

- Defining “Solver protobuf”

```protobuf
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"

# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100

test_interval: 500

# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005

# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75

display: 100

# The maximum number of iterations
max_iter: 10000

# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"

# solver mode: CPU or GPU
solver_mode: GPU
```
Pre-Processing of Data
- Once defining a model and its solver protbuf is done, then you need to preprocess the data.

Data Ingest Formats/choice of data Layers
- LevelDB or LMDB database
- In-memory (C++ and python only)
- HDF5
- Images files

Pre-Processing Tools
- LevelDB/LMDB creation from raw images
- Training and validation set creation with shuffling
- Mean Image Generation
- Mirroring

Data Transformation
- Image Cropping
- Resizing
- Scaling

These functions are available in $CAFFE_ROOT/build/tools
Example for MNIST

- Training a network
  - We can do it in two ways.
    - *Writing each command on shell*
  - Or
    - *Writing a shell script and than executing that.*
Let’s Practice

- Go to Caffe root where you have installed Caffe.
  - cd $CAFFE_ROOT

- Follow the following steps
  1. To get MNIST dataset
     * ./data/mnist/get_mnist.sh

  2. To create LMDB train and test files
     * ./examples/mnist/create_mnist.sh

  3. Training and Testing the model
     * ./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt -gpu 0

     * Or If you have train_lenet.sh script than
       - ./examples/mnist/train_lenet.sh
Let’s Practice Cont...

- Download Dataset
  - `./data/mnist/get_mnist.sh`
  - Loads data from a website and gunzip it

```sh
#!/usr/bin/env sh
# This script downloads the mnist data and unzips it.

DIR="$( cd $(dirname "$0") ; pwd -P )"

cd $DIR

echo "Downloading..."

for fname in train-images-idx3-ubyte train-labels-idx1-ubyte t10k-images-idx3-ubyte t10k-labels-idx1-ubyte do
  if [ ! -e $fname ]; then
    wget --no-check-certificate http://yann.lecun.com/exdb/mnist/$fname.gz
    gunzip $fname.gz
  fi
done
```
Training and Testing the model

- Convert to Caffe Readable Format (LMDB)
  
  ```
  #!/usr/bin/env sh
  # This script converts the mnist data into lmdb/leveldb format,
  # depending on the value assigned to $BACKEND.

  EXAMPLE=examples/mnist
  DATA=data/mnist
  BUILD=build/examples/mnist

  BACKEND="lmdb"

  echo "Creating $BACKEND..."

  rm -rf $EXAMPLE/mnist_train_${BACKEND}
  rm -rf $EXAMPLE/mnist_test_${BACKEND}

  $BUILD/convert_mnist_data.bin $DATA/train-images-idx3-ubyte \
  $DATA/train-labels-idx1-ubyte $EXAMPLE/mnist_train_${BACKEND} --backend=${BACKEND}

  $BUILD/convert_mnist_data.bin $DATA/t10k-images-idx3-ubyte \
  $DATA/t10k-labels-idx1-ubyte $EXAMPLE/mnist_test_${BACKEND} --backend=${BACKEND}

  echo "Done."
  ```
Let’s Practice Cont...

- **Start Training**
  
  ```bash
  ./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt --gpu 0
  ```

  - When you run the code, you will see a lot of messages flying by like this:

  ```
  I1203 net.cpp:66] Creating Layer conv1
  1203 net.cpp:76] conv1 <- data
  1203 net.cpp:101] conv1 -> conv1
  1203 net.cpp:116] Top shape: 20 24 24
  1203 net.cpp:127] conv1 needs backward computation.
  ```

- These messages provide details about each layer, its connection and its output shape, helpful in debugging.
Let’s Practice Cont...

- Monitoring the training process

Output to stdout:

```
I0814 14:44:33.410693 2026435328 solver.cpp:294] Iteration 0, Testing net (#0)  Test net output #0: accuracy = 0.0931
I0814 14:44:35.697690 2026435328 solver.cpp:343] Test net output #1: loss = 2.30247 (* 1 = 2.30247 * loss)
I0814 14:44:35.718361 2026435328 solver.cpp:214] Iteration 0, loss = 2.30184
I0814 14:44:35.718392 2026435328 solver.cpp:229] Train net output #0: loss = 2.30184 (* 1 = 2.30184 * loss)
I0814 14:44:35.718400 2026435328 solver.cpp:486] Iteration 0, lr = 0.001
I0814 14:44:41.550972 2026435328 solver.cpp:214] Iteration 100, loss = 1.72121
I0814 14:44:41.550999 2026435328 solver.cpp:229] Train net output #0: loss = 1.72121 (* 1 = 1.72121 * loss)
I0814 14:44:41.551007 2026435328 solver.cpp:486] Iteration 100, lr = 0.001
I0814 14:44:47.383386 2026435328 solver.cpp:214] Iteration 200, loss = 1.73216
I0814 14:44:47.384315 2026435328 solver.cpp:229] Train net output #0: loss = 1.73216 (* 1 = 1.73216 * loss)
I0814 14:44:47.384324 2026435328 solver.cpp:486] Iteration 200, lr = 0.001
I0814 14:44:53.220012 2026435328 solver.cpp:214] Iteration 300, loss = 1.30751
I0814 14:44:53.220772 2026435328 solver.cpp:229] Train net output #0: loss = 1.30751 (* 1 = 1.30751 * loss)
I0814 14:44:53.220782 2026435328 solver.cpp:486] Iteration 300, lr = 0.001
I0814 14:44:59.053917 2026435328 solver.cpp:214] Iteration 400, loss = 1.16627
I0814 14:44:59.053948 2026435328 solver.cpp:229] Train net output #0: loss = 1.16627 (* 1 = 1.16627 * loss)
I0814 14:44:59.053956 2026435328 solver.cpp:486] Iteration 400, lr = 0.001
I0814 14:45:04.833677 2026435328 solver.cpp:294] Iteration 500, Testing net (#0)  Test net output #0: accuracy = 0.5589
I0814 14:45:06.778378 2026435328 solver.cpp:343] Test net output #1: loss = 1.2699 (* 1 = 1.2699 * loss)
```

Network will stop when it reaches the maximum number of Iterations.
Pre-trained Deep Neural Network

- **Standard, Compact Model format**
  - Caffe train produces a binary `.caffemodel` file

- **Easily integrate trained models into data pipeline**
  - Deploy against new data using command line, python or Matlab wrappers
Interfaces

- **Matlab**
  - Scores = net.forward(input_data);
  - One thing to note that images are in BGR channel instead of RGB.

- **Python**
  - Out = net.forward()
There are several trained models available on Caffe Model Zoo

- Network in Network
- VGG
- GoogleNet
- AlexNet
- Etc.

You can download and retrain/test your data

https://github.com/BVLC/caffe/wiki/Model-Zoo
Designing pyramidal CNN for MNIST/Cifar10 based on:

- “A deep CNN architecture with specific number of layers and filters works on a specific task, while keeping the number of layers same and reversing the number of filters, provided that it forms pyramidal structure, the resultant pyramidal architecture will result in same or better performance”[10].

- For Example
  * Caffe LENET = 20-50-500-10
  * Pyramidal LENET = 50-20-500-10

  * Cifar-10 = 32-32-64-10
  * Pyramidal Cifar-10 = 64-32-32-10
Pyramidal CNN Structure.

Input

C1 = 384 x 3 x 11 x 11

C2 = 320 x 5 x 5

C3 = 256 x 3 x 3

C4 = 224 x 3 x 3

C5 = 192 x 3 x 3

Out

F7 = 3840

F6 = 4096
Pyramidal Models and their Impact

MNIST
(A) Caffe_LENET = 20-50-500-10
(B) SPyr_Rev_LENET = 50-20-500-10
(C) SPyr_LENET = 35-15-500-10
(D) SPyr_LENET* = 25-15-100-10
(E) SPyr_LENET** = 100-68-100-10

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>Size in MB</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>431080</td>
<td>1.685MB</td>
<td>99.1</td>
</tr>
<tr>
<td>(B)</td>
<td>191830</td>
<td>0.749MB</td>
<td>99.13</td>
</tr>
<tr>
<td>(C)</td>
<td>48910</td>
<td>0.191MB</td>
<td>99.14</td>
</tr>
<tr>
<td>(D)</td>
<td>35150</td>
<td>0.137MB</td>
<td>99.1</td>
</tr>
<tr>
<td>(E)</td>
<td>219250</td>
<td>0.857MB</td>
<td>99.24</td>
</tr>
</tbody>
</table>
Pyramidal Deep Models for Computer Vision: Lab Session (Tensorflow)
Tensorflow

- Open-source Machine Learning library
- Developed by Google
- Flexible architecture (works on CPU, GPU, and TPU)
- No need for writing back-propagation algorithm for new structures. Graph maintain the forward pass which help in backward pass
- Big Community
- Not as fast as Caffe
- [https://www.tensorflow.org/](https://www.tensorflow.org/)
Installing Tensorflow

- Go to Tensorflow website
  https://www.tensorflow.org/install/

Follow the steps according to your Operating System on machine
Installing TensorFlow

We've built and tested TensorFlow on the following 64-bit laptop/desktop operating systems:

- macOS 10.12.6 (Sierra) or later.
- Ubuntu 16.04 or later
- Windows 7 or later.
- Raspbian 9.0 or later.

Although you might be able to install TensorFlow on other laptop or desktop systems, we only support (and only fix issues in) the preceding configurations.

The following guides explain how to install a version of TensorFlow that enables you to write applications in Python:

- Installing TensorFlow on Ubuntu
- Installing TensorFlow on macOS
- Installing TensorFlow on Windows
- Installing TensorFlow on a Raspberry Pi
- Installing TensorFlow from Sources

Many aspects of the Python TensorFlow API changed from version 0.n to 1.0. The following guide explains how to migrate older TensorFlow applications to Version 1.0:

- Transitioning to TensorFlow 1.0

The following guides explain how to install TensorFlow libraries for use in other programming languages. These APIs are aimed at deploying TensorFlow models in applications and are not as extensive as the Python APIs.

- Installing TensorFlow for Java
Convolutional Layer
with tf.variable_scope('conv1') as scope:
    kernel = _variable_with_weight_decay('weights',
        shape=[5, 5, 3, 64],
        stddev=5e-2,
        wd=None)
    conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
    biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
    pre_activation = tf.nn.bias_add(conv, biases)
    conv1 = tf.nn.relu(pre_activation, name=scope.name)
    _activation_summary(conv1)
Pooling Layer

```
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                      padding='SAME', name='pool1')
```

Normalization Layer (Local Response Normalization)

```
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm1')
```
with tf.variable_scope('local3') as scope:
    # Move everything into depth so we can perform a single matrix multiply.
    reshape = tf.reshape(pool2d, [images.get_shape().as_list()[0], -1])
    dim = reshape.get_shape()[1].value
    weights = tf.contrib.layers.glorot_uniform_initializer()('weights', shape=[dim, 384],
                                         stddev=0.04, wd=0.004)
    biases = tf.Variable(tf.zeros([384]), name='biases')
    local3 = tf.nn.relu(tf.matmul(reshape, weights) + biases, name=scope.name)
    _activation_summary(local3)

# local4
with tf.variable_scope('local4') as scope:
    weights = tf.contrib.layers.glorot_uniform_initializer()('weights', shape=[384, 192],
                                         stddev=0.04, wd=0.004)
    biases = tf.Variable(tf.zeros([192]), name='biases')
    local4 = tf.nn.relu(tf.matmul(local3, weights) + biases, name=scope.name)
    _activation_summary(local4)

# linear layer(WX + b),
# We don't apply softmax here because
# tf.nn.sparse_softmax_cross_entropy_with_logits accepts the unscaled logits
# and performs the softmax internally for efficiency.
with tf.variable_scope('softmax_linear') as scope:
    weights = tf.Variable(tf.random_normal([192, NUM_CLASSES],
                                         stddev=1.0/tf.sqrt(192.0), wd=0.0)
    biases = tf.Variable(tf.zeros([NUM_CLASSES]),
                           tf.constant_initializer(0.0))
    softmax_linear = tf.add(tf.matmul(local4, weights), biases, name=scope.name)
    _activation_summary(softmax_linear)

return softmax_linear
EmoP3D for Emotion Recognition

Spatio-Temporal Pyramidal Deep Learning in Emotion Recognition (EmoP3D) [11]

- A pyramidal model for emotion recognition based on 3DPyraNet[9]

- Position oriented weight sharing scheme resulting in fewer parameters

- Tested on eNTERFACE and YouTube dataset

- Achieved state-of-the-art results.
eNTERFACE Dataset Samples

Fear

Happiness

Surprise

Anger
EmoP3D Architecture
EmoP3D

- Requirement:
  - Python
  - EmoP3D Repository
  - TensorFlow

- You must have Python if not Install it from their main webpage.
EmoP3D: Our Repository

- EmoP3D repository is located on Github:  
git clone https://github.com/CVPRLab-UniParthenope/EmoP3D.git

- Now go to EmoP3D directory to avoid mess on your file system 😊

- Create a virtual environment for (optional):  
  
  ➢ virtualenv --system-site-packages venv-vismac18

  ➢ source ~/EmoP3D/venv-vismac18/bin/activate # bash, sh, ksh, or zsh
  
  source ~/EmoP3D /venv-vismac18/bin/activate.csh # csh or tcsh

  . ~/EmoP3D / venv-vismac18/bin/activate.fish # fish
EmoP3D: Installing Tensorflow

- Use python package to install tensorflow:
  - It works with pip 8.1;
- CPU version:
  - pip install tensorflow;
- GPU version:
  - pip install tensorflow-gpu;
- Check if it is successful installed:
  (venv-vismac18)$ python -c "import tensorflow as tf; print(tf.__version__)"
EmoP3D: Installing additional libraries

- EmoP3D needs:
  - Numpy;
  - tqdm;
  - packaging;

- Use requirements file to install all libraries:
  - pip install -r requirements.txt
3D-Weighted Sum Layer

- A strict model is used in the graph (strict_norm_net);
- Each ws3d_layer applies a 3D Weighted Sum on the input:

```python
ws3d_layer(input, out_filters, rf=(3, 4, 4),
strides=(1, 1, 1, 1, 1), act_fn=lrelu,
initializer=None, weight_decay=None,
padding="VALID",
data_format="NDHWC",
log=False, reuse=False,
name="weighted_sum_3d_layer")
```
EmoP3D: Layer

3D-Weighted Sum Layer Cont...

- Out_filters: number of the output filters;
- Rf: receptive field size on each dimension (depth, height, width);
- Strides: sliding step over the input (first and last values MUST BE 1, like tensorflow standard);
- Act_fn: Activation function;

Please don’t touch data_format and reuse they are experimental;
EmoP3D: Layer

3D-Weighted Sum Layer Cont...

- Feature_maps: 3;
- Act_fn: lrelu;
- Initializer: tf.contrib.layers.variance_scaling_initializer(factor=1.0, mode='FAN_AVG', uniform=True, dtype=tf.float32).
  ✓ Xavier* weight initialization

Normalization layer applies a standardization over each frame along the depth;

* Xavier Glorot and Yoshua Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in 13th Inter. Conference on Artificial Intelligence and Statistics (AISTATS) 2010
EmoP3D: Layer

3D-Pooling Layer

- Each pool3d_layer applies the 3D Pooling Layer and the affine transformation to avoid translation problems due to biological neuron:
  ```python
  pool3d_layer(input_data, weight_depth=3, rf=(3, 2, 2), strides=(1, 1, 2, 2, 1), act_fn=lrelu, initializer=None, weight_decay=None, pool_type=max_pool3d, padding="VALID", data_format="NDHWC", reuse=False, log=False, name="pooling_3d_layer")
  ```
3D-Pooling Layer Cont...

- **Weight_depth**: same of first element of \texttt{rf} (it will be removed in future version);
- **rf**: receptive field size on each dimension (depth, height, width);
- **Strides**: sliding step over the input (first and last values MUST BE 1);
- **Pool_type**: available \texttt{max_pool3d} and \texttt{avg_pool3d};
- **Act_fn**: lrelu;
- **Pool_type**: \texttt{max_pool3d};
EmoP3D: Layer

3D-Pooling Layer Cont...

- **Initializer:** tf.contrib.layers.variance_scaling_initializer(factor=1.0, mode='FAN_AVG', uniform=True, dtype=tf.float32)
  - Xavier* weight initialization.

Don’t change data_format, just reuse as it is

---

* Xavier Glorot and Yoshua Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in 13th Inter. Conference on Artificial Intelligence and Statistics (AISTATS) 2010
EmoP3D: Train and Test

Train_3dpyranet.py

- We use TensorFlow flags to set up training parameters and hyper parameters;
- Train_3dpyranet has the right default values to work with emotion recognition:
  - If you want to experiment, you can check arguments in the documentation on GitHub;
- One needs to set only train/test path to the dataset:
  - train_path/train_labels_path;
  - val_path/val_labels_paths;
  - save_path;
Train_3dpyranet.py Cont...

- prepare_dataset():
  - Read the input;
  - Prepare information for training (number of epochs, batch size and so on);
  - Normalize dataset if needed;
  - Generate batch iterators;

- It is possible to use random input just to check if everything is set in the right way:
  - Use random_dataset();
EmoP3D Result for eNTERFACE

Confusion matrix

True label

Anger

Disgust

Fear

Happiness

Sadness

Surprise

Predicted label

Anger

Disgust

Fear

Happiness

Sadness

Surprise

219

19

13

11

7

17

0.77

0.07

0.05

0.04

0.02

0.06

10

20

17

2

4

0.05

0.09

0.08

0.01

0.02

162

172

13

8

10

0.75

0.72

0.09

0.06

0.07

6

3

101

108

4

16

11

11

10

0.04

0.02

0.72

0.78

0.03

0.08

0.01

0.03

2

4

10

15

0.01

0.03

0.08

0.02

0.07

1

4

12

20

3

88

0.01

0.03

0.09

0.16

0.02

0.69

25

50

75

100

125

150

175

200
## EmoP3D Result for eNTERFACE Cont...

<table>
<thead>
<tr>
<th>Model</th>
<th>eNTERFACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVER-Geometric</td>
<td>41.59%</td>
</tr>
<tr>
<td>KCMFA</td>
<td>58%</td>
</tr>
<tr>
<td>AVER-CNN</td>
<td>62%</td>
</tr>
<tr>
<td>EmoP3D</td>
<td>71.47%</td>
</tr>
</tbody>
</table>
Any Questions?