MagmaDNN-CNN Research Project
Final Presentation

Spencer Smith
Dept. of Computer Science
University of North Texas
Denton, USA
spencersmith4@my.unt.edu

Chow Tsz Ching
Dept. of Mathematics
Chinese University of Hong Kong
Hong Kong, China
nicolechow08@gmail.com

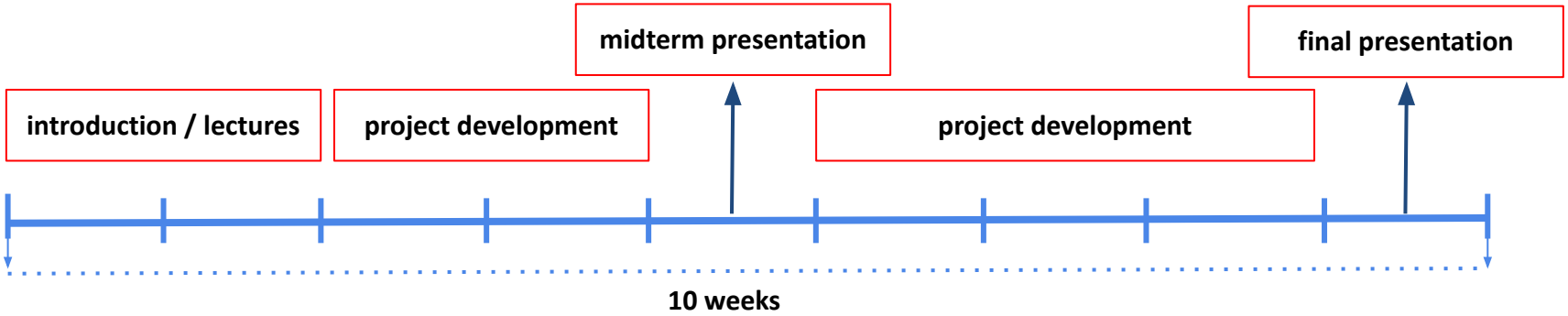
Edward Karak
Dept. of Mathematics.
Baruch College, City University of New York
New York, USA
edward.karak@baruchmail.cuny.edu

MagmaDNN: Table of Contents

MagmaDNN: Project Overview

Initial Research Goals for this Summer:

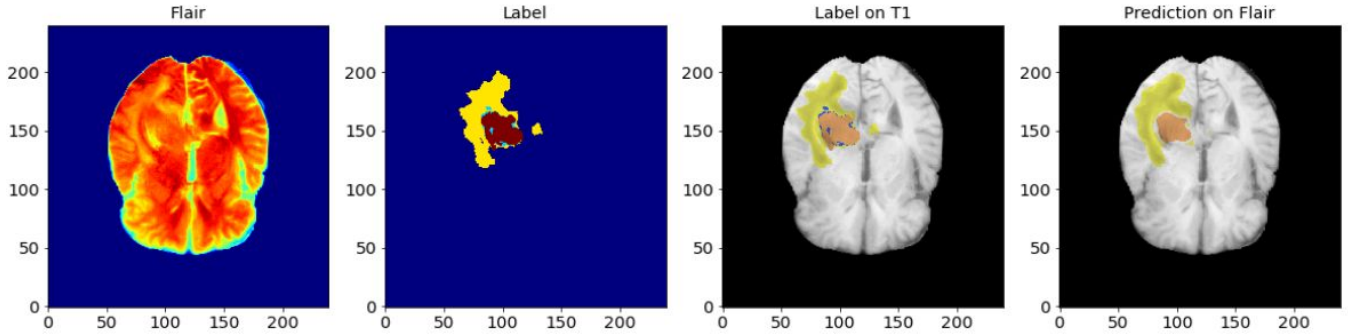
- Implement a U-Net architecture in MagmaDNN
- Extend I/O functions to handle HDF5 data
- Run Data Challenge #3 on MagmaDNN
- Build a UResNet



MagmaDNN: U-Net Background

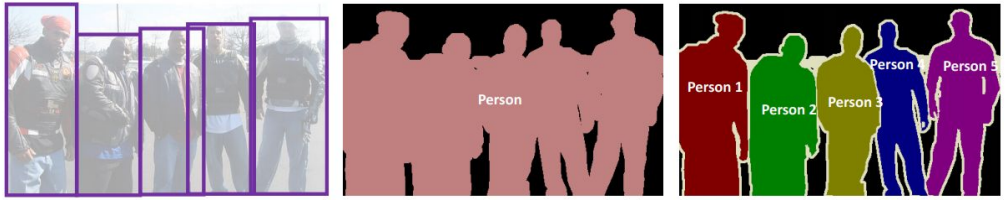
Why use a U-Net?

- Unet has many applications in the medical field as well as other fields.
  - Brain image segmentation
  - Liver image segmentation
  - Protein binding site prediction
- Given a labeled training set, a Unet is able to learn how to classify each object.
- After being trained, the model can take an image as input and then perform segmentation to a high degree of accuracy.



MAGMADNN-CNN Research Project

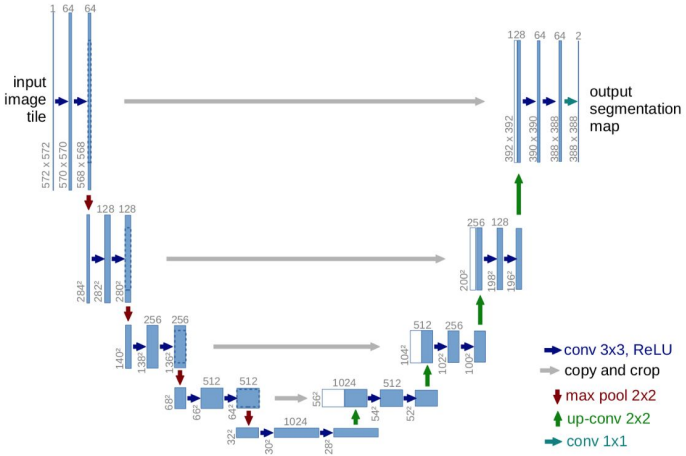
MagmaDNN: U-Net Background



Object Detection | Semantic Segmentation | **Instance Segmentation**

What is a Unet?

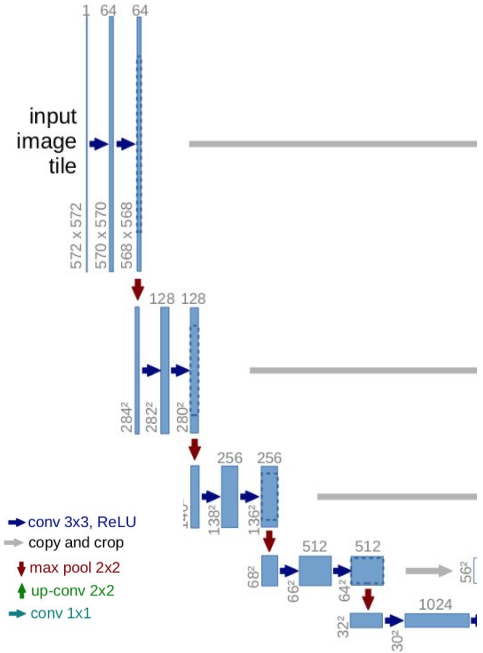
- A Unet is a neural network used to detect objects in an image: this is known as image segmentation.
- Identifies objects by downscaling the input image using convolutions.
- The network picks up on the different objects.
- The image is then upscaled and the pixels are given a classification based on the identified objects.



MagmaDNN: U-Net Background

Downsampling in U-Net

- It is done with a combination of convolutions, batch normalization and relu activation. This combination is called a convolution block.
- An Encoder block uses two convolution blocks and a maxpool to perform the downsampling.
- It is standard to adjust the number of Encoder blocks you need based on the dimensions of the input images.

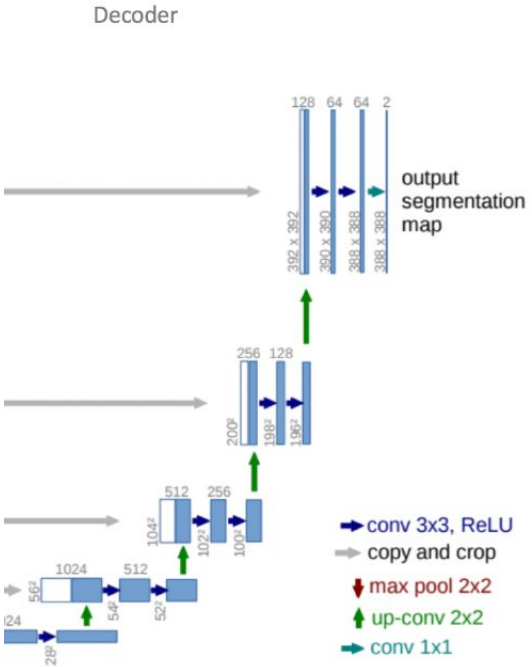


MagmaDNN: U-Net Background

Up-sampling in U-Net

Up-sampling is used in the decoder part of the Unet architecture. The goal is to transform the down-scaled input image to its original dimension.

- **There are many different ways to implement Up-sampling in a Unet.**
- **Nearest-Neighbor, Bi-linear interpolation, and transposed convolution are all methods of up-sampling.**



MagmaDNN: U-Net Background

- **Up-sampling in U-Net**

Bilinear

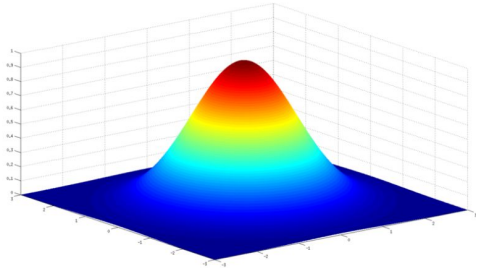
Conv2D Transpose

MagmaDNN: U-Net Background

Loss Functions for U-Net

- **Cross-Entropy Loss is the best loss function for image segmentation applications; however, it can be modified to converge prediction results onto the foreground of the image.**

$$L_{det} = \frac{-1}{N} \sum_{c=1}^{C} \sum_{i=1}^{H} \sum_{j=1}^{W} \begin{cases} (1 - p_{cij})^{\alpha} \log(p_{cij}) & \text{if } y_{cij} = 1 \\ (1 - y_{cij})^{\beta} (p_{cij})^{\alpha} \log(1 - p_{cij}) & \text{otherwise} \end{cases}$$



- **y_{cij} is a distance calculation based on the nearest foreground pixel. It is a 2D Gaussian distribution centered on the nearest foreground pixel.**
- **This allows for the loss function to be lenient on background pixels that are relatively close to the foreground.**

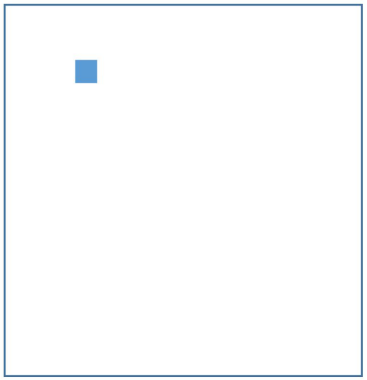


Figure 1: a labeled image for binary classification

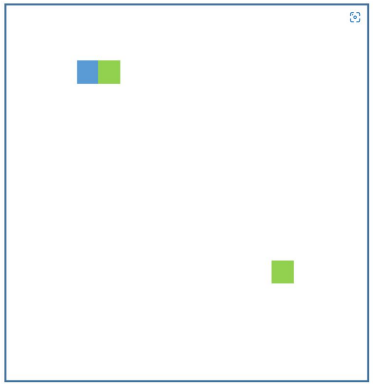
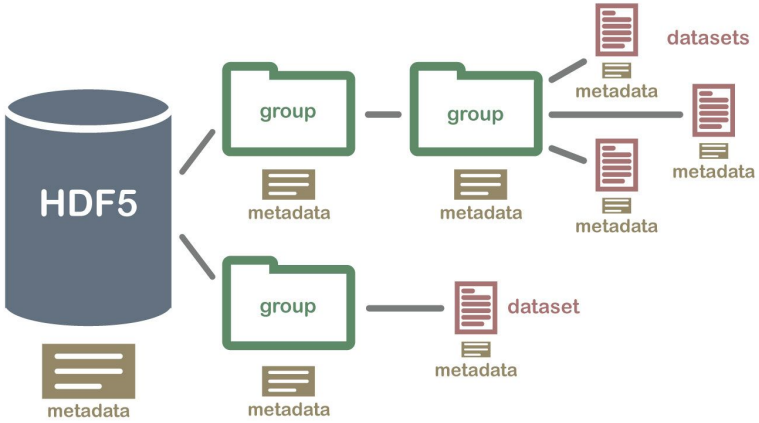


Figure 2: a prediction result for Figure 1

MagmaDNN: Background on HDF5

MagmaDNN I/O Capabilities

- HDF5 (Hierarchical Data Format 5) is a format for storing scalars, matrices and tensors on the disk.
- It is commonly used in computational sciences.
- MagmaDNN uses this format to store trained models.
- It is “hierarchical” because the file stores tensors in a tree structure
  - The tree consists of *groups*, which contains zero or more groups or zero or more *datasets*. A dataset stores a tensor and its metadata.



MagmaDNN-CNN Research Project

MagmaDNN: U-Net Implementation - Downsampling

Function Template

```
template <typename T>
std::vector<layer::Layer<T> *> EncoderMiniBlock(
    op::Operation<T> *input, int n_filters,
    float dropout_prob, bool max_pooling)
```

First Half of the Encoder

```
layer::Layer<T> *next_layer;
layer::Layer<T> *skip_connection;
std::vector<layer::Layer<T> *> layers;
/* apply double convolution with batchnorm + activation */
auto conv1 = layer::conv2d<T>(input, {3, 3}, n_filters, layer::SAME);
layers.push_back(conv1);
auto bn1 = layer::batchnorm(conv1->out());
layers.push_back(bn1);
auto act1 = layer::activation<T>(bn1->out(), layer::RELU);
layers.push_back(act1);
auto conv2 = layer::conv2d<T>(act1->out(), {3, 3}, n_filters, layer::SAME);
layers.push_back(conv2);
auto bn2 = layer::batchnorm(conv2->out());
layers.push_back(bn2);
auto act2 = layer::activation<T>(bn2->out(), layer::RELU);
layers.push_back(act2);
```

Second Half of the Encoder

```
/* apply dropout if it is specified */
if (dropout_prob > (float)0)
{
    skip_connection = layer::dropout(act2->out(), dropout_prob);
    layers.push_back(skip_connection);
}
else
    skip_connection = act2;

/* apply pooling if specified */
if (max_pooling)
{
    next_layer = layer::pooling(skip_connection->out(), {2, 2}, {0, 0}, {2, 2}, MAX_POOL);
    layers.push_back(next_layer);
}

layers.push_back(skip_connection);
return layers;
```

3x3 Convolution + BN + RELU (x2) → Dropout (Optional) → Max Pooling by 2* (no max pooling on last encoder)

MagmaDNN: U-Net Implementation - Downsampling

```
//////////////////////////        ENCODER        ////////////////////////

printf("Calling EncoderMiniBlock #1...\n");
auto encoder1 = EncoderMiniBlock(input->out(), n_filters, dropout_prob, true);
skip1 = encoder1.back(); encoder1.pop_back(); // get the skip connection, and remove it from the vector
printf("Calling EncoderMiniBlock #2...\n");
auto encoder2 = EncoderMiniBlock(encoder1.back()->out(), n_filters * 2, dropout_prob, true);
skip2 = encoder2.back(); encoder2.pop_back(); // get the skip connection, and remove it from the vector
printf("Calling EncoderMiniBlock #3...\n");
// no max pooling in the last encoder
auto encoder3 = EncoderMiniBlock(encoder2.back()->out(), n_filters * 4, dropout_prob, true);
skip3 = encoder3.back(); encoder3.pop_back();
printf("Calling EncoderMiniBlock #4...\n");
auto encoder4 = EncoderMiniBlock(encoder3.back()->out(), n_filters * 8, dropout_prob, true);
skip4 = encoder4.back(); encoder4.pop_back();
printf("Calling EncoderMiniBlock #5...\n");
auto encoder5 = EncoderMiniBlock(encoder4.back()->out(), n_filters * 16, dropout_prob, false);
encoder5.pop_back();
```

- **The number of filters doubles at each EncoderMiniBlock function call.**

```
skip1 = encoder1.back(); encoder1.pop_back();
```

- **The last value in the vector returned from EncoderMiniBlock is the skip connection. Store this value for later use and then remove it.**

MagmaDNN: U-Net Implementation - Upsampling

Function Template

```
template <typename T>
std::vector<layer::Layer<T> *> DecoderMiniBlock(
    op::Operation<T> *prev_layer_input,
    op::Operation<T> *skip_layer_input,
    int n_filters)
```

```
/* apply upsampling + 3x3 convolution */
auto up_scale = layer::conv2dtranspose(prev_layer_input, {3, 3}, n_filters * 2);
auto up_scale2 = layer::conv2d(up_scale->out(), {3, 3}, n_filters, layer::SAME);

/* concat the skip connection with the upsampled tensor */
auto concat = layer::concat(up_scale2->out(), skip_layer_input, 1);
/* apply double convolution with batch norm + RELU */
auto conv1 = layer::conv2d<T>(concat->out(), {3, 3}, n_filters, layer::SAME);
auto bn1 = layer::batchnorm(conv1->out());
auto act1 = layer::activation<T>(bn1->out(), layer::RELU);
// second convolution
auto conv2 = layer::conv2d<T>(act1->out(), {3, 3}, n_filters, layer::SAME);
auto bn2 = layer::batchnorm(conv2->out());
auto act2 = layer::activation<T>(bn2->out(), layer::RELU);
std::vector<layer::Layer<T> *> layers = {up_scale, up_scale2, concat, conv1, bn1, act1, conv2, bn2, act2};

return layers;
```

Upsampling (by 2) + 3x3 Convolution → Concat w/ Skip Connection → 3x3 Convolution + BN + RELU (x2)

MagmaDNN: U-Net Implementation - Upsampling

```
////////////////////////////        DECODER        ////////////////////////////////

printf("Calling DecoderMiniBlock #1...\n");
auto decoder1 = DecoderMiniBlock(encoder5.back()->out(), skip4->out(), n_filters * 8);
printf("Calling DecoderMiniBlock #2...\n");
auto decoder2 = DecoderMiniBlock(decoder1.back()->out(), skip3->out(), n_filters * 4);
printf("Calling DecoderMiniBlock #3...\n");
auto decoder3 = DecoderMiniBlock(decoder2.back()->out(), skip2->out(), n_filters * 2);
printf("Calling DecoderMiniBlock #4...\n");
auto decoder4 = DecoderMiniBlock(decoder3.back()->out(), skip1->out(), n_filters);
```

- **Unlike the Encoder block calls, where the number of filters doubles, the number of filters in the decoder gets cut in half at each subsequence Decoder block call.**
- **The first parameters is just the previous layers output.**
- **The second parameter is the skip connection from the Encoder.**
- **The number of Decoder blocks is dependent on the number of Encoder blocks with max pooling.**

MagmaDNN: U-Net Implementation - Transposed Convolution

The function that gets called in a MagmaDNN program.

```
template <typename T>
Conv2DTransposeLayer<T>* conv2dtranspose(op::Operation<T>* input, const std::vector<unsigned int>& filter_shape, int out_channels,
                        const std::vector<unsigned int>& padding, const std::vector<unsigned int>& output_padding,
                        const std::vector<unsigned int>& strides,
                        const std::vector<unsigned int>& dilation_rates, bool use_cross_correlation, bool use_bias,
                        tensor_filler_t<T> filter_initializer, tensor_filler_t<T> bias_initializer)
{
    return new Conv2DTransposeLayer<T>(input, filter_shape, out_channels, padding, output_padding, strides, dilation_rates,
                            use_cross_correlation, use_bias, filter_initializer, bias_initializer);
}
```

- **This makes it easy to use as long as you have included MagmaDNN into your project.**
- **Alot of the parameters have default values, which are specifically set to double to dimensions.**

```
template <typename T>
Conv2DTransposeLayer<T>* conv2dtranspose(op::Operation<T>* input, const std::vector<unsigned int>& filter_shape = {3, 3},
                        int out_channels = 1, const std::vector<unsigned int>& padding = {1, 1},
                        const std::vector<unsigned int>& output_padding = {1, 1},
                        const std::vector<unsigned int>& strides = {2, 2},
                        const std::vector<unsigned int>& dilation_rates = {1, 1}, bool use_cross_correlation = true,
                        bool use_bias = false, tensor_filler_t<T> filter_initializer = {GLOROT, {0.0, 0.2f}},
                        tensor_filler_t<T> bias_initializer = {GLOROT, {0.0, 0.2f}});
```

MagmaDNN: U-Net Implementation - Transposed Convolution

- The goal of a transposed convolution is to increase the height and width of the input tensor.
- Unlike in the normal convolution in MagmaDNN, where there is a cuDNN API function to calculate the output shape, we must use our own calculation.
- Explicitly calculating the output shape can be dangerous because it can cause problems in later steps.
- So, you must be careful when adjusting the default parameters of the transposed convolution function.

```
// calculate the shape of the output tensor

int in = 0, ih = 0, iw = 0;     // input tensor dimensions
int kh = 0, kw = 0;                     // kernel dimensions
in = this->input_tensor->get_shape(0);
ih = this->input_tensor->get_shape(2);
iw = this->input_tensor->get_shape(3);

kh = this->filter->get_output_shape()[2];
kw = this->filter->get_output_shape()[3];
// account for the case of dilation
int dkh = 1 + (kh - 1) * (dilation_h);
int dkw = 1 + (kw - 1) * (dilation_w);

// set the output shape of the transposed convolution
on = in;                                        // stays the same
oc = this->filter->get_output_shape()[0];   // equal to the number of filters
oh = (ih - 1) * vertical_stride             // output height
        - (2 * pad_h) + dkh + out_pad_h;
ow = (iw - 1) * horizontal_stride           //output width
        - (2 * pad_w) + dkw + out_pad_w;



this->output_shape =
{static_cast<unsigned int>(on), static_cast<unsigned int>(oc),
static_cast<unsigned int>(oh), static_cast<unsigned int>(ow)};
```

MagmaDNN-CNN Research Project

MagmaDNN: U-Net Implementation - Transposed Convolution

Normal Convolution in MagmDNN

```
template <typename T>
void conv2d_device(Tensor<T> *x, Tensor<T> *w, Tensor<T> *out, conv2d_cuc
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    cudnnErrchk(cudnnConvolutionForward(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        x->get_cudnn_tensor_descriptor(), x->get_ptr(),
        settings.filter_desc, w->get_ptr(), settings.conv_desc,
        settings.algo.algo, settings.workspace, settings.workspace_size,
        &beta, out->get_cudnn_tensor_descriptor(), out->get_ptr()));
}
```

- The forward pass of the normal convolution.
- Notice how it just uses the cudnnConvolutionForward to perform the forward pass, which is standard.

Transposed Convolution in MagmDNN

```
template <typename T>
void conv2dtranspose_device(Tensor<T> *x, Tensor<T> *w, Tensor<T> *out, conv2dtr
{
    // set alpha to one and beta to 0
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    // call the cudnn backward convolution
    // W represents the filter tensor
    // X represents the input tensor
    // out represents the output tensor
    cudnnErrchk(cudnnConvolutionBackwardData(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        settings.filter_desc, w->get_ptr(), x->get_cudnn_tensor_descriptor(),
        x->get_ptr(), settings.conv_desc, settings.bwd_data_algo.algo,
        settings.grad_data_workspace, settings.grad_data_workspace_size, &beta,
        out->get_cudnn_tensor_descriptor(), out->get_ptr()));
}
```

- The forward pass of the transposed convolution.
- Notice how we use the cudnnConvolutionBackward for the normal convolution to perform the forward pass for the transpose.

MagmaDNN-CNN Research Project

MagmaDNN: U-Net Implementation - Transposed Convolution

Normal Convolution in MagmDNN

```
cudnnErrchk(cudnnConvolutionForward(
    ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
    x->get_cudnn_tensor_descriptor(), x->get_ptr(),
    settings.filter_desc, w->get_ptr(), settings.conv_desc,
    settings.algo.algo, settings.workspace, settings.workspace_size,
    &beta, out->get_cudnn_tensor_descriptor(), out->get_ptr()));
```

```
template <typename T>
void conv2d_grad_data_device(Tensor<T> *w, Tensor<T> *grad, Tensor<T> *out, conv2
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    cudnnErrchk(cudnnConvolutionBackwardData(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        settings.filter_desc, w->get_ptr(), grad->get_cudnn_tensor_descriptor(),
        grad->get_ptr(), settings.conv_desc, settings.bwd_data_algo.algo,
        settings.grad_data_workspace, settings.grad_data_workspace_size, &beta,
        out->get_cudnn_tensor_descriptor(), out->get_ptr()));
}
```

- The gradient calculation of the input tensor for normal convolution.
- Again, it uses its standard cuDNN function for the computation.

Transposed Convolution in MagmDNN

```
template <typename T>
void conv2dtranspose_grad_data_device(Tensor<T> *w, Tensor<T> *grad, Tensor<T> *out, conv2dtra
{
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    // the gradient tensor from previous convolution is used as input
    // the weight tensor is used in its normal place
    // out is the output tensor
    cudnnErrchk(cudnnConvolutionForward(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        grad->get_cudnn_tensor_descriptor(), grad->get_ptr(), settings.filter_desc,
        w->get_ptr(), settings.conv_desc, settings.algo.algo, settings.workspace,
        settings.workspace_size, &beta, out->get_cudnn_tensor_descriptor(), out->get_ptr()));
}
```

- The gradient calculation of the input tensor for transposed convolution.
- Notice how it uses the cudnnConvolutionForward, which is the forward pass of normal convolution, to compute the backward pass of the transposed.
- The grad is equivalent to the x in the normal convolution forward pass, the w is just the weight tensor.

MagmaDNN-CNN Research Project

MagmaDNN: U-Net Implementation - Transposed Convolution

Normal Convolution in MagmDNN

```
template <typename T>
void conv2d_grad_filter_device(Tensor<T> *x, Tensor<T> *grad, Tensor<T> *out,
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    cudnnErrchk(cudnnConvolutionBackwardFilter(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        x->get_cudnn_tensor_descriptor(), x->get_ptr(),
        grad->get_cudnn_tensor_descriptor(), grad->get_ptr(),
        settings.conv_desc, settings.bwd_filter_algo.algo,
        settings.grad_filter_workspace, settings.grad_filter_workspace_size,
        &beta, settings.filter_desc, out->get_ptr()));
}
```

- The gradient calculation of the filter tensor for normal convolution.
- This is just the standard cudnn function for normal convolution

Transposed Convolution in MagmDNN

```
template <typename T>
void conv2dtranspose_grad_filter_device(Tensor<T> *x, Tensor<T> *grad, Tensor<T> *out, conv2dtr
{
    T alpha = static_cast<T>(1), beta = static_cast<T>(0);
    cudnnErrchk(cudnnConvolutionBackwardFilter(
        ::magmadnn::internal::MAGMADNN_SETTINGS->cudnn_handle, &alpha,
        grad->get_cudnn_tensor_descriptor(), grad->get_ptr(), x->get_cudnn_tensor_descriptor(),
        x->get_ptr(), settings.conv_desc, settings.bwd_filter_algo.algo,
        settings.grad_filter_workspace, settings.grad_filter_workspace_size, &beta,
        settings.filter_desc, out->get_ptr()));
}
```

- The gradient calculation of the filter tensor for transposed convolution.
- Notice how it uses the same function as the normal convolution to calculate the gradient of the filter.
- The only difference is the parameters. The positions of grad and x are flipped in the transposed function call.

MagmaDNN-CNN Research Project

MagmaDNN: U-Net Implementation - Transposed Convolution

Transposed Convolution in MagmDNN

```
/* initialize the conv2dtranspose params to match the Keras params */
auto convTran1 = layer::conv2dtranspose(inputop, {1, 1}, 1, {0, 0}, {1, 1}, {2, 2}, {1, 1}, true,
                                        false, {ONE, {}});
/* print the filter tensor for the layer */
printf("\033[0;31m");
printf("\n\tFilter Tensor\n");
printf("\033[0m");
auto filter_tensor = convTran1->get_filter()->get_output_tensor();
io::print_tensor(*filter_tensor);

/* evaluate the layer and print the result */
printf("\033[0;31m");
printf("\n\tOutput Tensor\n");
printf("\033[0m");
auto output_tensor = convTran1->out()->eval();
io::print_tensor(*output_tensor);
```

```
        Input Tensor
Tensor size of {1, 1, 2, 2}
{
  {
    | 1.00000,   2.00000, |
    | 3.00000,   4.00000, |
  }
}

        Filter Tensor
Tensor size of {1, 1, 1, 1}
{
  {
    | 1.00000, |
  }
}

        Output Tensor
Tensor size of {1, 1, 4, 4}
{
  {
    | 1.00000,   0.00000,   2.00000,   0.00000, |
    | 0.00000,   0.00000,   0.00000,   0.00000, |
    | 3.00000,   0.00000,   4.00000,   0.00000, |
    | 0.00000,   0.00000,   0.00000,   0.00000, |
  }
}
```

Transposed Convolution in Keras

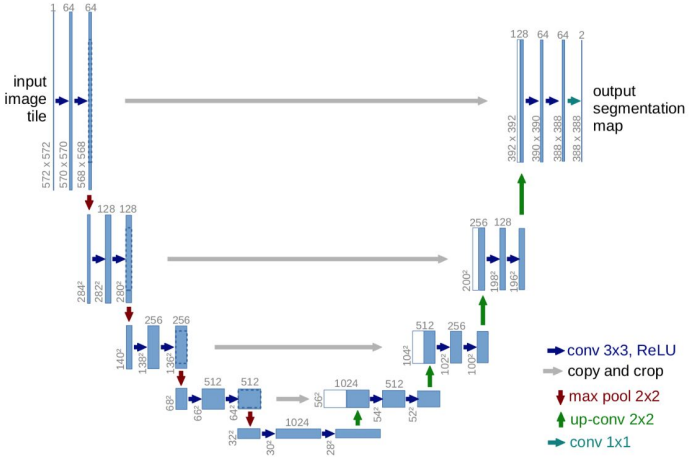
```
[1]: #import from keras
     from keras.models import Sequential
     from keras.layers import Conv2DTranspose
     from keras.layers import Reshape
     from numpy import asarray

[2]: # define the model
     model = Sequential()
     model.add(Conv2DTranspose(1, (1, 1), strides=(2, 2), input_shape=(2, 2, 1)))

[3]: model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      conv2d_transpose (Conv2DTra  (None, 4, 4, 1)           2
      nspose)

     =================================================================
     Total params: 2
     Trainable params: 2
     Non-trainable params: 0
     _________________________________________________________________

[4]: weights = [asarray([[[[1]]]]), asarray([0])]
     model.set_weights(weights)
```

```
[5]: X = asarray([[1, 2],
                  [3, 4]])
     # show input data for context
     print(X)
     # reshape input data into one sample a sample with a channel
     X = X.reshape((1, 2, 2, 1))
     yhat = model.predict(X)

     [[1 2]
      [3 4]]

[6]: yhat = yhat.reshape((4, 4))
     print(yhat)

     [[1. 0. 2. 0.]
      [0. 0. 0. 0.]
      [3. 0. 4. 0.]
      [0. 0. 0. 0.]]
```

MagmaDNN: U-Net Implementation - Concatenation

Skip Connections / Concatenation in U-Net

- The skip connection in a U-Net architecture is just a saved layer from the downsampling part of the network.
- Before max pooling is applied, the layer is stored away for later use in the up-sampling part.
- This allows for the original structure of the input image to be preserved through the countless convolutions and transposed convolutions.
- The skip connection is concatenated with the result of the transposed convolution, along the channel axis.

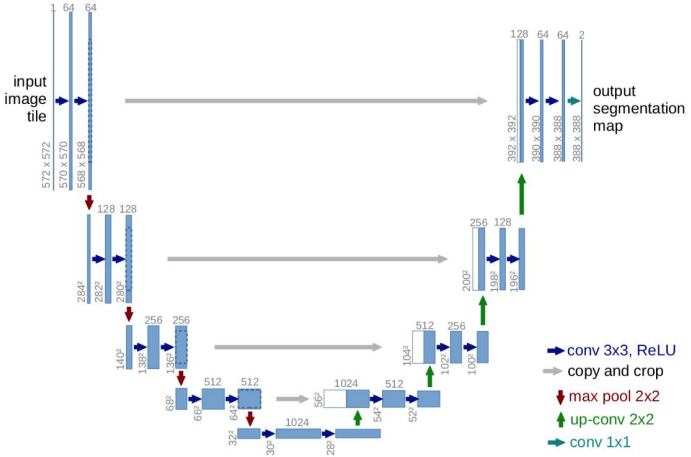


MagmaDNN-CNN Research Project

MagmaDNN: Extend MagmaDNN to run CNN Applications

Why do we need concatenation in the decoder?

- At a certain point, the error increases with network depth—*degradation problem*.
- The main advantage is to prevent vanishing gradient
- It allows the neutral network to have an alternative path to pass on gradient
- It helps U-Net that have a lot blocks of encoder and decoder to learn effectively
- It can help with the U-Net to upscale back to the original size after the feature is captured in the encoder part
- Without a proper concat, the output image will only give out features of the picture instead of the segmented of the image



MagmaDNN: U-Net Implementation - Concatenation

```
__global__ void kernel_concat4D(const T *a, unsigned chw_a, const T *b, unsigned chw_b,
                                T *c, unsigned nchw_c, unsigned chw_c)
{
    assert(a != nullptr); assert(b != nullptr); assert(c != nullptr);
    assert(chw_a > 0); assert(chw_b > 0); assert(nchw_c > 0); assert(chw_c > 0);

    unsigned i = blockDim.x * blockIdx.x + threadIdx.x;
    unsigned stride = blockDim.x * gridDim.x;

    for (; i < nchw_c; i += stride) {
        unsigned batchIdx = i/chw_c;
        int batchOff = i - batchIdx*chw_c;
        c[i] = (batchOff < chw_a) ?
                a[batchOff + batchIdx*chw_a] : b[(batchOff - chw_a) + batchIdx*chw_b];
    }
}
```

Credit: https://forums.developer.nvidia.com/t/concatenate-using-cudnn/63553

MagmaDNN: U-Net Implementation - Loss Function

```
template <typename T>
Tensor<T> *DistAwareCrossEntropyOp<T>::_eval(bool recompute) {
    x_tensor = x->eval(recompute);
    y_tensor = y->eval(recompute);
    T loss_sum = static_cast<T>(0);


    /* TEMPORARILY OUT OF COMMISSION, TAKES TOO LONG ON LARGE IMAGES */
    /* call the distance aware cross entropy */

    math::distawarecrossentropy(x_tensor, y_tensor, this->output_tensor);
```

- Everything for the distance aware cross entropy to work has been implemented; however, the calculation of the foreground pixel took a very looking time.
- Since the ground truth never changes. A fix to this would be to calculate the closest foreground pixel only once and store it somewhere, rather than calculating it every time the loss function gets called. This initial calculation would be slow, but the overall training speed would greatly increase.
- As a temporary fix, since time has ran out, we have just put in a normal cross-entropy calculation in the _eval of the distawarecrossentropy.
- It seems to work just fine, but the distance aware implementation would be ideal for image segmentation, as it would allow for the model to focus training on the foreground.

MagmaDNN: Implementing HDF5

MagmaDNN I/O Implementation

MagmaDNN's HDF5 library consists of standalone functions that wrap the C API such as:

- `hdf_open`: establish connection to HDF file (like `fopen`)
- `hdf_ds_open`: establish connection to dataset in a HDF file
- `hdf_ds_read` & `hdf_ds_write`: read and write to dataset

Atop these functions is a pair of classes: `HDF5` and `HDF5_DataSpace`, which refer to the file and dataspace/dataset, respectively.

MagmaDNN: Training Background

Training parameters in other U-Nets

- From the original U-Net paper, we learnt that the characteristic of a U-Net is it trains with a small dataset but with high resolution.
- For the purpose of biomedical image segmentation to identify problematic cell, it requires 30 512 x 512 images.
- The segmentation takes less than a second on a “recent GPU” in 2015

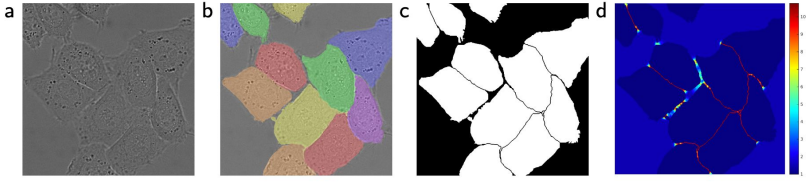
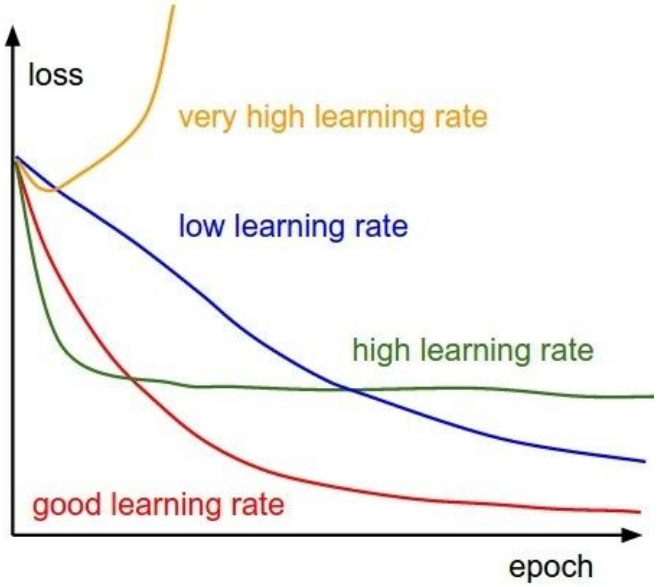


Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**d**) map with a pixel-wise loss weight to force the network to learn the border pixels.

MagmaDNN: Training

Training parameters in OURs U-Net

- We are using 31 256 x 256 images for the training set and 7 images for the testing set.
- We have trained our U-Net on a Nvidia RTX-3060 with 12GB of memory.
- Since our training set is so small, we went with a batch size of 1, which is standard with SGD.
- We have been doing alot of testing with various learning rates. With Adam we have been staying around 3e-4 and with SGD we have been staying around 1e-6.



MagmaDNN-CNN Research Project

MagmaDNN: Results of our U-Net using ADAM

	Sample #1	Sample #2	Sample #3	Sample #4	Sample #5	Sample #6	Sample #7
After 30 epochs	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...
After 60 epochs	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...
After 90 epochs	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...	 UNET_ ADAM30_ predicte...
Ground Truth	 Unet_ groundtrut h1.jpg	 Unet_ groundtrut h2.jpg	 Unet_ groundtrut h3.jpg	 Unet_ groundtrut h4.jpg	 Unet_ groundtrut h5.jpg	 Unet_ groundtrut h6.jpg	 Unet_ groundtrut h7.jpg

```
Accuracy for Sample 1: 0.74
Accuracy for Sample 2: 0.75
Accuracy for Sample 3: 0.85
Accuracy for Sample 4: 0.85
Accuracy for Sample 5: 0.75
Accuracy for Sample 6: 0.94
Accuracy for Sample 7: 0.72
```

```
Accuracy for Sample 1: 0.65
Accuracy for Sample 2: 0.75
Accuracy for Sample 3: 0.93
Accuracy for Sample 4: 0.83
Accuracy for Sample 5: 0.67
Accuracy for Sample 6: 0.94
Accuracy for Sample 7: 0.75
```

```
Accuracy for Sample 1: 0.76
Accuracy for Sample 2: 0.78
Accuracy for Sample 3: 0.84
Accuracy for Sample 4: 0.75
Accuracy for Sample 5: 0.66
Accuracy for Sample 6: 0.96
Accuracy for Sample 7: 0.79
```

- 80/20 training-testing split
- 31 total samples in the training set
- Batch Size: 1
- Learning Rate: 3e-4
- Loss Function = Cross-Entropy

MagmaDNN-CNN Research Project

MagmaDNN: Results of our U-Net using SGD w/ Momentum

	Sample #1	Sample #2	Sample #3	Sample #4	Sample #5	Sample #6	Sample #7
After 30 epochs	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...
After 60 epochs	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...
After 90 epochs	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...	 UNET_ SGD30_ predicte...
Ground Truth	 Unet_ groundtrut h1.jpg	 Unet_ groundtrut h2.jpg	 Unet_ groundtrut h3.jpg	 Unet_ groundtrut h4.jpg	 Unet_ groundtrut h5.jpg	 Unet_ groundtrut h6.jpg	 Unet_ groundtrut h7.jpg

```
Accuracy for Sample 1: 0.74
Accuracy for Sample 2: 0.66
Accuracy for Sample 3: 0.83
Accuracy for Sample 4: 0.77
Accuracy for Sample 5: 0.77
Accuracy for Sample 6: 0.92
Accuracy for Sample 7: 0.63
```

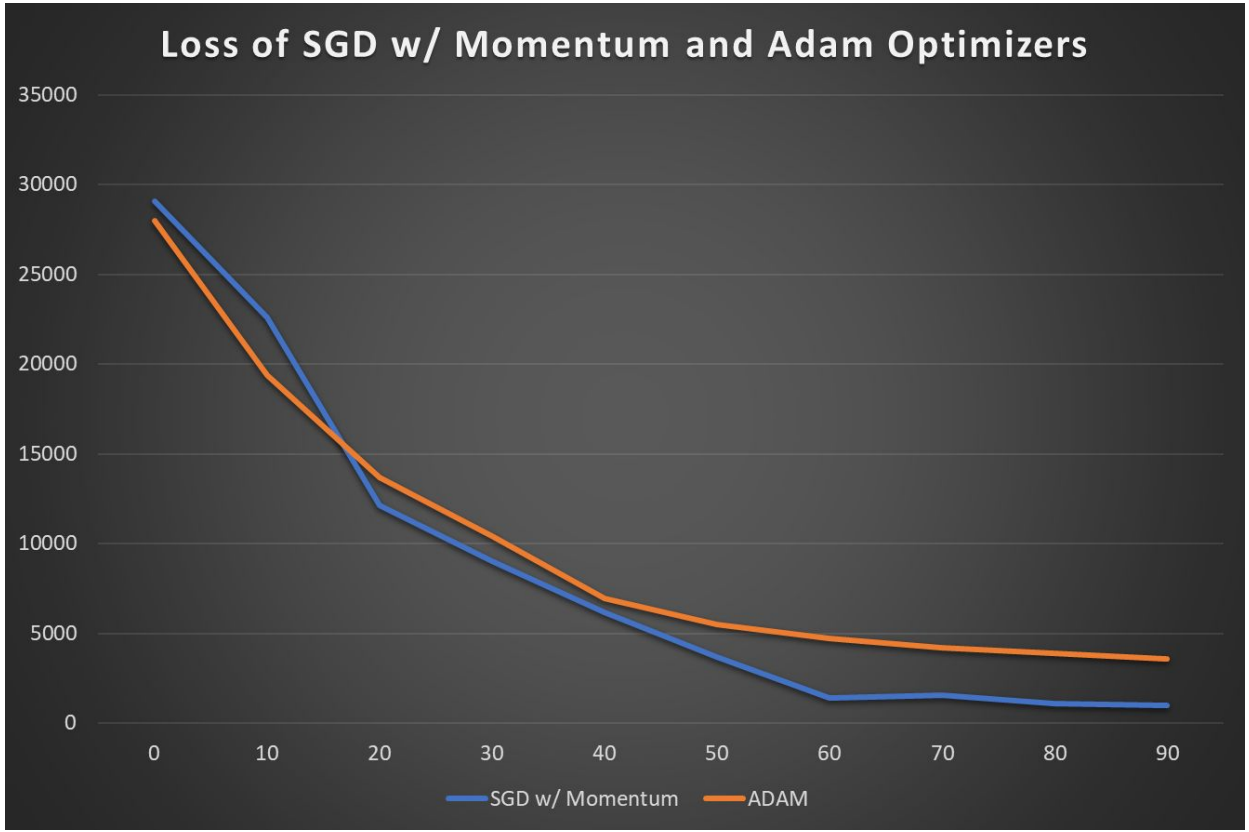
```
Accuracy for Sample 1: 0.68
Accuracy for Sample 2: 0.77
Accuracy for Sample 3: 0.83
Accuracy for Sample 4: 0.79
Accuracy for Sample 5: 0.70
Accuracy for Sample 6: 0.94
Accuracy for Sample 7: 0.72
```

```
Accuracy for Sample 1: 0.66
Accuracy for Sample 2: 0.75
Accuracy for Sample 3: 0.84
Accuracy for Sample 4: 0.78
Accuracy for Sample 5: 0.69
Accuracy for Sample 6: 0.93
Accuracy for Sample 7: 0.71
```

- 80/20 training-testing split
- 31 total samples in the training set
- Batch Size: 1
- Learning Rate: 3e-6
- Momentum = 0.9
- Loss Function = Cross-Entropy

MagmaDNN-CNN Research Project

MagmaDNN: Comparing Loss of SGD w/ Momentum and ADAM



SGD w/ Momentum

- **Batch Size: 1**
- **Learning Rate: 3e-6**
- **Momentum = 0.9**
- **Loss Function = Cross-Entropy**

Adam

- **Batch Size: 1**
- **Learning Rate: 3e-4**
- **Beta1 = 0.9**
- **Beta2 = 0.999**
- **Loss Function = Cross-Entropy**

MagmaDNN-CNN Research Project

THE UNIVERSITY of TENNESSEE KNOXVILLE

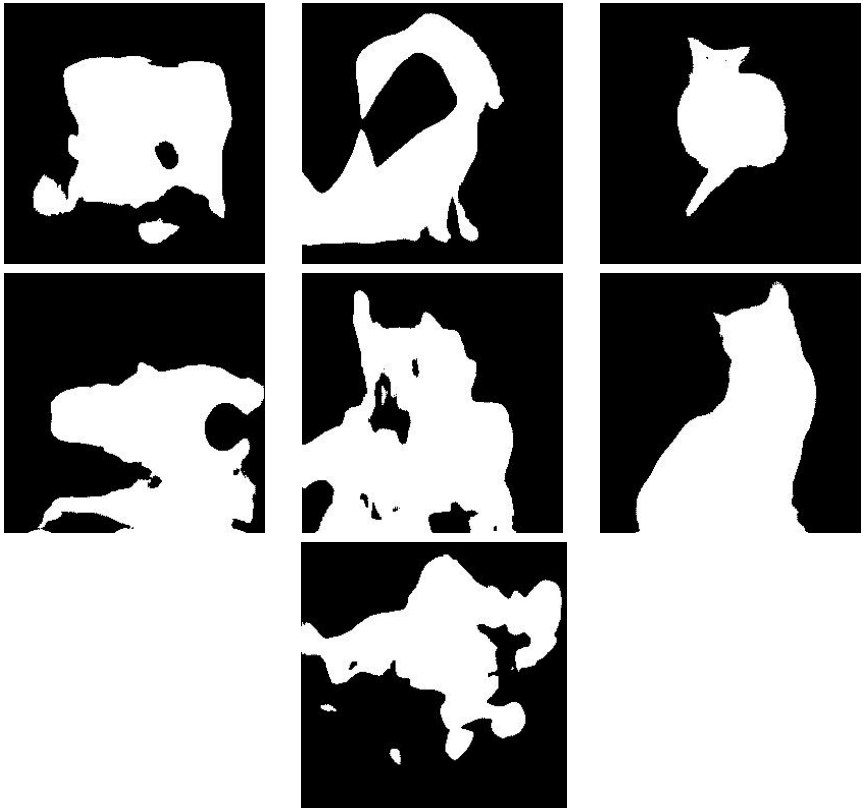
MagmaDNN: Results of our U-Net using SGD / ADAM vs. PyTorch

	IoU	Dice	Pixel
Pytorch model w/ Car dataset	0.705533955	0.536302446	0.405787962
U-Net-like model w/ full Oxford Pet Set (7349)	0.924337826	0.2089358	0.480340
Our Model w/ SGD	0.697989235	0.732576941	0.408179616
Our Model w/ ADAM	0.7270899	0.74655103	0.4190377

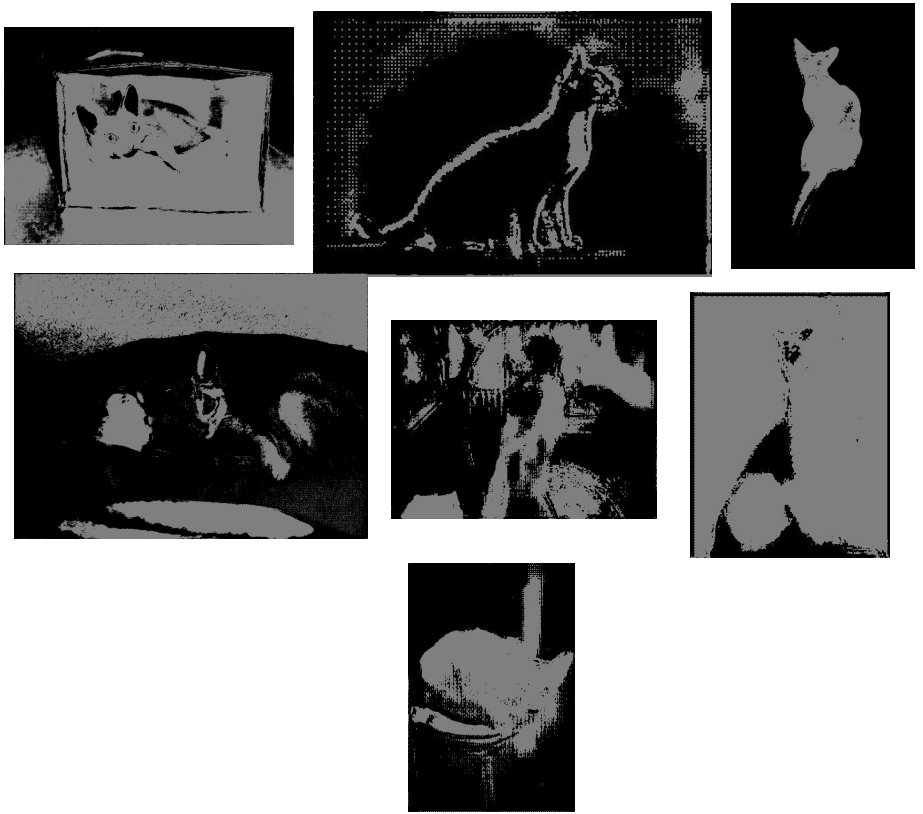
MagmaDNN-CNN Research Project

MagmaDNN: Results of our U-Net using ADAM vs. PyTorch

Predicted Images from our U-Net



Predicted Images from PyTorch



MagmaDNN-CNN Research Project

MagmaDNN: Results of our U-Net using SGD / ADAM vs. PyTorch

Accuracy from our U-Net Model

Loss Function: Cross-Entropy
Optimizer: Adam
Training Set Size: 31
Image Dimensions: 256x256
Testing Set Size: 7

	IoU	Dice	Pixel
30 epochs	0.4309651	0.50146092	0.2908932
60 epochs	0.7270899	0.74655103	0.4190377
90 epochs	0.6567489	0.70364418	0.3932231

Accuracy from the PyTorch Model

Loss Function: Cross-Entropy
Optimizer: Adam
Training Set Size: 31
Image Dimensions: 256x256
Testing Set Size: 7

	IoU	Dice	Pixel
30 epochs	0.2703989	0.23404101	0.189038
60 epochs	0.7130330	0.26042986	0.4059788
90 epochs	0.7820648	0.25603748	0.4331681

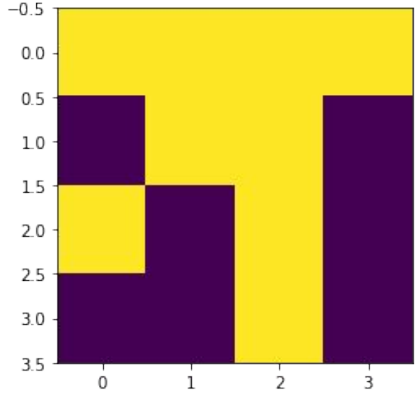
MagmaDNN: Results of our U-Net using SGD

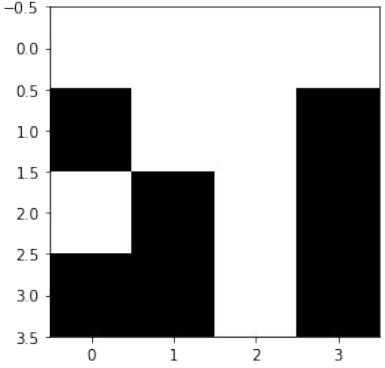
- 80/20 training-testing split
- 31 total samples in the training set
- Batch Size: 1
- Learning Rate: 3e-6
- Momentum = 0.9
- Loss Function = Cross-Entropy

	IoU	Dice	Pixel
30 epochs	0.394911641	0.45857022	0.272548213
60 epochs	0.697989235	0.732576941	0.408179616
90 epochs	0.677266841	0.717351087	0.400935175

MagmaDNN-CNN Research Project

MagmaDNN: Testing with a simple example






```
Name                          Output Shape              # Params
=================================================================
InputLayer                    (1, 1, 4, 4)                     0
Conv2d                        (1, 1, 4, 4)                     9
RELU                          (1, 1, 4, 4)                     0
Conv2d                        (1, 1, 4, 4)                     9
RELU                          (1, 1, 4, 4)                     0
Pooling                       (1, 1, 2, 2)                     0
Conv2d                        (1, 2, 2, 2)                    18
RELU                          (1, 2, 2, 2)                     0
Conv2dTranspose               (1, 2, 4, 4)                    36
Conv2d                        (1, 1, 4, 4)                    18
Concat                        (1, 2, 4, 4)                     0
Conv2d                        (1, 1, 4, 4)                    18
RELU                          (1, 1, 4, 4)                     0
Conv2d                        (1, 1, 4, 4)                    18
RELU                          (1, 1, 4, 4)                     0
Conv2d                        (1, 1, 4, 4)                     9
RELU                          (1, 1, 4, 4)                     0
SIGMOID                       (1, 1, 4, 4)                     0
OutputLayer                   (1, 1, 4, 4)                     0
=================================================================
Total number of params : 135
n_samples: 2048
```

```
Epoch (40/50): accuracy=59.31 loss=2.416 time=101
Epoch (41/50): accuracy=81.06 loss=2.416 time=104
Epoch (42/50): accuracy=98.75 loss=2.416 time=106
Epoch (43/50): accuracy=120.4 loss=2.416 time=109
Epoch (44/50): accuracy=154.2 loss=2.416 time=111
Epoch (45/50): accuracy=166.5 loss=2.416 time=114
Epoch (46/50): accuracy=198.6 loss=2.416 time=116
Epoch (47/50): accuracy=207 loss=2.416 time=119
Epoch (48/50): accuracy=236.5 loss=2.415 time=121
Epoch (49/50): accuracy=245.1 loss=2.415 time=124
Epoch (50/50): accuracy=252.1 loss=2.415 time=126
Final Training Metrics: accuracy=0 loss=0 time=126
Tensor size of {1, 1, 4, 4}
{
  {
    | 0.00000,   1.00000,   1.00000,   1.00000, |
    | 1.00000,   1.00000,   1.00000,   1.00000, |
    | 0.00000,   1.00000,   1.00000,   1.00000, |
    | 1.00000,   0.00000,   1.00000,   1.00000, |
  }
}
Tensor size of {1, 1, 4, 4}
{
  {
    | 0.00000,   1.00000,   1.00000,   1.00000, |
    | 1.00000,   1.00000,   1.00000,   1.00000, |
    | 0.00000,   1.00000,   1.00000,   1.00000, |
    | 1.00000,   0.00000,   1.00000,   1.00000, |
  }
}
```

MagmaDNN: Future Directions

- Implement URes-Net using the successfully implemented U-Net and Resnet models.
- Implement bilinear interpolation and compare the result with Convolution Transpose.
- Adjust the distance aware cross entropy algorithm to calculate the closest foreground pixel only once.
- Write the distance aware cross entropy in GPU code so it is more efficient.
- Write the _grad of the concat in GPU code so it is more efficient.

MagmaDNN: References

[1] Ronneberger, Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Medical Image Computing and Computer-Assisted Intervention MICCAI 2015 (pp. 234241). Springer International Publishing.

[2] Dumoulin, & Visin, F. (2016). A guide to convolution arithmetic for deep learning.

[3] Nichols, Wong, K., Tomov, S., Ng, L., Chen, S., & Gessinger, A. (2019). MagmaDNN: Accelerated Deep Learning Using MAGMA. Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), 16. https://doi.org/10.1145/3332186. 3333047

[4] https://developer.download.nvidia.com/compute/DevZone/docs/html/C/ doc/CUDA C Programming Guide.pdf

[5] Sergey Ioffe, & Christian Szegedy. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv.org.