ECE 1718 Assignment #3

Group #1
April 13, 2023

Members:
Teng Shu shuteng 1003533651
Yih Cheng cheng442 1009206878
Yang Chen chenya26 999551783
Chen Liang liangc78 1006621443

Milestone #1

1. Explore the “Canny Edge Detector” and “Skin Tone Detector” samples by using different input images.

Canny Edge Detector detects the edge of the image or live stream. Skin Tone Detector detects the skin tone of the person in the image or live stream.

2. Go to the source code “canny.cpp” and “skinToneDetector.cpp” and identify what functions are used to create the image object.

In these source codes, there are 2 functions that are used to create the image object: `vxCreateImage` and `vxCreateVirtualImage`. ‘vxCreateImage’ is used to create input and output image objects in the OpenVX context. ‘vxCreateVirtualImage’ is used to create virtual image objects within a graph that do not need to be accessed by the user. This API can create a data reference to link two or more nodes together when the intermediate data are not required to be accessed by outside entities.

The function documentation in OpenVX shows the APIs as below:

```
vx_image VX_API_CALL vxCreateImage (   vx_context       context,
                                       vx_uint32        width,
                                       vx_uint32        height,
                                       vx_df_image      color
                                   )

vx_image VX_API_CALL vxCreateVirtualImage ( vx_graph         graph,
                                            vx_uint32        width,
                                            vx_uint32        height,
                                            vx_df_image      color
                                          )
```

3. Identify how the memory objects are created.

The input image data was passed in by the Mat type from OpenCV. The function ‘imread’ will read the input image data and creates a Mat object which allocates memory for this image. When the image object has been created by ‘vxCreateImage’ and ‘vxCreateVirtualImage’, the memory is also allocated but not yet used before ‘vxMapImagePatch’ has been called. After using the memory, ‘vxUnmapImagePatch’ will be called to unmap the memory.

4. Identify how the nodes are created.

Nodes are created by functions ‘vxColorConvertNode’, ‘vxChannelExtractNode’, ‘vxCannyEdgeDetectorNode’, ‘vxSubtractNode’, ‘vxThresholdNode’, ‘vxAndNode‘. They are used to create nodes for different use and also need different parameters.
Nodes were released after making sure there is no error in each node (‘canny.cpp’ did this).

5. Identify how the nodes are connected to become an OpenVX graph.

An OpenVX graph object was passed in as a parameter during node creation, and this connects the node and the OpenVX graph.

6. (Optional) If you have a camera, play around with the “Bubble Pop” sample. However, you will need to use a webcam that is external to your computer. Using a built-in camera may not work.

We have tested and successfully ran this application on our local machine.

7. Conclude your findings and what you have learned so far about OpenVX.

Functions ‘vxCreateImage’ and ‘vxCreateVirtualImage’ can create OpenVX image objects.
There are different nodes available that can be connected to the OpenVX graph to be used. Different nodes will do different computations and also need different inputs. Nodes can be released after checking all nodes for any errors.
Function ‘vxCopyImagePatch’ copies image data from memory to the OpenVX image. This can be used to pass in the input image data before computation.
The function ‘vxProcessGraph’ will start the computation based on the nodes defined previously.
The function ‘vxMapImagePatch’ maps the image data to the memory allocated by the creation of the OpenVX image object. A pointer points to the beginning of the image data and will be returned by a parameter.
After using the mapped image memory, the function ‘vxUnmapImagePatch’ can be used to unmap the image data.

Milestone #2

1. Convert the given “SRCNN” Python source code to a more efficient CPU-based baseline implementation and make sure you can run it and verify that its run-time is shorter than the Python implementation (assuming you chose to convert the given Python code).

Python code uses PyTorch to build the network. PyTorch is built upon lots of C++ codes and functions which makes it as fast as regular C++ programs [1]. PyTorch also takes advantage of

multithreading as it has a function ‘set_num_threads()’ which is been used to set the number of threads that can be used for computation, and the default number of threads is set to the number of CPU cores [2]. Based on all this information, it was realized that building a C++ program that does convolution manually and wishing it runs faster than PyTorch is fairly unrealistic.

It was decided to make two versions of C++ SRCNN programs. One uses LibTorch (the C++ version of PyTorch) to take advantage of the execution speed from PyTorch and wishes it can be a little faster than the Python version. Another one does convolution manually so to prove that PyTorch is highly efficient even compares to C++ programs.

Note that the CNN network is having input images and output images of the same size, which means the network would optimize the image without increasing its resolution. So the complete SRCNN process would use bicubic interpolation first to increase the resolution, then use the trained network to enhance the image.

Note the two C++ version was implemented in one single C++ file, and the execution time was recorded by ‘std::chrono’ inside the code, and SSIM calculation was disabled since the PyTorch version doesn’t have SSIM calculated. The time recorded only counts the inference time, so it would be slightly shorter than the time recorded by perf.
The execution time, PSNR and SSIM for one epoch of testing (19 images) is shown in the table below:

	PyTorch Version	LibTorch Version	C++ Manual Convolution Version
Execution Time [sec]	1.399	1.277	152

The table above shows that the LibTorch C++ version is slightly faster than the PyTorch version. It also shows that the C++ manual convolution version is extremely slow compared to the PyTorch version or LibTorch version.

2. For the CPU-based baseline implementation, record the execution time, data footprint, CPU utilization, and other hardware counter statistics using “perf”.

Perf output for LibTorch implementation:

```
     15,188.67 msec task-clock:u              #    6.897 CPUs utilized
    388,930,123      cache-references:u        #   25.607 M/sec
    229,077,040      cache-misses:u            #   58.899 % of all cache refs
 44,371,509,848      cycles:u                  #    2.921 GHz
 24,142,285,604      instructions:u            #    0.54  insn per cycle
  1,809,458,686      branches:u                #  119.132 M/sec
```

```
      2.202352444 seconds time elapsed

     12.562738000 seconds user
      2.633992000 seconds sys
```

Perf output for manual convolution implementation:

```
      2,288.58 msec task-clock:u              #    0.976 CPUs utilized
     19,856,308      cache-references:u        #    8.626 M/sec
      3,461,280      cache-misses:u            #   17.432 % of all cache refs
 10,251,360,300      cycles:u                  #    4.453 GHz
 27,116,386,700      instructions:u            #    2.65  insn per cycle
  3,612,889,575      branches:u                # 1569.490 M/sec

      2.339856941 seconds time elapsed

      2.214838000 seconds user
      0.079958000 seconds sys
```

Discussion

The data above shows that the LibTorch version is having CPU utilization of 6.897 which is a way better result than the manual convolution version. This CPUs utilization also shows that the LibTorch takes advantage of parallelism.
The number of cache references in LibTorch implementation is way smaller than the number of cache references in manual convolution implementation, this may be due to the convolution in LibTorch being optimized for memory reading, which requires fewer cache references and taking less time.
The number of branches in LibTorch implementation is also way smaller than the number of branches in manual convolution implementation. Fewer branches mean it utilizes the instruction pipeline more, which can lead to a smaller ISP and faster execution.

3. Record the PSNR and SSIM metrics versus the original high-res image(s).

	PyTorch Version	LibTorch Version	C++ Manual Convolution Version
Average PSNR	29.613	29.9176	29.9176
Average SSIM	Not calculated	0.99997	0.99997

Regarding the average PSNR, the LibTorch version is slightly higher than the PyTorch version although they are using the same model for inference. This may be due to the image being read by different libraries in these two versions. The PyTorch version uses PIL to read and process

images, however, PIL is not available in C++, so LibTorch uses OpenCV for image processing. These two libraries may have some slight differences in rounding values and in bicubic interpolation which results in a small difference at PSNR.

In the C++ manual convolution version, the output result for PSNR and SSIM is exactly the same as the LibTorch version. This means that there is no mistake made on the manual convolution, but it is still way slower than the PyTorch/LibTorch implementation.

4. Research different DL-based approaches that can solve SR problems and describe their advantages and disadvantages.

According to our search, we are going to focus on the 4 different DL-based approaches that can solve SR problems: SCUNet, SwinIR, BSRGAN, and ESRGAN.

Swin-Conv-UNet (SCUNet)[3] is a neural network architecture which combines the Swin transformer block and U-Net to help denoise the input noisy images.
Advantage

1. SCUNet is specifically designed for denoising tasks, whereas SRCNN is primarily focused on super-resolution tasks. This gives SCUNet an edge in effectively removing noise and recovering details in blind image-denoising applications. SCUNet demonstrates superior performance in noise removal and detail preservation in real images, and real applications which suggest better adaptability and applicability to real-world scenarios compared to SRCNN.
2. SCUNet is more efficient than traditional transformer networks because it is easy to process the design in parallel.

Disadvantage

1. It is difficult to implement SCUNet since it has a more complex architecture than traditional models.
2. SCUNet requires large datasets and significant computational resources to train the model for optimal performance.

SwinIR[4] applies the Swin transformer in a cascading manner to perform multi-scale image restoration. The input image is downsampled and then restored at a low resolution. The restored image will be upsampled and used as input for the next stage of restoration at a higher resolution.
Advantage

1. Compared with state-of-art image restoration methods based on convolutional neural networks, SwinIR has achieved a slightly better performance in various image restoration tasks, such as image super-resolution, image denoising and JPEG compression artifact reduction, while the total number of parameters is reduced by up to 67%
2. Because of the usage of Swin transformer blocks, SwinIR is designed to be processed in parallel which improves efficiency.

Disadvantage

1. SwinIR is difficult to understand and implement because of the complex network design.

BSRGAN[5] designs a new degradation model to synthesize low-resolution images from given high-resolution images, which makes blur, downsampling and noise more practical. It also applies randomly shuttled degradations to synthesize low-resolution images. Next, a deep blind super-resolution model is trained based on this new degradation model with given high-resolution images and synthesized low-resolution images.

Advantage

1. The deep blind model trained from the synthesized data from the degradation model performs better on images corrupted by diverse degradation compared with other SR image generation.
2. Because of the newly designed degradation model, BSRGAN can improve its performance over time.

Disadvantage

1. It consumes great computational resources to train the deep blind super-resolution model because it needs the degradation model to generate the datasets.
2. As BSRGAN is trained on specific datasets from the degradation model, it may have performance issues outside of the datasets.

ESRGAN[6] is an improved model based on SRGAN from adopting a deeper model using Residual-in-Residual Dense Block without batch normalization layers. It also applies relativistic average GAN instead of vanilla GAN and improves the perceptual loss by using the features before activation.

Advantage

1. ESRGAN has much better performance than other SR methods with easy training.
2. It has high accuracy on brightness and textures because of using relativistic GAN as the discriminator to guide the generator.

Disadvantage

1. It consumes great computing power to train model
2. It requires large datasets to train the model.

5. Compare traditional, and DL-based SR approaches quality and speed by using FFmpeg implementation of traditional approaches.

Traditional SR methods are generally based on interpolation techniques such as bicubic, bilinear, and nearest-neighbour; these techniques are fast and easy to implement but often produce lower-quality results. Compared to traditional methods, DL-based methods, such as SRCNN, have shown superior performance in image and video upscaling.

Quality

DL-based SR methods can generate better-quality images and videos compared to traditional methods. Traditional methods such as bicubic interpolation or Lanczos resampling may introduce artifacts and blurriness. These methods work well for small-scale upscaling but

struggle to generate high-frequency details when upscaling to a much larger resolution. DL-based methods produce fewer artifacts and sharper details.

Speed
DL-based SR methods are generally slower than traditional methods due to the complexity of neural networks and the need for high computational resources. Real-time applications can be challenging unless optimized models and hardware acceleration are used. Traditional methods are much more computationally efficient, and they can be easily implemented using FFmpeg, which is a well-optimized, cross-platform, and widely-used multimedia framework. It is important to note that FFmpeg does not natively support DL-based SR methods and would require PyTorch to implement.

6. Do some research on what OpenVX API can be used to replace existing parts of your rewritten code.

There are two possible ways of using OpenVX API to do CNN inference.

One way is to use the custom convolution. OpenVX has a ‘vx_convolution’ object that is designed for linear filters. It can be created using a kernel that’s greater than 3x3, and then do convolution on ‘vx_image’ using ‘vxConvolveNode()’. However, this approach has lots of limitations that make it difficult to be used here.
The kernel size is limited to be greater than 3x3, however, the second convolution layer on the SRCNN network only has a kernel size of 1x1.
The SRCNN network has padding equal to 2 for all three convolution layers, however, custom convolution in OpenVX does not seem to naturally support padding, so a conversion like coping the original image to a bigger-sized image needs to be done for all channels, in all layers. There will be 99 (=3+64+32) different ‘vx_image’ objects created just for padding, which seems very inefficient and memory-demanding.
OpenVX convolution only supports convolution on ‘vx_image’ in the format of uint8 or int16, which means conversion of image data will be needed.
OpenVX convolution only supports the ‘vx_int16’ type of weights, so all the weights from the network need to be converted as well, and there will be precision missing.
OpenVX convolution only supports single-channel convolution, so to perform inference for the SRCNN network, each output channel needs to be separated based on the number of input channels, then summed up after single-channel convolution is done. (For example, if a convolution layer has 3 input channels and 64 output channels, then there will be 3*64 intermediate channels to do single-channel convolution, then combined into 64 channels to get the output of this convolution layer). This will need an unreasonable amount of memory to hold all these values.

Another way is to use the ‘NNEF Import Conformance Feature Set’ which seems to be a better way. To use this feature, OpenVX needs to be in version 1.1 or higher, and have the ‘NNEF Import Conformance Feature Set’ installed.

It will enable OpenVX to import an NNEF format neural network trained elsewhere and do inferences on it without manually setting the layer parameters or convolution parameters. It does have some limitations, but it is way better than the custom convolution method. The main challenge with this method is the lack of information, as there is very limited related work that can be found on the Internet, and the official document for 'NNEF Import Conformance Feature Set' was not even found.

7. Comment on all your findings and results.

Comments and discussions were added to each section of the result.

8. Come up with a plan on how to construct your OpenVX graph-based model.

As of now, it was decided to use the 'NNEF Import Conformance Feature Set' to do the inference in OpenVX.
This feature set only support the NNEF format of the neural network, so the '.pth' model needs to be converted to NNEF format somehow. A possible way to do this might be to convert the '.pth' model to ONNX format first which is supported by PyTorch. Then convert the ONNX format neural network to NNEF format using the 'NNEF-Tools' available on GitHub provided by KhronosGroup. The ONNX format only supports inference for fixed-size input tensors, so it would become a limitation of this method using the NNEF format.

If something goes wrong and it turns out that the 'NNEF Import Conformance Feature Set' cannot be adapted in this assignment, then the OpenVX custom convolution method will be considered.

Milestone #3

1. Implement all functions in your baseline model in OpenVX using your choice of node(s). Make sure your OpenVX implementation is using GPU and is fully functional.

The inference of the network was implemented in OpenVX, but due to the OpenVX is not well supported by NVIDIA with respect to recent versions of CUDA, and the lack of access to AMD GPUs, it was not able to test the implementation on GPU.
All of the following discussions on OpenVX are using CPU as the computing device.

2. Record the PSNR and SSIM metrics of your GPU-accelerated program.

	OpenVX Version

Average PSNR	30.0739
Average SSIM	0.999968
Execution Time [sec]	4202

The result shows that the OpenVX version of SRCNN is having a comparable result compared to the PyTorch version and LibTorch version, which means the functionality is been implemented correctly.

The small difference may be due to the workaround of the limitation of using NNEF model. The NNEF model does not support various sizes of input tensors, but the test images do not have the same sizes. Therefore, pre-processing of images to resize them to the expected size would be needed before doing inference. This pre-processing makes the input image to NNEF model not exactly the same as input images from PyTorch/LibTorch version, which may cause a slight difference in PSNR and SSIM.

3. Record the execution time, data footprint, utilization, and other hardware counter statistics.

```
 Performance counter stats for './SRCNN_openVX':

      4,210,135.17 msec task-clock:u              #    1.002 CPUs utilized
     1,779,613,274      cache-references:u        #    0.423 M/sec
       603,479,123      cache-misses:u            #   33.911 % of all cache refs
19,710,661,876,613      cycles:u                  #    4.682 GHz
52,551,239,426,990      instructions:u            #    2.67  insn per cycle
 7,088,306,036,632      branches:u                # 1683.629 M/sec

    4201.020342619 seconds time elapsed

    4209.702427000 seconds user
       0.467970000 seconds sys
```

This OpenVX uses CPU as computation device, so it is expected to be very slow.

CPU utilization was about 1, which means the CPU computation on OpenVX is not benefit from multithreading. This implementation is actually slower than the CPU-based manual convolution inference. This may be due to the complexity of OpenVX infrastructure. It was shown by perf that OpenVX implementation is executing about 70% more instructions and having about 130% more branches which will slow down the calculation.

4. Comment on all your findings and results.

Comments and discussions were added to each section of the result.

5. Perform comprehensive comparisons between your accelerated and non-accelerated model.

The OpenVX implementation gives comparable results on image quality as the two CPU-based C++ inference implementations, but it is less flexible as it does not support variable-sized input tensors.

Regarding the execution time, OpenVX is actually slower than both CPU-based C++ inference implementations, which makes it not favourable.

In terms of support of GPU, in the field of machine learning, both PyTorch/LibTorch and Tensorflow support variables version of CUDA which means they also support GPU calculation and are better maintained than OpenVX for NVIDIA machines.

6. Repeat all the above steps for a candidate CPU-based bilinear and/or bicubic Interpolation approach.

PSNR and SSIM for Bicubic Interpolation

	CPU-based Bicubic Interpolation
Average PSNR	29.461
Average SSIM	0.999974
Execution Time [sec]	0.789

Record the execution time, data footprint, utilization, and other hardware counter statistics

```
 Performance counter stats for './bicubic':

          6,193.39 msec task-clock:u              #    6.257 CPUs utilized
       181,272,518      cache-references:u        #   29.269 M/sec
        85,119,896      cache-misses:u            #   46.957 % of all cache refs
    21,547,372,310      cycles:u                  #    3.479 GHz
    12,638,815,820      instructions:u            #    0.59  insn per cycle
     1,029,456,560      branches:u                #  166.219 M/sec

       0.989764361 seconds time elapsed

       5.957637000 seconds user
       0.241632000 seconds sys
```

This CPU-based bicubic interpolation algorithm was implemented by OpenCV. Based on the report given by perf, it is highly CPU utilized and likely used multithreading. The execution time for interpolation was only taking about 0.789s, which is very fast.

Comparisons between different models

This OpenCV bicubic interpolation is faster than any CPU implementations of SRCNN that are tested in this assignment, so it would be suitable for some speed-critical use cases.

The image quality, however, is slightly lower than any of the SRCNN implementations. This means for some use cases that focus more on image quality and are willing to sacrifice some speed, then SRCNN would be a better option. Note that the current SRCNN model was only trained on a relatively small training set for only 10 hours. If the training set can be enhanced and let it be trained longer, it is expected to produce images that have even better quality and only needs the same time for inference.

7. A report describing your findings and what you have learned throughout this assignment.

In this assignment, the team gained practical experience in implementing a Convolutional Neural Network in C++. The team gets more familiar with PyTorch and LibTorch, as well as various convolution concepts such as strides and padding.

Additionally, it was learned how to use CMake to build the environment and include different libraries necessary for the C++ code, which helped to set up an efficient and reliable development environment for future projects.

Throughout the assignment, research skills were also improved, particularly in finding and utilizing unfamiliar libraries. Team members were able to effectively explore the official API of various libraries and identify useful functions for the task. This experience will prove valuable for future projects that require similar research efforts.

8. Explore GPU Acceleration on PyTorch Implementation (not required)

As it is difficult to use OpenVX on NVIDIA GPUs, it was decided to try to run PyTorch on GPU for acceleration as PyTorch supports CUDA relatively well. All tests except the C++ CUDA version are performed on RTX A5000, whereas the C++ CUDA version is verified and tested on RTX 3080.

	PyTorch Version (CPU)	PyTorch Version (GPU)	C++ LibTorch Version	C++ OpenVX Version	C++ CUDA Version
Average PSNR	29.613	29.613	29.9176	30.0739	31.1663
Average SSIM	N/A	N/A	0.99997	0.999968	N/A

Execution Time [sec]	1.399	0.417	1.277	4202	0.124

The execution result is shown in the table above, along with most of the SRCNN versions that are implemented and tested. The GPU accelerates the CPU PyTorch version more than 3 times and outputs the exact same outcome, while the CPU PyTorch version, was already the fastest in CPU implementation that has been tested. In addition, we have achieved a significantly faster implementation on CUDA, which may contribute to around 5-10% better performance of RTX 3080 compared to RTX A5000. However, it is more likely that CUDA has been well-optimized for NVIDIA GPUs, including threads, blocks, and shared memory structures, which resulted in a significant increase in speed. Interestingly, the CUDA version resulted in a slightly higher PSNR compared to the other versions. We believe that this is due to the architecture of CUDA. Since it utilizes grids and blocks, there are idle processing units that will still contribute to the image if the dimensions of the image do not match the grids and blocks exactly, especially the edge areas. As a result, it might pad the borders of the images. This can be verified, as multiple of the output images have borders of random colours, while the image itself is correct. In conclusion, this result shows that the idea of using GPU for CNN acceleration is sensible, even though OpenVX might not be the best option for inferencing on the NVIDIA platform.

Milestone #4

1. Complete requirements of previous milestones

Completed above.

2. Explore Data Augmentation Techniques and Frameworks

1.Why is data augmentation important in the DL-based SR problem?

Data augmentation is essential in DL-based SR problems when small datasets are not enough for the deep model to be trained sufficiently[7]. It is beneficial to increase the effective size of the dataset by transforming each data sample in different ways and adding all the augmented samples to the dataset. The data augmentation techniques include two main transformations: cleaning transformations and augmentation transformations. We can find rescaling, grayscaling, samplewise centering, samplewise std normalization, futurewise centering, Featurewise std normalization for the cleaning transformations. We also use rotation, flipping and scaling to generate additional training data.
The deep model can also be improved by augmenting the training data with different variations or adding noise or blur. This data augmentation will help the deep learning model to solve more complicated problems in the real world.

2. How does the data augmentation technique improve the SR model training?

There are multiple data augmentation techniques that can improve the SR model training. Below are some of their intended purposes and use cases.

Rescaling: Adjusting the size of images (scaling them up or down) during preprocessing helps the SR model to handle images with different resolutions.
Grayscaling: Converting images to grayscale can help the SR model focus on structural information and texture, which is essential for reconstructing fine details.
Samplewise Centering and Samplewise Standard Deviation Normalization: These techniques involve normalizing each image in the dataset independently. This will enhance the model's ability to learn features and patterns from the images, leading to a better high-frequency in component recovery.
Featurewise Centering and Featurewise Standard Deviation Normalization: These methods normalize the entire dataset. This will help the model to focus on the most relevant features in the input data, potentially leading to improved reconstruction quality.
Flipping: flipping of the input images can help the SR model become invariant to the direction of the image content, leading to better generalization with diverse content orientations.
Noise addition & Blurring: Adding noise and blur to the training data helps the SR model learn to be more robust to various noise types and levels, leading to better performance on real-world images with noise and blur artifacts.

3. [For each data augmentation approach] What is the novel idea of the approach, and how does it work better than the others?

Rescaling: The novel idea is to adjust the size of images, training the model to process images of different resolutions, which helps improve its generalization across diverse image scales.
Grayscaling: The novelty is in converting images to grayscale, enabling the model to focus on structural information and texture over colour, which may be particularly beneficial when dealing with monochrome images or when colour information is less critical.
Samplewise/Featurewise Normalization: The concept is to normalize the input data, enhancing the model's ability to learn features and patterns from images with varying intensities and contrasts.
Rotation: The novel idea is to introduce rotated input images, making the model more robust to variations in orientation and helping it handle different orientations effectively.
Flipping: The innovation is flipping the input images, enabling the model to become invariant to the content direction and improving across different content orientations.
Scaling: The novelty lies in varying scales during training, which improves the ability of the model to recover high-resolution details and ensures it can handle varying different scaling factors.
Noise and Blur addition: The idea is to add noise or blur to the training data, making the model more robust to various noise and blur types and levels, and enhancing its performance.

The new MIDAS (Multi-Image Data Augmentation for Super-resolution) is using the degradation model from pairs of LR and HR input images and extracts shift distribution among LR images.

Then the new augmented scenes will be generated with the extracted shifts distribution and the learned degradation model The trained high-resolution to low-resolution relation can be used to generate HR reference from low-resolution images which can be used to train a CNN for MISR when there are not sufficient training data sets. This approach is better because this is the first method which can generate training datasets when there are insufficient amounts of HR-LR images available for training a CNN for MISR. [8]

CutBlur is using the LR image region and replacing the corresponding region on HR images to generate new datasets. The mixture of augmentation (MoA) is a strategy which determines whether to apply data augmentation on inputs and then randomly selects the data augmentation methods that will be used among the DA pool. This idea is better than other models because CutBlur helps the network to understand which region it needs to apply the super-resolution in the given images. MoA strategy also helps to improve the network's performance. [9]

4. Compare different data augmentation approaches and summarize each method's advantages/disadvantages.

Data Augmentation Approach	Advantage	Disadvantage
Rescaling	Allows the model to handle various resolutions and improves generalization to diverse image scales.	Introduction of distortion which can degrade quality if not applied carefully
Grayscaling	Forces the model to focus on structural information and texture, potentially useful for monochrome images or when colour is not crucial.	Not suitable where colour information is essential may limit the model's performance in colour-dependent tasks.
Samplewise Centering and Samplewise Standard Deviation Normalization	Enhance the model's ability to learn features and patterns, improving high-frequency components recovery.	May be sensitive to outliers, as each image is processed individually.
Featurewise Centering and Featurewise Standard Deviation Normalization	Help the model focus on the most relevant features, which can lead to improved reconstruction quality.	Assumes that the entire dataset has similar characteristics, and may not perform well on diverse datasets.
Flipping	Enhances the model's invariance to the content direction and improves generalization to diverse content orientations.	May not provide significant improvements if the input data does not have diverse orientations.
Noise & Blur addition	Improves the model's	Introduction of artifacts which

	robustness to noise and blur artifacts, resulting in better performance on real-world images.	can degrade quality if not applied carefully
Multi-Image Data Augmentation for Super-resolution	Generates new augmented images for training with the given HR-LR pairs. It significantly helps to train the CNN when there are not enough datasets.	The new method heavily depends on degradation but not using any other methods like shifting, downscaling etc. This will limit the application of the new method.
CutBlur & MoA	CutBlur performs cut-and-paste with the given LR and HR images so that it will not introduce any unrealistic patterns or information loss. MoA's strategy is to introduce randomness to the new augmented images to improve the performance of CNN.	This method may not be applied to all given datasets if the image data contains important contextual information outside of the cropped region, or if the images are already low-resolution or heavily blurred, CutBlur may not provide significant benefits

5. Do all data augmentation techniques that work well for other models such as classification apply to SR models?

Some data augmentation techniques work well for other models, such as classification. It can definitely be applied to SR models. However, not all of them may be equally effective. The effectiveness of a data augmentation technique for SR model training really depends on the specific requirements of the output, the quality of the input data, the model architecture and the application domain. It is crucial to choose the right data augmentation technique for the SR model otherwise, the disadvantages of using data augmentation will outweigh its benefits.

6. Explore the rocAL framework and investigate how it can be extended to help the DL-based SR problem using an existing data augmentation approach. Summarize your findings about the framework in your survey.

rocAL is an augmentation library which is based on AMD ROCm to efficiently process the image data with the help of hardware acceleration. To pre-process the image dataset, we can leverage the rocAL python API (amd.rocal.fn) for image augmentation. This API can be used to read the image and process the image like blurring, rotation etc. This is important to create new training datasets for the DL-based SR problem. rocAL also has PyTorch support (amd.rocal.plugin.pytorch) which might be used to train the DL network with the generated augmented data created by rocAL.

Reference

[1] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 8024-8035. https://pytorch.org/tutorials/advanced/cpp_frontend.html

[2] PyTorch. (2021). CPU threading and TorchScript inference — PyTorch 2.0 documentation. https://pytorch.org/docs/stable/notes/cpu_threading_torchscript_inference.html

[3] Qiangui Huang, Yuanyuan Liu, Qi Wang, Wenqing Jiang, Haojie Li, and Yiran Chen, Practical Blind Denoising via Swin-Conv-UNet and Data Synthesis, https://github.com/cszn/SCUNet, https://arxiv.org/pdf/2203.13278.pdf

[4] Xiaoyu Xiang, Chunxiao Liu, Xin Jin, Xianming Liu, and Wenxiu Sun, SwinIR: Image Restoration Using Swin Transformer, https://github.com/JingyunLiang/SwinIR, https://arxiv.org/pdf/2108.10257.pdf

[5] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn, Designing a Practical Degradation Model for Deep Blind Image Super-Resolution, https://arxiv.org/pdf/2103.14006.pdf, https://github.com/cszn/BSRGAN

[6] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, and Yu Qiao, ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,https://github.com/xinntao/ESRGAN, https://arxiv.org/pdf/1809.00219.pdf

[7] *Hands-on AI Part 14: Image Data preprocessing and Augmentation*. Intel. (n.d.). Retrieved April 10, 2023, from https://www.intel.com/content/www/us/en/developer/articles/technical/hands-on-ai-part-14-image-data-preprocessing-and-augmentation.html

[8] M. Ziaja, J. Nalepa and M. Kawulok, "Data Augmentation for Multi-Image Super-Resolution," IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, Kuala Lumpur, Malaysia, 2022, pp. 119-122, doi: 10.1109/IGARSS46834.2022.9884609.

[9]. Yoo. Jaejun, Ahn. Namhyuk, Sohn. Kyung-Ah: “Rethinking Data Augmentation for Image Super-resolution: A Comprehensive Analysis and a New Strategy.”.