Fast Depth Coding Using Deep Learning Documentation

Release 0.1.0

Pharrell.zx WANG

Mar 27, 2018

Propose the Algorithm

Fast Depth Coding Using Deep Learning is the thesis topic. The details of the implementation are documented here for reference.

This documentation is organized into a couple sections:

CHAPTER 1

Flow Chart

In 3D-HEVC, the wedgelet searching process in the depth map coding consumes a lot of time. We propose an algorithm in this work to balance the trade-off between **coding efficiency** and **computational complexity** using deep learning.

Fig. 1: Figure 1: Flowchart for Proposed Algorithm

Fig. 2: Figure 2: Detailed Flowchart for Proposed Algorithm

CHAPTER 2

Description

`step 1` Get the Luma pixel values from one depth block. (The block can be of size 8x8, 16x16, 32x32)

`step 2` Feed the 2D matrix of Luma pixels into **learned model** for getting the **top-16 predictions**.

`step 3` Add **top 16 predictions** into the **RMD LIST**.

`step 4` Check whether mode 2 is inside **RMD LIST**. If yes, add mode 34 into **RMD LIST**; otherwise jump to *step 5*.

`step 5` Add mode 0, 1, DMM1, DMM4 into **RMD LIST**.

`step 6` `Do RMD`. For DMM1, only check the directions covered by **top-16 predictions**.

`step 7` Add two modes into **FULL RDO LIST**. `Do full RDO`.

`step 8` Obtain the **best mode** for the depth block.

Note: The above process can be applied to a batch of blocks, in which case the time cost of prediction can be optimized. For details, see *Time Cost of TF in C++*

CHAPTER 3

[Deprecated]Flow chart

This chart has been deprecated. Kept here only for reference.

Deprecation Summary

- For below `reason 1` and `reason 2`, we remove **edge strength analysis**;
- For below `reason 3`, we remove **the implementation to texture**.

Reasons

1. Edge strength analysis is not innovative.
2. Besides, removing it from the flow chart only will decrease the accuracy of ResNet prediction by roughly 2%~3%.
3. And according to **Dr.Tsang**, since we are only using luma pixel values, it seems we should not apply our model into the texture blocks.

Fig. 1: Figure D-1: Flowchart for Proposed Fast Intra Mode Decision Algorithm

CHAPTER 4

Data Collection

This document will show you how to collect the data.

The source code of the project for processing the data is in GitHub: https://github.com/PharrellWANG/data-processing-for-fdc

4.1 Training Data Source

We collect the data by encoding the video sequences.

Data are collected from four video sequences.

#	Name of the Sequence	Resolution	Usage	Frames
1	Balloons	1024x768	train/test/validation	300
2	kendo	1024x768	train/test/validation	300
3	poznan_street	1920x1088	train/test/validation	250
4	undo_dancer	1920x1088	train/test/validation	250

4.2 Method for Collecting the data

Based on the *Effort from Ho*:

When encoding the video sequences, for every block (of size 4x4, 8x8, 16x16, 32x32, 64x64):

- `if` DIS has been assigned (where `DIS_FLAG == 1`), we **skip** it (since none of the conventional intra modes including DMMs will be used). *Skip it* means we don't collect data from it.
- `else if` DIS has **not** been assigned (where `DIS_FLAG == 0`), let's identify the partition mode:
  - `if` HTM encoder decides to implement a partition for the block (where `partition_number == 4` (NXN)):

    * **collect** the `INTRA_PRED[1]`, `INTRA_PRED[2]`, `INTRA_PRED[3]`, `INTRA_PRED[4]` for each sub parts along with their **1-D Depth Data**.

  - `else if` HTM encoder decides **not** to implement a partition:

    * let's **collect** the `INTRA_PRED[0]` along with the **1-D Depth Data**.

Note:

1. **1-D Depth Data** means the pixel value of the depth block being **flattened** into 1 dimension. For example, to store an M x N matrix of pixel values (you can imagine those pixels forming an image, hence it is like we are storing an image), the **1-D Depth Data** (pixel values) must contain M*N values, with M rows of N contiguous values each. That is, the 1-D data must store the matrix as: `....  row 0 .... .... row 1 .... // .......... // ... row M-1 ....`
2. when collecting the data, I have made it to write 35 for mode 37, and 36 for mode 38. Hence a little time/energy is saved for the data processing.

4.3 Effort from Ho

The pdf file contributed by Ho are provided for downloading.

`20170621 Fast Depth Coding Via TensorFlow (Data Collection) v1`

CHAPTER 5

Processing Steps and Statistics

After encoding the sequences, we obtained size 4x4, 8x8, 16x16, 32x32 and 64x64 for each sequence.

5.1 Step 1 Merging

Then we do the merging. i.e.,

1. Merge the data of block size 4x4 from four sequences together.
2. Merge the data of block size 8x8 from four sequences together.
3. Merge the data of block size 16x16 from four sequences together.
4. Merge the data of block size 32x32 from four sequences together.
5. Merge the data of block size 64x64 from four sequences together.

After merging, we obtained five csv files:

#	Name of the Files	Size	Samples	Usage
1	size_04.csv	206.7 MB	3675428	train/test/validation
2	size_08.csv	513.6 MB	2372324	train/test/validation
3	size_16.csv	1.25 GB	1439773	train/test/validation
4	size_32.csv	2.02 GB	567554	train/test/validation
5	size_64.csv	1.85 GB	125141	train/test/validation

Mode distribution data after merging are provided for downloading as text format.

```
mode distribution of block size 04x04
mode distribution of block size 08x08
mode distribution of block size 16x16
mode distribution of block size 32x32
```

```
mode distribution of block size 64x64
```

5.2 Step 2 Removing some modes

Remove mode 0, 1, 34, 35, 36. We only do deep learning for angular modes. Mode 34 is removed because mode 34 has the same direction feature as mode 2.

After removing 0, 1, 34, 35, 36:

#	Name of the Files	Size	Samples	Usage
1	m_size_04.csv	75.7 MB	1335970	train/test/validation
2	m_size_08.csv	130.8 MB	600187	train/test/validation
3	m_size_16.csv	377.3 MB	430302	train/test/validation
4	m_size_32.csv	708.7 GB	195943	train/test/validation
5	m_size_64.csv	1.37 GB	92034	train/test/validation

Percentage of non-angular-removed data:

size 4: (3675428 - 1335970) / 3675428.0 = 0.64

size 8: (2372324 - 600187) / 2372324.0 = 0.74

size 16: (1439773 - 430302) / 1439773.0 = 0.70

size 32: (567554 - 195943) / 567554.0 = 0.65

size 64: (125141 - 92034) / 125141.0 = 0.26

The mode distribution data are provided for downloading.

```
mode distribution of block size 04x04 AFTER non-angular removing
mode distribution of block size 08x08 AFTER non-angular removing
mode distribution of block size 16x16 AFTER non-angular removing
mode distribution of block size 32x32 AFTER non-angular removing
mode distribution of block size 64x64 AFTER non-angular removing
```

5.3 Step 3 Removing Smooth Blocks

Perform *Edge Strength Analysis* for each block sample of all sizes. Observing the histogram distribution.

Flat regions will trap CNN into ill condition. I decided to remove the regions where the edge strength is under 50.

And for the blocks where the edge strength is above 25000, we only consider four modes: VER, HOR, Wedgelet, Contour.

After removing the smooth areas,

#	Name of the Files	Size	Samples	Usage
1	sm_size_04.csv	36.3 MB	616281	train/test/validation
2	sm_size_08.csv	91.2 MB	403277	train/test/validation
3	sm_size_16.csv	210.9 MB	232806	train/test/validation
4	sm_size_32.csv	235.4 MB	65481	train/test/validation
5	sm_size_64.csv	271.8 MB	19244	train/test/validation

Percentage of smooth-removed data:

size 4: (1335970 - 616281) / 1335970.0 = 0.54

size 8: (600187 - 403277) / 600187.0 = 0.33

size 16: (430302 - 232806) / 430302.0 = 0.46

size 32: (195943 - 65481) / 195943.0 = 0.67

size 64: (92034 - 19244) / 92034.0 = 0.79

The mode distribution data are provided for downloading.

`mode distribution of block size 04x04 AFTER smooth removing`

`mode distribution of block size 08x08 AFTER smooth removing`

`mode distribution of block size 16x16 AFTER smooth removing`

`mode distribution of block size 32x32 AFTER smooth removing`

`mode distribution of block size 64x64 AFTER smooth removing`

5.4 Step 4 Imbalanced Learning

Please notice we are facing a problem of **imbalanced learning** which means the data sizes of each class vary in a large scale. To tackle the issue: we use equal data sizes for each class, abandon the extra data.

5.5 Step 5 Tagging

Tag the mode to start from 0, end with 31. Just use `modeIdx - 2` to obtain the new index of each mode for deep learning.

5.6 Final Data Description

#	Name of the Files	Size	Samples	Usage
1	train_04x04.csv	7.9 MB	4092*32	train
2	test_04x04.csv	1.1 MB	600*32	test
3	val_04x04.csv	1.1 MB	600*32	validation

#	Name of the Files	Size	Samples	Usage
1	train_08x08.csv	46.3 MB	6303*32	train
2	test_08x08.csv	3.8 MB	600*32	test
3	val_08x08.csv	3.8 MB	600*32	validation

#	Name of the Files	Size	Samples	Usage
1	train_16x16.csv	89.9 MB	3075*32	train
2	test_16x16.csv	15.9 MB	600*32	test
3	val_16x16.csv	15.6 MB	600*32	validation

1. After trying to train data of size 04x04, it does not learn well, only top-28 accuracy is around 0.95.

2. Too few data for size 32x32 and 64x64 after smooth removing. We resize them using Bilinear Interpolation for employing learned model for size 16x16

CHAPTER 6

Data Visualization

6.1 Summary

The modes are chosen based on RD cost, which is a function of distortion and coding rate.

Each CU will choose the optimal mode which has the least RD Cost.

There might be a chance to choose the mode with distortion but lower coding rate.

The distortion would also depend on the distortion of synthesized texture view if VSO is on.

6.2 Memo

Note: Please notice this visualization is for the data without any pre-processing, i.e., the data that we obtained right after encoding the video sequences. You can get a sense of what kind of data that we are dealing with. And you might know the reason for the techniques executed in the pre-processing steps, such as edge strength analysis for smooth-removing.

In the python project for pre-processing the data, which is data-processing-for-fast-depth-coding, you can visualize the collected data by typing commands from terminal:

```
python data_visu/visualize_blocks.py --block_size=32 --mode=2
```

Figure 0-1. Illustration of Intra modes [0, 34] and *Figure 0-2. Examples of 8x8 luma prediction blocks generated with all the HEVC intra prediction modes.* are **the illustrations of Intra modes in HEVC** in case you want to refresh your memory. (DMMs are not illustrated here).

Other figures are **the visualization of some collected data**.

Note: Posting all the visualization images here would be possibly overwhelming and unnecessary. However, if you want to see all the visualizations, that is also doable (by spending a *little* more time running the python script then

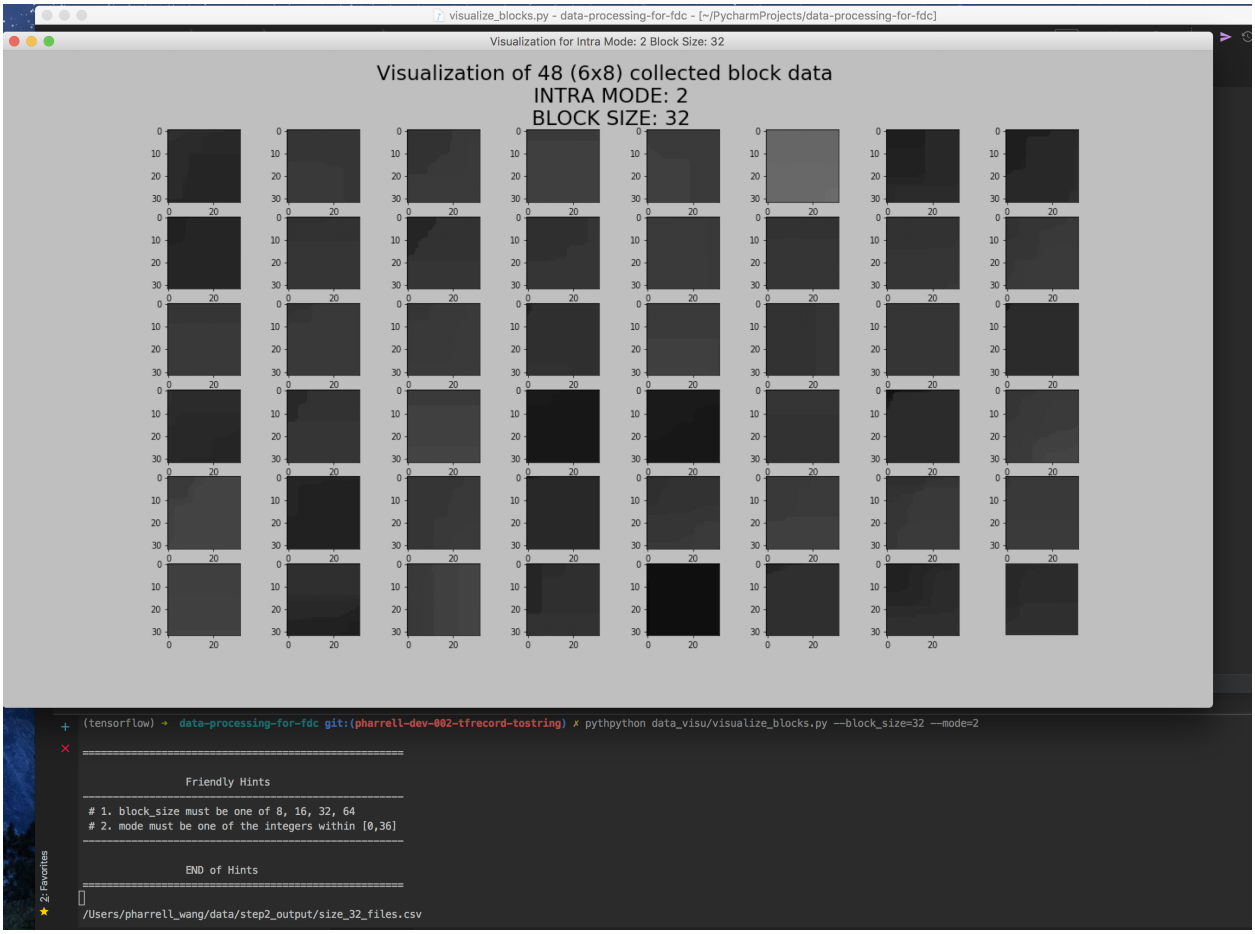


Fig. 1: Figure 0-0. running the visualization

capturing screen). Please just let me know if there's a need :D

Important: Please compare *Figure 2-2. Intra Mode: 2, Block Size: 16x16* with *Figure 2-34. Intra Mode: 34, Block Size: 16x16*. Compare the two blocks in red circle. You should be able to find the patterns are very similar to each other while their modes are totally different. This is the reason why we need to use TOP-5 accuracy instead of TOP-1 for evaluating the machine learning model.

TOP-5 To compare models, we examine how often the model fails to predict the correct answer as one of their top 5 guesses – termed "top-5 error rate".

Tip: Feel free to click on the figures for inspecting their features carefully by enlarging or downloading. They are licensed under MIT License.

6.3 Intra modes in HEVC

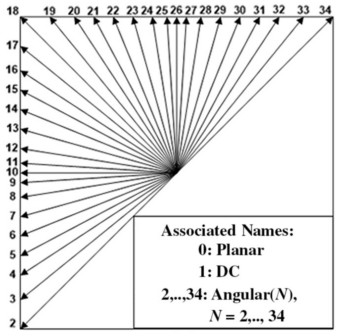


Fig. 2: Figure 0-1. Illustration of Intra modes [0, 34]

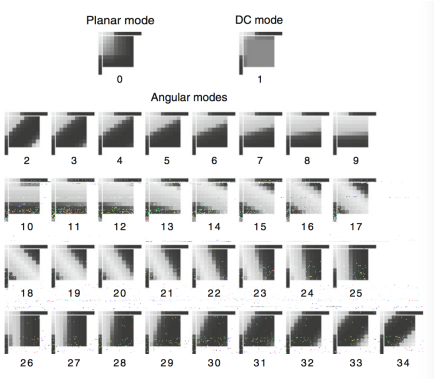


Fig. 3: Figure 0-2. Examples of 8x8 luma prediction blocks generated with all the HEVC intra prediction modes.

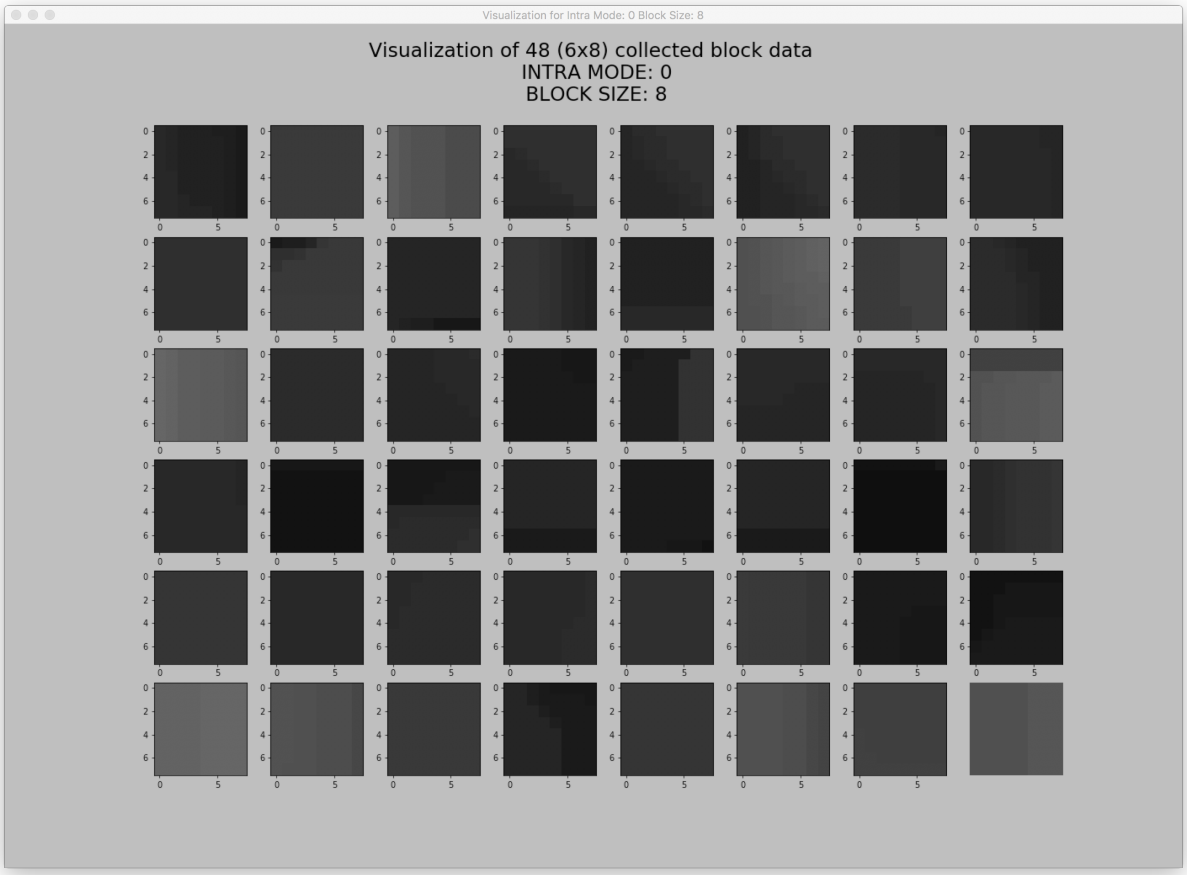


Fig. 4: Figure 1-0. Intra Mode: 0, Block Size: 8x8

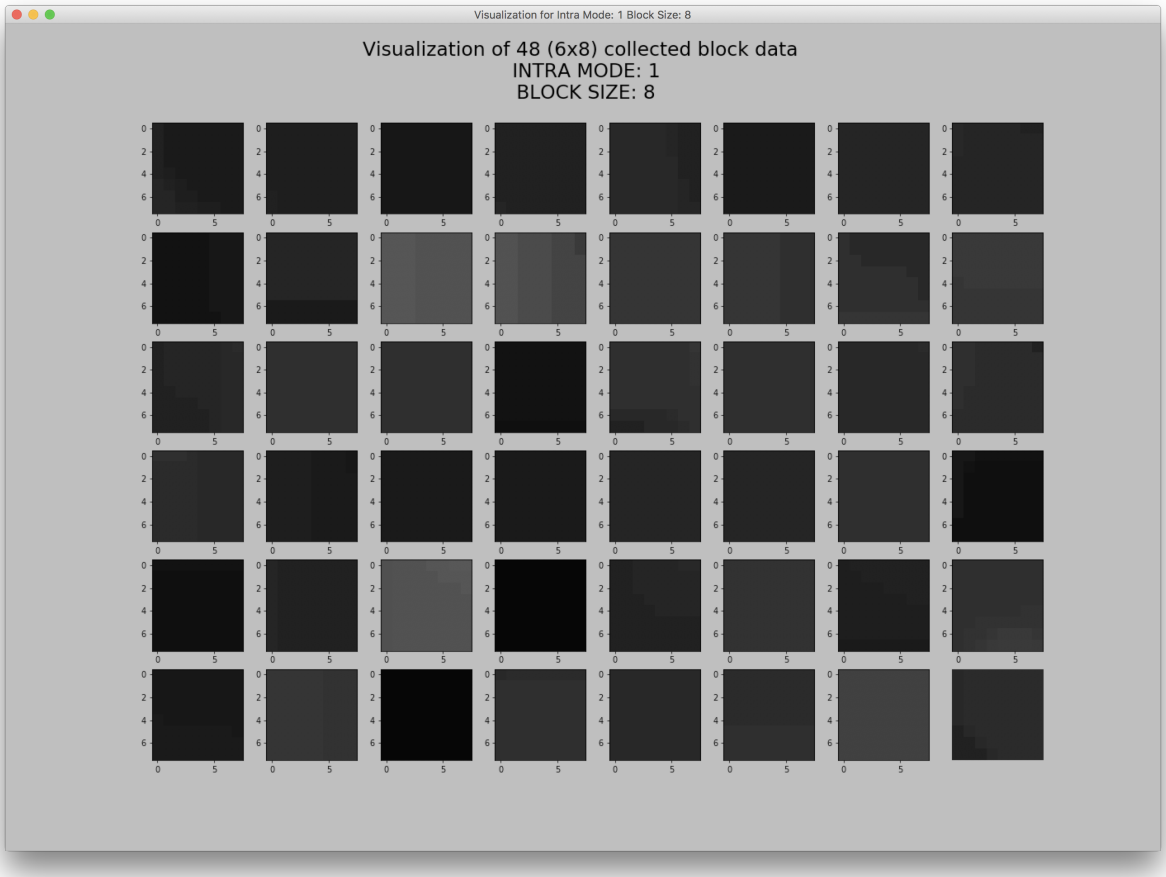


Fig. 5: Figure 1-1. Intra Mode: 1, Block Size: 8x8

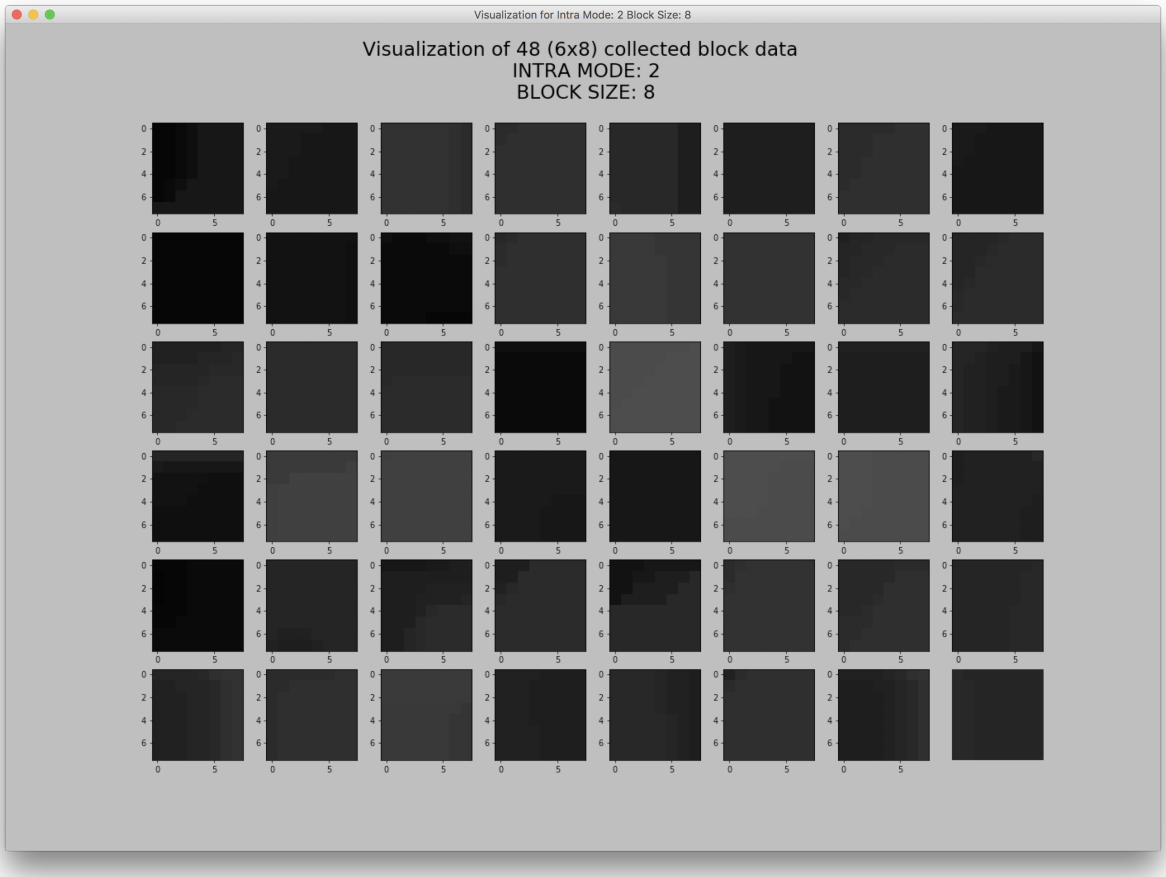


Fig. 6: Figure 1-2. Intra Mode: 2, Block Size: 8x8

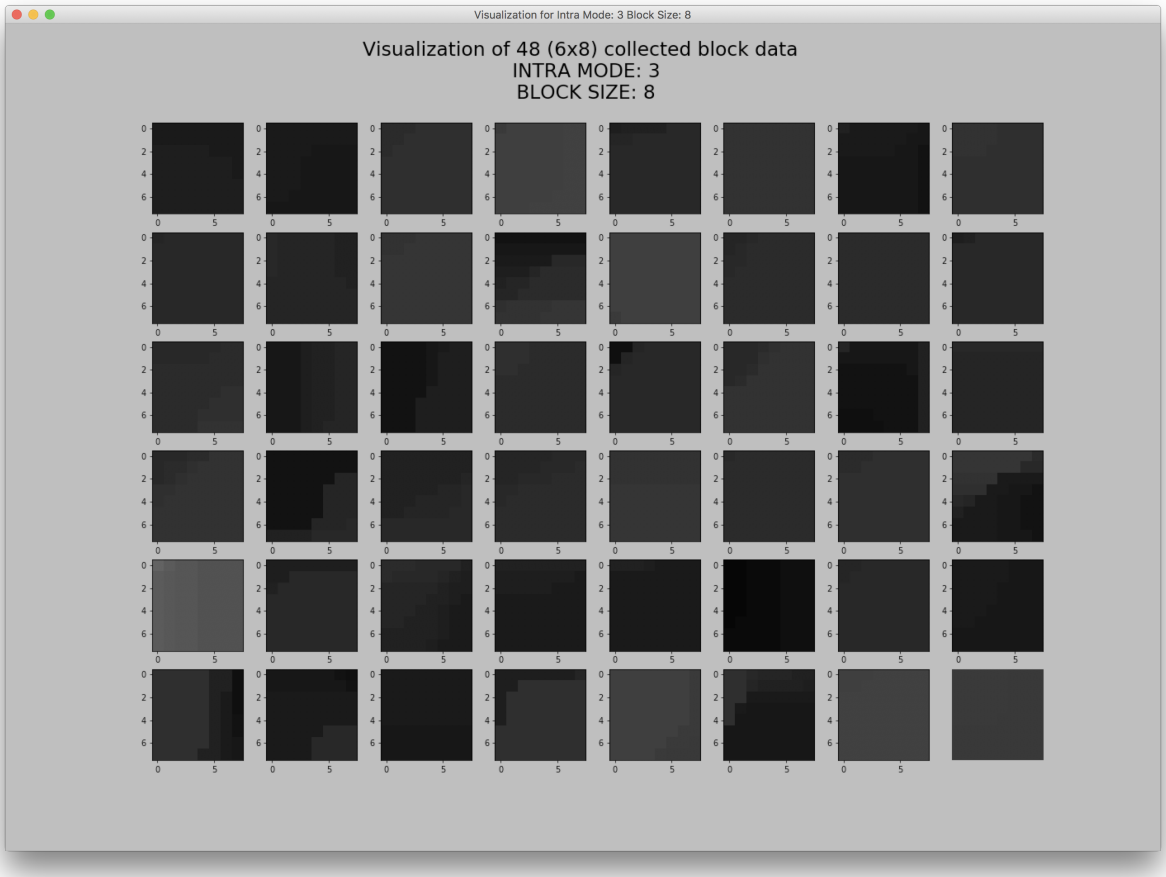


Fig. 7: Figure 1-3. Intra Mode: 3, Block Size: 8x8

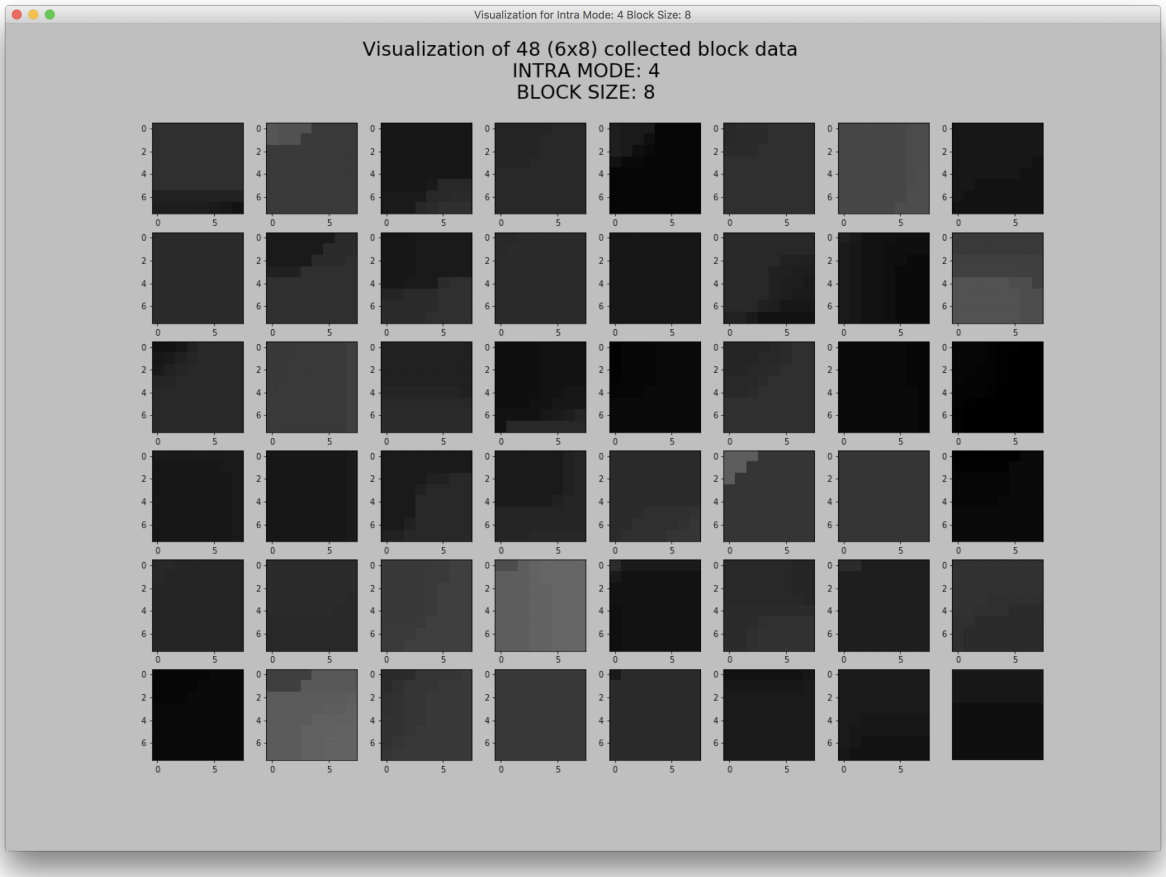


Fig. 8: Figure 1-4. Intra Mode: 4, Block Size: 8x8

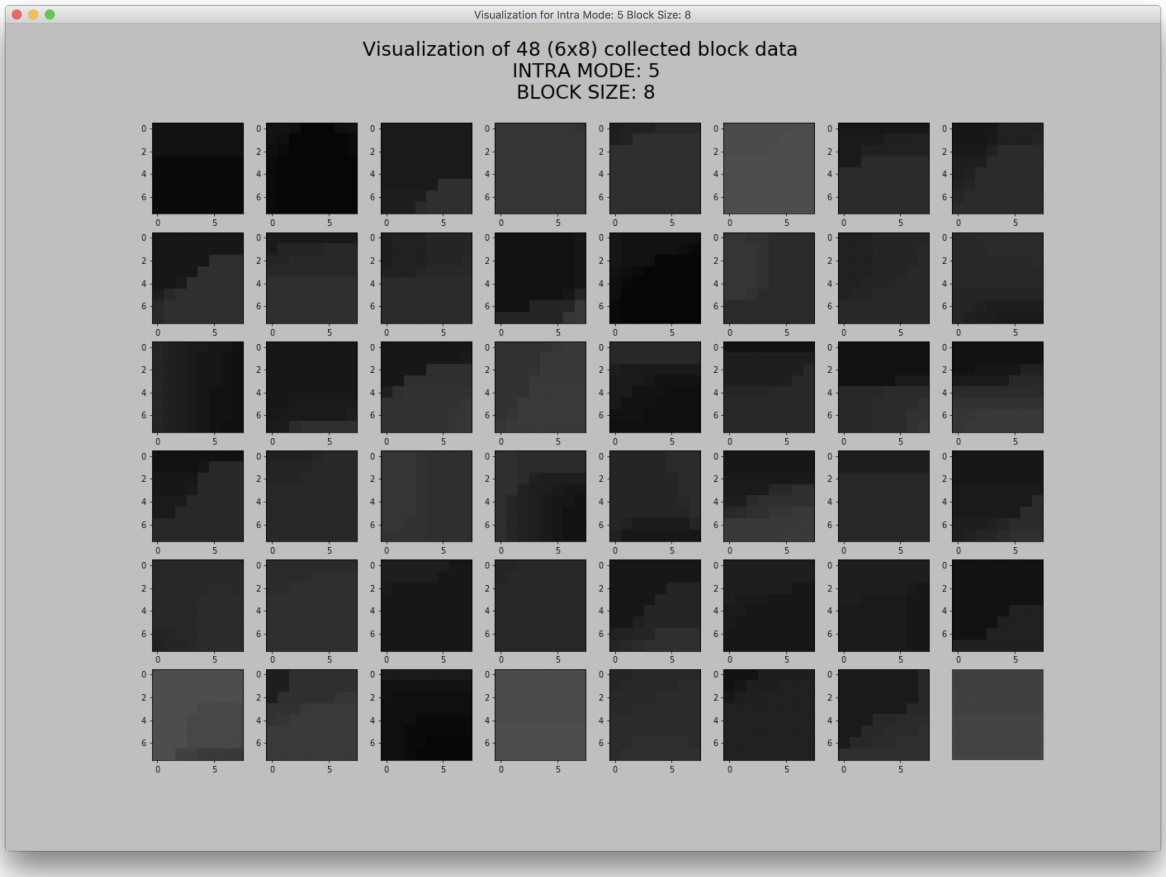


Fig. 9: Figure 1-5. Intra Mode: 5, Block Size: 8x8

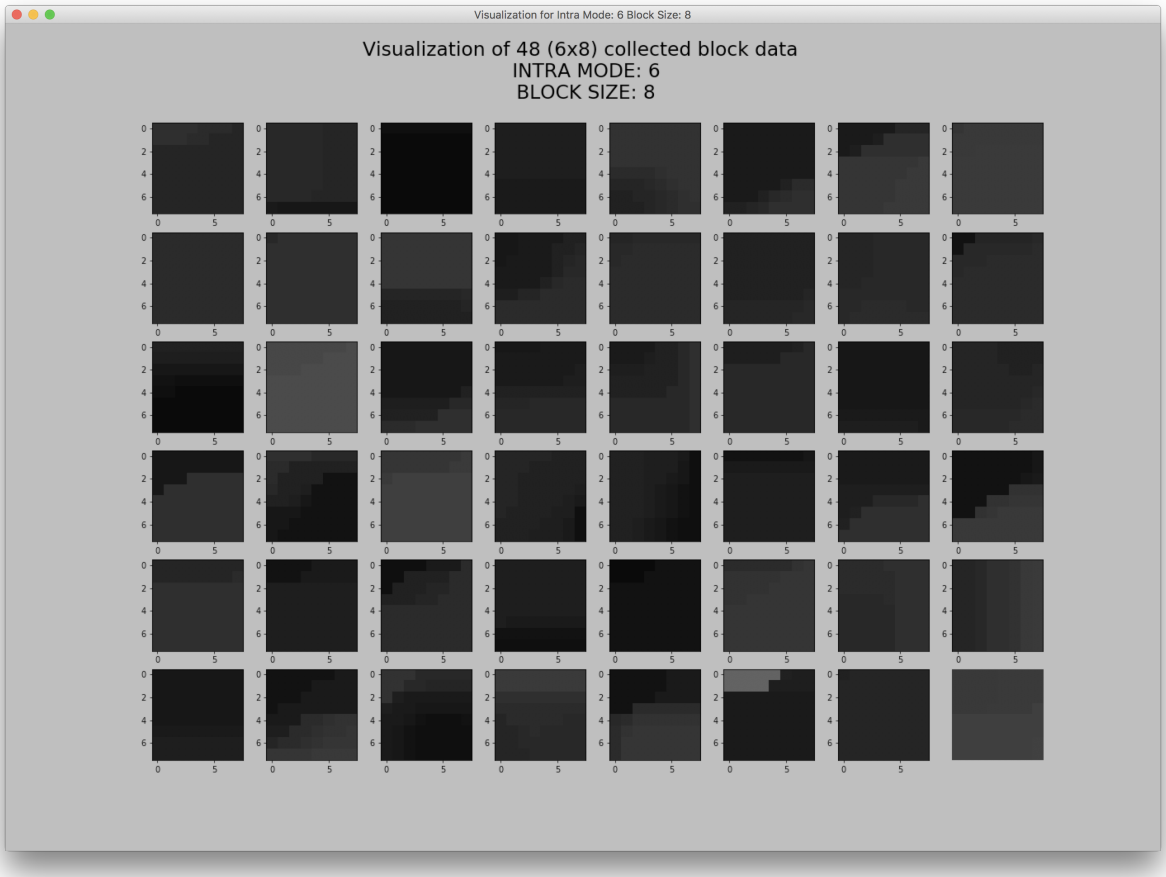


Fig. 10: Figure 1-6. Intra Mode: 6, Block Size: 8x8

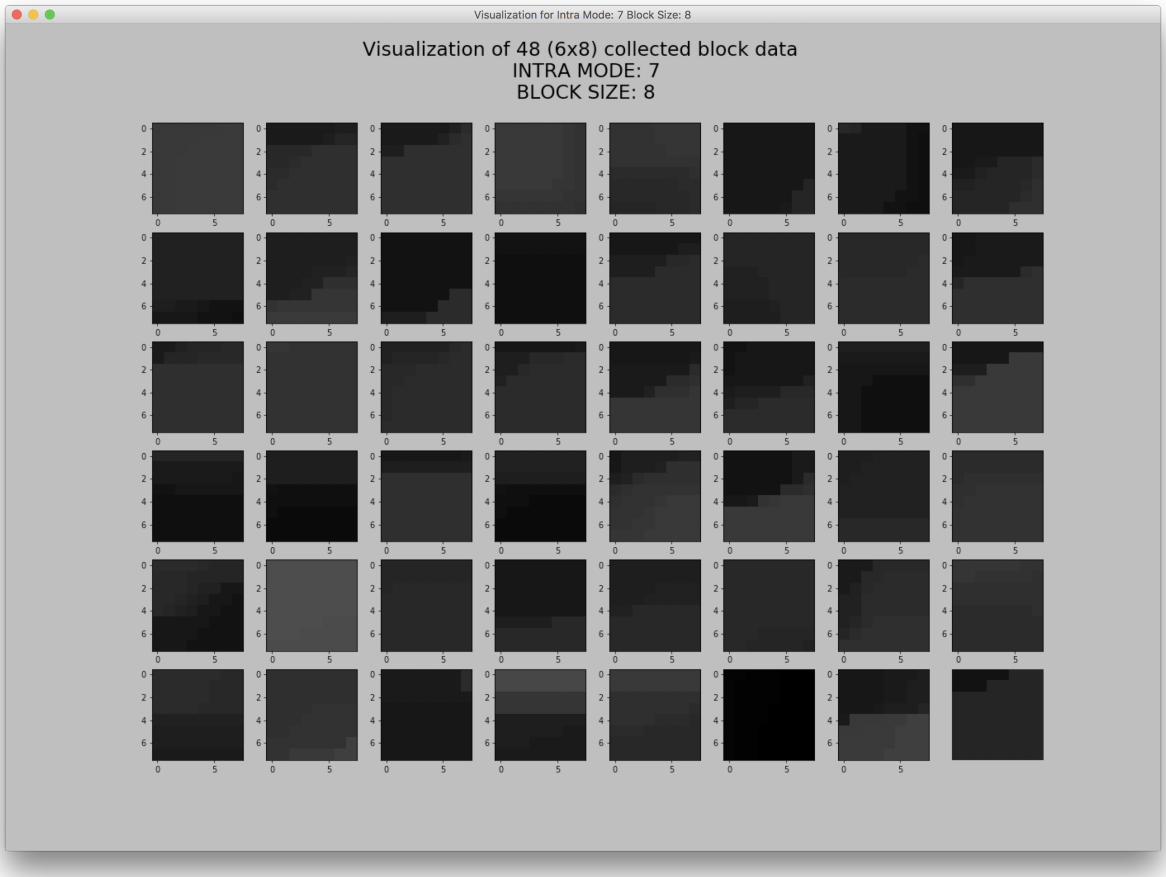


Fig. 11: Figure 1-7. Intra Mode: 7, Block Size: 8x8

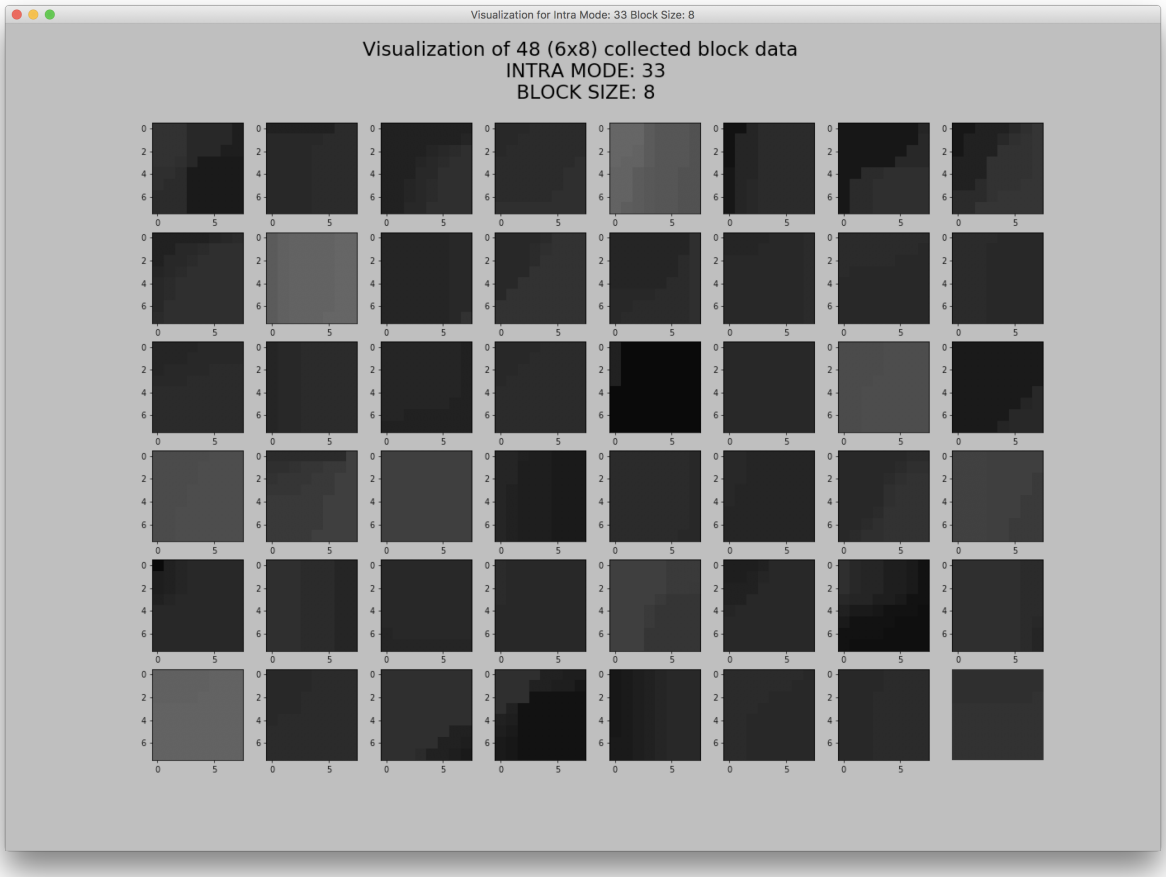


Fig. 12: Figure 1-33. Intra Mode: 33, Block Size: 8x8

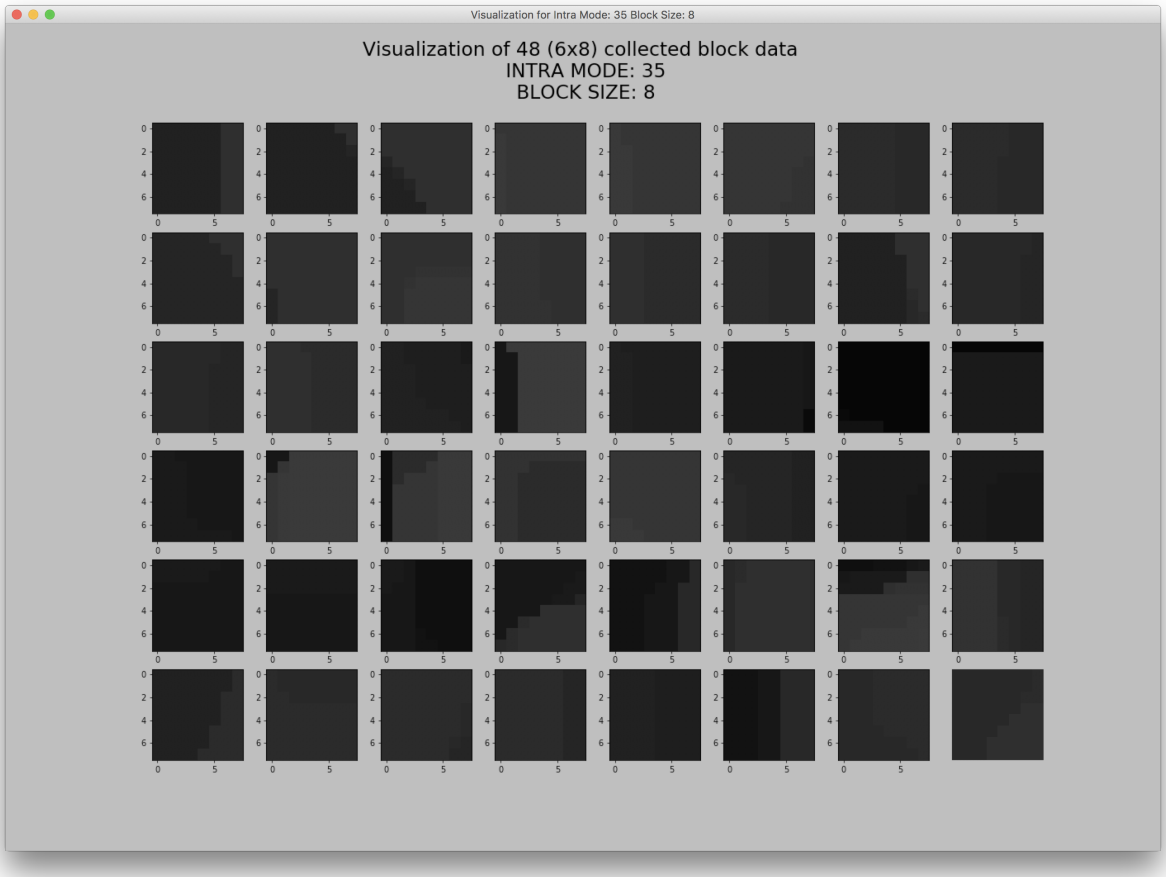


Fig. 13: Figure 1-35. Intra Mode: 35, Block Size: 8x8

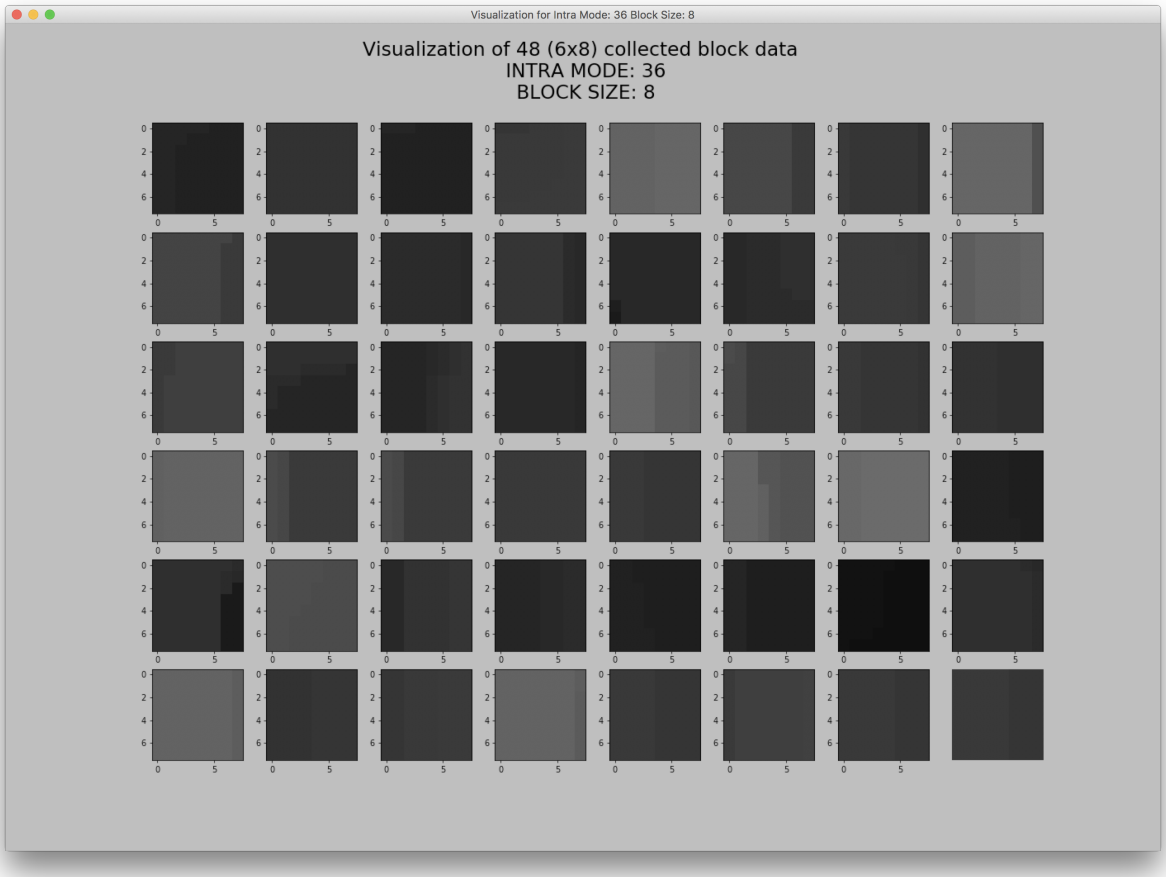


Fig. 14: Figure 1-36. Intra Mode: 36, Block Size: 8x8

6.4 Examples for block size 8x8

6.5 Examples for block size 16x16

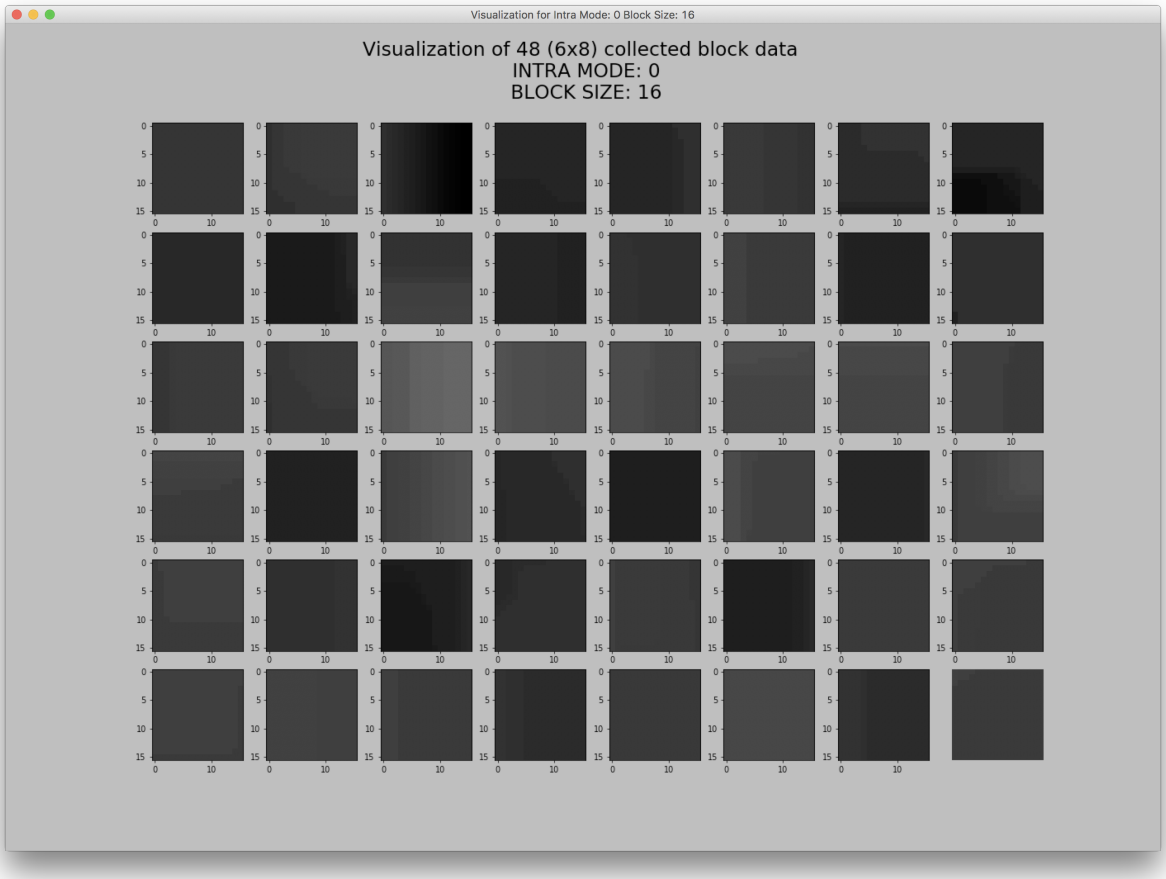


Fig. 15: Figure 2-0. Intra Mode: 0, Block Size: 16x16

6.6 Examples for block size 32x32

6.7 Examples for block size 64x64

Figure 4-0. Intra Mode: 0, Block Size: 64x64

Figure 4-1. Intra Mode: 1, Block Size: 64x64

Figure 4-2. Intra Mode: 2, Block Size: 64x64

Figure 4-3. Intra Mode: 3, Block Size: 64x64

Figure 4-4. Intra Mode: 4, Block Size: 64x64

Figure 4-5. Intra Mode: 5, Block Size: 64x64

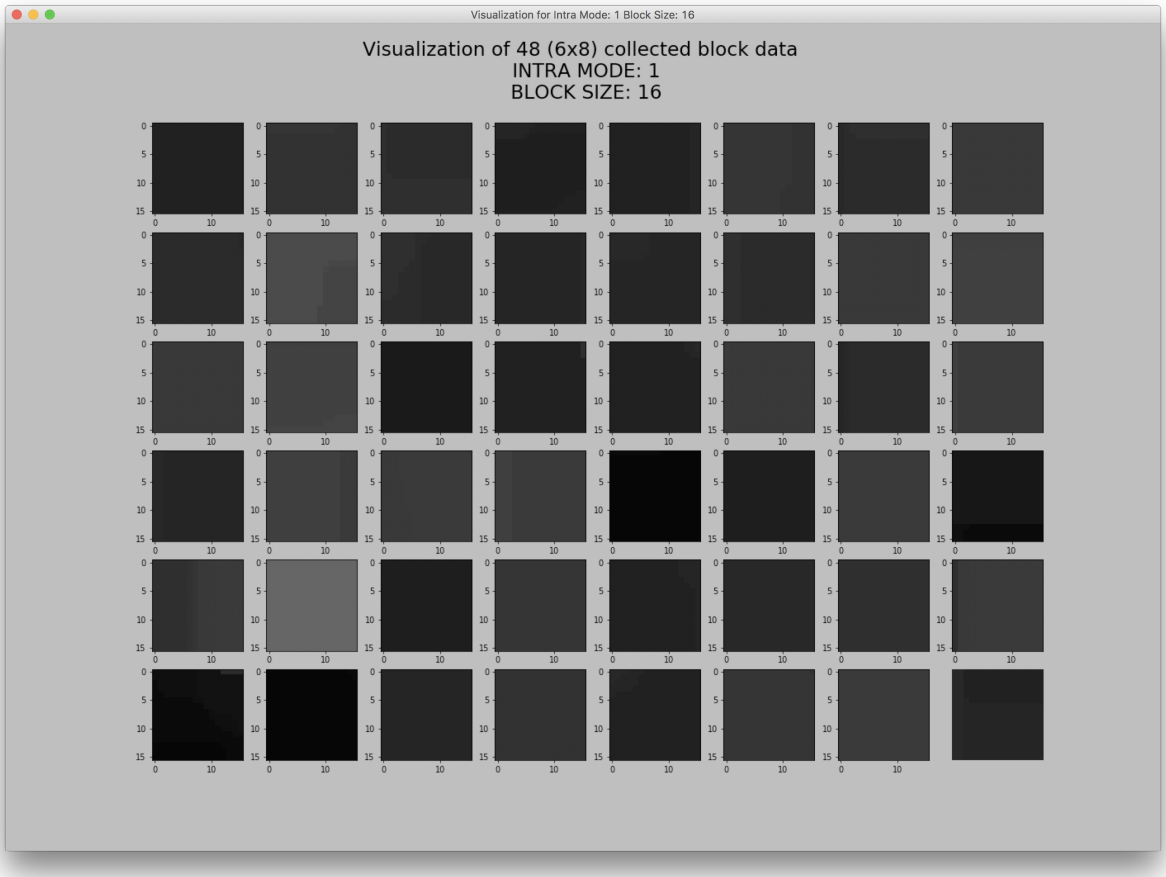


Fig. 16: Figure 2-1. Intra Mode: 1, Block Size: 16x16

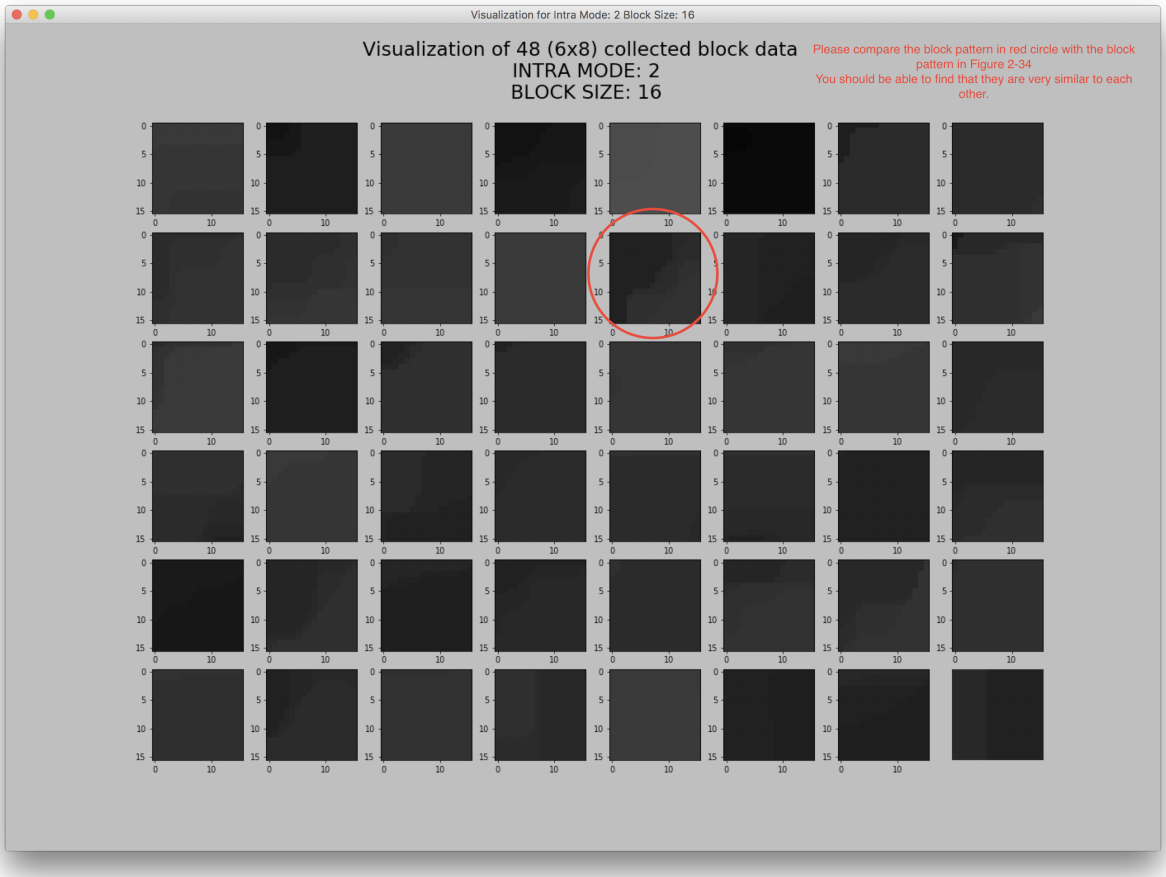


Fig. 17: Figure 2-2. Intra Mode: 2, Block Size: 16x16

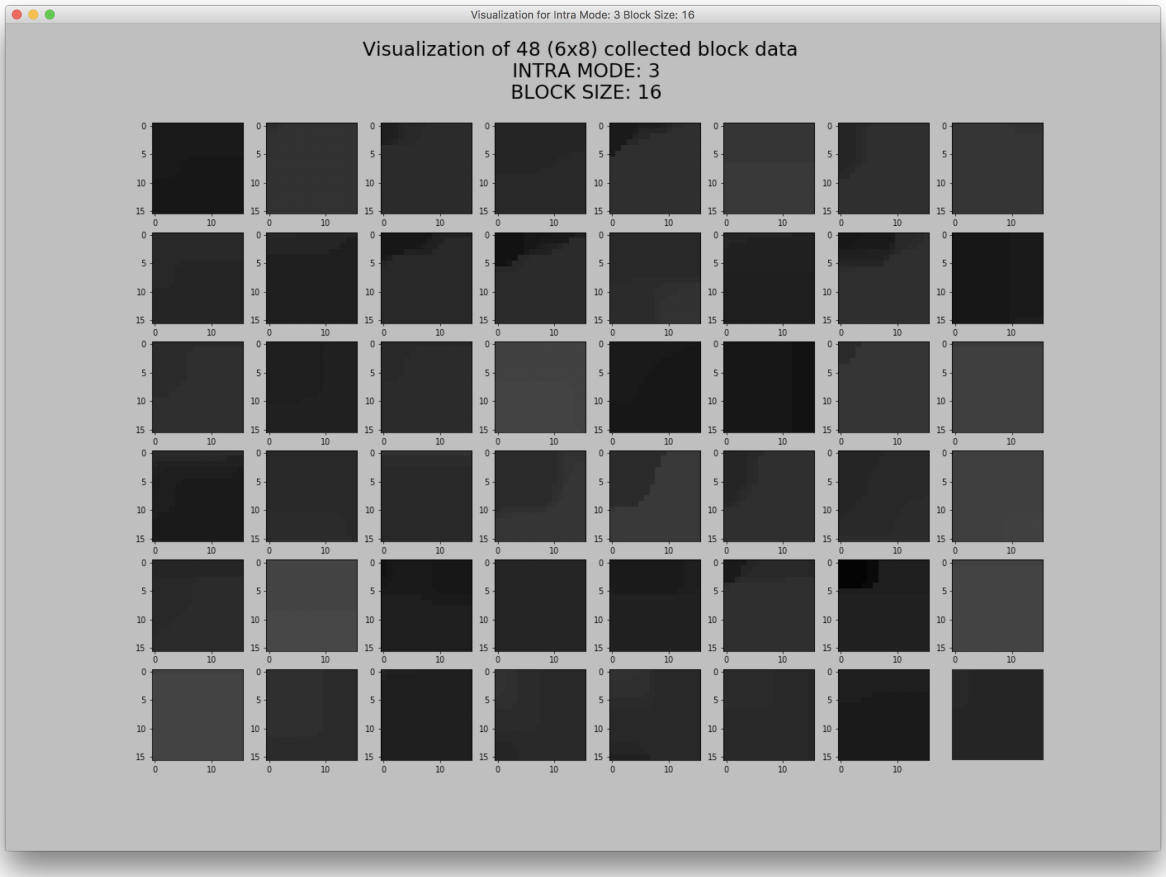


Fig. 18: Figure 2-3. Intra Mode: 3, Block Size: 16x16

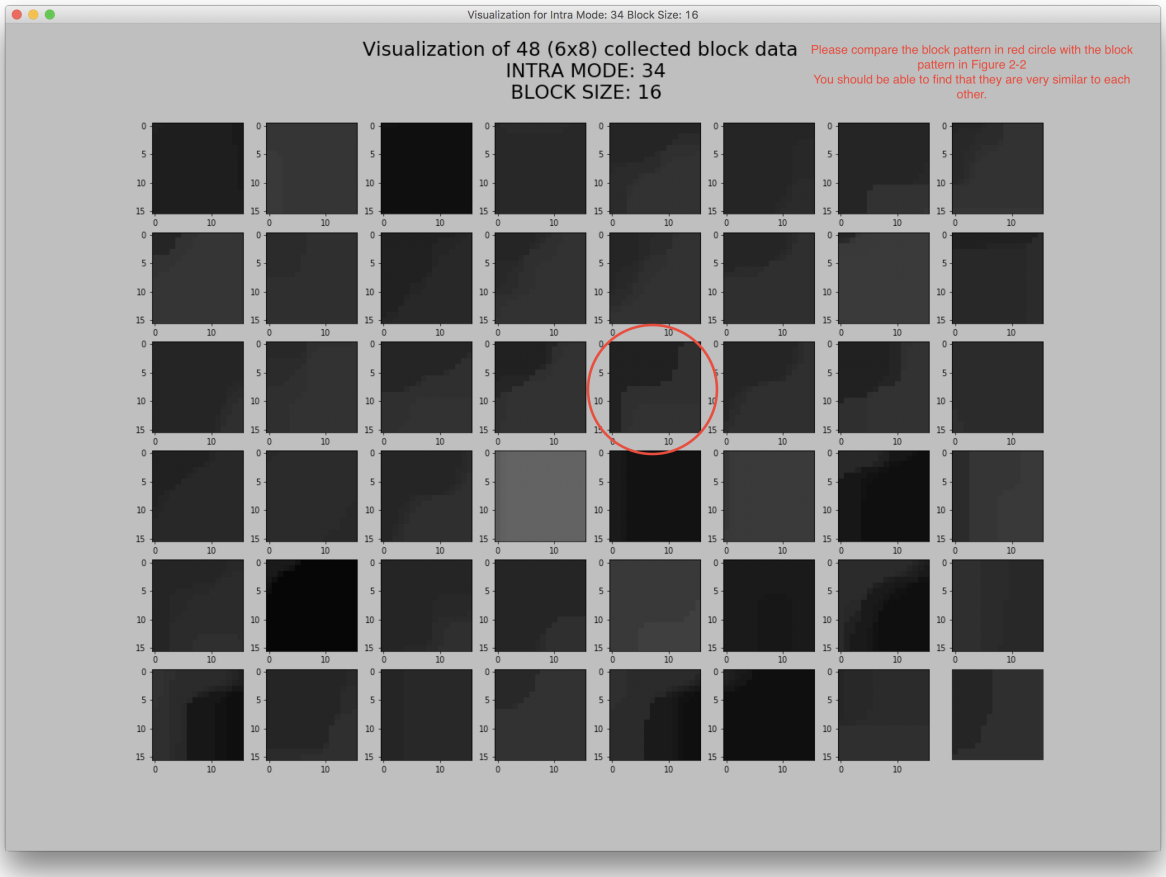


Fig. 19: Figure 2-34. Intra Mode: 34, Block Size: 16x16

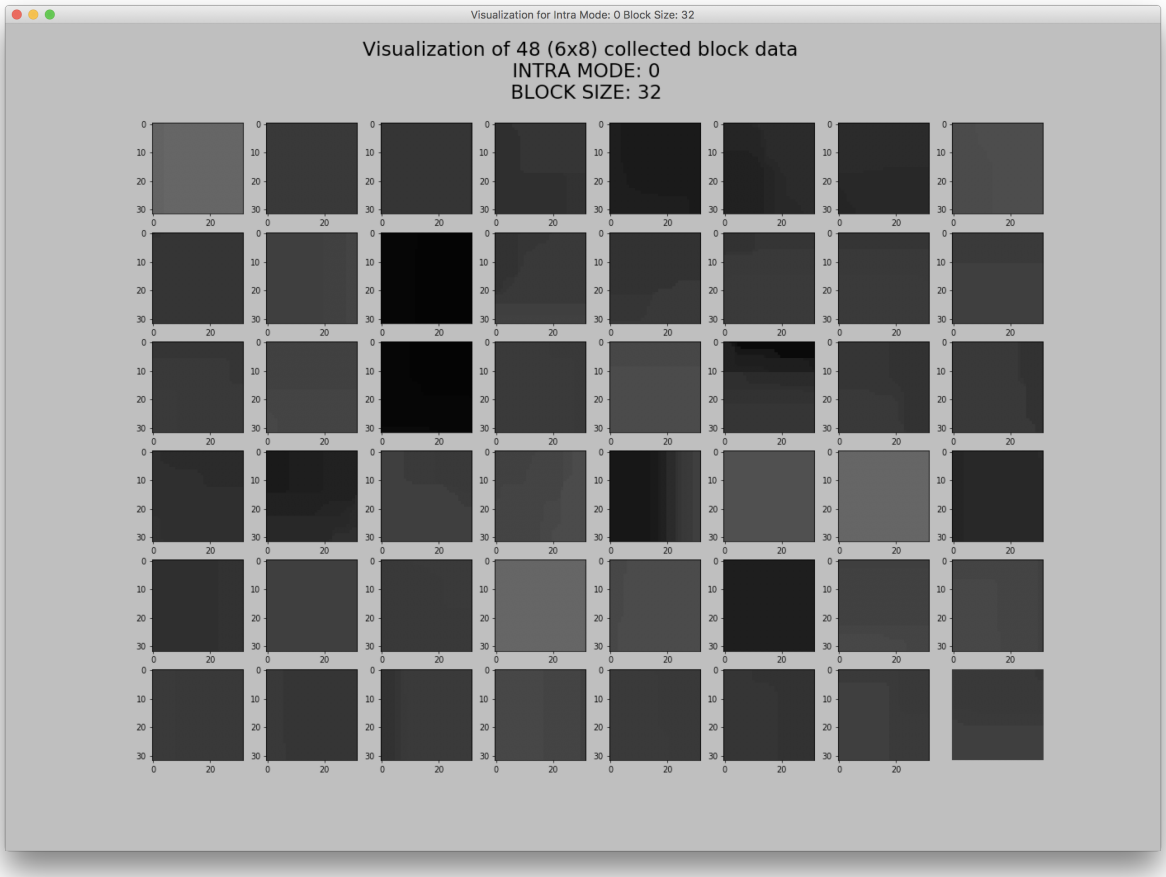


Fig. 20: Figure 3-0. Intra Mode: 0, Block Size: 32x32

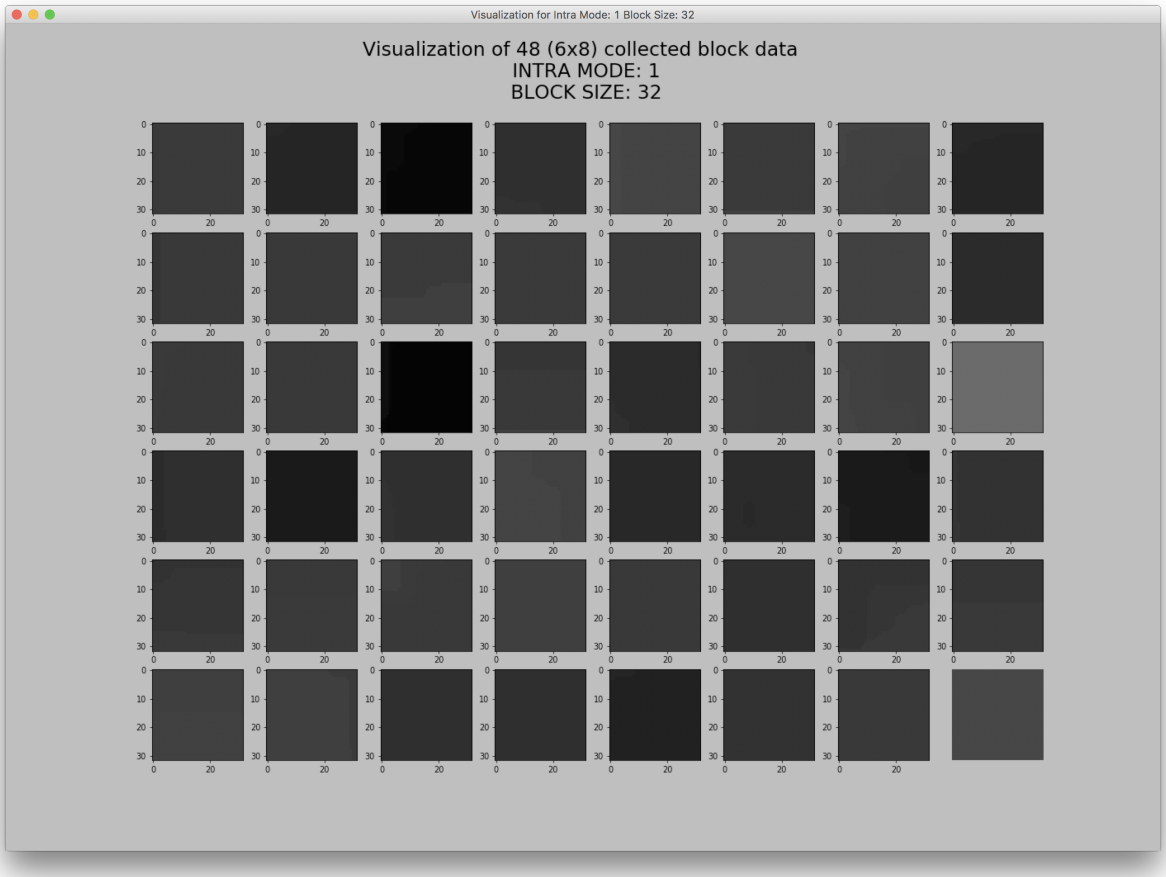


Fig. 21: Figure 3-1. Intra Mode: 1, Block Size: 32x32

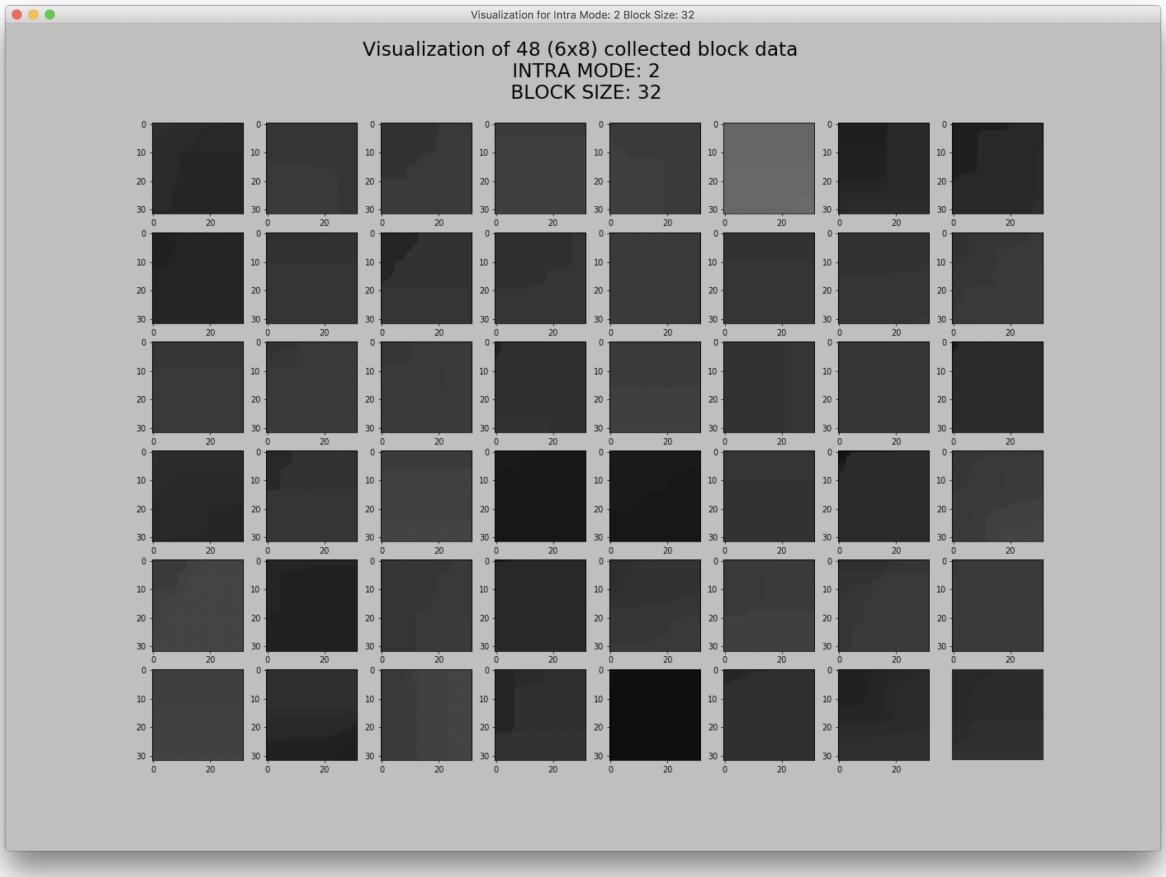


Fig. 22: Figure 3-2. Intra Mode: 2, Block Size: 32x32

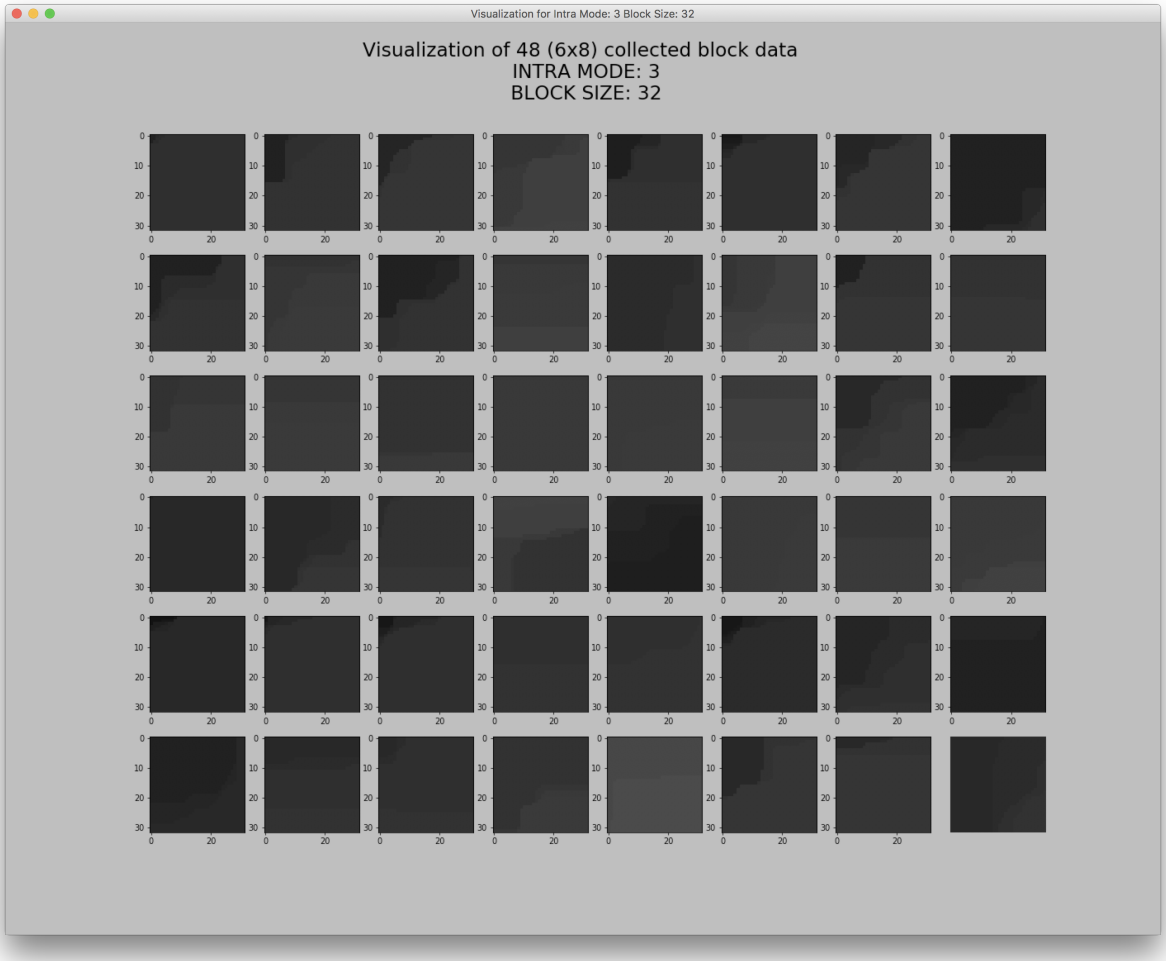


Fig. 23: Figure 3-3. Intra Mode: 3, Block Size: 32x32

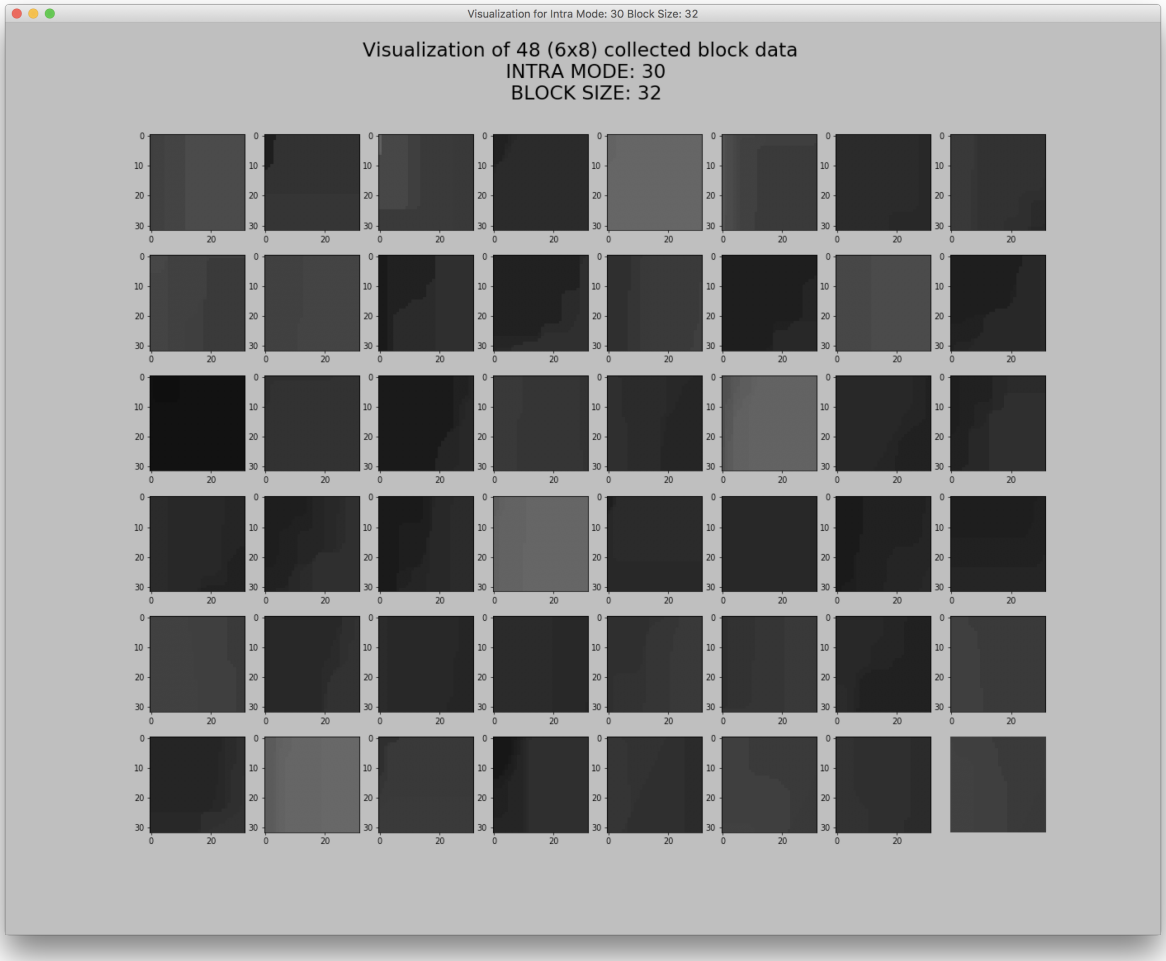


Fig. 24: Figure 3-30. Intra Mode: 30, Block Size: 32x32

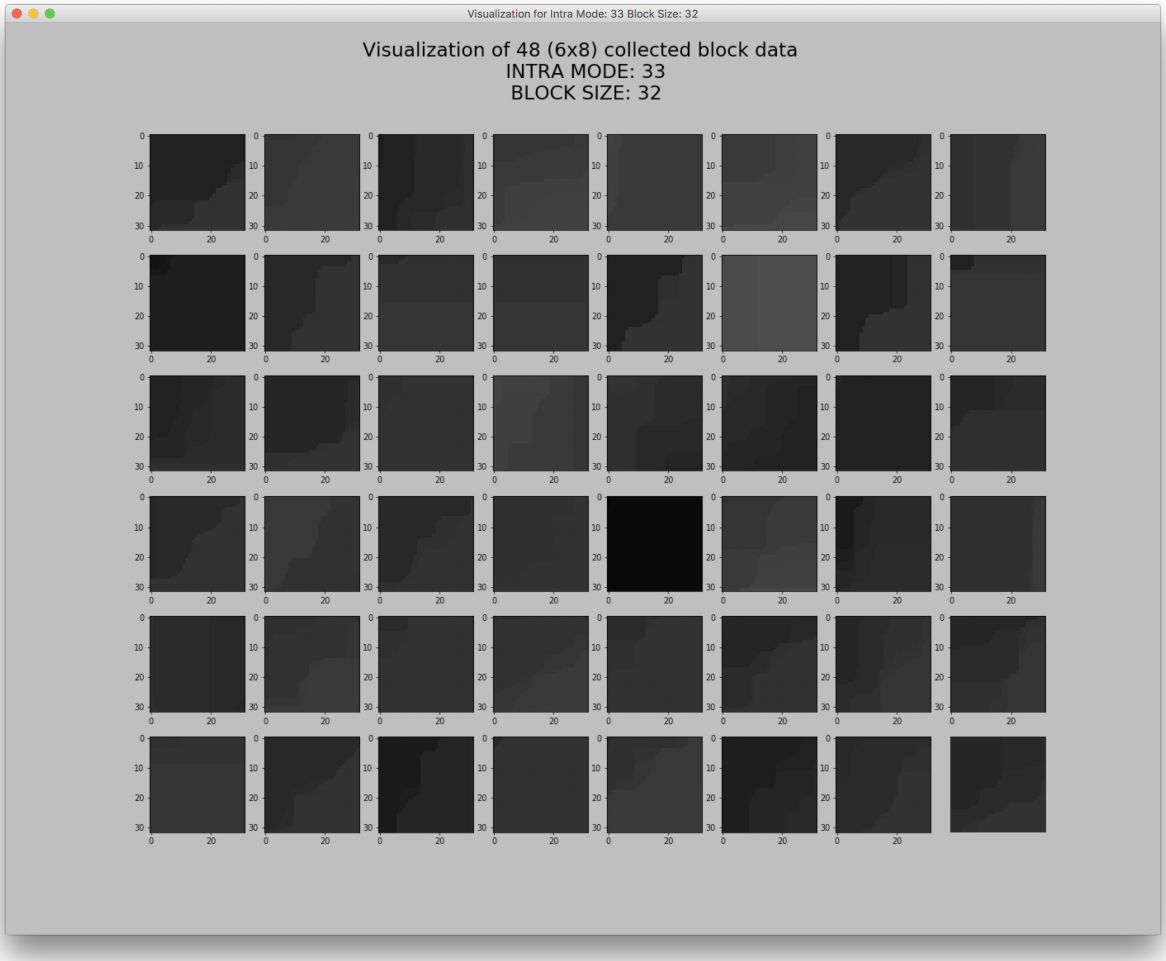


Fig. 25: Figure 3-33. Intra Mode: 33, Block Size: 32x32

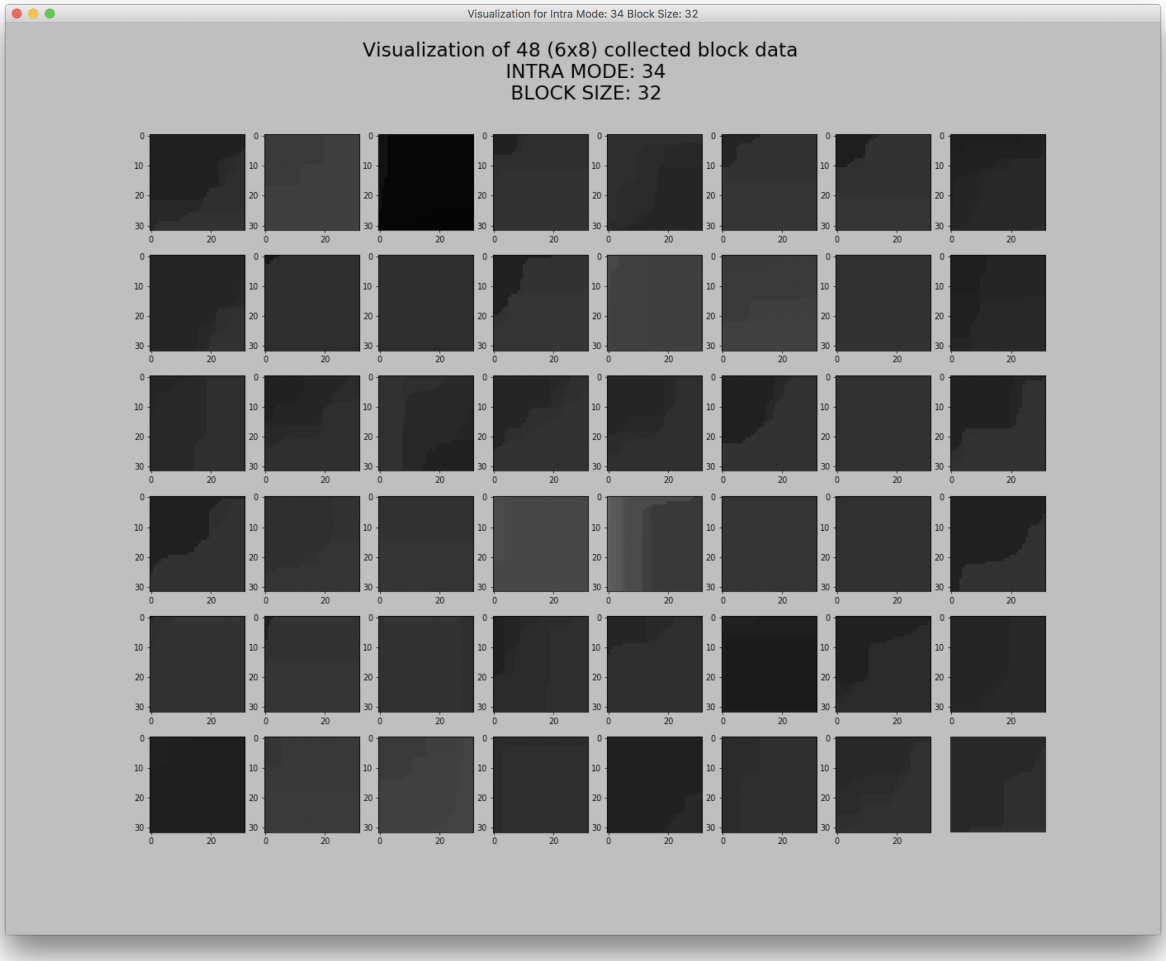


Fig. 26: Figure 3-34. Intra Mode: 34, Block Size: 32x32

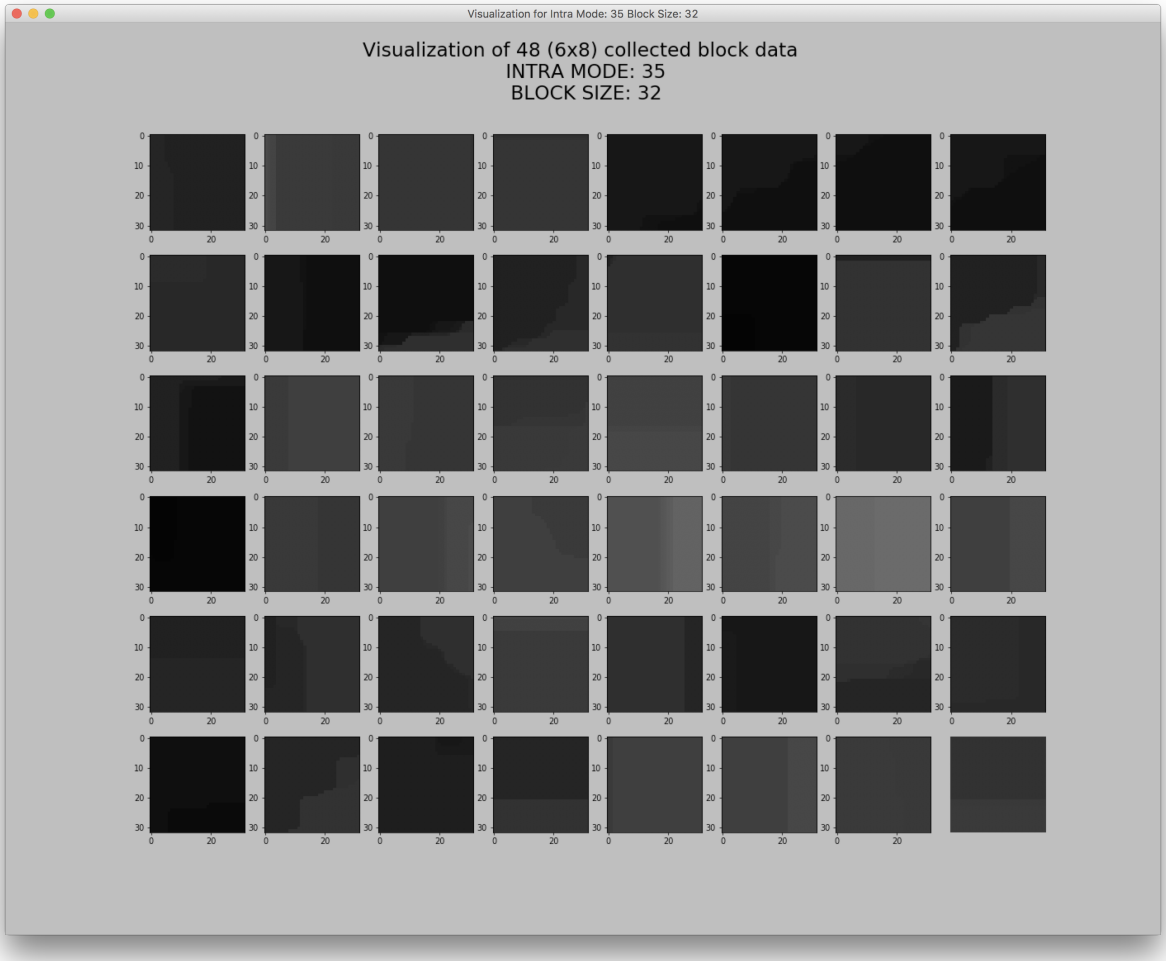


Fig. 27: Figure 3-35. Intra Mode: 35, Block Size: 32x32

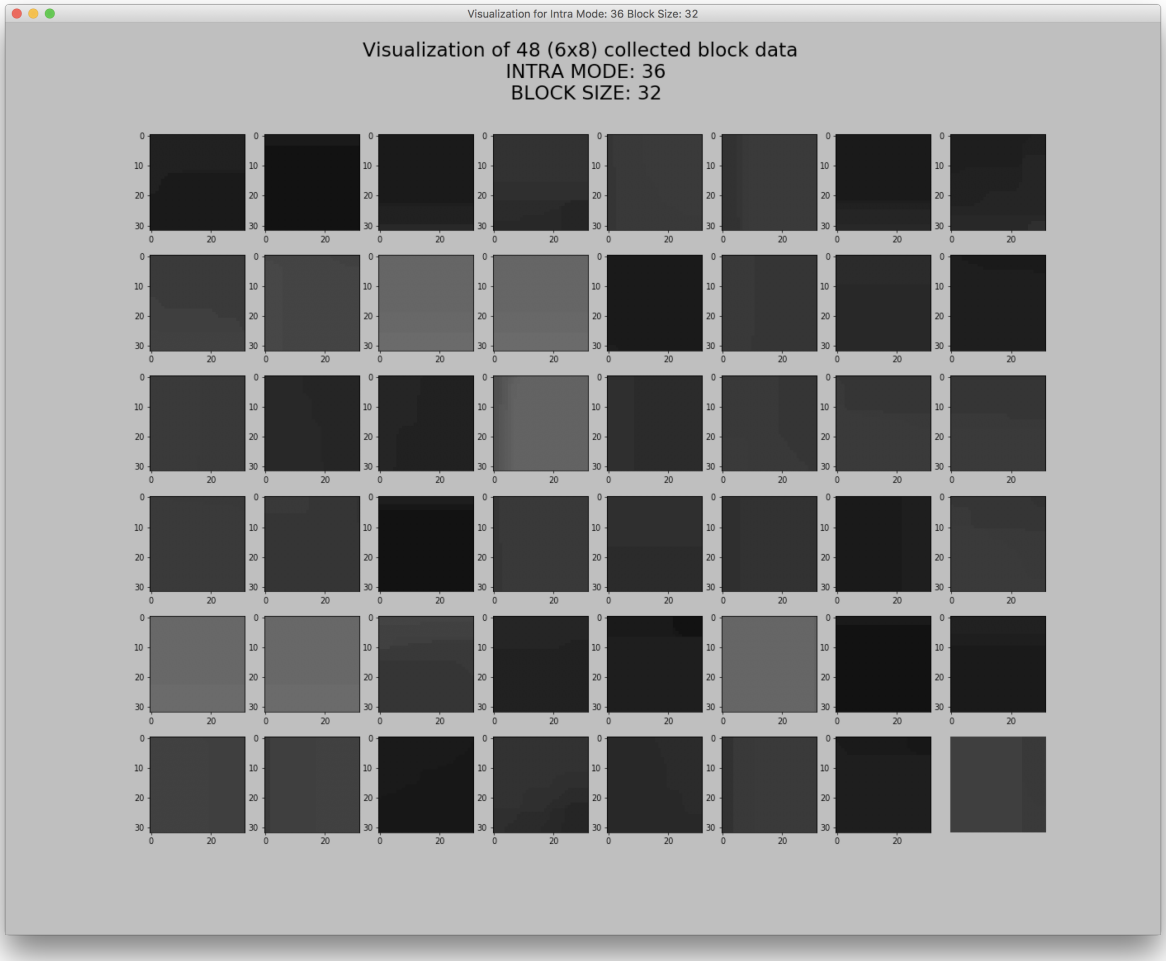
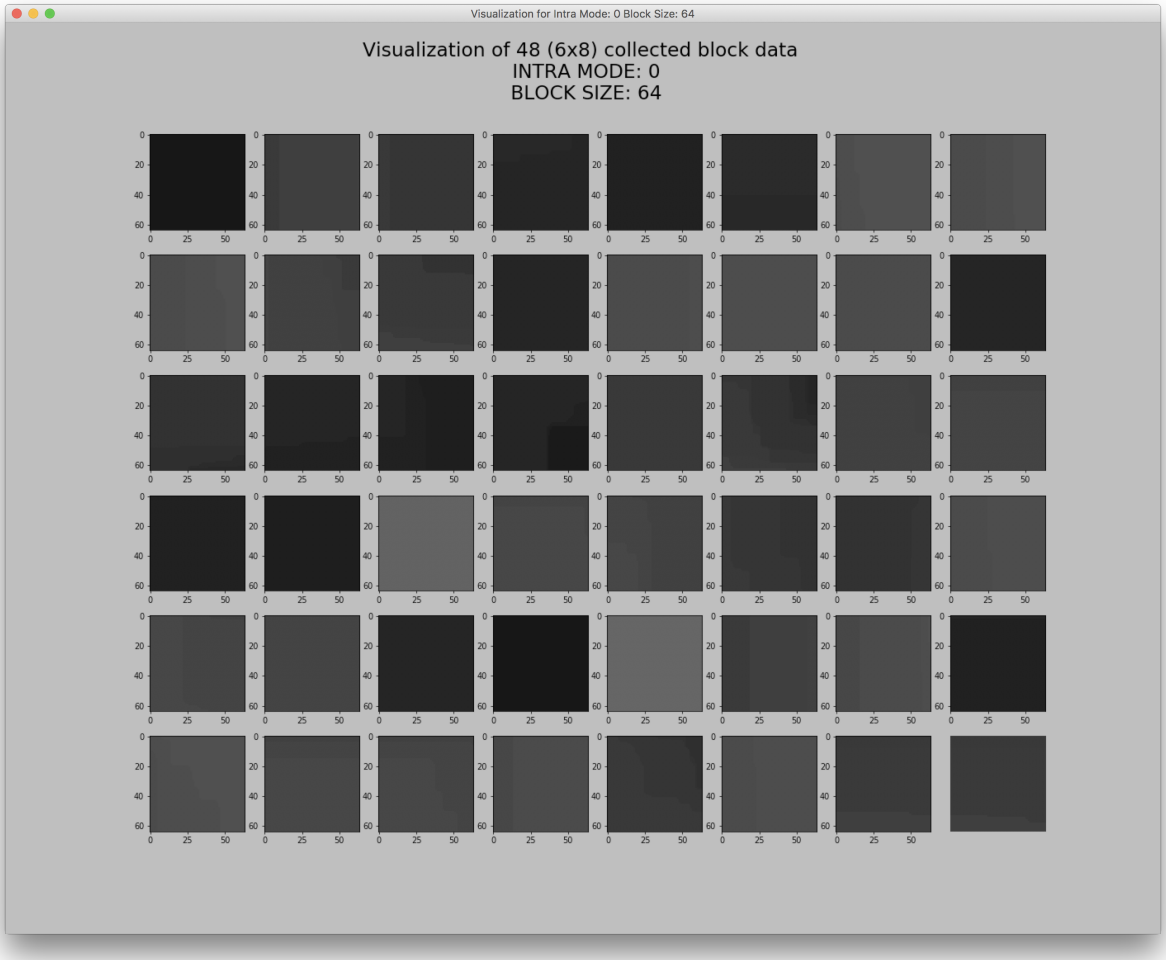
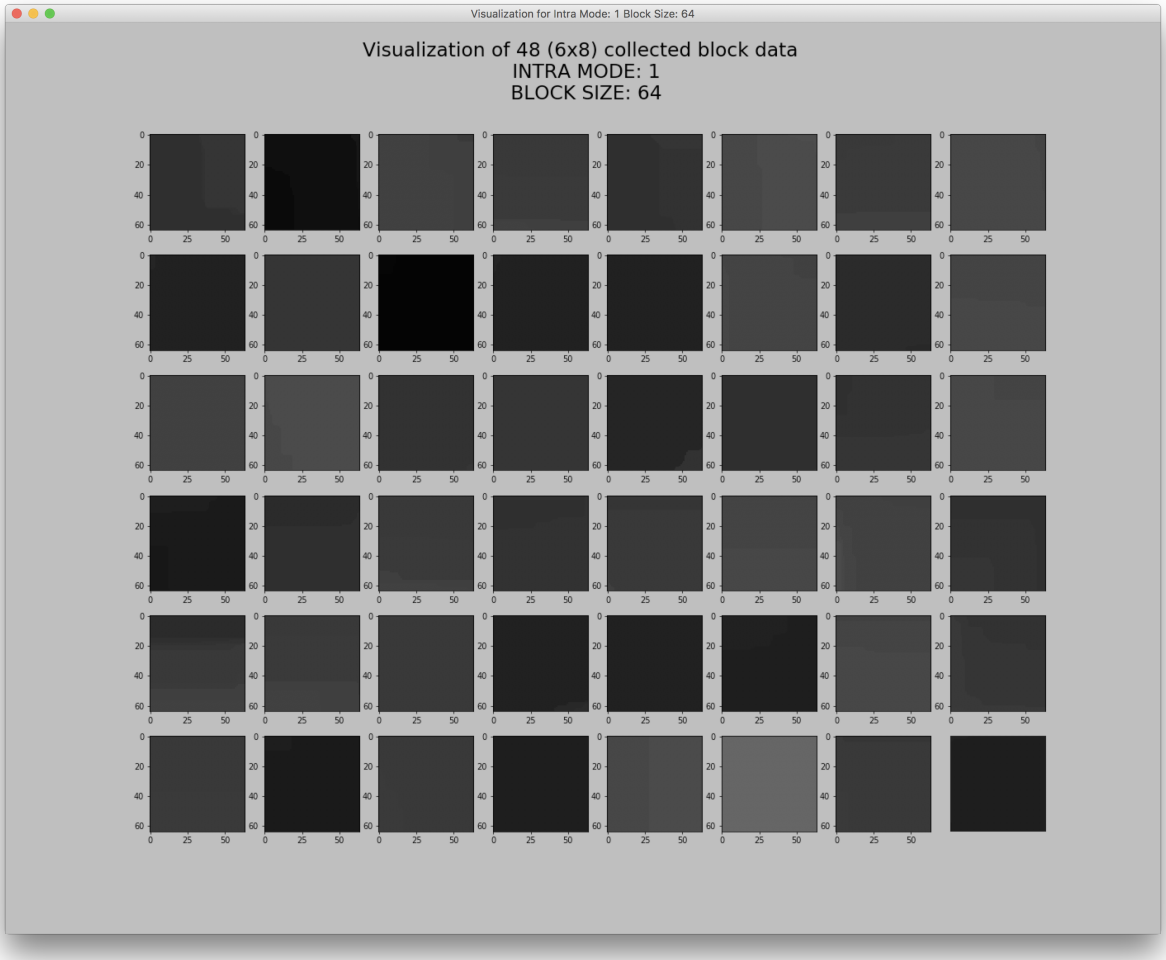
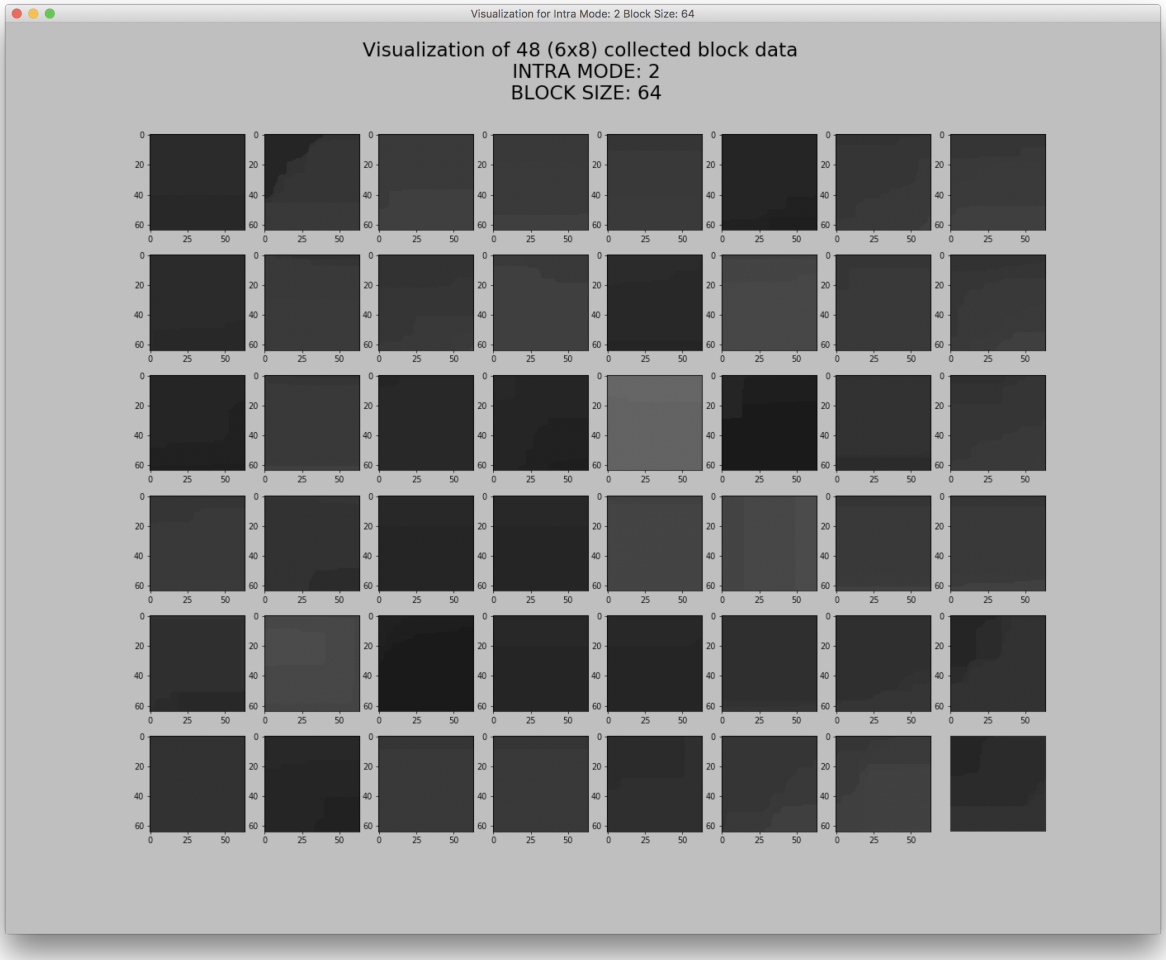
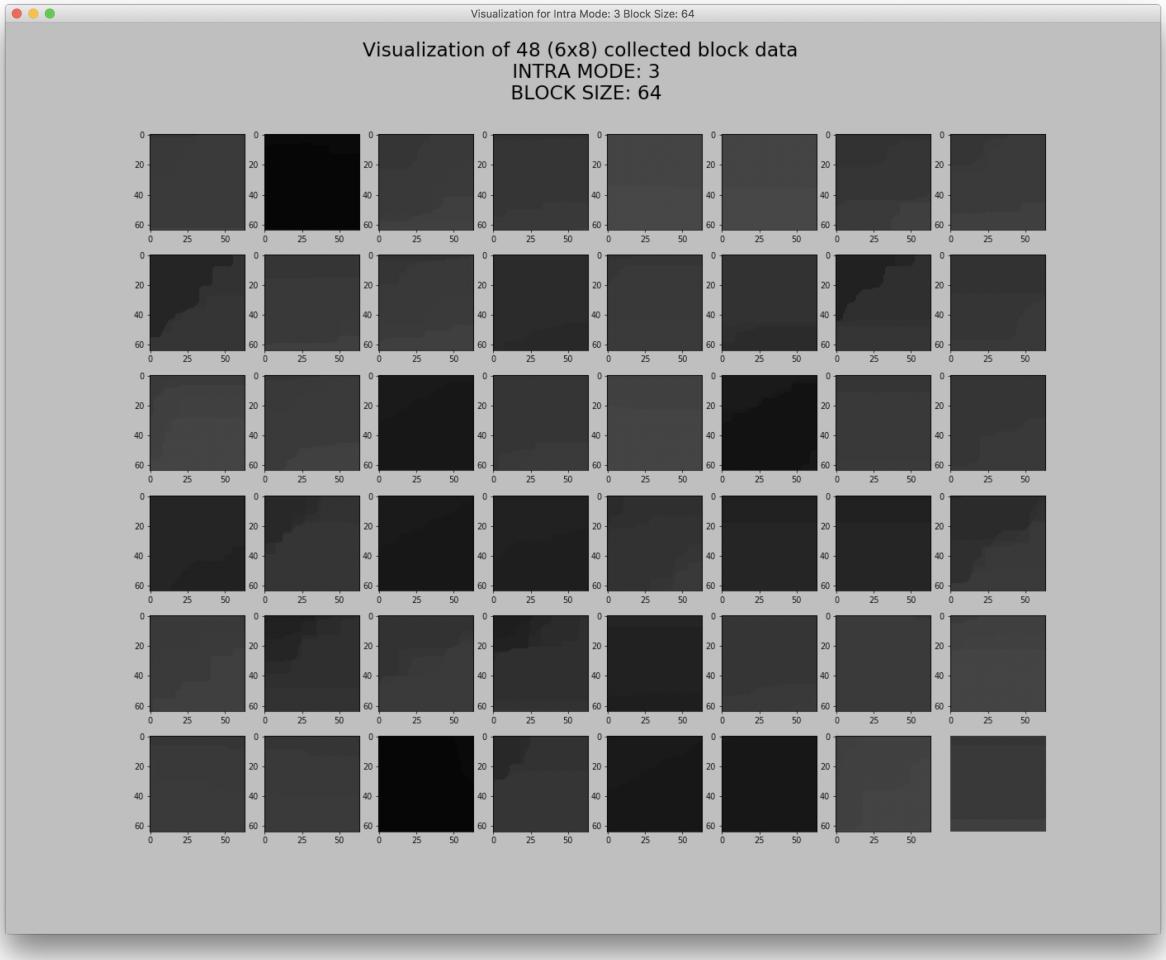


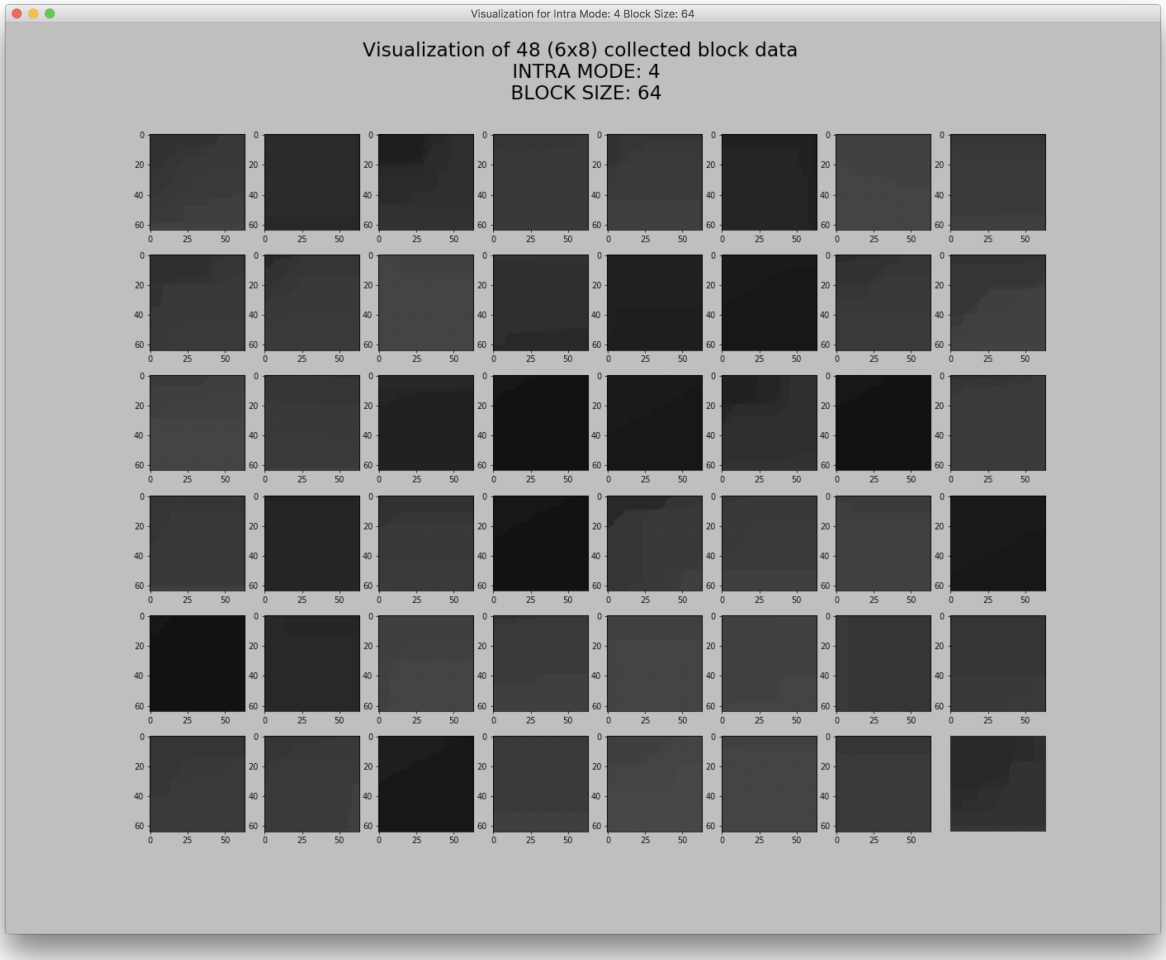
Fig. 28: Figure 3-36. Intra Mode: 36, Block Size: 32x32

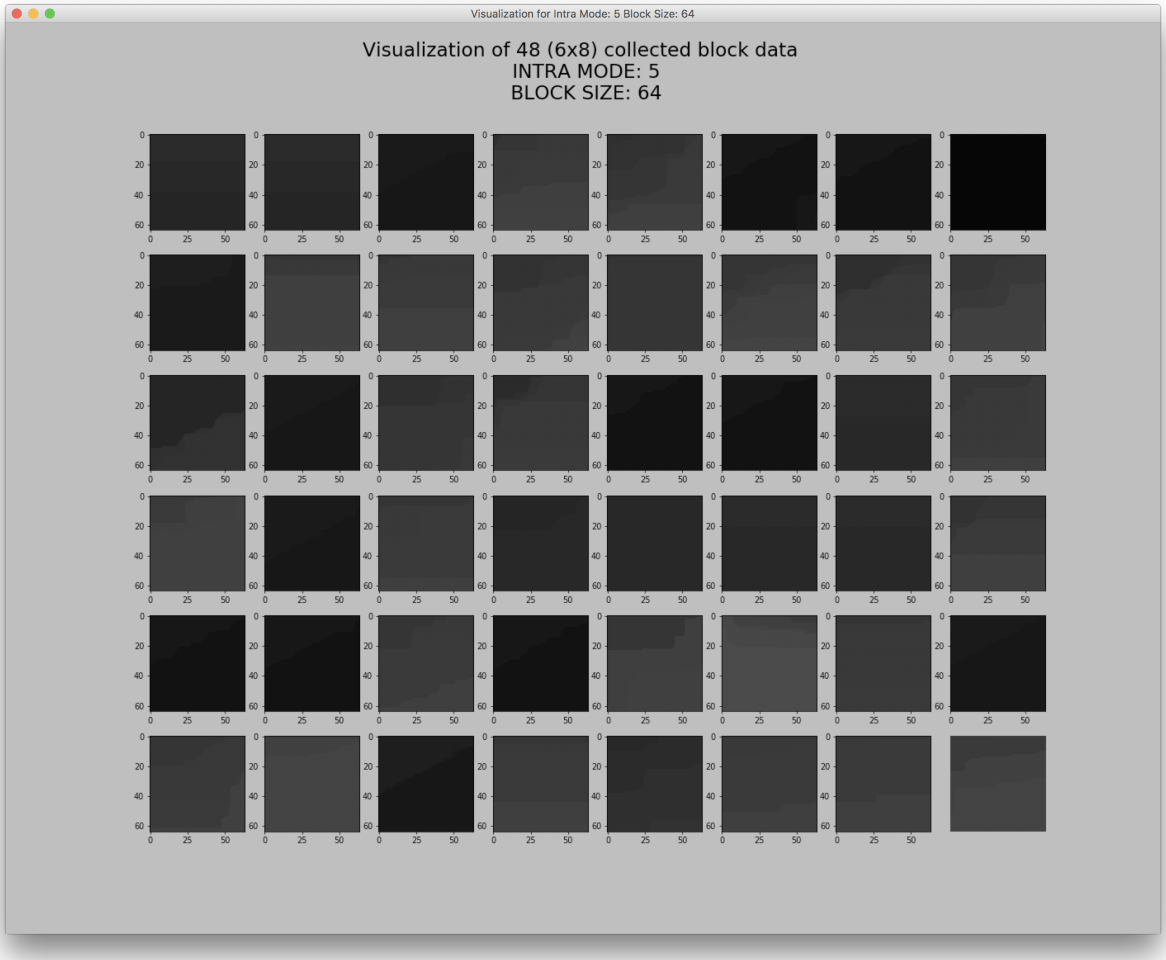

Visualization for Intra Mode: 0 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 0
BLOCK SIZE: 64


Visualization for Intra Mode: 1 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 1
BLOCK SIZE: 64


Visualization for Intra Mode: 2 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 2
BLOCK SIZE: 64


Visualization for Intra Mode: 3 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 3
BLOCK SIZE: 64


Visualization for Intra Mode: 4 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 4
BLOCK SIZE: 64


Visualization for Intra Mode: 5 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 5
BLOCK SIZE: 64

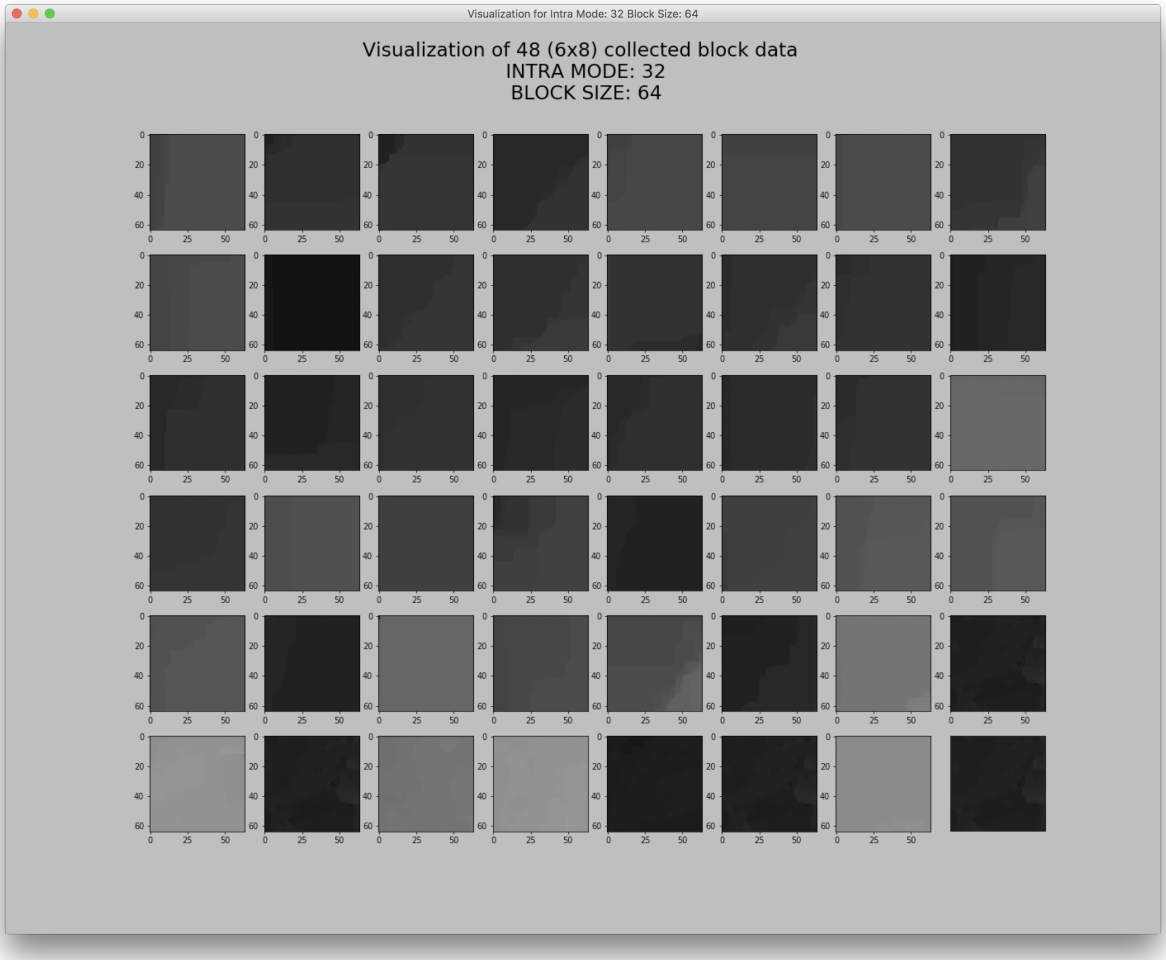

Visualization for Intra Mode: 32 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 32
BLOCK SIZE: 64

Figure 4-32. Intra Mode: 32, Block Size: 64x64

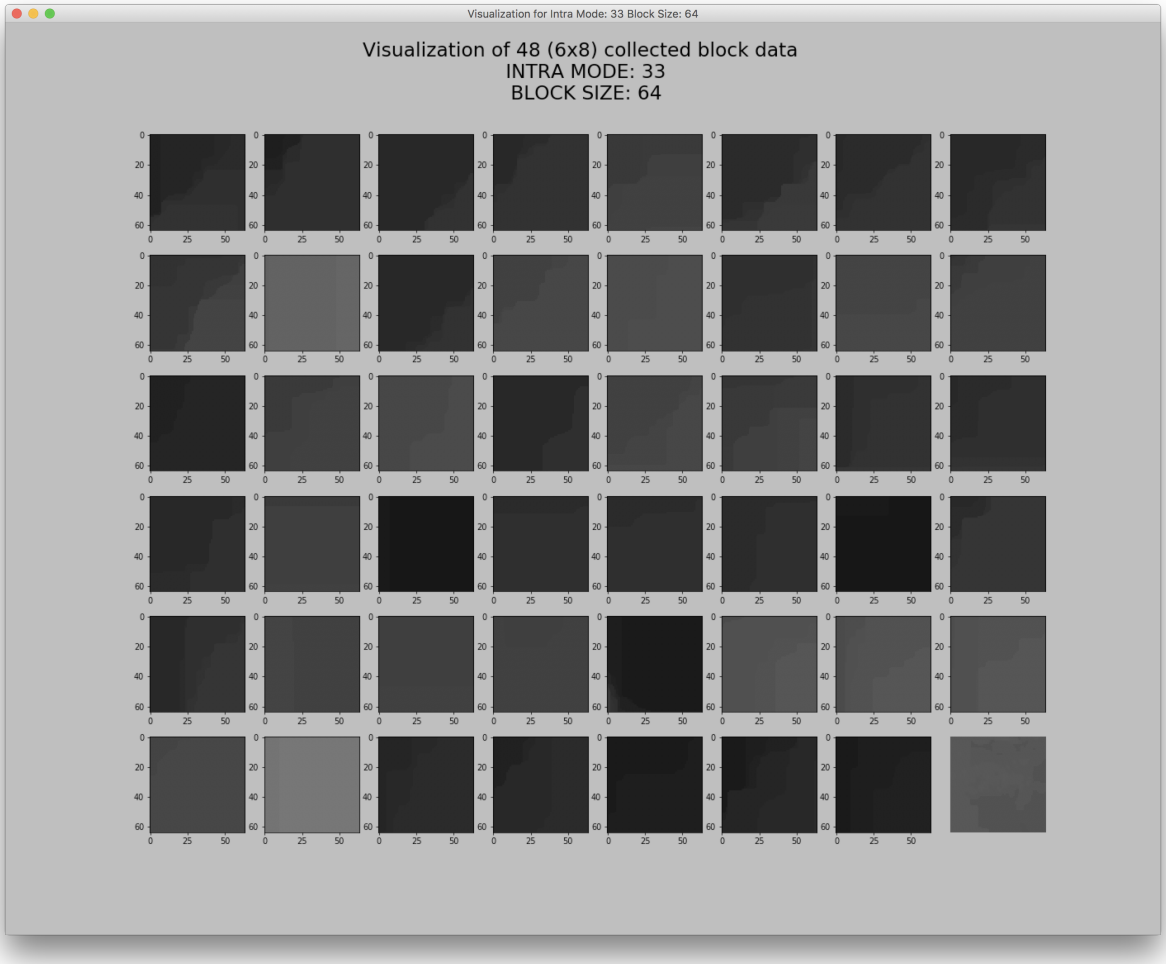
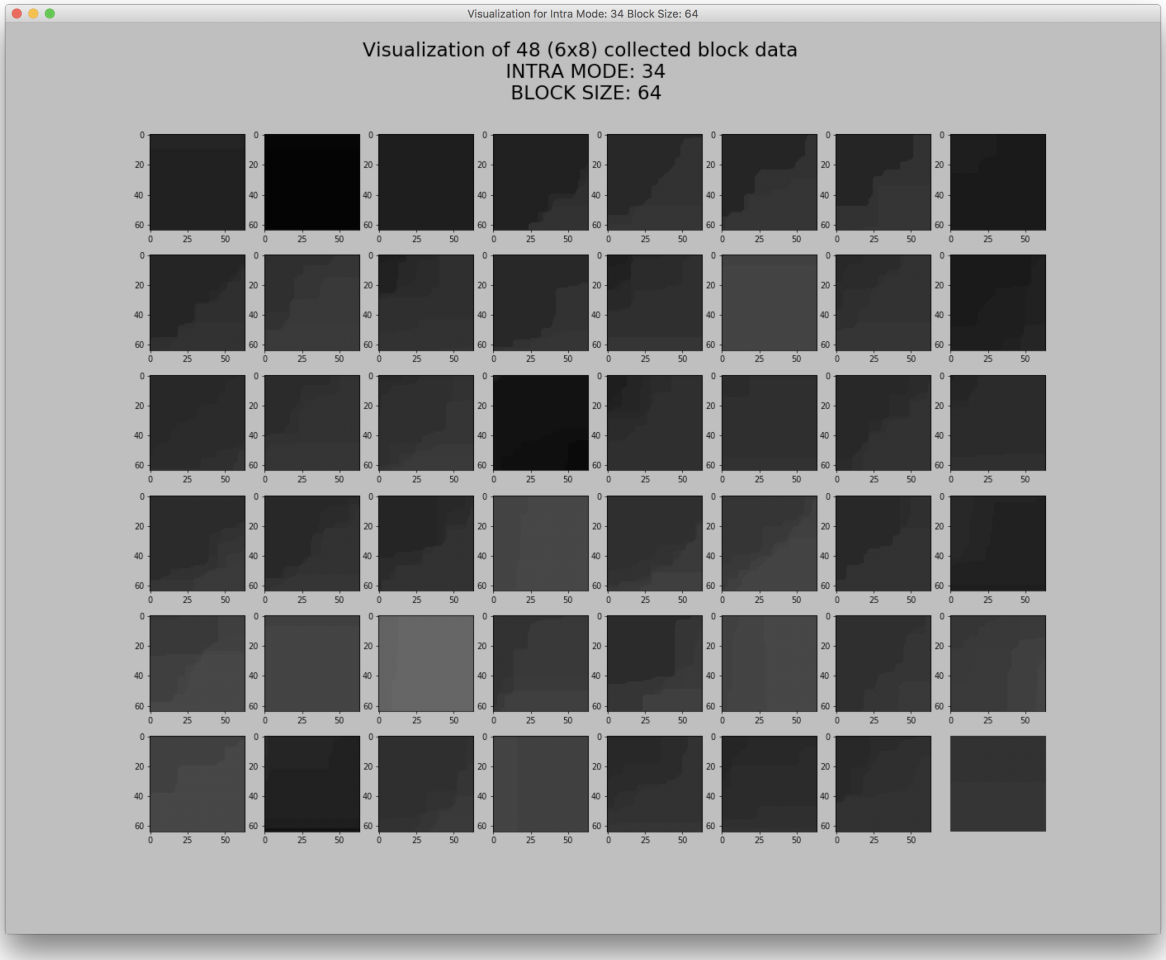


Figure 4-33. Intra Mode: 33, Block Size: 64x64

Figure 4-34. Intra Mode: 34, Block Size: 64x64


Visualization for Intra Mode: 34 Block Size: 64
Visualization of 48 (6x8) collected block data
INTRA MODE: 34
BLOCK SIZE: 64

CHAPTER 7

Edge Strength Analysis

7.1 Motivation originated from experience

To know the motivation, first you have to know some experience.

I have tried to train the data by using 37 classes, including mode [0, 1, 2, . . . , 34, DMM1, DMM4]. (2 + 33 + 2 = 37)

7.1.1 Confusion matrix obtained during training process

Confusion matrix provided for downloading.

```
Confusion matrix after 10342 steps
Confusion matrix after 20622 steps
Confusion matrix after 30757 steps
Confusion matrix after 40884 steps
Confusion matrix after 51009 steps
Confusion matrix after 61168 steps
```

7.1.2 Statistics

1. total train samples: 2900*37
2. Batch size:128
3. 61168*128/2900/37 = 73 epoch

- Refer to the **confusing matrix**, after 73 epochs, the model still don't understand the features of mode 0, 1, 35, 36, which possibly means the CNN is not capable of learning features for 0, 1, 35, 36. Or it can mean my network size is not large enough to learn the features inside mode 0, 1, 35, 36. Or it can mean CNN is not enough for learn them. I think RNN is able to gain much lifting in classification accuracy by taking the information related to time into consideration. But due to the limited time, i have not tried it yet.

- Refer to the **confusing matrix**, the CNN must be able to distinguish between angular modes.

After visualizing the angular modes, the blocks with very weak edges for each mode are frequently observed. They looks like the planar or DC mode.

I've tried to train the model while the blocks with weak edges are kept in the training data. Results are not good.

Then it is natural to come to the conclusion that it is not good to mix smooth block with angular blocks for classification under the confidence that our neural net is good in the sense of both architecture and hyper params. (The confidence originated from the high accuracy on CIFAR-10/100 data set, and the conclusion in the paper of **wide residul network** are verified clearly.)

The conclusion `smooth regions will trap CNN in ill condition` is also found in the journal paper below:

`CU Partition Mode Decision for HEVC Hardwired Intra Encoder Using CNN`

So we decided to remove the smooth regions.

When trying to remove the smooth region, we are facing a question:

`How to define the smooth regions?`

Well, see below for the answer.

7.2 Algorithm designed for edge analysis

For answering the question of `How to define the smooth regions`, we can think like this: can we define the sharpness of the edges?

Yes. We can.

See below code snippets for a quick understanding of how we define the edge strength:

```
for each sample (a row) in the collected data set (a csv file):
    feature = the_pixel_data_of_a_square_block_as_a_matrix
        for i in range(width_of_the_block - 1):
            for j in range(width_of_the_block - 1):
                #calculating the hor and ver strength
                horizontal_strength = \
                    features[i][j] + \
                    features[i + 1][j] - \
                    features[i][j + 1] - \
                    features[i + 1][j + 1]
                vertical_strength = \
                    features[i][j] + \
                    features[i][j + 1] - \
                    features[i + 1][j] - \
                    features[i + 1][j + 1]
                # calculating the power
                strength = horizontal_strength ** 2 + vertical_strength ** 2
                # put each strength into an numpy array to get the
                # total strength of a block (or you can say a line
                # in the csv file)
                data = np.append(data, np.array([strength]))
                total_strength += strength

        assert (data.ndim == 1)
```

Then calculate top (width*2 && non-zero) average.

```
# calculating top (width*2 && non-zero) average.
# step1: top width*2 values in the numpy arrary
top_k = data[np.argsort(data)][data.size - RESHAPE * 2:]
assert (top_k.ndim == 1)
# step2: non-zero values (because sometimes the edge length can be
# short. We only want the sharpness. We do not want smooth regions
# to affect the sharpness.)
data = top_k[top_k.nonzero()]
# e.g., [[2, 0], [0, 0]], i exclude it from the concept of sharp
data = data[np.where(data > 8)]
# all the strength are zero. (that is to say , it is like DC mode)
if data.size == 0:
    ave = 0
    data = np.array([0])
else:
    ave = np.mean(data)
    data = np.array([ave])

# add ave of the blocks grouping by each mode.
# calculate the ave by dividing the number of blocks of each mode
```

We encouraging the readers to check the python codes provided below for downloading to understand the algorithm used by us. (The python codes for the algorithm is short and easy to understand!)

Edge analysis algorithm implemented in python is provided for downloading.

`Edge Analysis in Python`

CHAPTER 8

Convolutional Neural Network

Download codebase from GitHub: https://github.com/PharrellWANG/fdc_resnet_v3

ConvNet architectures make the explicit assumption that the inputs are images/blocks.

Fig. 1: Figure 1: Convolutional Neural Networks

why resnet? 1. easier to train 2. faster to converge

Fig. 2: Figure 2: Basic Unit of ResNet

Fig. 3: Figure 3: Our Neural Network Structure

CHAPTER 9

Settings

- All the models are trained from scratch.
- Since the datasets for block size 32x32 and size 64x64 are too small, models are not trained for them. Instead, we use Bilinear Interpolation to resize the block to do the prediction for them using learned model for block size 16x16.

1. No padding/cropping/flipping applied. No data augmentation applied. The orignal data is distorted enough by nature. See *Data Visualization* section to get a taste.
2. Momentum optimizer 0.9.
3. Learning rate schedule: 0.01 (<20k), 0.001 (<40k), 0.0001 (<60k), 0.00001 (else).
4. Weight decay rate: 0.0002.
5. Batch size 128.
6. Filters [16, 16, 32, 64], residual units for last three layers: 5

Note:

1. Block size 4x4 is for **PU**, while the smallest size of **CU** is 8x8.
2. From below training results, our model is not so applicable to blocks of size 4x4.
3. DMM is not applied for size 64x64.

Our **deep learning** strategy is targeted to **CU** from **size 8x8** to **size 64x64**, both *texture* and *depth*.

CHAPTER 10

Training for blocks of size 4x4

10.1 Results

The model **cannot** learn well for size 4x4, only top-28 is fine.

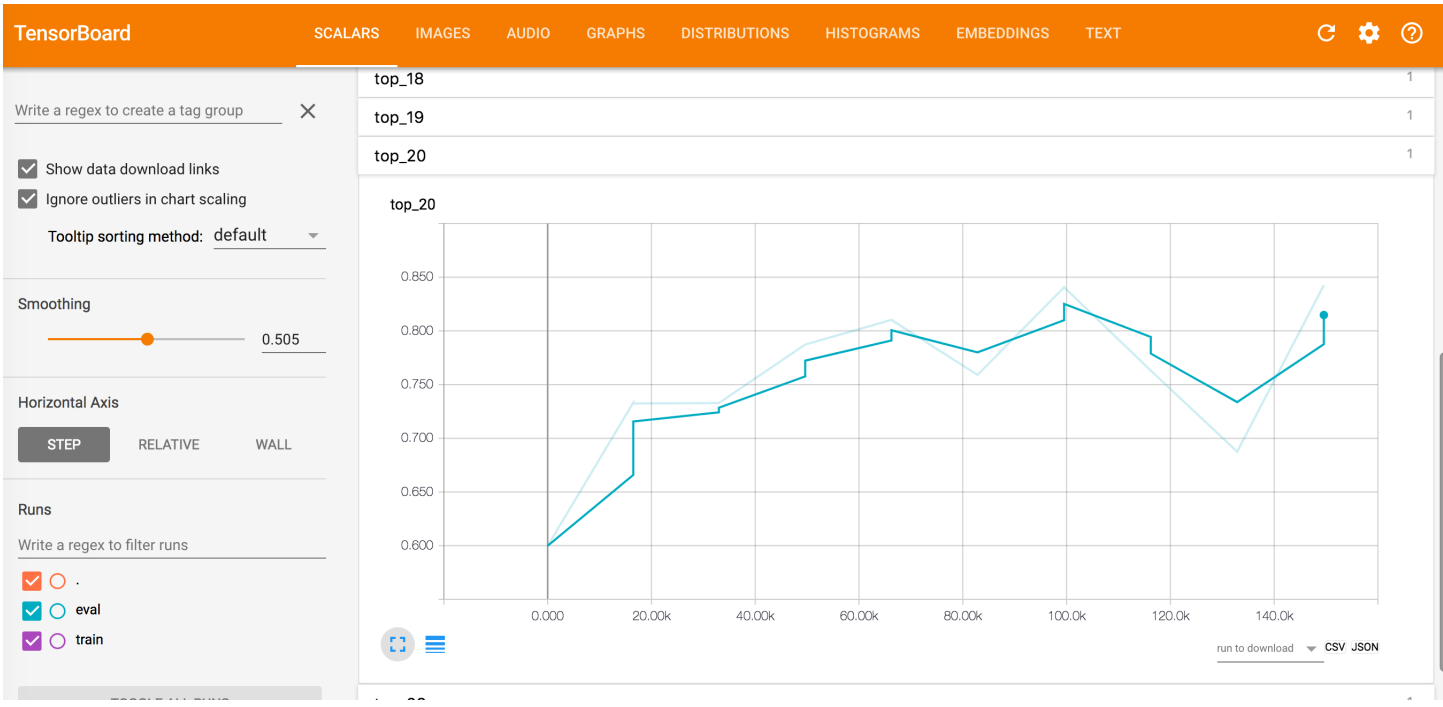


Fig. 1: Figure 1.1 Top 20 Accuracy

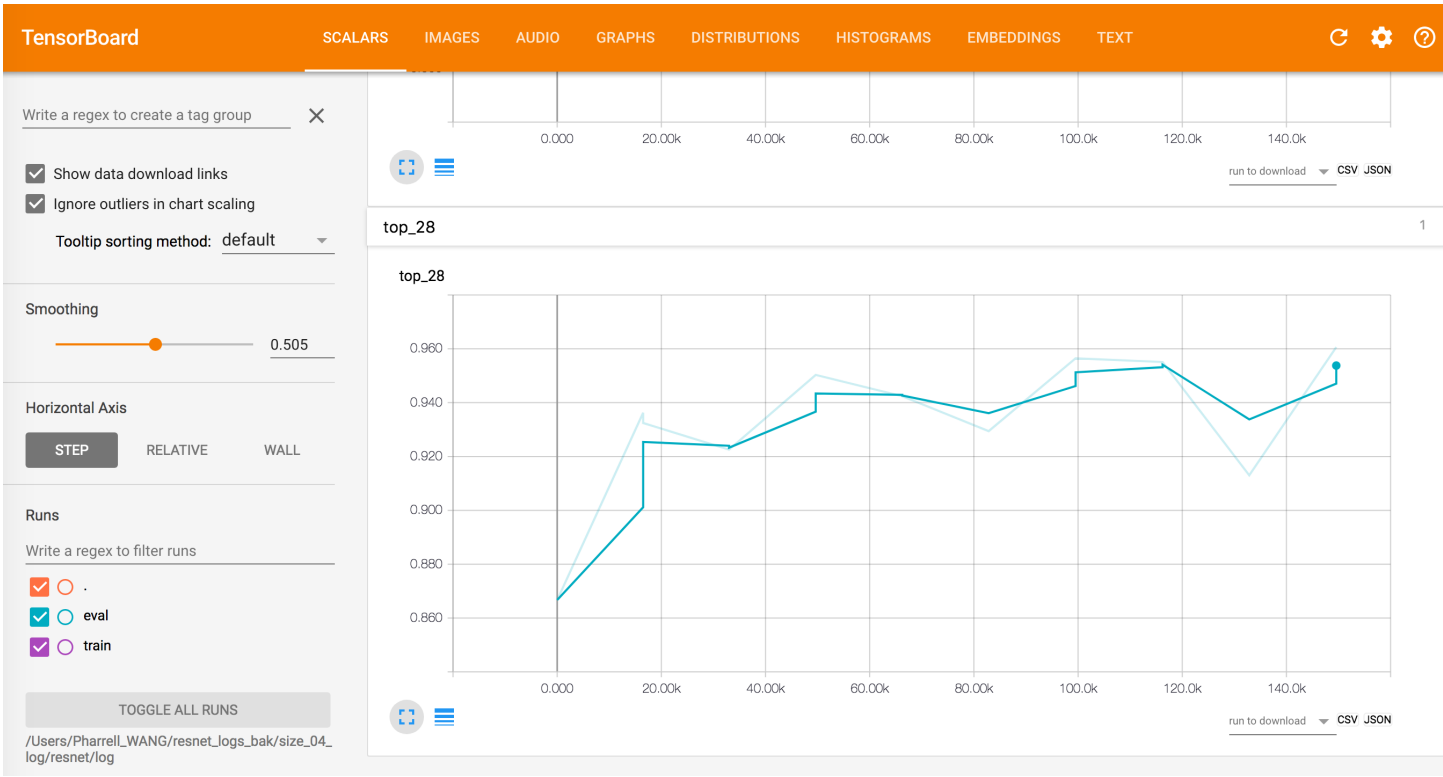


Fig. 2: Figure 1.2 Top 28 Accuracy

CHAPTER 11

Training for blocks of size 8x8

11.1 Results

The model indeed **can** learn something for size 8x8. Top 16 is fine, which can reduce the angular modes by half.

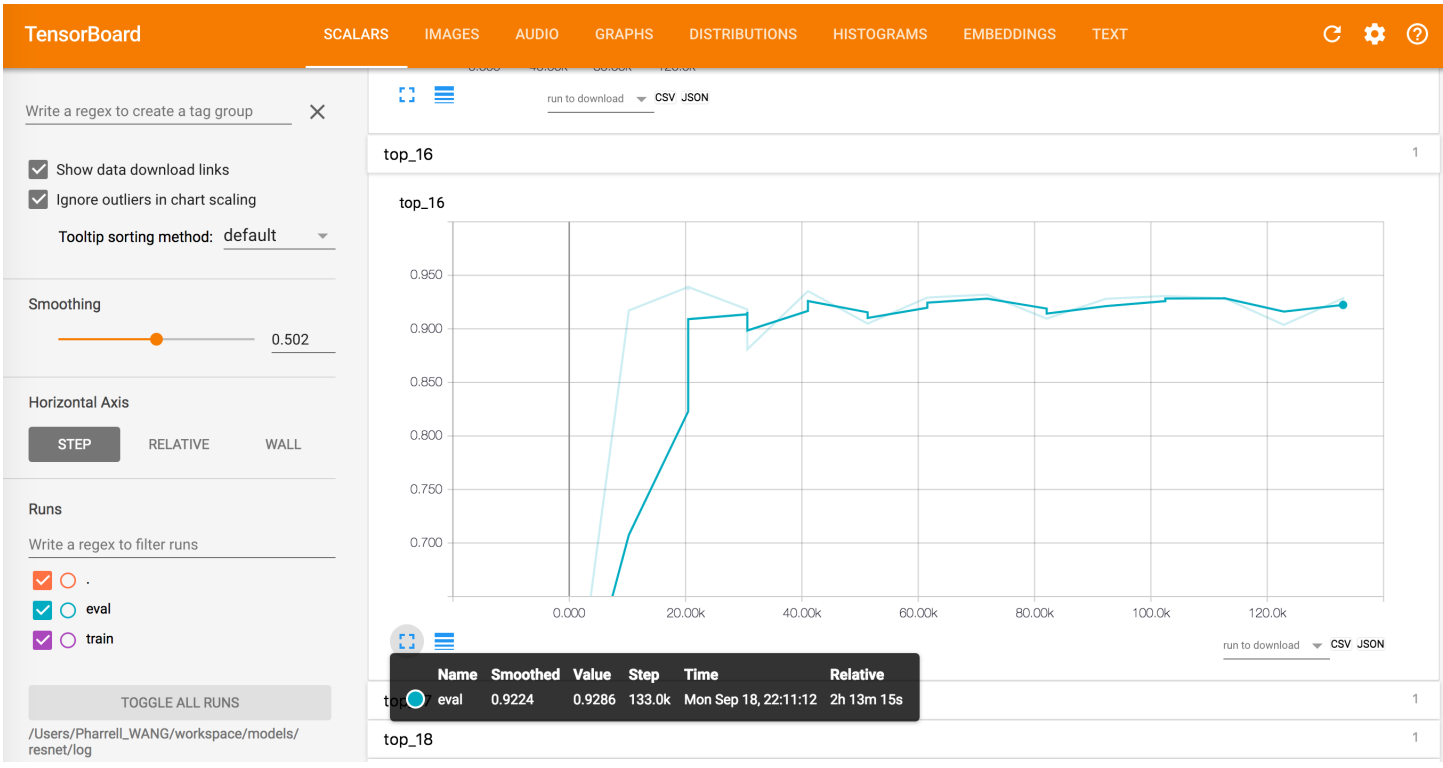


Fig. 1: Figure 2.1 Top 16 Accuracy for block size 08x08

CHAPTER 12

Training for blocks of size 16x16

12.1 Results

The model indeed **can** learn something for size 16x16. Top 16 is fine, which can reduce the angular modes by half.

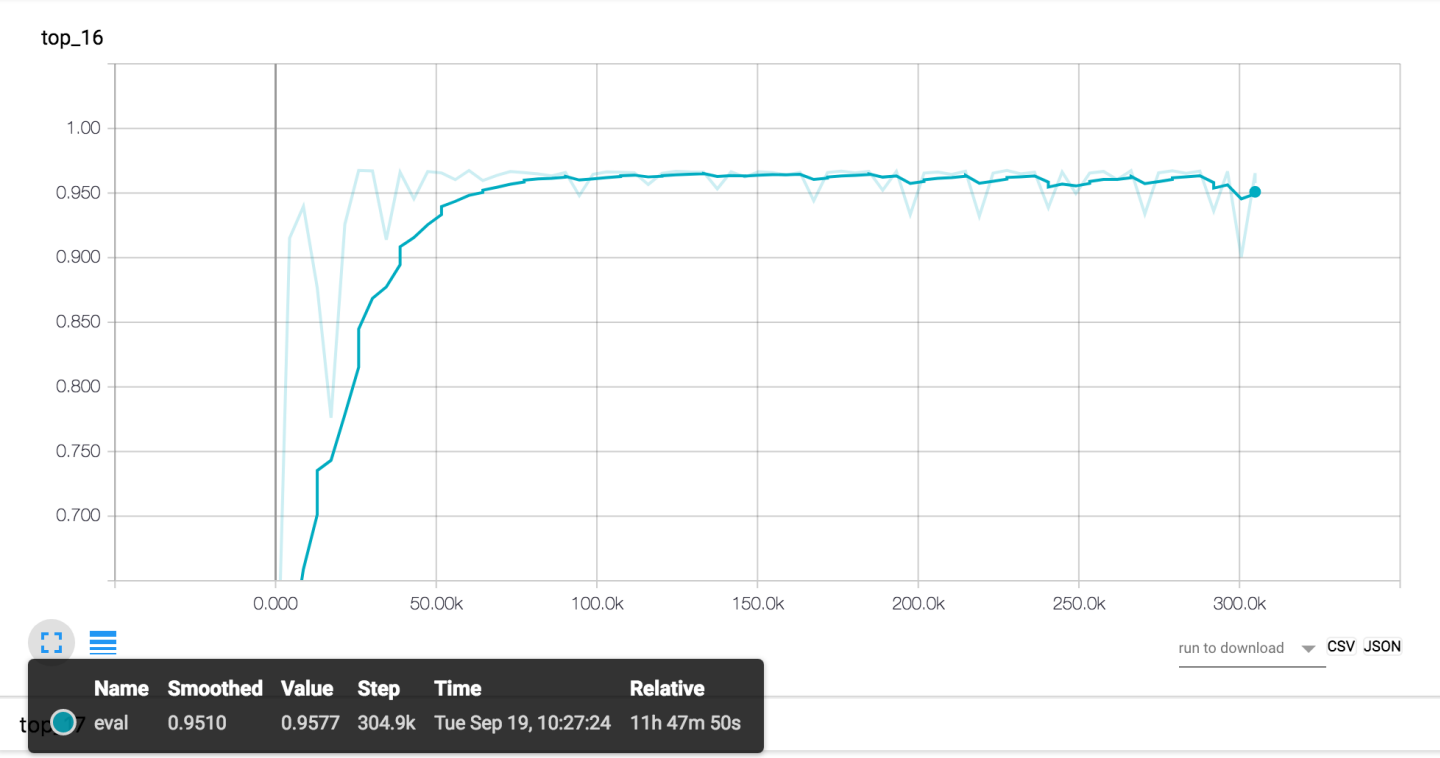


Fig. 1: Figure 2.1 Top 16 Accuracy for block size 16x16

CHAPTER 13

Training for blocks of size 32x32

Dataset obtained after pre-processing is too small for using deep learning to train a model. We use Bilinear Interpolation to resize the block to employ model trained for size 16x16.

CHAPTER 14

Training for blocks of size 64x64

Dataset obtained after pre-processing is too small for using deep learning to train a model. We use Bilinear Interpolation to resize the block to employ model trained for size 16x16.

CHAPTER 15

Model trained from blocks of size 08x08

Note: This model is trained using the data of size 08x08. It is used for the predictions of blocks with size 08x08.

Statistics

1	global step	133000
2	batch size	128
3	samples each class	3075
4	number of classes	32
5	training time	16h 45m 48s
6	epoch	173

Epoch calculating

```
>>> 133000*128/3075/32.0
173.0081300813008
```

15.1 Model Performance on Validating Dataset

Evaluation batch size 100, number of batches 192.

Using validating dataset, the details are documented below:

0.1	Name of dataset	val_08x08.csv
0.2	Size of dataset	4.8 MB
0.3	Samples	600*32
0.4	Usage of dataset	validation
1	top 5 accuracy	0.650
2	top 6 accuracy	0.711
3	top 7 accuracy	0.759
4	top 8 accuracy	0.823
5	top 9 accuracy	0.846
6	top 10 accuracy	0.846
7	top 11 accuracy	0.867
8	top 12 accuracy	0.884
9	top 16 accuracy	0.929
10	top 17 accuracy	0.936
11	top 18 accuracy	0.944
12	top 19 accuracy	0.950
13	top 20 accuracy	0.955
14	top 28 accuracy	0.980

15.2 Model Performance on Testing Dataset

Evaluation batch size 100, number of batches 192.

Using testing dataset, the details are documented below:

0.1	Name of dataset	test_08x08.csv
0.2	Size of dataset	4.7 MB
0.3	Samples	600*32
0.4	Usage of dataset	test
1	top 5 accuracy	0.651
2	top 6 accuracy	0.710
3	top 7 accuracy	0.756
4	top 8 accuracy	0.791
5	top 9 accuracy	0.820
6	top 10 accuracy	0.843
7	top 11 accuracy	0.863
8	top 12 accuracy	0.879
9	top 16 accuracy	0.922
10	top 17 accuracy	0.931
11	top 18 accuracy	0.938
12	top 19 accuracy	0.944
13	top 20 accuracy	0.951
14	top 28 accuracy	0.988

CHAPTER 16

Model trained from blocks of size 16x16

Note: This model is trained using the data of size 16x16. But the evaluation results clearly proved: this model is applicable to size 32x32 and size 64x64 by using Bilinear Interpolation to do resizing for the larger blocks.

Statistics

1	global step	304857
2	batch size	128
3	samples each class	3075
4	number of classes	32
5	training time	11h 47m 50s
6	epoch	396

Epoch calculating

```
>>> 304857*128/3075/32.0
396.53125
```

CHAPTER 17

Using blocks of size 16x16

17.1 Model Performance on Validating Dataset

Evaluation batch size 100, number of batches 192.

Using validating dataset, the details are documented below:

0.1	Name of dataset	val_16x16.csv
0.2	Size of dataset	15.6 MB
0.3	Samples	600*32
0.4	Usage of dataset	validation
1	top 5 accuracy	0.801
2	top 6 accuracy	0.842
3	top 7 accuracy	0.873
4	top 8 accuracy	0.895
5	top 9 accuracy	0.912
6	top 10 accuracy	0.924
7	top 11 accuracy	0.934
8	top 12 accuracy	0.942
9	top 16 accuracy	0.965
10	top 17 accuracy	0.970
11	top 18 accuracy	0.974
12	top 19 accuracy	0.977
13	top 20 accuracy	0.980
14	top 28 accuracy	0.995

17.2 Model Performance on Testing Dataset

Evaluation batch size 100, number of batches 192.

Using testing dataset, the details are documented below:

0.1	Name of dataset	test_16x16.csv
0.2	Size of dataset	15.9 MB
0.3	Samples	600*32
0.4	Usage of dataset	test
1	top 5 accuracy	0.739
2	top 6 accuracy	0.794
3	top 7 accuracy	0.829
4	top 8 accuracy	0.859
5	top 9 accuracy	0.877
6	top 10 accuracy	0.894
7	top 11 accuracy	0.911
8	top 12 accuracy	0.924
9	top 16 accuracy	0.958
10	top 17 accuracy	0.963
11	top 18 accuracy	0.970
12	top 19 accuracy	0.974
13	top 20 accuracy	0.977
14	top 28 accuracy	0.995

CHAPTER 18

Using blocks of size 32x32

We have tried four resizing method:

1. Bilinear interpolation.
2. Nearest neighbor interpolation.
3. Bicubic interpolation.
4. Area interpolation.

Note: All the data for size 32x32 after pre-processing are used for evaluation. We just named it as `val_32x32.csv`, no need for anther `test_32x32.csv`.

Evaluation batch size 100, number of batches 192.

18.1 Performance with Bilinear Interpolation

Using validating dataset, with Bilinear interpolation, the details are documented below:

0.1	Name of dataset	val_32x32.csv
0.2	Size of dataset	104.9 MB
0.3	Samples	872*32
0.4	Usage of dataset	validate&test
1	top 5 accuracy	0.812
2	top 6 accuracy	0.855
3	top 7 accuracy	0.887
4	top 8 accuracy	0.908
5	top 9 accuracy	0.924
6	top 10 accuracy	0.936
7	top 11 accuracy	0.946
8	top 12 accuracy	0.954
9	top 16 accuracy	0.972
10	top 17 accuracy	0.976
11	top 18 accuracy	0.979
12	top 19 accuracy	0.982
13	top 20 accuracy	0.984
14	top 28 accuracy	0.996

18.2 Performance with Nearest Neighbor Interpolation

Almost the same performance as using Linear Interpolation! Omitted here for clarity.

18.3 Performance with Bicubic Interpolation

Almost the same performance as using Linear Interpolation! Omitted here for clarity.

18.4 Performance with Area Interpolation

Almost the same performance as using Linear Interpolation! Omitted here for clarity.

CHAPTER 19

Using blocks of size 64x64

Based on the observations of the testing results of block size 32x32, we believe there should not be such differences among different interpolation method.

Here we only use **Bilinear Interpolation**.

19.1 Performance with Bilinear Interpolation

Using validating dataset, with Bilinear interpolation, the details are documented below:

Total samples: 1728

```
>>> 54*32
1728
```

Evaluation batch size 100, number of batches 17.

0.1	Name of dataset	val_64x64.csv
0.2	Size of dataset	24.5 MB
0.3	Samples	54*32
0.4	Usage of dataset	validate&test
1	top 5 accuracy	0.764
2	top 6 accuracy	0.821
3	top 7 accuracy	0.868
4	top 8 accuracy	0.892
5	top 9 accuracy	0.916
6	top 10 accuracy	0.932
7	top 11 accuracy	0.946
8	top 12 accuracy	0.956
9	top 16 accuracy	0.973
10	top 17 accuracy	0.979
11	top 18 accuracy	0.982
12	top 19 accuracy	0.984
13	top 20 accuracy	0.987
14	top 28 accuracy	0.994

CHAPTER 20

Time Cost of TF in C++

20.1 Motivation

When using Tensorflow(TF) C++ APIs, if you want to run the prediction, you have to call `session->Run()`. (When you call the `session->Run()`, TF will initialize a session and run the prediction in that session for you.)

We want to know the time cost of `session->Run()`, since initializing 1 session for 1 block is easier to implement in HTM (It has already been implemented). But that can be expensive when we talk about time cost. See experiments for details.

20.2 Experiments

The experiments in this section are performed by loading ResNet graph of size [16, 16, 32, 64], units 5.

Here we present the experiments for evaluating the time cost of `session->Run()`.

Now we want to run predictions for blocks of size 8x8 from a frame in one video sequence.

Total 12288 predictions. (`1024/8*768/8=12288`)

20.2.1 Session Run

Now we only use normal CPU computing, which means our binary is not compiled with the AVX and SSE4.2 offered by Intel CPU. (AVX and SSE4.2 are CPU infrastructures for faster matrix computations)

Two choices:

1. Running all samples from one video frame in one session (*Figure 1. Running all samples in one session*)
2. Running one sample in one session (*Figure 2. Running one sample in one session*)

Look at the time cost.

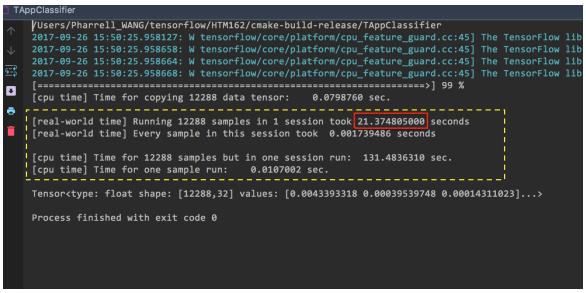


Fig. 1: Figure 1. Running all samples in one session

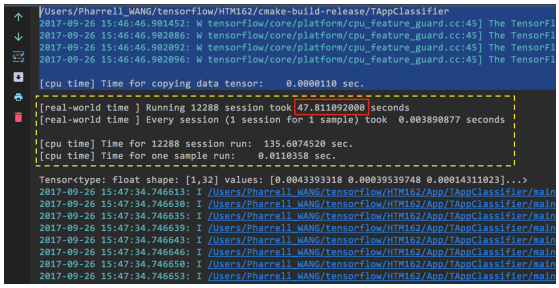


Fig. 2: Figure 2. Running one sample in one session

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	21.37 s	not intuitive
2	Init 12288 sessions for 12288 blocks	47.81 s	intuitive

20.2.2 Session Run with AVX, AVX2, SSE4.2

Now we are only using the benefits offered by CPU.

Two choices:

1. Running all samples from one video frame in one session (*Figure 3. Running all samples in one session*)
2. Running one sample in one session (*Figure 4. Running one sample in one session*)

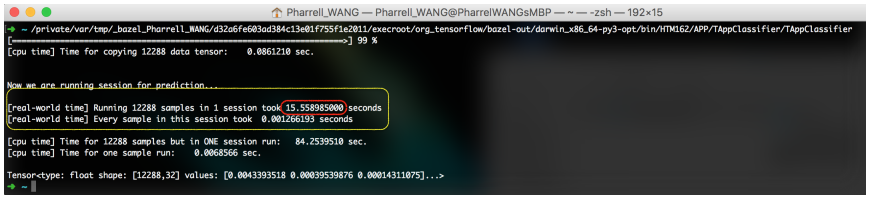


Fig. 3: Figure 3. Running all samples in one session

Look at the time cost.

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	15.56 s	not intuitive
2	Init 12288 sessions for 12288 blocks	33.91 s	intuitive

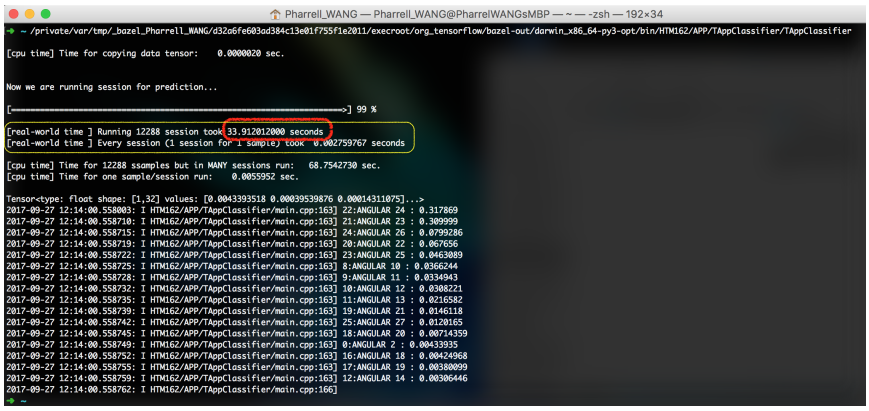


Fig. 4: Figure 4. Running one sample in one session

20.2.3 Session Run with AVX, AVX2, SSE4.2 and GPU Support

Now we are using both the benefits offered by CPU and GPU.

Two choices:

1. Running all samples from one video frame in one session (*Figure 5. Running all samples in one session*)
2. Running one sample in one session (*Figure 6. Running one sample in one session*)

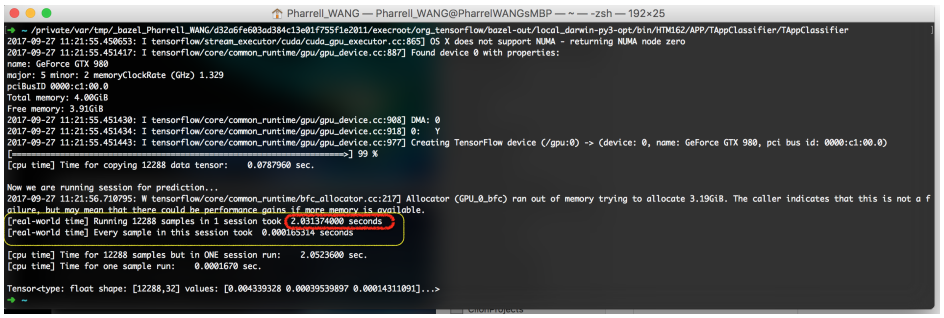


Fig. 5: Figure 5. Running all samples in one session

Look at the time cost.

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	**2.03** s	not intuitive
2	Init 12288 sessions for 12288 blocks	55.26 s	intuitive

20.3 Conclusions

Now I put the three tables above together for comparison:

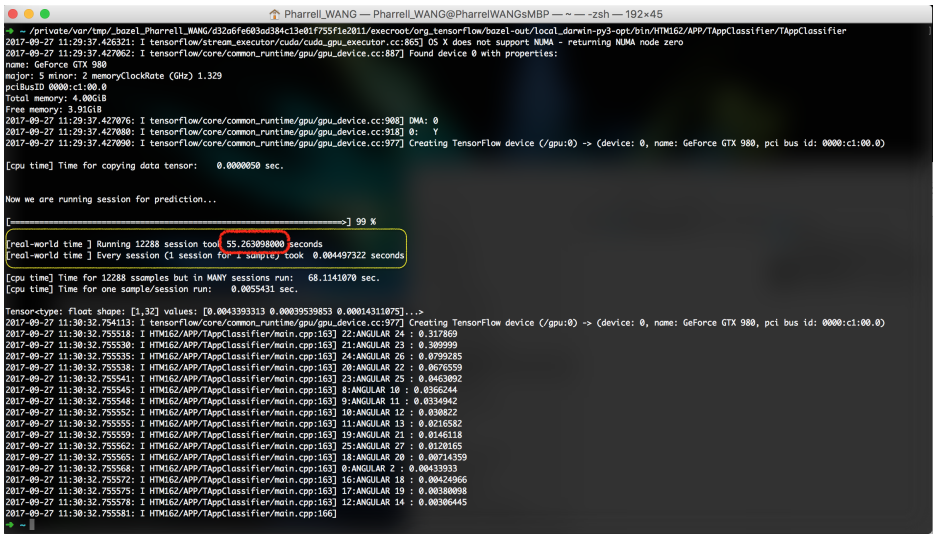


Fig. 6: Figure 6. Running one sample in one session

20.3.1 Plain CPU Config

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	21.37 s	not intuitive
2	Init 12288 sessions for 12288 blocks	47.81 s	intuitive

20.3.2 Employing AVX, SSE4.2

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	15.56 s	not intuitive
2	Init 12288 sessions for 12288 blocks	33.91 s	intuitive

20.3.3 Employing AVX, SSE4.2 and GPU(Parallel computing)

#	Scenario	Time Cost	the difficulty of implementation
1	Init 1 session for 12288 block	**2.03** s	not intuitive
2	Init 12288 sessions for 12288 blocks	55.26 s	intuitive

Apparently, the fastest way is **running a large batch of predictions in a single session using GPU**.

Further more, consider this situation:

- 300 frames to process.
- 12288 8x8 blocks for 1 frame. Time cost 2.03 s for such a frame.
- Then, do a calculation:

```
>>> 300 * 2.03 / 60
10.149999999999999 minutes
```

That is to say, 10 minutes for a video sequence of 300 frames only for processing size 8x8 blocks

We also want to do predictions for size 16x16 and 32x32. Hence the time cost are roughly 30 ~ 60 minutes.

CHAPTER 21

Encoder Integration in C++

- **Tensorflow r1.1** is used in this work. (Tensorflow r1.3 is the newest stable version; Tensorflow r1.1 is the last version that Tensorflow supports Mac GPU.)
- **macOS** is used in this work.

Note: Linux desktop with GPU is highly recommended (Windows OS is not recommended). (Ubuntu is the first choice since it has the largest community support.)

21.1 Pre-requisites

1. Build tensorflow from source
2. Build shared library for using the TensorFlow C++ library

Note: **Archive/Static library** (.a) VS **Shared library** (.so)

Archive libraries (.a) are statically linked i.e when you compile your program with -c option in gcc. So, if there's any change in library, you need to compile and build your code again.

The advantage of .so (*shared object*) over .a library is that *they are linked during the runtime*, i.e. after creation of your .o file -o option in gcc. So, if there's any change in .so file, you don't need to recompile your main program. But make sure that your main program is linked to the new .so file with in command.

21.2 Integrate the model into HTM

Download codebase from GitHub: https://github.com/PharrellWANG/HTM162-Bazel-Cmake

There are two Apps in the above project.

- TAppClassifier
- TAppEncoder

TAppClassifier is the skeleton code which can help you understand how to load graph in C++ and run the prediction using Tensorflow. It is a self-contained c++ Application which can be built from both Bazel and CMake.

ResNet engine has been integrated to **TAppEncoder** for *depth map angular modes [2, 34] prediction* and *the DMM1 searching process*.

For the DMM1 searching process, we are making use of wedgelet slope to reduce the number of wedgelet candidates to be evaluated in DMM1 searching process. If top-16 is used, then almost half of the candidates will be skipped. Hence the time reduction for wedgelet decision shall be reduced roughly by half.

Note: If have time, try to estimate the time cost of ResNet size [4, 4, 8, 16], units 3. Prediction accuracy will be decreased by 2%~3%. But since flops has been reduced from 600k to 130k, the speed of prediction in c++ should be faster.

CHAPTER 22

Simulation Results

We have integrated the learned ResNet model into `HTM16.2` (which is the reference software of 3D-HEVC).

Several simulations are carried out to further evaluate the performance of the proposed algorithm.

BD-BR and **BD-PSNR** metrics *[REF2]* are employed.

22.1 Simulation Environments

22.1.1 Device

- **Macbook Pro (15-inch, Mid 2015)**
- Processor 2.2GHz Intel Core i7
- Memory 16GB 1600MHz DDR3
- Nvidia GTX980, Memory 4GB (External GPU)

22.1.2 Video Sequences

Data are collected from four video sequences.

(This table is copied from *Training Data Source*)

#	Name of the Sequence	Resolution	Usage	Frames
1	Balloons	1024x768	train/test/validation	300
2	Kendo	1024x768	train/test/validation	300
3	PoznanStreet	1920x1088	train/test/validation	250
4	UndoDancer	1920x1088	train/test/validation	250

We want to make sure every sample that will be predicted has never been seen by the learned model. Otherwise it will be cheating.

Anther four sequences which have never been seen by the learned ResNet model are used for simulation:

#	Name of the Sequence	Resolution	Usage	Frames
1	Newspaper	1024x768	Simulation	300
2	GhostTownFly	1920x1088	Simulation	250
3	PoznanHall2	1920x1088	Simulation	200
4	Shark	1920x1088	Simulation	300

22.2 Configuration

The common test condition defined in *[REF1]* are used.

All the sequences are encoded as I-Frame.

22.3 Simulation Results

Time Saving for DMM1 Wedgelet Searching

Sequence	Time Saving (%)				BD-BR	BD-PSNR
	QP 34	QP 39	QP 42	QP 45		
newspaper	63.76	64.94	71.98	74.14	0.98%	-0.02 dB
poznan_hall2	71.80	71.80	66.36	71.27	1.64%	-0.05 dB
ghost_fly	62.00	56.20	51.60	58.87	0.65%	-0.02 dB
shark	63.55	58.66	63.26	63.34	1.04%	-0.03 dB

Fig. 1: Figure 1. Time Saving for DMM1 Wedgelet Searching and Coding Performance of the Proposed Method

Time Saving for Total Encoding

Sequence	Time Saving (%)				BD-BR	BD-PSNR
	QP 34	QP 39	QP 42	QP 45		
newspaper	27.33	27.20	27.78	27.38	0.98%	-0.02 dB
poznan_hall2	29.00	28.34	28.75	19.04	1.64%	-0.05 dB
ghost_fly	32.52	31.19	31.25	32.07	0.65%	-0.02 dB
shark	38.94	31.26	37.33	36.86	1.04%	-0.03 dB

Fig. 2: Figure 1. Time Saving for the total encoding process and Coding Performance of the Proposed Method

CHAPTER 23

About Compilation

23.1 Debug and Release

- Typically *Release* mode is used to run multiple binaries simultaneously for collecting the data. Speed matters.
- *Debug* mode is used for inspecting/debugging the codebase. You can stop at breakpoint and check values of vars.

Few points to notice:

1. In *TypeDef.h*, the `MAC_DEBUG_PATH` need to be toggled for configuring the path when you switch from debug mode and release mode.
2. Remember to toggle the *Release/Debug* Config in **AppCode**.

CHAPTER 24

About GPU

24.1 NVIDIA vs AMD

For machine learning, just two words: choose NVIDIA.

24.2 GPU-Accelerated Computing

NVDIA has a good explanation about What is GPU-Accelerated Computing.

In summary, GPU-accelerated computing offloads compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run much faster.

24.3 Performance

There is a video (length: 1m 33s) GPU VS CPU. It might have been exaggerated for marketing purpose. However the acceleration provided by GPU is something that you cannot deny/ignore if you are doing serious work using deep learning.

24.3.1 Time Saving for Training

The speed of acceleration for training a deep neural network depends on the GPU model.

According to my experience, typically the training process will be accelerated 4~6 times with NVIDIA GTX980 if you are training a 50 layer Convolutional Neural Network.

24.3.2 Time Saving for Prediction

I've documented the prediction acceleration in *Time Cost of TF in C++*. Please check it out.

Here's a quick note (prediction using CPU `Intel Quad-Core i7` vs GPU `NVIDIA GTX980`):

Intel Quad-Core i7 (CPU)	15.56 s
NVIDIA GTX 980 (GPU)	2.03 s

24.4 CUDA and cuDNN

NVIDIA Official Blog has a good explanation about What is CUDA.

Note: CUDA is a parallel computing platform and programming model that makes using a GPU for general purpose computing simple and elegant. The developer still programs in the familiar C, C++, Fortran, or an ever expanding list of supported languages, and incorporates extensions of these languages in the form of a few basic keywords. These keywords let the developer express massive amounts of parallelism and direct the compiler to the portion of the application that maps to the GPU.

CUDA stands for Compute Unified Device Architecture developed by Nvidia . In CUDA basic idea is to use GPU (Graphical Processing Unit) for parallel programming which provides better performance for solving complex problems.

- CUDA : the API/language you talk to Nvidia GPUs.
- cuDNN: library for Deep Learning using CUDA.

You could use CUDA/cuDNN directly yourself, but other libraries like TensorFlow already have built abstractions backed by cuDNN. Tensorflow will handle the device assignments for you as long as you provide a little configurations by writing a few lines of codes.

24.5 Computation Capabilities of CUDA GPUs

Readers may refer to CUDA GPUs to know the computation capability of each type of CUDA GPUs.

In short, The high-end GPU models such as **Tesla P series** will be suitable for **data centre**; **Tesla K series** will be suitable for **work station**. **Geforce series** will have some entertainment elements inside the design. **Tesla series** is recommended while Geforce is not recommended if you are not going to play video games.

CHAPTER 25

Thesis

25.1 Outline

The outline below is subjected to further modifications whenever needed.

- Abstract
- Acknowledgements
- List of Tables (shall not be shown in TOC)
- List of Figures (shall not be shown in TOC)
- Abbreviations (shall not be shown in TOC)
- Introduction
  - Motivation and the Proposed Algorithm
  - Contributions and Organization of the Following Chapters
- Background
  - Video Coding
  - Deep Learning
  - Related Work
- Prepare the Data for Deep Learning
  - Introduction
  - Collecting the Data
  - Pre-processing the Data
  - Visualizing the Data
  - Discussion
  - Conclusion

- Train the Deep Model for Prediction
  - Introduction
  - The Architecture of the Deep Convolutional Neural Network
  - The Hyper-parameters of the Deep Convolutional Neural Network
  - Stopping criteria and Training Results
  - Discussion
  - Conclusion
- Evaluate the Learned Deep Model
  - Introduction
  - Evaluate the Deep Model trained using blocks of size 08x08
  - Evaluate the Deep Model trained using blocks of size 16x16
    - Evaluate the Deep Model on blocks of size 16x16
    - Evaluate the Deep Model on blocks of size 32x32
  - Discussion
  - Conclusion
- Employ the Learned Deep Model
  - Introduction
  - The Analysis and Optimisation for the Time of Prediction
  - The Integration of the Learned Model
  - Simulation Results
  - Discussion
  - Conclusion
- Conclusion
- Bibliography

25.2 Abstract

The Abstract below is subjected to further modifications whenever needed.

Abstract

The 3D Extension of the High Efficiency Video Coding standard (3D-HEVC), which has been finalized by the Joint Collaborative Team on Video Coding (JCT-VC) in February 2015, is the new industry standard for 3D applications. The 3D-HEVC provides plenty of advanced coding tools specifically for addressing the coding of auto-stereoscopic videos which have the format of multiple texture views along with the depth maps which are responsible for synthesising intermediate views with sufficient quality for auto-stereoscopic display. The provided tools take advantage of the statistical redundancies amongst texture views and depth maps in the video sequences, as well as the unique characteristics of depth maps to significantly shrink the bit-rate while preserving the objective visual quality of the 3D videos. However, those tools with high capability in terms of compression come with the high complexity of computation which has made the encoding time of the 3D video sequences much longer than ever by traversing a lot more candidates, calculating time-consuming RD Cost for each of them, especially in the wedgelet searching process for depth maps. While this full-search style method can promise to find the best candidate in depth intra mode decision, the time cost is expensive.

In this dissertation we address the time cost by presenting a new intra mode decision method for depth maps, leveraging the deep convolutional neural networks to predict the wedgelet angles for the depth blocks. The predictions from the learned models are capable of reducing the number of wedgelet candidates by half as well as the angular modes in depth map coding. The size of the neural network has been carefully designed to balance the trade-off between the time cost of model prediction and the model prediction accuracy. Confusion matrix is used to monitor the training process. Top-K criteria is employed for the prediction. We have integrated the learned models into the reference software of 3D-HEVC for the experiments. The compiled executable binaries are able to harness the power of the simultaneous computation of CPU, as well as the parrallel computation of GPU to accelerate the predictions. The simulation results show that the proposed algorithm provides 64.6% time reduction in average while the BD performance has a tiny decrease comparing with the state-of-the-art 3D-HEVC standard.

Fig. 1: Figure 1. Abstract Screen Capture.

CHAPTER 26

How to write thesis

26.1 How to write abstract

Ref: *[REF3]*

26.1.1 What is an Abstract

1. It is a summary of the whole thesis.
2. It presents all the major elements of your work in a highly condensed form.
3. It often functions, together with the thesis title, as a stand-alone text.
4. In addition to prepare the reader for the thesis, it must be capable of substituting for the whole thesis when there is insufficient time and space for the full text.

26.1.2 Size and Structure

1. PolyU requirements for MSc thesis is 200 to 500 words for abstract.
2. Currently, the maximum sizes for abstracts submitted to Canada's National Archive are 150 words (Masters thesis) and 350 words (Doctoral dissertation).
3. To preserve visual coherence, you may wish to limit the abstract for your doctoral dissertation to one double-spaced page, about 280 words.
4. The structure of the abstract should mirror the structure of the whole thesis, and should represent all its major elements.
5. For example, if your thesis has five chapters (introduction, literature review, methodology, results, conclusion), there should be one or more sentences assigned to summarize each chapter.
6. Clearly Specify Your Research Questions

26.1.3 Don't Forget the Results

1. The most common error in abstracts is failure to present results.
2. The primary function of your thesis (and by extension your abstract) is not to tell readers what you did, it is to tell them what you discovered.
3. Approximately the last half of the abstract should be dedicated to summarizing and interpreting your results.

26.2 How to write the Introduction

Ref: *[REF4]*

You need to get your introduction sorted. You need to get your brain in gear.

You need to know:

- What the purpose of an introduction is
- How it should work in the first place

Well,

1. An introduction should introduce.
2. It needs to explain what's coming, and
3. What the reader can expect. Similarly,
4. It needs to explain why the work that's been done has been worth doing,
5. What new contribution to knowledge this thesis is going to make
6. What does the reader get out of reading it.

To let the reader know what to expect,

1. Provide key concepts, defining terms, explaining basic theory

2. Explain your scope limitations, e.g., clear it up that I am dealing with mode decision rather than CU depth decision. 3. Highlight key themes and ideas that unite the chapters as a whole; the introduction should flag up the Important Ideas in a general form so that the reader has a vague idea of the shape that the chapters are going to take.

The final part of the introduction is the road map. Here is a list of the chapters with a paragraph summary of what you will find in each.

Another way to think about what you need to cover in your introduction is to consider your scope, your aims and your methodology. That sounds a bit scary, but can be broken down into simple questions – what are you talking about? What were you trying to find out? How were you trying to find it out? Once I'd written my introduction, I went back and made sure I had answered those questions to the best of my ability, rather than trying to write to answer them in the first place, which seemed the more helpful way of going about it. I should also note that methodology is a word that tends to put my nerves on edges, because I am a text-based analysis person. My methodology – I look at texts, I analyse, what more do you want? Obviously methodology is more important in fields where the ways of doing things are less fixed, even in classics, but it's still important to talk about how you did the research you are going to tell people about, and what your guiding principles are.

To sum up – introductions lay the ground, highlight the important ideas, argue the case for the importance of the work, lay out the stall, sell the product. They also, as subtly as possible, make it clear what a work is not going to offer – but an introduction is not apologetic or flimsy. That said, neither is it overbearing and arrogant, convinced it's introducing the most important piece of writing on the topic ever written. It makes a calm, considered case for the value of what the reader is about to read, and should whet said reader's appetite to find out more about the details of this Important Idea. An introduction should be an invitation, like an appetizer that makes you want to see what else the chef can do.

CHAPTER 27

How to Obtain BD-BR & BD-PSNR

PSNR and bitrate are two important metrics to obtain the BDBR, BDPSNR which are two common criteria for measuring the average PSNR differences between RD-curves.

step 1

use `TAppRenderer` to synthesize intermediate views using

- original YUVs
- (four QPs) the YUVs obtained by your method
- (four QPs) the YUVs obtained by standard method (or any other method that you want to compare with)

separately.

step 2

1. (four QPs) Calculate PSNR using [synthesized_views_from_origin, synthesized_views_from_your_method]
2. (four QPs) Calculate PSNR using [synthesized_views_from_origin, synthesized_views_from_ref_method]

step 3

calculate BD-BR, BD-PSNR for each sequence.

CHAPTER 28

Terms

1. **Bitrate**: Bitrate refers to the number of bits or the amount of data that are processed over a certain amount of time.

CHAPTER 29

References

Bibliography

[REF1] 6. Jager, “Description of Core Experiment 5 (CE5) on Depth Intra Modes,” ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, JCT3V-E1105, 5th Meeting: Vienna, Austria, Aug. 2013

[REF2] 7. Bjontegaard,“Calculation of Average PSNR Differences Between RD Curves,” 2001: ITU-T Video Coding Experts Group (VCEG).

[REF3] 10. (c) Nesbit. (2008.09.11). How to Write an Abstract for Your Thesis or Dissertation. Available: https://www.sfu.ca/~jcnesbit/HowToWriteAbstract.htm

[REF4] (2011). How to write a thesis introduction. Available: https://lizgloyn.wordpress.com/2011/06/01/how-to-write-a-thesis-introduction/