Model documentation and write-up (3rd place)

Deep Chimpact: Depth Estimation for Wildlife Conservation
Author: Igor Ivanov
Email: vecxoz@gmail.com

1. Who are you (mini-bio) and what do you do professionally?

My name is Igor Ivanov. I'm a deep learning engineer from Dnipro, Ukraine. I specialize in CV and NLP and work at a small local startup.

2. What motivated you to compete in this challenge?

Environment conservation is a very important effort. I'm happy to take part and contribute. Additional inspiration came from the cute chimpanzees and other animals featured in the videos.

3. High level summary of your approach: what did you do and why?

Solution is an ensemble of 12 models each of which is a self-ensemble of 5 folds. All models are based on CNN-LSTM architecture with EfficientNet-B0 backbone. Input data is a sequence of 7 or 9 video frames taken with an interval (time-step) of 1 or 2 seconds. Each sequence has an equal number of frames taken before and after the target frame. Each frame is a 3-channel image with a resolution 512 x 512. Optimization performed using Adam optimizer and MAE loss. Significant improvements were obtained using different types of augmentation, especially affine transformations (flips, rotations) and Gaussian blur. Another source of improvement is gamma correction. Around 10% of all videos are overexposed, and gamma correction allows to extract much more information from such examples.

3.1 Data pipeline description

In my solution I used 4 datasets:

- Images extracted from downsampled videos, time-step 1 second
- Images extracted from downsampled videos, time-step 2 seconds
- Images extracted from original videos with gamma correction, time-step 1 second
- Images extracted from original videos with gamma correction, time-step 2 seconds

In general validation scores are significantly better for images extracted from original (full size) videos. From the data analysis it's evident that about 10% of videos are overexposed. Frames extracted from such videos with standard parameters are almost completely white. In this case gamma correction is very helpful. To find videos which need gamma correction I computed the proportion of white pixels (value 250) for target frames extracted from downsampled videos. If this proportion is greater than 20% then video needs gamma correction. I copied “white” target frames into a dedicated directory and used file ids from this directory to identify which original videos need

gamma correction. It's important to note that gamma correction is impossible for downsampled videos because information is lost.

3.2 Models description

All models share the same architecture and hyperparameters: CNN-LSTM architecture with EfficientNet-B0 backbone, batch size 96, learning rate 1e-3, image size 512 x 512. The main differences are augmentation, augmentation intensity (percentage of modified images), gamma correction, number of frames in a sequence, and time step between frames. Model ID (directory name) contains all model specific information encoded in the following way:

down/full – images were extracted from downsampled/full size videos

a1 – augmentation version 1: 2 flips, 7 rotations multiple of 45 degrees

a2 – augmentation version 2: all from version 1, hue levels, grayscale, grayscale +/- brightness

a4 – augmentation version 4: all from version 1, grayscale, Gaussian blur

p50/p80 – probability of getting augmented image (across all types of modifications)

f7/f9 – total number of frames (3-channel images) in a sequence (7 or 9). E.g. 7-frame sequence contains: 3 previous frames, target frame, 3 next frames.

s1/s2 – time step between frames (1 or 2 seconds)

Model ID (dir name)	**Cross-val MAE (15k examples)**
run-20211101-1526-down-a1p50-f7s1	1.4054
run-20211101-2141-down-a1p50-f9s1	1.4302
run-20211103-1552-down-a2p50-f7s1	1.3936
run-20211103-1744-down-a2p80-f7s1	1.4306
run-20211103-1749-down-a1p80-f7s1	1.4645
run-20211104-1815-down-a4p50-f7s1	1.3861
run-20211107-0019-down-a4p50-f7s2	1.4173
run-20211113-1639-full-a1p50-f7s1	1.3903
run-20211113-1642-full-a1p50-f9s1	1.3874
run-20211113-1645-full-a2p50-f7s1	1.3588
run-20211113-1648-full-a4p50-f7s1	**1.3382**
run-20211113-1847-full-a4p50-f7s2	1.3806

To train models I used MAE loss and Adam optimizer. Learning rate is constant with reduction on plateau. Number of epochs is fixed to 40, and weights are saved only if validation loss improves. It is possible to reduce training time by setting a smaller number of epochs (e.g. 30) or early stopping. Score degradation should not be significant.

3.3 Ensemble and post-processing

Ensemble is a weighted arithmetic average optimized using pair-wise greedy search over sorted predictions.

As a post-processing I rounded predicted values to the nearest values present in the training set. Probably this is not significant for production but it gave me some improvement in the leaderboard.

4. Do you have any useful charts, graphs, or visualizations from the process?

The following pictures demonstrate significant effect of gamma correction:

5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

1) Gaussian blur is a very efficient augmentation for this type of data. I used “tensorflow_addons” package:

```
image = tfa.image.gaussian_filter2d(image, filter_shape=10, sigma=5.0)
```

2) I found it convenient to use the command line “ffmpeg” utility for gamma correction. I called it from Python using “subprocess” package in the following way:

```
cmd_list = ['ffmpeg', '-hide_banner', '-loglevel', 'error', '-y', '-i', file, '-start_number',
            '0', '-r', '1', '-vf', 'eq=gamma=0.1', '-q:v', '1', out_template]
_ = subprocess.run(cmd_list, stdout=subprocess.PIPE).stdout.decode('utf-8')
```

3) “TimeDistributed” layer allows for very concise model definition:

```
x = tf.keras.layers.TimeDistributed(effnet)(inp)
x = tf.keras.layers.TimeDistributed(pool_2d)(x)
x = tf.keras.layers.LSTM(1024, return_sequences=False)(x)
x = tf.keras.layers.Dense(300, activation='relu')(x)
out = tf.keras.layers.Dense(1, activation='linear')(x)
```

6. Please provide the machine specs and time you used to run your model.

OS: Ubuntu 18.04

Data preparation:
12 CPU, 16 GB RAM, 1 TB HDD
Prepare inference (test set): 4 hours
Prepare training (train set): 6 hours

Inference:
1x V100-16GB, 12 CPU, 16 GB RAM, 1 TB HDD
Inference time: 5 hours total (all models)

Training (in fact):
TPUv3-8, 1 CPU, 4 GB RAM, 1 TB HDD
Training time: 120 hours total (all models)

Training (possible alternative):
It's possible to use the following configuration with 3x smaller batch size
and explicit mixed precision "mixed_float16":
8x V100-16GB, 12 CPU, 48 GB RAM, 1 TB HDD
Training time: 300 hours total (all models)

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

1) There is a known edge case related to LSTM model inference performed on multiple GPUs. In this scenario the number of validation or test examples must be divisible by batch size. The most straightforward solution for inference is to run on a single GPU, and for training to set “drop_remainder=True”. All my scripts have this argument. On a TPU or single GPU everything works without any such limitations.

2) There is a small discrepancy (probably less than 1 sec time shift) between frames extracted using “ffmpeg-python” package and command line tool “ffmpeg”. It is observed only when processing full size videos. For downsampled videos both methods give identical results. From my experiments this discrepancy has no effect on model performance. Also it’s important to note that sometimes the command line tool “ffmpeg” may return one more frame compared to “ffmpeg-python”.

3) In my data pipeline I create TFRecord files in two stages: 1) create sequence of images as Numpy arrays of byte-strings and save in .npy files; 2) then create TFRecord files by reading Numpy files. When TFRecord files are ready all Numpy files are deleted by default. To retain Numpy files for further experiments just set the argument “–delete_numpy=False” of the script “create_tfrecords.py”.

8. **Did you use any tools for data preparation or exploratory data analysis that aren’t listed in your code submission?**

No

9. **How did you evaluate performance of the model other than the provided metric, if at all?**

I used the provided metric MAE as loss and RMSE as additional metric.

10. **What are some other things you tried that didn’t necessarily make it into the final workflow (quick overview)?**

1) *Audio (no improvement)*. I had one interesting idea to try features extracted from soundtracks. Please note that only original (full size) videos have audio. I watched several examples and had mixed conclusions. From one side animals make noises, and loudness is related to distance, which is good. But from the other side sometimes animals walk in silence, oftentimes sound level depends on the type of surface coverage (rocks vs. dry leaves), different videos have different quality of the sound. Eventually I extracted audio features as Log-Mel spectrograms and trained EfficientNet-B0. I did not obtain any promising results. But still this idea may be relevant in some environments.

2) *TTA (no improvement)*. I implemented TTA (test-time augmentation) corresponding to transformations applied as augmentation. The scores from the average of all TTA predictions were worse than without TTA.

11. **If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?**

Some videos from the dataset have low resolution (probably some older cameras). I would try to improve the quality of such examples using different techniques as a preprocessing step.