Training a CNN for emotion detection

Talk given at course
Machine Learning in HPC@GRNET

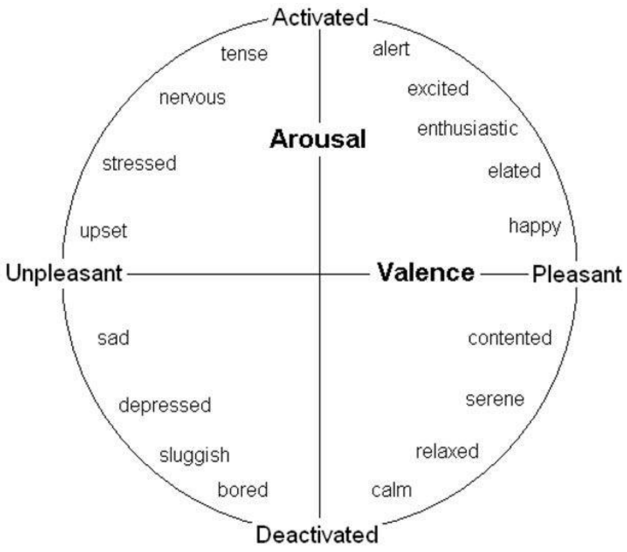
Dimitrios Michail
Dept. of Informatics & Telematics
Harokopio University of Athens, Greece
michail [at] hua.gr

Who we are

- Based on work in progress
  - DeeLeaVER: Deep Learning for Video Emotion Recognition
- Authors:
  - Evan B. Markou,
  - Dimitrios Michail,
  - Iraklis Varlamis
- Affiliation
  - Dept. of Informatics & Telematics,
  - Harokopio University of Athens,
  - Greece

Emotion Models

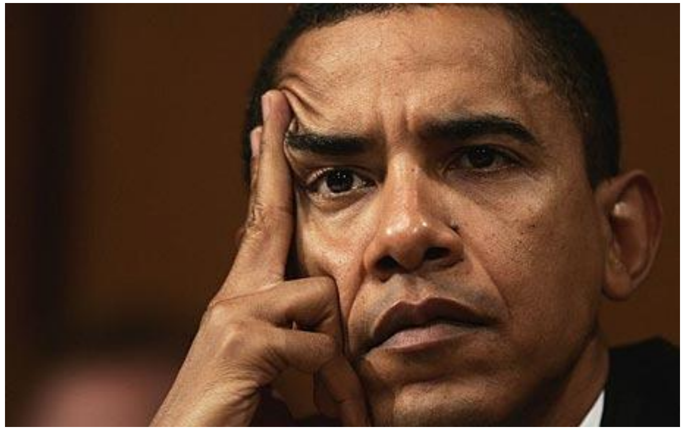
- Categorical theory
  - happiness, anger, surprise, sadness, disgust and fear
- Dimensional Theory
  - valence & arousal



- Facial Action Coding System (EMFACS)
  - correlates muscular face activities (Action Units) with the sentiment expressions

The Problem

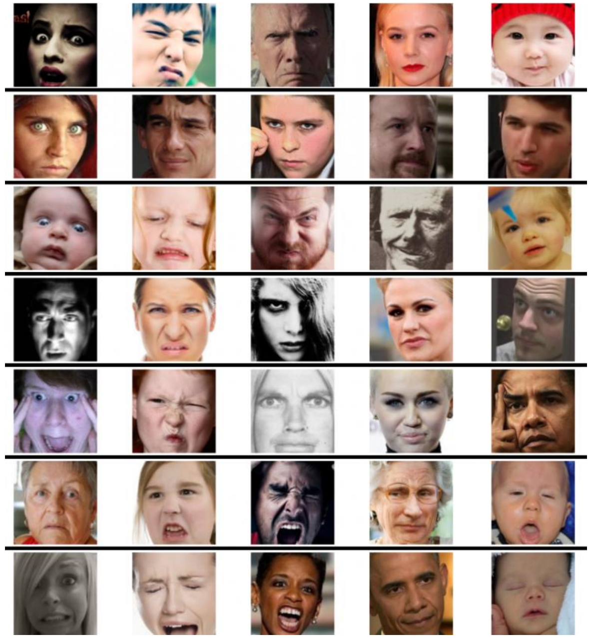
- Given a facial expression image
  - with its location inside a bigger image



- Determine emotions in terms of both categorical and dimensional models

Dataset

- AffectNet contains 1M images annotated with
  - 11 discrete emotion categories (categorical model)
  - valence and arousal scores in the range of [-1, 1]
- Manually annotated image in training and validation sets (test set is not released)

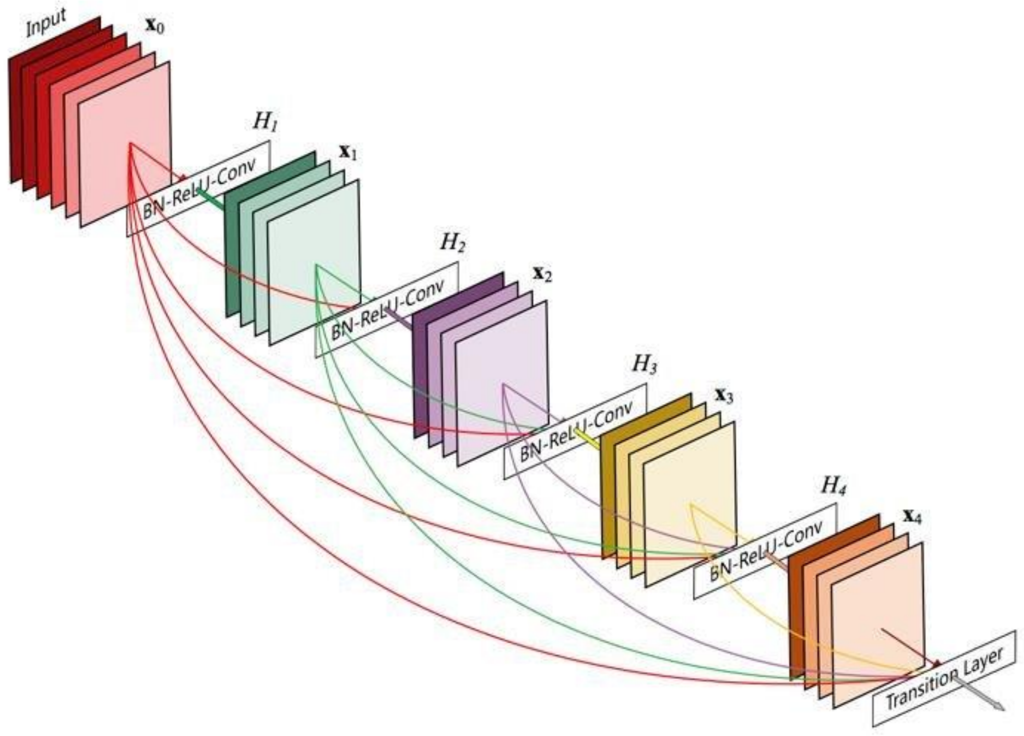


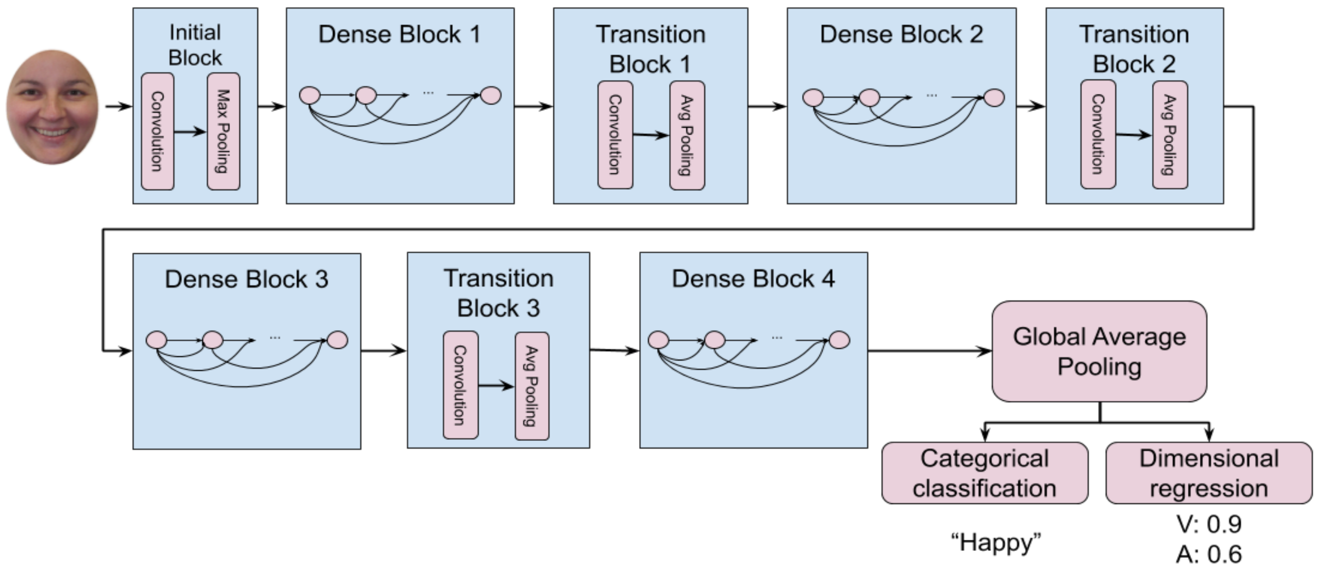
Neutral	75374
Happy	134915
Sad	25959
Surprise	14590
Fear	6878
Disgust	4303
Anger	25382
Contempt	4250
None	33588
Uncertain	12145
Non-Face	82915
Total	420299

Ali Mollahosseini, Behzad Hasani, and Mohammad H. Mahoor, "AffectNet: A New Database for Facial Expression, Valence, and Arousal Computation in the Wild", IEEE Transactions on Affective Computing, 2017.

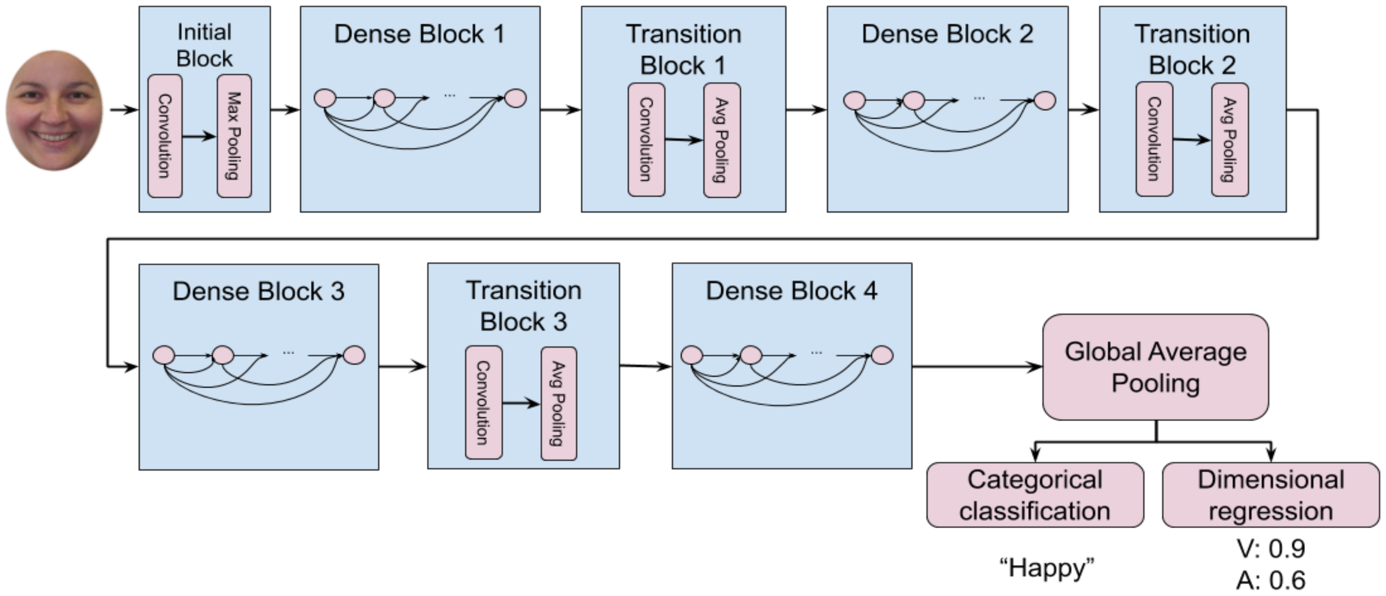
ML Architecture

- Based on DenseNet [Huang et al. 2017]
- Joint model (categorical and valence/arousal)
- The layers outlook follows DenseNet-161 with growth rate of k = 32.
- Each dense block possesses a different number of layers (6, 12, 24, 16).
- Also added bottleneck layers inside each dense block, and a compression of 0.5 in the transition layers.





Loss Functions



- Categorical -> (weighted) softmax cross-entropy
- Regression -> Concordance Correlation Coefficient (CCC) [Mollahosseinit et al. 2017]
- Train with joint loss function (weighted average, 50-50)

Optimizer & Learning Rate

- SGDR (Stochastic Gradient Descent with Restarts) [Loshchilov & Hutter ‘2016]
- Total 48 epochs -- 2 HPC Nodes x 2 GPUs
- Batch Size 64 -- Initial learning rate 0.025

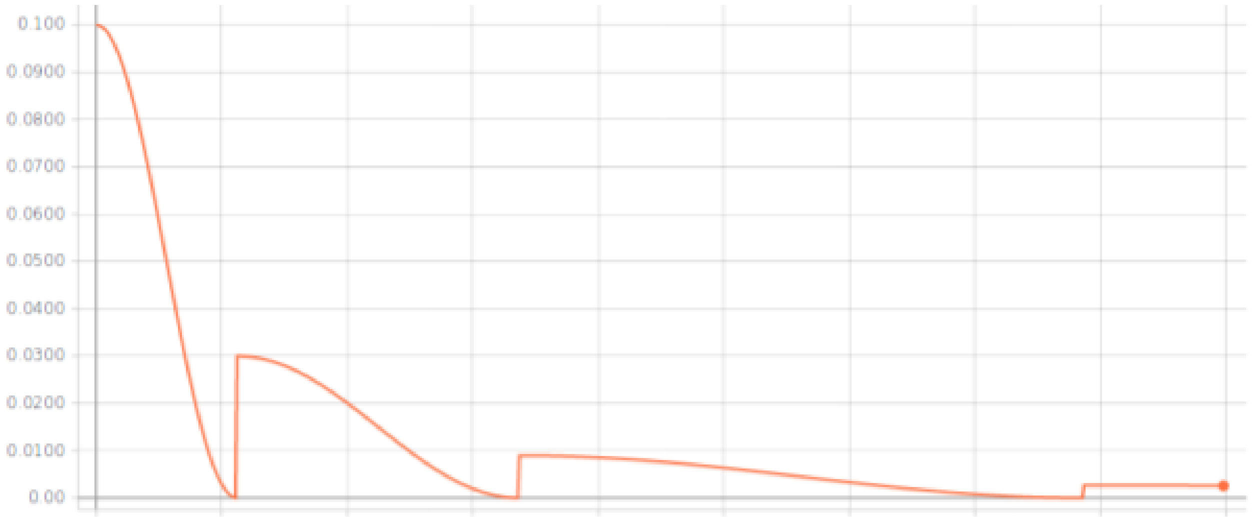


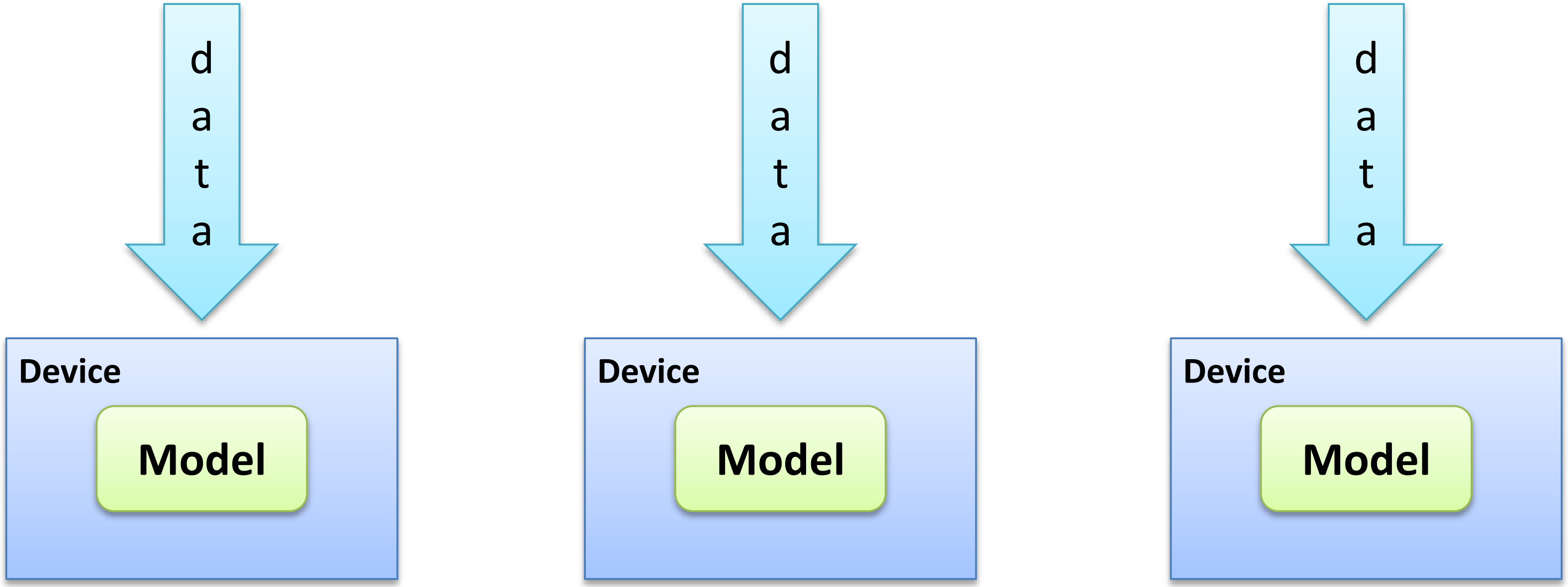
Figure 4. SGDR learning rate scheduling policy

- The first decay steps were executed for 12 epochs
- after that, they continued for yet another 12 epochs with 70% smaller learning rate.
- this policy was followed up until the end of the 48 epochs.

Technology Stack

- TensorFlow 1.10
  - at the time 1.12 had some problems on Aris but `tf.data.experimental` contained backports
  - at the time 2.x was beta
- TFrecords and Data Pipelines
- Horovod for distributed computation

Data Parallelism



Horovod

- Horovod [Sergeev & Del Balso ‘2018]
  - Open-sourced by Uber
  - Uses MPI concepts (size, rank, local rank, allreduce, allgather & broadcast)
  - Supports all major frameworks (including TensorFlow)
  - Uses the ring-allreduce algorithm created by Baidu
  - Uses the nvidia nccl-2 tool to ensure peer-to-peer GPU connectivity
- Distributed Optimizer
  - delegates gradient computation to the original optimizer, averages gradients using allreduce or allgather, and applies averaged gradients

Horovod Usage

- Basic steps from the Horovod website:
  1. Run hvd.init() to initialize horovod
  2. Pin each GPU to a single process
  3. Scale the learning rate by the number of workers
     - Effective batch size in synchronous distributed training is scaled by the number of workers. An increase in learning rate compensates for the increased batch size.
  4. Wrap optimizer in hvd.DistributedOptimizer
  5. Broadcast the initial variable states (random weights or checkpoint restore) from rank 0 to all other processes
  6. Modify code to save checkpoints only on worker with rank 0

Optimizer

```python
def sgd_w_optimizer(loss, lr, use_nesterov, momentum, params):
    """
    Use the SGD with Momentum optimizer (optionally with nesterov) to minimize the loss function.

    :param loss: The calculated loss function based on the true labels and logits to be passed in SGD optimizer
    :param lr: The learning rate. Could be fixed or use a learning rate annealing scheduler for a decayed learning rate
    :param momentum: The momentum value
    :param use_nesterov: A boolean value that use nesterov optimizer or not
    :param params: The json hyperparameter file
    :return: the train operation to be passed into session for execution
    """
    # learning rate scheduler on effective learning rate
    weight_decay_multiplier = lr / params.initial_learning_rate * hvd.size()  # equal with learning rate multiplier
    weight_decay = params.weight_decay * weight_decay_multiplier

    optimizer = tf.contrib.opt.MomentumWOptimizer(weight_decay, lr, momentum, use_nesterov=use_nesterov)
    global_step = tf.train.get_global_step()

    # Add Horovod Distributed Optimizer
    optimizer = hvd.DistributedOptimizer(optimizer)

    # Batch norm requires update ops to be added as a dependency to the train_op
    update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
    with tf.control_dependencies(update_ops):
        # Minimize the loss of the list of variables in the graph under the key GraphKeys.TRAINABLE_VARIABLES
        train_op = optimizer.minimize(loss, global_step=global_step)

    return train_op
```

Horovod GPU Pinning

```
# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))
```

- We use data-parallelism
- Make every process see only one GPU, as '/gpu:0'

Training

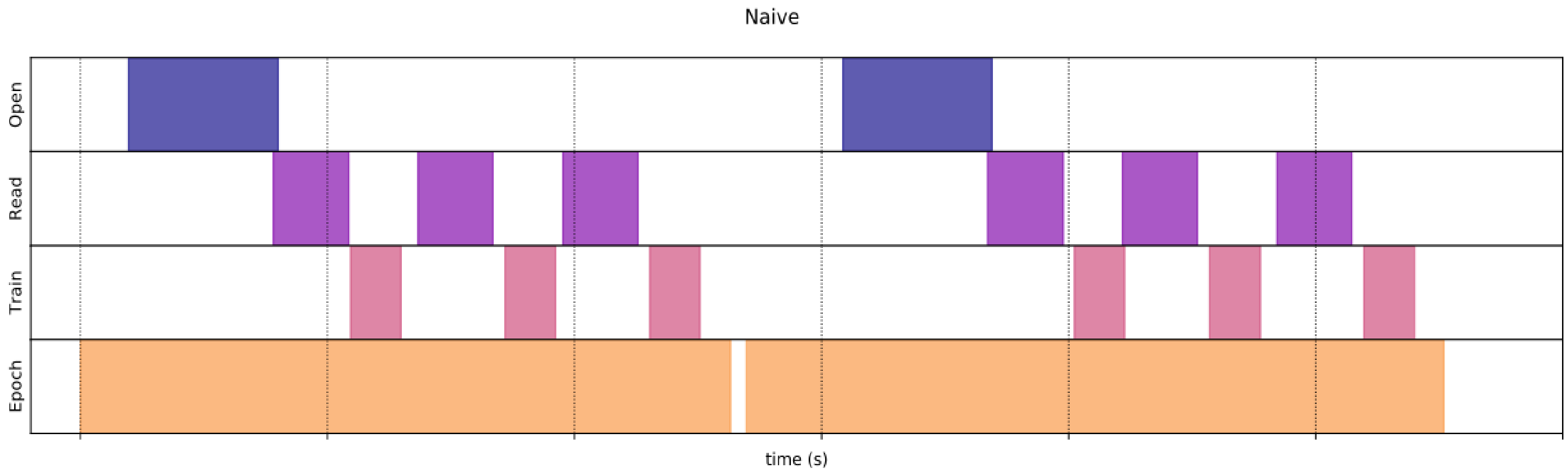
```
# Assume data parallelism and GPU pinning
device = '/gpu:0'

# define the model layers
logits = _build_model(training, trainable, inputs, params)

with tf.device(device):
    # ... code to decouple predictions

    # Define loss for every case (training and validation)
    va_loss = valence_arousal_ccc_loss(dim_sep_labels, dim_sep_predictions)

    if training:
        # Define optimiser for training
        global_step = tf.train.get_or_create_global_step()
        num_steps_per_epoch = int(math.ceil(params.train_size / params.batch_size / hvd.size()))

        with tf.device(device):
            with tf.name_scope(name='L_Rate_Logic'):
                # Use the effective learning rate from [Goyal et al. 2017]
                # Equivalent to effective_lr = params.init_lr * (gpus*batchsize/32)
                effective_lr = params.initial_learning_rate * hvd.size()
                lr_decayed = tf.train.cosine_decay_restarts(learning_rate=effective_lr, global_step=global_step,
                                                            first_decay_steps=params.num_epochs * num_steps_per_epoch,
                                                            t_mul=1.0, m_mul=0.7, alpha=0.0)
                learning_rate = lr_decayed

        # Define training step that minimizes the loss using SGD with Momentum (optional Nesterov) optimiser
        train_op = sgd_w_optimizer(loss=va_loss, lr=learning_rate, use_nesterov=True, params=params)

    # ... code for metrics, summaries and update ops
```

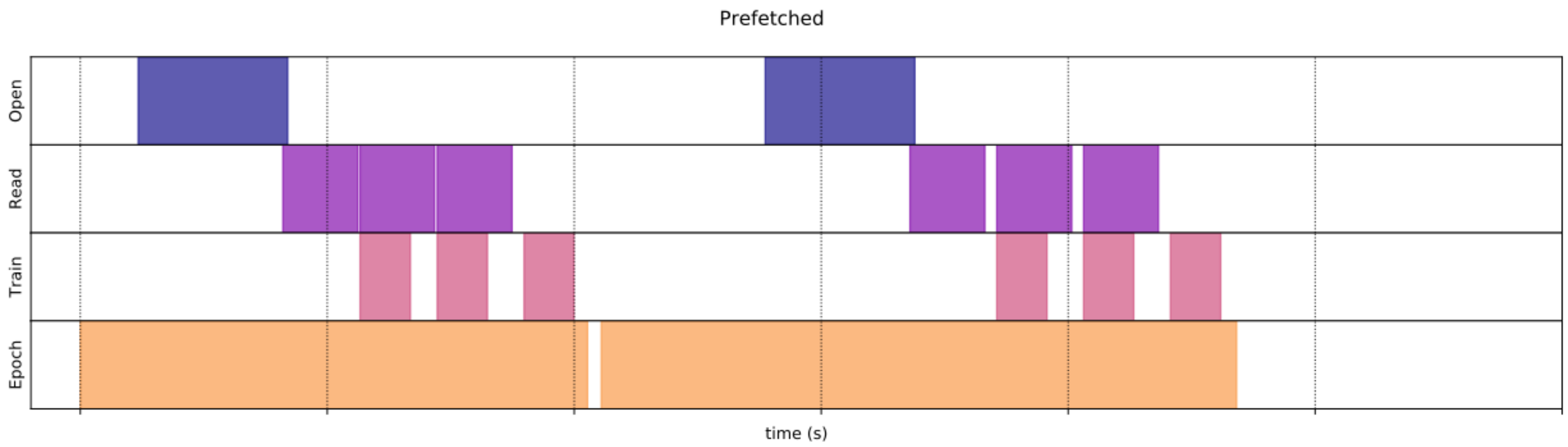
Data Pipelines

- Need to setup data pipelines
  - Create tfrecords
  - Perform preprocessing
  - Data augmentation
  - Efficient parallel batch creation

- https://www.tensorflow.org/guide/data_performance

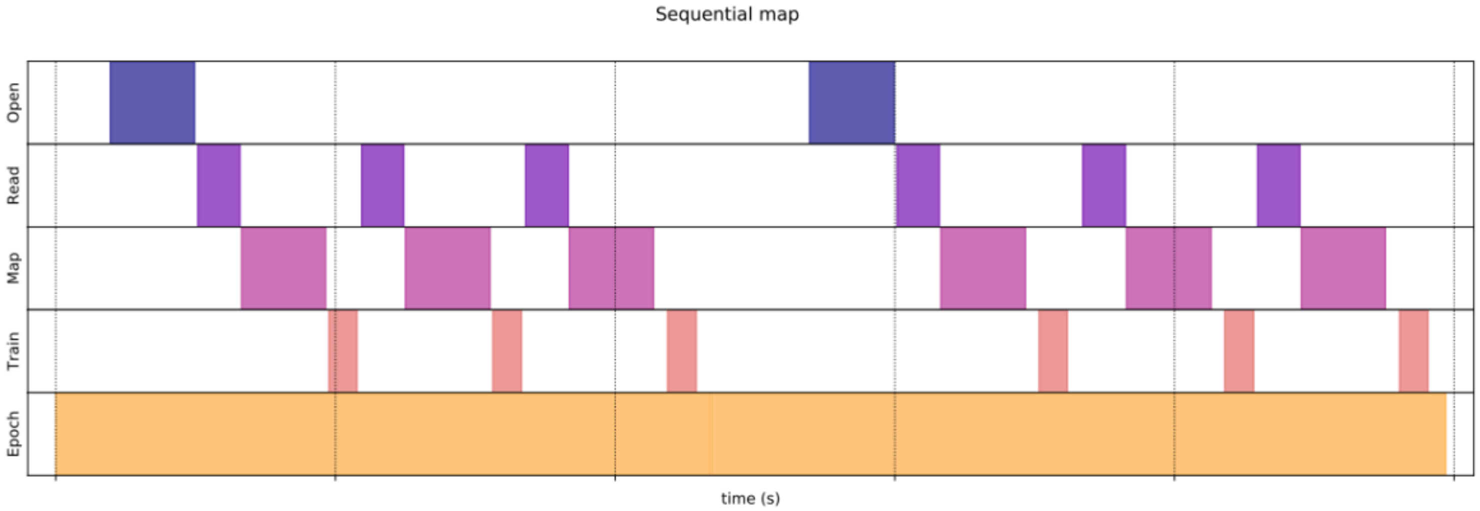
Prefetching

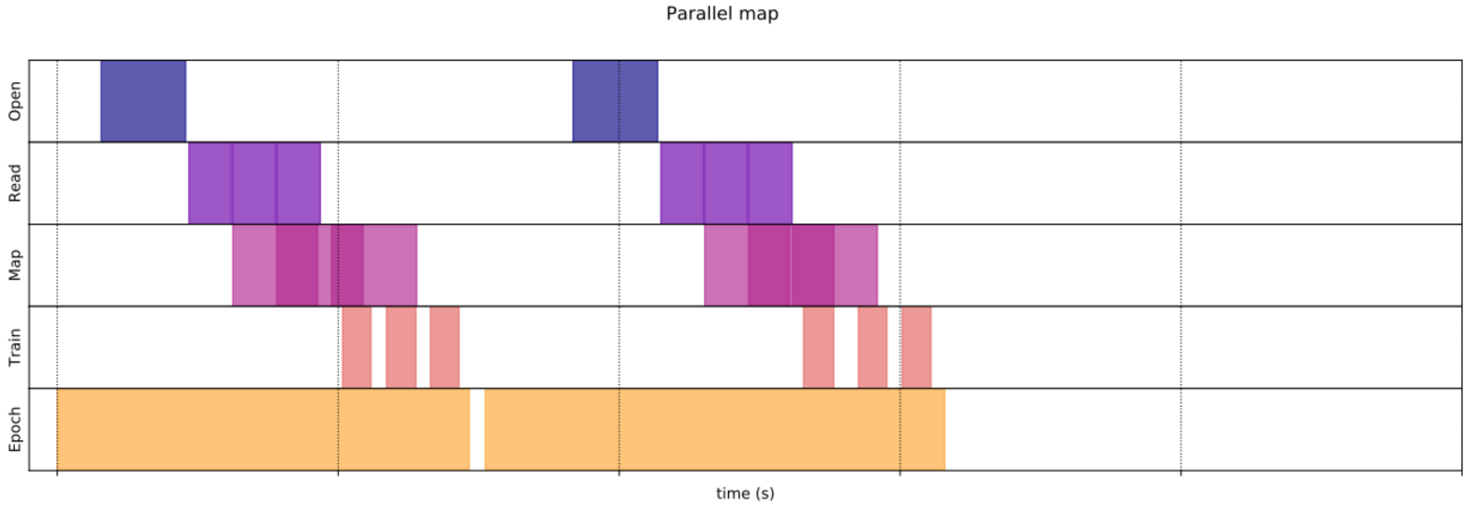




Figures from https://www.tensorflow.org/guide/data_performance

Parallel Data Transformation





Figures from https://www.tensorflow.org/guide/data_performance

Data Preparation

```python
def _convert_to_example(filename, image_buffer, expression_label,
  valence_label, arousal_label, height, width):

    colorspace = 'RGB'
    channels = 3
    image_format = 'jpg'

    example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': _int64_feature(height),
        'image/width': _int64_feature(width),
        'image/class/expression_label': _int64_feature(expression_label),
        'image/class/valence_label': _float_feature(valence_label),
        'image/class/arousal_label': _float_feature(arousal_label),
        'image/colorspace': _bytes_feature(colorspace),
        'image/channels': _int64_feature(channels),
        'image/format': _bytes_feature(image_format),
        'image/filename': _bytes_feature(os.path.basename(filename)),
        'image/encoded': _bytes_feature(image_buffer)
    }))
    return example
```

- Use custom python script which converts our dataset into a collection of tfrecord files.
- Do the reverse when parsing data in the input pipelines

Input Pipeline - Preprocess

```python
def _preprocess_data(image, exp_label, dim_label):
    """
    Image processing for training
    :param image: Tensor with shape [batch, height, width, channels]
    :return: processed image
    """
    image = tf.image.per_image_standardization(image)

    # Make sure that the image is still in [0, 1]
    image = tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)

    return image, exp_label, dim_label
```

Input Pipeline - Augment

```python
def _augment_data(image, exp_label, dim_label, image_size):
    """
    Image augmentation for training
    Apply the following operations:
        - Horizontally flip the image with probability 0.5
        - Rotate the image with a random angle taken by a uniform
          distribution from range (-20, 20)

    :param image: Tensor with shape [batch, height, width, channels]
    :return: augmented image
    """
    image = tf.image.random_flip_left_right(image)
    image = tf.contrib.image.rotate(image, angles=math.radians(
                  int(random.uniform(-20, 20))),
                  interpolation='BILINEAR')

    tx = random.uniform(-0.1, 0.1) * image_size
    ty = random.uniform(-0.1, 0.1) * image_size
    translation_matrix = [1, 0, -tx, 0, 1, -ty, 0, 0]
    image = tf.contrib.image.transform(image, transforms=translation_matrix,
                                       interpolation='BILINEAR')

    return image, exp_label, dim_label
```

Input Pipeline

```
with tf.variable_scope(name_or_scope='Train_Dataset'):
    files = tf.data.Dataset.list_files(file_pattern=filepath + '*.tfrecord',
                                       shuffle=False)
    files = files.shard(num_shards=hvd.size(), index=hvd.rank())
    # number of files divided by number of workers
    files = files.shuffle(buffer_size=240 // hvd.size())
    dataset = files.apply(
        # tf.data.experimental.parallel_interleave for tensorflow 1.12.0 version
        transformation_func=tf.contrib.data.parallel_interleave(
          lambda f: tf.data.TFRecordDataset(f),
          cycle_length=multiprocessing.cpu_count() // hvd.local_size(),
          block_length=1, sloppy=True,
          buffer_output_elements=2,
          prefetch_input_elements=2*(multiprocessing.cpu_count() // hvd.local_size()))
    dataset = dataset.map(parse_fn, num_parallel_calls=
                            multiprocessing.cpu_count() // hvd.local_size())
    dataset = dataset.shuffle(buffer_size=10000, reshuffle_each_iteration=True)
    dataset = dataset.map(preprocess_fn, num_parallel_calls=
                            multiprocessing.cpu_count() // hvd.local_size())
    # tf.data.experimental.map_and_batch for tensorflow 1.12.0 version
    dataset = dataset.apply(tf.contrib.data.map_and_batch(augment_fn,
                              batch_size=params.batch_size, num_parallel_calls=
                                  multiprocessing.cpu_count() // hvd.local_size()))
    # tf.data.experimental.copy_tp_device for tensorflow 1.12.0 version
    dataset = dataset.apply(tf.contrib.data.copy_to_device(target_device='/gpu:0'))
    # make sure you always have at least one batch ready to serve
    # buffer_size = None or -1 | This is the sentinel for auto-tuning.
    dataset = dataset.prefetch(buffer_size=-1)
```

Best Practices

Best practice summary from Tensorflow:

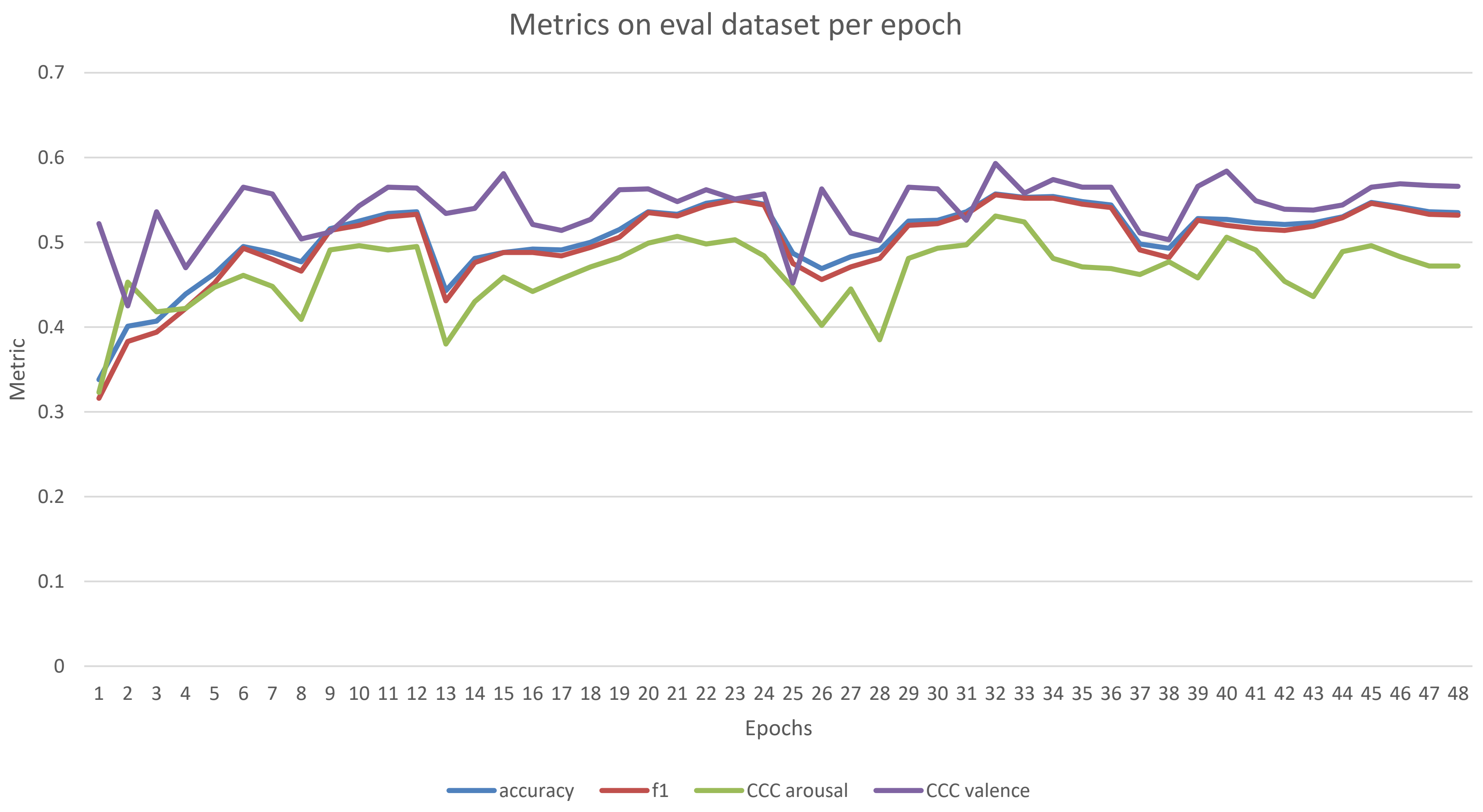
- Use the prefetch transformation to overlap the work of a producer and consumer.
- Parallelize the data reading transformation using the interleave transformation.
- Parallelize the map transformation by setting the num_parallel_calls argument.
- Use the cache transformation to cache data in memory during the first epoch
- Vectorize user-defined functions passed in to the map transformation
- Reduce memory usage when applying the interleave, prefetch, and shuffle transformations.

Execution on Aris

```bash
#!/bin/bash

#SBATCH --job-name=emotions    # Job name
#SBATCH --output=emotions.%j.out # Stdout (%j expands to jobId)
#SBATCH --error=emotions.%j.err # Stderr (%j expands to jobId)
#SBATCH --ntasks=4     # Number of tasks(processes)
#SBATCH --nodes=2     # Number of nodes requested
#SBATCH --ntasks-per-node=2     # Tasks per node
#SBATCH --gres=gpu:2 # GPUs per node -- must be equal to ntasks per node
#SBATCH --cpus-per-task=10     # Threads per task
#SBATCH --time=40:00:00   # walltime find approximate time
#SBATCH --mem=56G   # memory per NODE
#SBATCH --partition=gpu    # Partition
#SBATCH --account=foo    # Replace with your system project
#SBATCH --export=ALL,HOROVOD_CYCLE_TIME=1,NCCL_DEBUG=INFO,HOROVOD_MPI_THREADS_DISABLE=1

export I_MPI_FABRICS=shm:dapl

if [ x$SLURM_CPUS_PER_TASK == x ]; then
  export OMP_NUM_THREADS=1
else
  export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
fi

# change dir to source code
cd /users/emotions/src

# load necessary modules
module purge
module load gnu/6.4.0
module load intel/15.0.3
module load intelmpi/5.0.3
module load java/1.8.0
module load cuda/9.2.148
module load tensorflow/1.10.1gpu

# train
srun python train.py
```

- Total 48 epochs
- 2 HPC Nodes x 2 GPUs
  - Each node Intel Xeon E5-2660v3, 64 GB
  - Each node 2 GPUs Nvidia K40
- Batch Size 64 -- Initial learning rate 0.025
- Data-parallelism

48 Training Epochs



Results

	AffectNet	This work
Accuracy	0.58	0.5557
F1-score	0.58	0.5551
CCC-valence	0.541	0.5954
CCC-arousal	0.450	0.5358
RMSE-valence	0.394	0.42
RMSE-arousal	0.402	0.39
CORR-valence (Pearson CC)	0.602	0.604
CORR-arousal (Pearson CC)	0.539	0.545
SAGR-valence (Sign Agreement Metric)	0.728	0.604
SAGR-arousal (Sign Agreement Metric)	0.670	0.545

Not exactly comparable, as we are running on the validation dataset.

Test dataset was never published.

Future & Ongoing Work

- Train 3dconv for video (frames)
- Use the aff-wild dataset
- Use best frozen model for images.
- Feed layers before final classifiers to 3dconv layer.
- Train on ML node on Aris
  - 2 Intel E5-2698v4 (20 cores each), 512GB, 8 GPUs Nvidia V100

The End

- Thank you for your attention
- Questions?
- Suggestions?