Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

FINAL REPORT

REVISION – 01
24 April 2022

FINAL REPORT

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

Project Leader Date

Prof. Kalafatis Date

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	12/04/2021	Will Cooper		Initial Draft Release
01	04/24/2022	Will Cooper		Final Draft

Table of Contents

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

VALIDATION PLAN

REVISION – 02
24 April 2022

VALIDATION PLAN

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

Project Leader Date

Prof. Kalafatis Date

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	10/03/2021	Will Cooper		Initial Draft Release
01	12/04/2021	Will Cooper		Revision for Fall Report
02	04/24/2022	Will Cooper		Final Draft

Validation Plan - Spring 2022

FSR Paragraph #	Test Name	Success Criteria	Methodology	Status	Responsible Engineer(s)
2.1.1	System Input	The system must be able to autonomously import .obj/.mtl files cohesively, without error or data loss, given a directory file path.	Run the synthetic image generation subsystem; pause execution after IMPORT_OBJECT() function. Verify if object appears with matching textures/shaders. Also check for any data loss during importation.	SUCCESSFUL	Will Cooper
2.1.2	System Output	The system must be able to autonomously export .png images to a specified directory given a file path.	Run the image generation program entirely for one cycle. Verify if new .png image appears in desired directory.	SUCCESSFUL	Will Cooper
2.1.3	Image Parameters	The system should easily allow the user to adjust different parameters such as camera height, angle, maximum vehicle distance, etc.	Run the program with these different parameters, changing only one at a time, and verify each change results in its intended alteration of the exported image.	SUCCESSFUL	Will Cooper
2.1.4	Resolution	The system must be able to produce any resolution/dimension of image as desired by the user (up to 8k).	Test the image generation subsystem with four different resolutions and verify exported images follow suit.	SUCCESSFUL	Will Cooper
2.1.5	Internal Aliasing	The system must be able to adjust internal aliasing parameters such that the user can control how similar the spectral properties of the synthetic image match that of a real image.	Produce two identical images with one image having aliasing and one without. Run these two images through the aliasing test program to verify their differences.	SUCCESSFUL	Will Cooper
2.1.6	Render Time	The system must be able to produce an HD image within a reasonable amount of time given common computer hardware.	Using an intel core i7 processor and an Nvidia 1070 graphics card, the system should produce a single image in less than a minute.	SUCCESSFUL	Will Cooper
2.2.1	Antialisasing	The program must be able to import .png files and reduce aliasing noticeably (>10%).	Aliasing will be graded via ARL's alias detection program, with 1.00 being a perfectly unaliased image.	SUCCESSFUL	Harry Abbott
2.2.2	Filtering	The program must be able to take in any image file, grade the image for aliasing, execute the filter as necessary, then produce and regrade the output filtered image and store the image on the hard drive.	Execute the program on test images, and adjust the filter as desired for an appropriate cutoff frequency, due to the natural distortion-filtration tradeoff that exists in image processing	SUCCESSFUL	Harry Abbott

2.2.3	Unfiltered Recognition	ARL-proposed filter must increase object recognition distance compared to unfiltered images by at least 20%	Execute the testing on ARL-filtered images and compare to unfiltered images	SUCCESSFUL	Harry Abbott
2.2.4	Filtered Recognition	ARL-proposed filter must increase object recognition distance compared to other filtration types by at least 10%	Execute the testing on ARL-filtered images and compare to images filtered by other means	SUCCESSFUL	Harry Abbott
2.2.5	Accuracy & Improvement	ARL-proposed filter must not reduce validation accuracy by more than 5%	Train the ML algorithm with a mixture of real and synthetic filtered images and record the validation accuracy.	SUCCESSFUL	Harry Abbott
2.3.1	Training of CNN	A Convolutional Neural Network should yield a validation accuracy of 70% or above when trained with synthetic images	Build a CNN model and train it using synthetic vehicles (in addition to four other real classes) and plot training and validation accuracies & losses	SUCCESSFUL	Alexia Perez
2.3.2	Testing Synthetic Images	A pre-trained CNN should be able to identify the synthetic vehicles with an accuracy of at least 70%.	Execute the program using synthetic image library with and without filtering, and the results after filtering should recognize vehicles for at least 70% of the tested library	SUCCESSFUL	Alexia Perez
2.3.3	Testing Real Images	A pre-trained CNN should be able to identify real vehicles with an accuracy of at least 70% when trained with synthetic images	Execute the program using real image library after training with filtered and unfiltered synthetic images, and the model trained with filtered synthetic cars should recognize 70% or more of the total real cars tested	SUCCESSFUL	Alexia Perez
N/A	Full System Demo	Similar results of the CNN will be yielded when tested with both a training library of all real images and a training library of a mix of real and synthetic images	Each subsystem must be able to demonstrate its required function or present data as proof of performance.	SUCCESSFUL	Full Team

Validation Timeline - Spring 2022

Task	2/2/22	2/16/22	3/2/22	3/23/22	4/6/22	4/13/22	4/20/22	Date Completed
Status Update 1	🟩							2/1/22
"One Button Execution"	🟩				⬜ Not Started			1/31/22
Re-Test CNN on Textured Vehicles	🟩				🟨 In Progress			1/31/22
Have a minimum of 7 car models	🟩				🟩 Completed			1/31/22
Render new test libraries with textured vehicles	🟩				🟥 Behind Schedule			1/31/22
Status Update 2	⬜	🟩						2/15/22
Evaluate tested images	🟨	🟩						2/11/22
Re-Test CNN on Textured Vehicles if needed	⬜	🟩						2/11/22
Fix any issues with sub-system integration	🟨	🟩						2/11/22
Adjust Filter parameters as needed	⬜	🟩						2/11/22
Status Update 3		⬜	🟩					3/2/22
Evaluate tested images		🟨	🟩					3/2/22
Re-test CNN on textured vehicles if needed		⬜	🟩					3/7/22
Render new test libraries as needed		🟨	🟩					3/24/22
Train CNN with synthetic images		🟨	🟨	🟨	🟨	🟩		4/16/22
Status Update 4			⬜	🟨	🟨	🟩		3/22/22
Improve CNN learning rates + generate plots			⬜	⬜	🟨	🟨	🟩	4/15/22
Evaluate and test newly trained CNN			⬜	⬜	🟨	🟨	🟩	4/17/22
Finalize integration			⬜	⬜	🟨	🟨	🟩	4/19/22
Status Update 5					⬜	🟩	🟩	4/13/22
Complete Final Report					⬜	🟨	🟩	4/24/22
Complete Engineering Project Showcase Poster					⬜	🟨	🟩	4/24/22
Final Project Update						🟨	🟩	4/21/22
System Demo						🟨	🟩	4/25/22
Project Showcase						🟨	🟩	4/29/22
Final Report						🟨	🟩	4/24/22

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

CONCEPT OF OPERATIONS

REVISION – 03
24 April 2022

CONCEPT OF OPERATIONS

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

Project Leader Date

Prof. Kalafatis Date

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	09/15/2021	Will Cooper		Initial Draft Release
01	10/03/2021	Will Cooper		Major reformatting, added ICD, FSR, Validation Plan
02	12/04/2021	Will Cooper		Revision for Fall Report
03	04/24/2022	Will Cooper		Final Draft

Table of Contents

1) Introduction

1.1 Executive Summary

With access to enough training data, machine learning algorithms can solve an extremely wide range of problems that traditional computer programs cannot. However, in the absence of sufficient training data, or in the case of faulty training data, Machine Learning (ML) is useless. Sometimes, collecting real data is not feasible for economic, technological, environmental, or temporal reasons. There are a multitude of programs available which are capable of creating Computer Generated Imagery (CGI), hereafter referred to as synthetic images. However, these programs often generate aliased images with erroneous high frequency data embedded within them. If a ML algorithm is trained using this data, it can yield erroneous information when it is presented with real data.

Our goal is to use spectral filtering to create a large number of synthetic images when provided with a 3D model of a target of interest. These synthetic images will be free of the aliasing that is currently present in many synthetic images and will be used to train an image-recognition ML algorithm. Using this system to train ML algorithms would not only save time and money, but allow scientists, engineers, military, and industry to capture data sets that were once impossible or impractical to obtain.

1.2 Background

In optics, a “real” image is defined as light striking a sensor, which is then encoded into a multidimensional array. A “synthetic” image, however, is defined as a multidimensional array created by mathematical manipulation on data. For example, a real image would be a picture of a car taken with a DSLR camera, and a synthetic image could be an image of a 3D model of a car that was taken in a modeling software.

In these modeling softwares, there is a lot of aliasing present, especially when models are rendered at long ranges. ML algorithms might train to recognize these aliasing artifacts when learning how to classify targets. After training, when presented with real images of the target, these aliasing artifacts will not be present, and the ML algorithm might make mistakes in classification.

Our proposed system will take in a 3D model of an object of interest. It will then generate numerous synthetic images of that object. These images will likely have erroneous high-frequency (aliased) information encoded within them. However, we will pass these images through a spectral filter. This will rid the images of any high-frequency artifacts that should not be present. Finally, these filtered images will be used to explore the capability of synthetically training an image-based ML algorithm.

1.3 Abbreviations

ConOps	Concept of Operations
ICD	Interface Control Document
FSR	Functional System Requirements
ML	Machine Learning
3D	Three Dimensional
2D	Two Dimensional
CAD	Computer Aided Design
CGI	Computer Generated Imagery. Synonymous with Synthetic Imagery

1.4 Definitions

Machine Learning:
A class of computational artificial intelligence wherein an algorithm is trained on a set of data, learns from it, and can improve itself automatically. In the scope of this project, we are especially interested in machine learning algorithms which can identify objects in images.

Synthetic Image:
A computer-generated image.

1.5 References

Susan's Abstract:
https://docs.google.com/document/d/1LLITiCUDm1ER-I-2Hx8ivNuIs-21aJmy/edit?usp=sharing&ouid=102047605495244699269&rtpof=true&sd=true

Fourier Transforms:
https://plus.maths.org/content/fourier-transforms-images

2) Operating Concept

2.1 Scope

Our proposed system will operate with synthetic object input (3D models) and output a large sample of high-fidelity synthetic images. These synthetic images can then be used to train ML algorithms that previously had insufficient training data. These synthetic models should be indistinguishable from real models in terms of the physics which define a real image (particularly with respect to spectral properties). The project as a whole is divided into three categories: a synthetic image generator, a spectral filter, and convolutional neural network. These each have their own subsystem design and will be discussed more in depth within their respective sections of this report.

2.2 Constraints

Our team's goal is to create a system which receives a 3D model as input and outputs a large sample of synthetic images. Additionally, our subsystem is not intended to improve ML algorithms as a whole, but rather improve the ability of proven ML algorithms to identify and be trained with filtered synthetic images. We hope that users will be able to use the synthetic images generated by our algorithm to effectively train any ML technology. Our focus is on images in the visible spectrum.

2.3 Hardware

Our project is entirely software-based, and is designed to run on any computer with modern graphics capabilities.

2.4 User Interface

The system will have a single mode of operation; namely, when provided with a 3D model, it will save many synthetic images of that model to a location specified by the user. The user can specify how many images, and the randomness of each scene by defining metric ranges in the Synthetic Generator.

2.5 Users

Our system is of interest to anyone who needs to train a machine learning algorithm for image recognition but has a limited data set of real images. With our system, users could generate a plethora of synthetic images without fear of training their algorithm on faulty data. Ideally, this system will be easy to use; a user could simply provide a 3D model of an object of interest, and our program would output large quantities of clean, filtered, synthetic images. This could be ideal when you have very few images available for training but need many images to train the ML algorithm.

3) Subsystem Allocation

3.1 Synthetic Image Generation

Will Cooper was responsible for the initial synthetic generation of the images. He used Python automation to control the popular modeling software Blender. This code automatically imported models provided by the user, randomly translated these models within a background scene, and took images of it at different ranges, angles, and positions.

3.2 Fourier-Based Spectral Filtering

Harry Abbot was responsible for developing the spectral filter. The filter takes in the raw synthetic images from Blender and removes the aliasing artifacts, resulting in an effectively realistic image.

3.3 ML Testing: Convolutional Neural Network

Alexia Perez was responsible for exploring the capability of a ML algorithm to be trained using synthetic images. It was trained and evaluated by comparing three convolutional neural networks: one being trained using real images, one being trained with unfiltered synthetic images, and one being trained with filtered synthetic images. In this way, we were able to compare the effectiveness of the spectral filtering against real photographs.

4) Implementation

4.1 Car Identification

Say a police station is looking for a unique stolen vehicle. They only have one picture of the car so they use it to develop a 3D model. The police station could then feed that model of the vehicle to our system, and it will generate a plethora of filtered synthetic images. The police could then use those images to train a ML algorithm so that the stolen vehicle could automatically be identified from street cameras.

4.2 Foreign Entity Recognition

Assume that the user of our algorithm is the U.S Air Force. If they were to capture a real image of a foreign entity, such as an airplane, and wanted to then use that image to train a ML algorithm, they would need to generate additional images (since one is not enough). The issue arises when these synthetic images are not accurate in comparison to the original (real) image, as the AI might not be able to effectively recognize the objects it was trained to recognize. In this scenario, our team's system would provide a solution for the user. If the user was able to generate a 3D model of the aircraft, our program could create enough synthetic images to effectively train an ML algorithm to detect the aircraft.

5) Analysis

5.1 Future Improvements

Different cameras process the images that they capture in slightly different ways. In the future, it might be beneficial to build different models of camera Modulation Transfer Functions into our software. In this way, we could generate images that exactly match the spectral properties of the real cameras that our users' algorithms will be using. This would further reduce errors in identification. Additionally, expanding the amount of training models, shaders, and background scenes would help increase the accuracy of a synthetically trained ML algorithm.

5.2 Limitations

Our software is limited by the quality of the 3D model it is provided as input. A poorly designed model that is not reflective of reality will not effectively train any ML algorithm. We eventually learned that models with textures and shaders are necessary for accurate ML training/recognition.

5.3 Alternative Designs

Initially, we planned to use a Generative Adversarial Network to generate synthetic images. GANs have been used in the past to generate synthetic images. However, we learned that GANs themselves need to be trained with real data in order to produce their synthetic images. We determined that this would be counterproductive for our project. If one already has a large sample of real data on hand, they could simply use it to train their ML algorithm directly. There is no need to feed those images to a GAN and generate more synthetic images.

5.4 Potential Impact

As stated above, this proposed system could revolutionize how machine learning algorithms are trained. It would save society time, money, and allow never-before captured images to be used to train machine learning and artificial intelligence.

From our understanding, there doesn't seem to be any environmental impact or physical safety concerns associated with this system. However, using this system with models of people may result in some ethical concerns. For example, if an airport were to train security cameras to identify a certain race or gender, the unpredictability of ML algorithms could result in negative associations that might victimize certain classes of people.

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 02
24 April 2022

FUNCTIONAL SYSTEM REQUIREMENTS

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

____
Project Leader Date

____
Prof. Kalafatis Date

____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	10/03/2021	Will Cooper		Initial Draft Release
01	12/04/2021	Will Cooper		Revision for Final Report
02	04/24/2021	Will Cooper		Final Draft

Table of Contents

1) Introduction

1.1 Executive Summary

The purpose of this document is to illustrate the requirements needed to run the system. Each subsystem has both hardware and software requirements, and requires certain parameters be met in order to run smoothly and entirely.

1.2 Abbreviations

FSR	Functional System Requirements
ML	Machine Learning
FFT	Fast Fourier Transform
RAM	Random Access Memory
3D	Three Dimensional
2D	Two Dimensional
GB	Gigabyte
MatLab	Matrix Laboratory (Software)
PC	Personal Computer
FBX	Filmbox (3D model) Filetype
JPG	Joint Photographic Experts Group (Image) Filetype

1.3 Definitions

Fourier Transform:
The mathematical decomposition of a function from the time domain to the frequency domain

Machine Learning:
Having a computer learn and adapt without explicit instruction, but instead adapting by extracting information from data via algorithms and statistical models.

Fidelity:
An aliasing property of images in spectral analysis that is used to distinguish how realistic an image is.

1.4 References

Study of Fidelity of Synthetic Imagery for Artificial Intelligence (AI) and Machine Learning (ML) Problems by Susan Young, et al.
*****<i>Not in the public domain. Email shiqiong.s.young.civ@army.mil to request access.</i>

MatLab Wikipedia:
https://en.wikipedia.org/wiki/MATLAB

Blender Wikipedia:
https://en.wikipedia.org/wiki/Blender_(software)

Python Wikipedia:
https://en.wikipedia.org/wiki/Python_(programming_language)

Visual Studio Code Wikipedia:
https://en.wikipedia.org/wiki/Visual_Studio_Code

2) System Requirements

2.1 Synthetic Image Generation

2.1.1 Subsystem Input:

The system must be able to autonomously import .obj/.mtl files cohesively, without error or data loss, given a directory file path.

2.1.2 Subsystem Output:

The system must be able to autonomously export .png images to a specified directory given a file path.

2.1.3 Image Parameters:

The system should easily allow the user to adjust different parameters such as camera height, angle, maximum vehicle distance, etc.

2.1.4 Resolution:

The system must be able to produce any resolution/dimension of image as desired by the user (up to 8k).

2.1.5 Internal Aliasing:

The system must be able to adjust internal aliasing parameters such that the user can control how similar the spectral properties of the synthetic image match that of a real image.

2.1.6 Render Time:

The system must be able to produce an HD image within a reasonable amount of time given common computer hardware.

2.2 Spectral Filtering

2.2.1 Antialiasing:
The program must be able to import .png files and noticeably reduce aliasing (greater than 10%).

2.2.2 Filtering:
The program must be able to take in any image file, grade the image for aliasing, execute the filter as necessary, then produce and regrade the output filtered image and store the image on the hard drive.

2.2.3 Unfiltered Recognition:
ARL-proposed filter must increase object recognition distance compared to unfiltered images by at least 20%

2.2.4 Filtered Recognition:
ARL-proposed filter must increase object recognition distance compared to other filtration types by at least 10%

2.2.5 Accuracy & Improvement:
ARL-proposed filter must not reduce validation accuracy by more than 5%.

2.3 Convolutional Neural Network

2.3.1 Training of CNN:
A Convolutional Neural Network should yield a validation accuracy of 70% or above when trained with synthetic images

2.3.2 Testing Synthetic Images:
A pre-trained CNN should be able to identify the synthetic vehicles with an accuracy of at least 70%.

2.3.3 Testing Real Images:
A pre-trained CNN should be able to identify real vehicles with an accuracy of at least 70% when trained with synthetic images

3) Equipment

3.1 Hardware

3.1.1 Custom Built Desktop:

To generate synthetic datasets in a reasonable amount of time, high-end computer hardware is desirable. In our case, we used a custom built computer consisting of an AMD Ryzen 3970x Processor and an Nvidia dual RTX Titan Graphics Card.

3.1.2 Macbook Pro (2021):

To edit code, access Blender, and for general purpose use, a Macbook Pro running Windows Bootcamp with an intel core i7 processor and a discrete Nvidia 1070ti graphics card was used.

3.2 Software

3.2.1 MatLab:

MatLab is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages (Wikipedia). We will use MatLab scripts to construct and test our Fourier-based filter..

3.2.2 Blender/Cycles:

Blender is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, virtual reality, and computer games (Wikipedia). We use Blender to automate scene/model setup for our synthetic images. Cycles is a built in render engine within Blender that uses ray tracing to realistically reflect real light physics.

3.2.3 Python:

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its defined language constructs and its object-oriented approach help programmers write clear, logical code for small and large-scale projects (Wikipedia). We use python scripts to automate the synthetic generation process so that we can quickly output thousands of synthetic images.

3.2.4 Google Colaboratity:

Colaboratory, or “Colab” for short, is a product from Google Research that allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education, since it provides users with the opportunity to pay for access to powerful GPU and TPU nodes.

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

INTERFACE CONTROL DOCUMENT

REVISION – 02
24 April 2022

INTERFACE CONTROL DOCUMENT

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

____

Project Leader Date

____

Prof. Kalafatis Date

____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	10/03/2021	Will Cooper		Initial Draft Release
01	12/04/2021	Will Cooper		Revision for Final Report
02	04/24/2022	Will Cooper		Final Draft

Table of Contents

1) Introduction

1.1 Executive Summary

The intent of this document is to describe the interface between the multiple subcomponents of the project. Because the objective of the system is not reliant on the interaction of hardware components, this document focuses instead on how each step of the software interfaces with the step before and after it, how the images progress through each subsystem, and ultimately how the resulting datasets interact with machine learning algorithms.

1.2 Abbreviations

ICD	Interface Control Document
ML	Machine Learning
3D	Three Dimensional
STL	Standard Triangle Library Filetype
FBX	Filmbox (3D model) Filetype
JPG	Joint Photographic Experts Group (Image) Filetype
MatLab	Matrix Laboratory (Software)

1.3 Definitions

Render:
To take a simulated three dimensional scene and convert it into a visual representation that our eyes can interpret. This involves many complex algorithms that alter how light is projected and reflected into a scene, and typically displayed as an image on screen.

Fourier Filter:
A filter that is applied to an image or 3D scene that modifies the resulting output image to alias in a way that closely mimics how images are captured with cameras in the real world.

1.4 References

No external references were used for this version of the ICD.

2) Synthetic Library Development Interface

2.1 Synthetic Generation File Input

To simplify the scope of this project, all input models must first be converted to an .OBJ/.MTL file type before importing. In future modifications of this project, the synthetic generator could accept .FBX files as input as well. Ideally, these .FBX files would contain pre-defined shaders and textures to maintain realism to the scene.

2.2 Modeling Software

This project uses Blender as its primary means of 3D object manipulation. Files will be imported into the software through automated Python scripting, and these models will be pulled from a directory specified by the user.

2.3 Automation Scripting

The ability to automate the importation and render process streamlines the ability to create thousands of images for machine learning training. Since Blender is open source, the internal code can be manipulated to import images and adjust certain characteristics of the models automatically. These characteristics include, but are not limited to, object location, orientation, background, lighting, and rendering properties. The script will then render the scene and export the resulting image to a defined directory. This stage essentially sets each unique scene before rendering the image.

2.4 Rendering Engine

Rendering software is the necessary backbone that transforms a synthetic scene to useful visual data that our eyes (and visual/recognition ML) can interpret. It takes the kinesthetic information from the modeling software and applies its own algorithms that output visual data. Certain characteristics of the renderer can be manipulated to make an image more or less realistic, typically at the cost of rendering time. Once the rendering engine is finished, the image will be exported to a user-specified directory.

2.5 Spectral Filtering

This stage of the process takes in the image from the export directory of the scene from the rendering software and applies the spectral filter to the image. This is used to help remove the unwanted aliasing in the images and prepare them for testing in the convolutional neural network.

2.6 CNN Testing

The final stage of the program takes these filtered images and uses them to train a convolutional neural network. This trained CNN is then compared to a CNN that is trained with real images to determine any increase or decrease in image recognition accuracy.

3) Machine Learning Interface

3.1 Training Data

Again, the purpose of this project is to provide spectrally realistic image libraries to successfully train arbitrary image recognition machine learning algorithms. To achieve this, the program as a whole will take in a 3D model in .OBJ format and output a desirable number of filtered, realistic images in .PNG format.

3.2 Image Recognition

Machine learning requires training data to be trained successfully. ML algorithms that take real word, real time camera images as input must use similar input data for training. Thus, any arbitrary image recognition ML could utilize this project by taking the resulting synthetic image dataset as training input, and then applying that training to consistently identify objects in real world, camera-captured images. Our project specifically uses a Convolutional Neural Network to identify the type of object within an image.

Spectral Filtering of Synthetic Images for Machine Learning Applications

Will Cooper

SYNTHETIC IMAGE GENERATION SUBSYSTEM

REVISION – 01
24 April 2022

Change Record

Rev.	Date	Originator	Approvals	Description
00	12/04/2021	Will Cooper		Initial Draft Release
01	04/24/2022	Will Cooper		Final Draft

Table of Contents

List of Figures

1) Introduction

1.1 Executive Summary

The synthetic image generation subsystem is designed to generate near photo-realistic images to train machine learning algorithms. The subsystem must be able to quickly and autonomously generate these images using imported 3D models and store them into a designated directory. The generator must be adjustable to control the amount of images produced, as well as the randomness of the model position, camera angle, and camera height. Additionally, every model imported into the scene should have no triangulation or data loss. Although the generator can adjust aliasing manually using the built in *Grease-Pencil* software, that obstacle is addressed specifically in the Fourier Filtering section. The object of this subsystem is rather to generate numerous synthetic images that are as realistic as possible.

1.2 Abbreviations

ML	Machine Learning
3D	Three Dimensional
2D	Two Dimensional
OBJ	Object (Filetype)
CAD	Computer Aided Design
FBX	Filmbox (Filetype)
API	Application Programming Interface
CNN	Convolutional Neural Network

1.3 Definitions

Machine Learning:
A class of artificial intelligence wherein an algorithm is trained on a set of data, learns from it, and can improve itself automatically. In the scope of this project, we are especially interested in machine learning algorithms which can identify objects in images.

Synthetic Image:
A computer-generated image.

.OBJ/.MTL:
Object/Material Files. An 3D object file paired with an associated material file.

API:
Application Programming Interface: A library of specific language commands that directly modifies or executes actions within a specific program or application.

2) Operating Concept

2.1 Scope

This subsystem operates by importing a 3D model, translating and rotating that object within a three dimensional space (or scene), translating and rotating a camera viewport within the same scene, rendering a ray-traced exposure from the perspective of the camera, and exporting that render into a designated directory.

2.2 Constraints

This generator is restricted to importing .OBJ/.MTL files. All 3D models must be converted to these file pairs before the system will function properly.

2.3 User Interface

All aspects of this subsystem were constructed using Blender v2.93, and all operations are performed within its user interface. This includes modifying models, file locations, object randomness, etc. These elements are stored as python script constants, which can be viewed within Blender through a script window.

3) Equipment

3.1 Software

The whole subsystem runs exclusively on Blender v2.93 using Python v3.10.0. All scripting was written and executed using built in scripting windows in Blender. “Cycles” was the render engine of choice because it is the most commonly used ray traced engine, providing more realistic images with plenty of online community support.

3.2 Blender Python API

Blender has a large library of local commands that allow Python to execute changes within the application without having to use the traditional user interface. This is what allows automation to take place, however the Blender Python API is critical to finding which commands are linked to each desired action. To use this library, the command “import bpy” is necessary at the beginning of any python script.

3.3 Hardware

This generator can be run on any computer with a processor and graphics card that can run Blender v2.93. However, the more advanced/powerful the computer hardware is, the faster the generator will be able to produce images. When using this generator to create a six thousand image training set, I used a dual Nvidia Titan RTX graphics card with a 3970x processor. This allowed me to produce an HD quality library set in approximately 14 hours.

4) Data

4.1 3D Models

For testing, I used pre-made .OBJ files of cars downloaded from the internet. I found 10 different automobiles including sedans, pickup trucks, jeeps, and racecars. Each object file had a corresponding material file which gave the vehicle its appropriate shaders/textures.

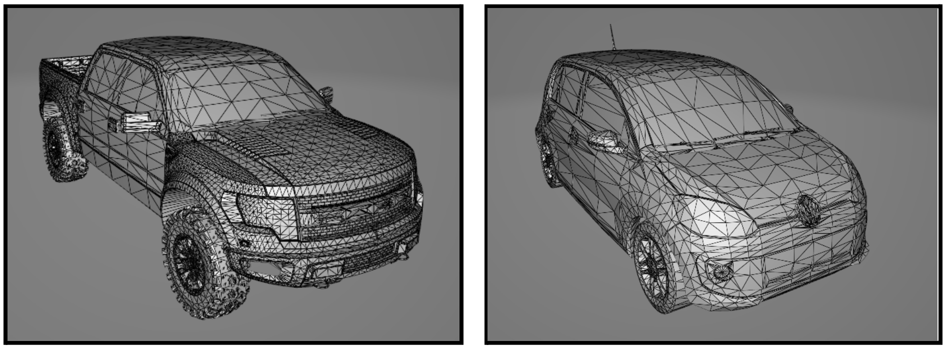
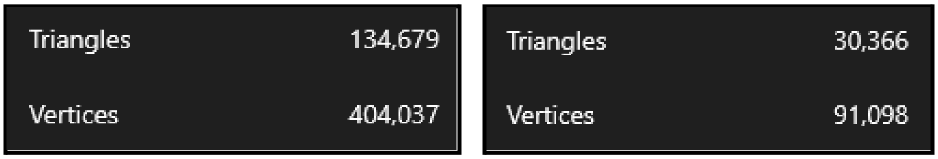


Figure 1: Examples of .OBJ Models

4.2 Triangulation

After importing these models into blender, the triangle count and vertice count remained the same. This implies there was no compression or data loss during the import process. Below are the triangle and vertice count for the two models shown above.

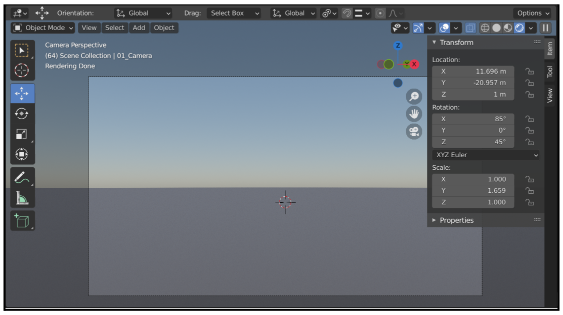


Triangles	134,679	Triangles	30,366
Vertices	404,037	Vertices	91,098

Figure 2: Examples of Model Triangulation

4.3 Background Scene

This generator uses a stagnant background scene of a basic gray plane. This road has built in shaders/textures and does not move during the generation process. However, this could be changed in the future as is discussed in the "Future Implementations" section later in the paper.



Orientation: Global Drag: Select Box Global

Camera Perspective
(64) Scene Collection | 01_Camera
Rendering Done

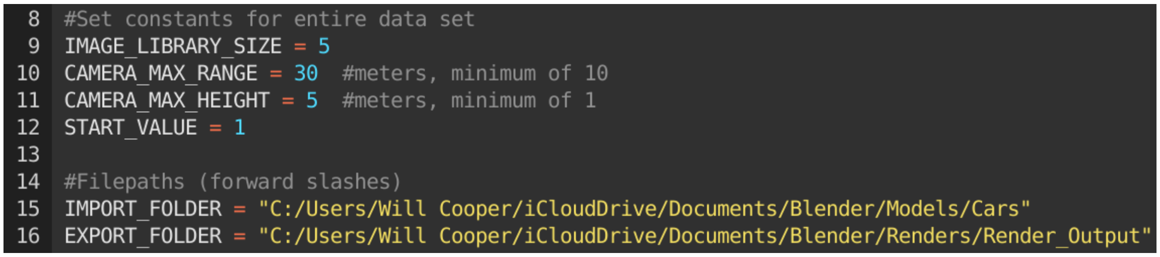
Transform
Location:
X 11.696 m
Y -20.957 m
Z 1 m
Rotation:
X 85°
Y 0°
Z 45°
XYZ Euler
Scale:
X 1.000
Y 1.659
Z 1.000

Figure 3: Background Scene

4.4 Python

4.4.1 Constants:

Before executing the Python script, the user must first define the four constants and two file paths at the top of the script. These constants control the output library size, the camera's maximum range, the camera's max height, and the numerical starting value for the library output (for labeling purposes). The user must also define the model library file path as well as the desired export library path.

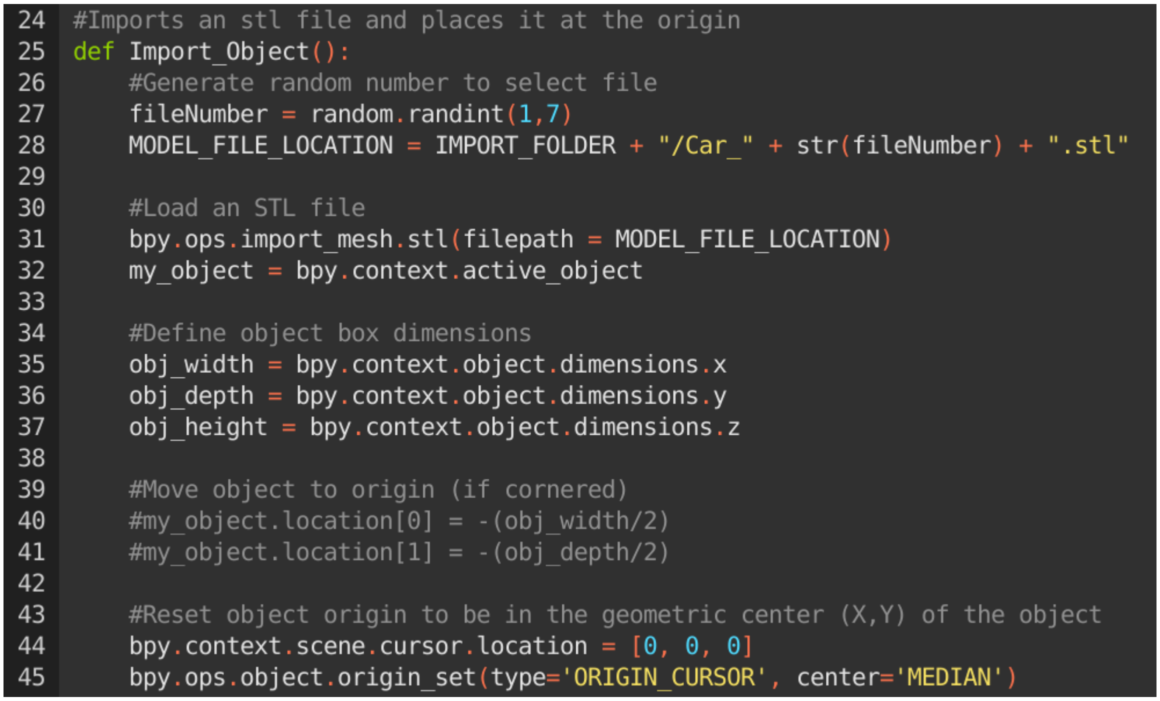


```python
#Set constants for entire data set
IMAGE_LIBRARY_SIZE = 5
CAMERA_MAX_RANGE = 30  #meters, minimum of 10
CAMERA_MAX_HEIGHT = 5  #meters, minimum of 1
START_VALUE = 1

#Filepaths (forward slashes)
IMPORT_FOLDER = "C:/Users/Will Cooper/iCloudDrive/Documents/Blender/Models/Cars"
EXPORT_FOLDER = "C:/Users/Will Cooper/iCloudDrive/Documents/Blender/Renders/Render_Output"
```

Figure 4: Python Constants

4.4.2 Importing Model:

The following script searched through the given model library path and randomly imported one of the seven available vehicles. Each vehicle was labeled "Car_X" where X was a single digit 1 through 10. It then translated that model to sit at the origin and redefined the object's origin to its geometric center on the XY plane.

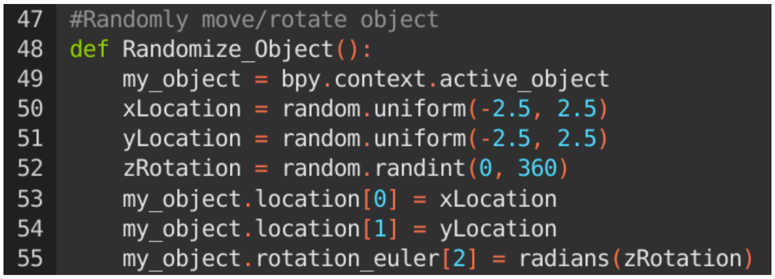


```python
#Imports an stl file and places it at the origin
def Import_Object():
    #Generate random number to select file
    fileNumber = random.randint(1,7)
    MODEL_FILE_LOCATION = IMPORT_FOLDER + "/Car_" + str(fileNumber) + ".stl"

    #Load an STL file
    bpy.ops.import_mesh.stl(filepath = MODEL_FILE_LOCATION)
    my_object = bpy.context.active_object

    #Define object box dimensions
    obj_width = bpy.context.object.dimensions.x
    obj_depth = bpy.context.object.dimensions.y
    obj_height = bpy.context.object.dimensions.z

    #Move object to origin (if cornered)
    #my_object.location[0] = -(obj_width/2)
    #my_object.location[1] = -(obj_depth/2)

    #Reset object origin to be in the geometric center (X,Y) of the object
    bpy.context.scene.cursor.location = [0, 0, 0]
    bpy.ops.object.origin_set(type='ORIGIN_CURSOR', center='MEDIAN')
```

Figure 5: Import Object Script

4.4.3 Model Translation & Rotation:

This small snippet of the script simply translates the imported model randomly within a 2.5 square meter. It also randomly rotates the vehicle. This allows the model to better reflect reality since the vehicle is not always in the center of a photograph and is not always facing one direction.

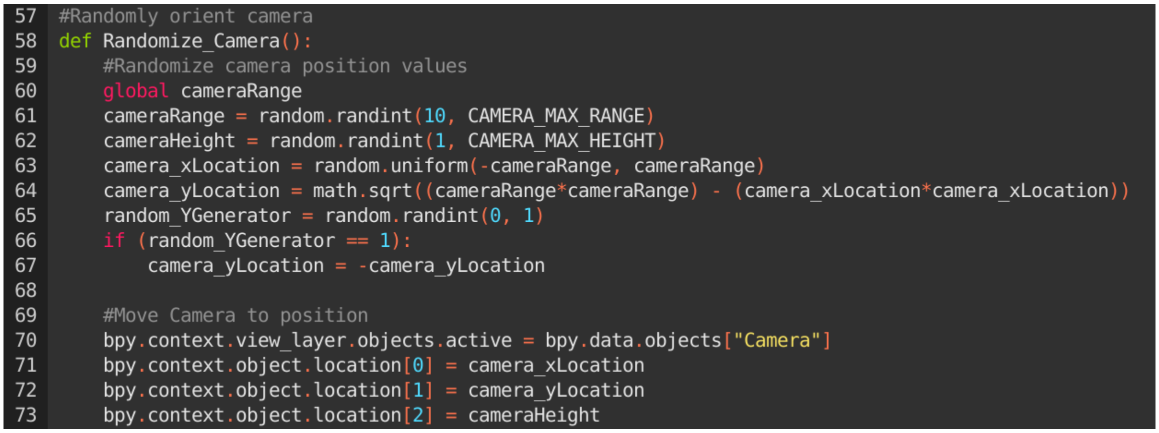


```
#Randomly move/rotate object
def Randomize_Object():
    my_object = bpy.context.active_object
    xLocation = random.uniform(-2.5, 2.5)
    yLocation = random.uniform(-2.5, 2.5)
    zRotation = random.randint(0, 360)
    my_object.location[0] = xLocation
    my_object.location[1] = yLocation
    my_object.rotation_euler[2] = radians(zRotation)
```

Figure 6: Object Randomization Script

4.4.4 Camera Translation & Rotation:

The camera viewport in this scene is always oriented to face one meter above the origin. When selecting a range for the camera, pythagorean calculations are executed so that the camera can be positioned anywhere around the model while maintaining a specific distance from the origin. The camera height and distance are selected randomly within the range determined by the constants at the beginning of the script.

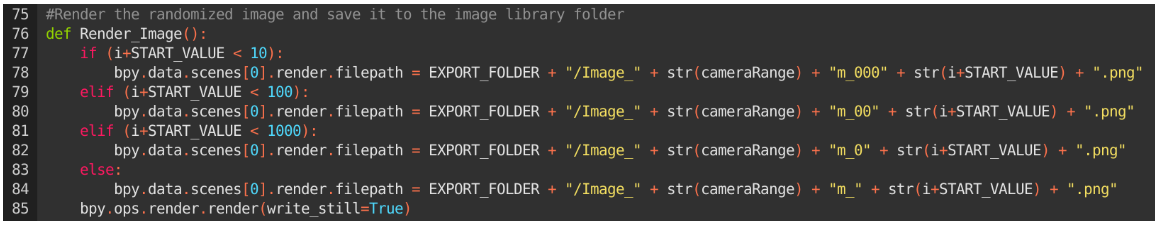


```
#Randomly orient camera
def Randomize_Camera():
    #Randomize camera position values
    global cameraRange
    cameraRange = random.randint(10, CAMERA_MAX_RANGE)
    cameraHeight = random.randint(1, CAMERA_MAX_HEIGHT)
    camera_xLocation = random.uniform(-cameraRange, cameraRange)
    camera_yLocation = math.sqrt((cameraRange*cameraRange) - (camera_xLocation*camera_xLocation))
    random_YGenerator = random.randint(0, 1)
    if (random_YGenerator == 1):
        camera_yLocation = -camera_yLocation

    #Move Camera to position
    bpy.context.view_layer.objects.active = bpy.data.objects["Camera"]
    bpy.context.object.location[0] = camera_xLocation
    bpy.context.object.location[1] = camera_yLocation
    bpy.context.object.location[2] = cameraHeight
```

Figure 7: Camera Randomization Script

4.4.5 Rendering:

Once the model and camera have been positioned, it is time for Cycles to begin rendering the image. Executing a render in Python does not open the rendering window and automatically exports the render into the specified file at the beginning of the script.

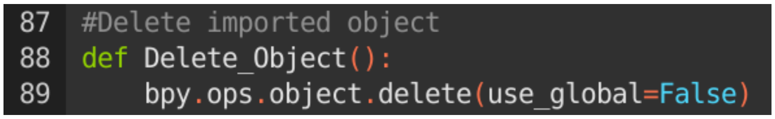


```python
#Render the randomized image and save it to the image library folder
def Render_Image():
    if (i+START_VALUE < 10):
        bpy.data.scenes[0].render.filepath = EXPORT_FOLDER + "/Image_" + str(cameraRange) + "m_000" + str(i+START_VALUE) + ".png"
    elif (i+START_VALUE < 100):
        bpy.data.scenes[0].render.filepath = EXPORT_FOLDER + "/Image_" + str(cameraRange) + "m_00" + str(i+START_VALUE) + ".png"
    elif (i+START_VALUE < 1000):
        bpy.data.scenes[0].render.filepath = EXPORT_FOLDER + "/Image_" + str(cameraRange) + "m_0" + str(i+START_VALUE) + ".png"
    else:
        bpy.data.scenes[0].render.filepath = EXPORT_FOLDER + "/Image_" + str(cameraRange) + "m_" + str(i+START_VALUE) + ".png"
    bpy.ops.render.render(write_still=True)
```

Figure 8: Rendering Script

4.4.6 Resetting the Scene:

Finally, once the image has been rendered, the model is deleted and the scene is reset for the next model import. With the exception of the rendering, this entire process runs in tenths of a second.



```python
#Delete imported object
def Delete_Object():
    bpy.ops.object.delete(use_global=False)
```

Figure 9: Delete/Reset Scene

4.5 Render Optimization

4.5.1 Discussion:

With advanced graphics cards/processors, this generator can output an HD quality image in less than 20 seconds. However, when it comes to ray tracing, there are dozens of settings you can tweak including particle bouncing, denoising algorithms, tile size, antialiasing, and sampling rates that affect the quality and render time of the images. This section of the paper analyzes how changing these settings affects the output render time.

4.5.2 Analysis:

Below are four different settings that significantly impacted the render performance. All external variables were kept constant and each setting was tested on the same image five times, with the resulting time being averaged.

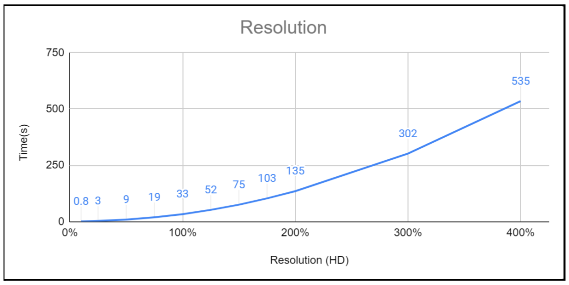


Figure 10: Image Resolution vs Time

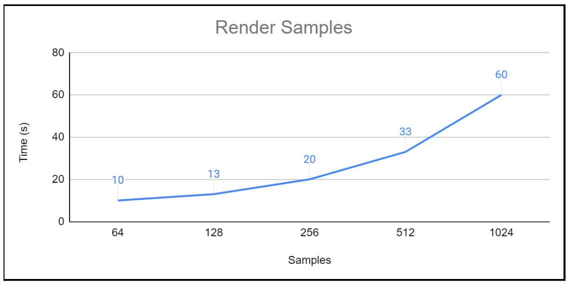


Figure 11: Render Samples vs Time

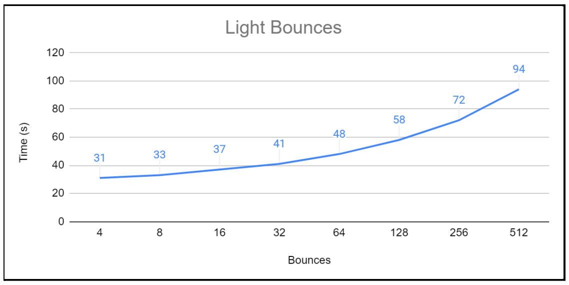


Figure 12: Light Bounces vs Time

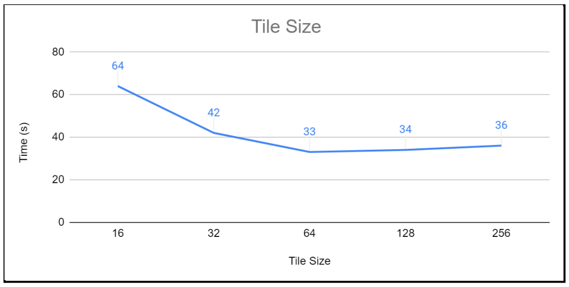


Figure 13: Tile Size vs Time

4.5.3 Conclusion:

Ultimately, each setting came down to a tradeoff between render time and image quality. The lower the user can make the quality of an image, the faster that image will render. In the context of this project, the ideal settings allow for as low quality of an image as possible while maintaining constant recognition by the machine learning algorithm. During ML training, higher quality renders will most likely be required.

5) Analysis

5.1 Functionality

From an individual perspective, this subsystem performed perfectly. The generator met all of the respective requirements and was able to produce a desired library size of synthetic images with random characteristics.

5.2 Future Implementations

One possible implementation for the future would be to add a dynamic background. Adding a changing background would prevent any ML algorithm from fixating on an incorrect part of the scene. This would most likely increase recognition rates in the machine learning validation phase of the system. I had originally tried to implement this with basic images of city roads, however, the still images did not have any depth and I realized it was more difficult than anticipated. This is due to the fact that it requires a flat image to be transposed onto a spherical object background. While possible, I couldn’t get the proportions to align correctly and the images never seemed to align correctly.

6) Validation Plan - Spring 2022

FSR Paragraph #	Test Name	Success Criteria	Methodology	Status
2.1.1	System Input	The system must be able to autonomously import .obj/.mtl files cohesively, without error or data loss, given a directory file path.	Run the synthetic image generation subsystem; pause execution after IMPORT_OBJECT() function. Verify if object appears with matching textures/shaders. Also check for any data loss during importation.	SUCCESSFUL
2.1.2	System Output	The system must be able to autonomously export .png images to a specified directory given a file path.	Run the image generation program entirely for one cycle. Verify if new .png image appears in desired directory.	SUCCESSFUL
2.1.3	Image Parameters	The system should easily allow the user to adjust different parameters such as camera height, angle, maximum vehicle distance, etc.	Run the program with these different parameters, changing only one at a time, and verify each change results in its intended alteration of the exported image.	SUCCESSFUL
2.1.4	Resolution	The system must be able to produce any resolution/dimension of image as desired by the user (up to 8k).	Test the image generation subsystem with four different resolutions and verify exported images follow suit.	SUCCESSFUL
2.1.5	Internal Aliasing	The system must be able to adjust internal aliasing parameters such that the user can control how similar the spectral properties of the synthetic image match that of a real image.	Produce two identical images with one image having aliasing and one without. Run these two images through the aliasing test program to verify their differences.	SUCCESSFUL
2.1.6	Render Time	The system must be able to produce an HD image within a reasonable amount of time given common computer hardware.	Using an intel core i7 processor and an Nvidia 1070 graphics card, the system should produce a single image in less than a minute.	SUCCESSFUL

Spectral Filtering of Synthetic Images
for Machine Learning Applications

Harry Abbott

FOURIER-BASED SPECTRAL FILTER SUBSYSTEM

REVISION – 01
24 April 2022

Change Record

Rev.	Date	Originator	Approvals	Description
00	12/04/2021	Harry Abbott		Initial Draft Release
01	04/24/2022	Harry Abbott		Final Report

Table of Contents

Lists of Tables

List of Figures

1) Subsystem Introduction

The purpose of this subsystem is to design an anti-aliasing frequency-based spectral filtering program for ML applications. It will be validated by finding aliasing based on the 95% energy level of the near and far ranges of the image for various images in an effort to become spectrally indistinguishable from real images. During the next semester the filter will be tested on an ML algorithm, however the scope of the current project (with respect to ECEN 403) is to design and validate the anti-aliasing properties in preparation for testing on an ML algorithm designed by another group member. The program in its entirety is available in Appendix A.

1.1 Aliasing

In synthetic image generation, there is a common phenomena known as aliasing which negatively impacts ML training/testing. In essence, aliasing is high-frequency information in the frequency domain of an image. This is a result of the individual pixels of an image being much sharper in contrast than is common in real image photographs. This can most easily be described by the one-dimensional Fourier Transform of a rectangular pulse. It is common knowledge that a “hard pulse” (i.e. a rectangular pulse) in the time domain yields a sinc (sin(x)/x) function in the frequency domain as seen below:

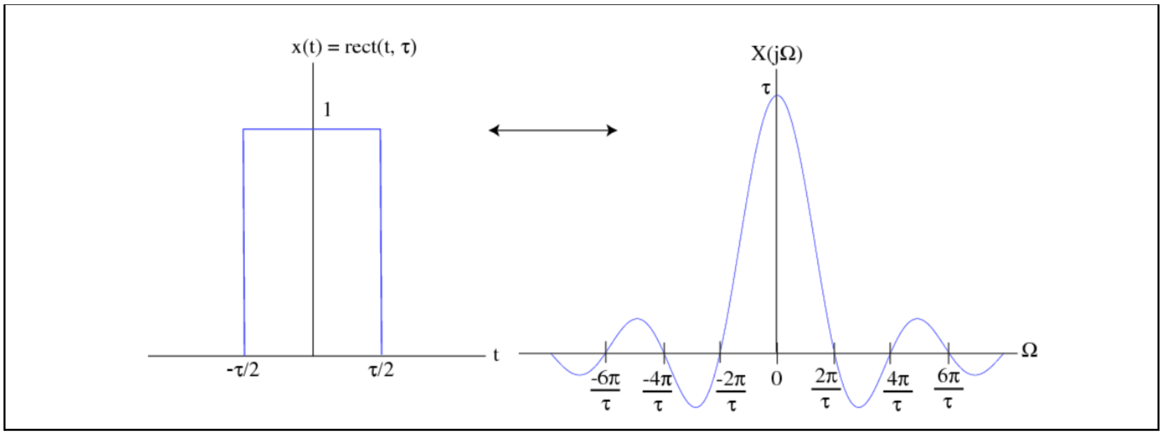


Figure 1: Fourier Transform of a Rectangular Pulse

In real images, the gradient of these pixels is gradual. This means that the peaks in the frequency domain are much higher than the sidelobes, and there is a much more consistent tapering. This is not the case in synthetic images, where synthetic images have much higher sidelobe “ringing” which affects ML algorithms, since these algorithms usually analyze images in the frequency domain. A pixel can be modeled as a 2-dimensional hard pulse, and the high-frequency information in the sinc function can fold into the low frequency band of neighboring pixels, yielding aliasing. In images, there is a fundamental tradeoff between these sidelobe levels and image quality, so we therefore must taper the sidelobes without either destroying them or overtapering, else we will yield a distorted or destroyed image.

1.2 Proposed Solution to Aliasing

The main tradeoff with improving anti-aliasing comes with image distortion. As you attenuate the higher-frequency information, you do irrecoverable damage to the individual signals, thus causing image distortion. A conventional FIR filter does not work well for this reason, as it destroys information beyond the passband. Therefore image processing makes use of "windowed" filters, which are filters which attenuate signals outside the passband without zeroing them out. .Therefore it is the goal of this subsystem to develop a "windowed" filter, which will attenuate high-frequency information without necessarily discarding it, so the image can retain its original qualities without much (if any) distortion. The proposed filter design will take the following form:

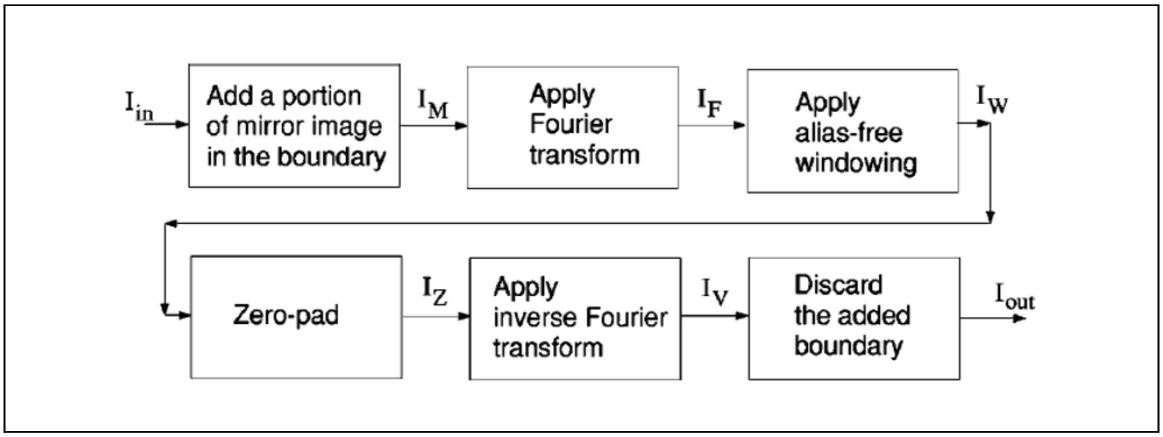

I_{in}
Add a portion of mirror image in the boundary
I_M
Apply Fourier transform
I_F
Apply alias-free windowing
I_W
Zero-pad
I_Z
Apply inverse Fourier transform
I_V
Discard the added boundary
I_{out}

Figure 2: Proposed Filter Design

1.3 ARL Spectral Filter

The proposed filter design from 1.2 was accomplished by using a unique filter design developed by Army Research Labs, because conventional spectral window filters such as Hamming windows cause too much sidelobe attenuation, which leads to image distortion above the threshold ML algorithms typically operate. The new proposed filter design takes the form of:

mask(i,j)=exp(-p*(z(i,j))^n);

Figure 3: Proposed ARL filter function

where p is the cutoff percent of pixel energy (0.95 according to ARL's proposition that the 95% energy level should be the minimum for aliased images), i and j are the pixel positions, z is the pixel frequency and n is the filter order. This yields a kernel with radial exponential attenuation beyond the passband p. The window works quite well with sinc functions due to the lack of attenuation at the center frequency, and the decimation of ultra-high frequency which isolates pixels better than a traditional window filter, while also attenuating the more central sidelobes without complete decimation (and thus avoiding image distortion) The kernel produced by figure 3 is present here:

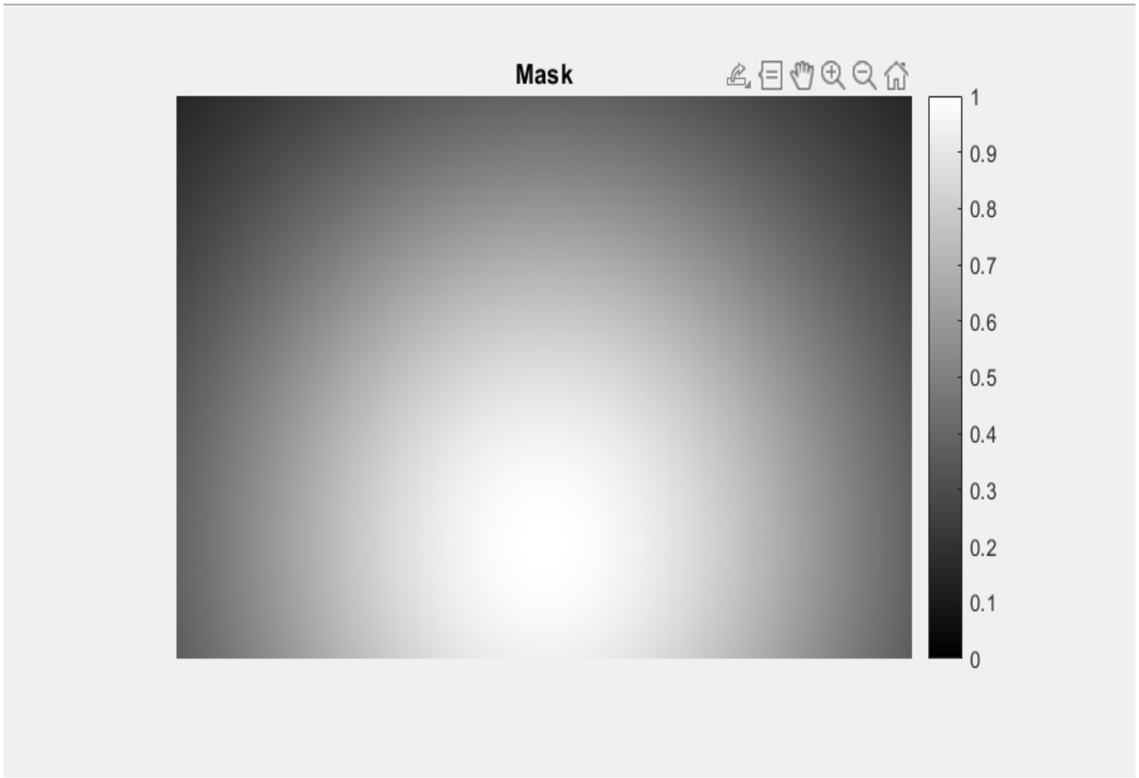


Figure 4: ARL Filter Mask Coefficient Map

2) Filter Design and Implementation

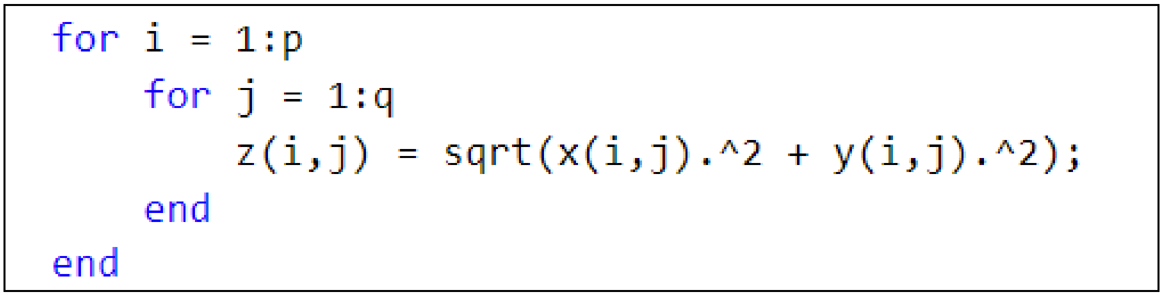
See Appendix A for the source code written to filter synthetic images. This program was written with MATLAB R2021b September 2021 and was last updated April 2022. This section will detail a step-by-step analysis of the program.

2.1 Image Reading and Filter Selection

The program begins by intaking an image and converting it to grayscale. Due to the nature of color images, a 2-D DFT cannot be conducted on a color image, since a color image is a 3-D matrix, where you have x, y, and color weighting. By making the image grayscale you eliminate the color weighting dimension, and can then conduct a 2-D FFT on the image. Also at this time the image is converted to a double, because the grayscale conversion converts to discrete grayscale values, so therefore it must be converted to a double to conduct the FFT. The select option determines which of the 5 filter masks to use (unfiltered, ARL, Gaussian, Hamming, and Blackman-Harris). If the unfiltered option is selected, the input image is returned.

2.2 Image Padding and FFT

Once the image has been read and converted to a double grayscale, the image is padded with one layer around the border. This padded layer essentially repeats the first and last row and column of the image. The purpose of this layer is to be the principal axes for x=0 and y=0 once the image has been fft-shifted. This padding protects the image from distortion caused by an FFT-shift and is necessary to properly recover the image from the frequency domain back to the time domain. After padding, the fft2() command built into MATLAB is called, which performs a discrete-time 2-D FFT on the image, transferring it from the time to the frequency domain. Then, an fft-shift is conducted, which essentially splits the image into 4 quadrants, and switches the top-right quadrant with the bottom-left quadrant, and the top-left quadrant with the bottom-right quadrant. Due to the nature of the DFT, an FFT-shift is necessary to make the (0,0) point in the center of the image. Now with a centered image in the frequency domain, we can call the freqspace() matlab command to extract the x- and y- spatial frequencies, and create a vector “z” which can be used to determine cutoff frequencies, where:

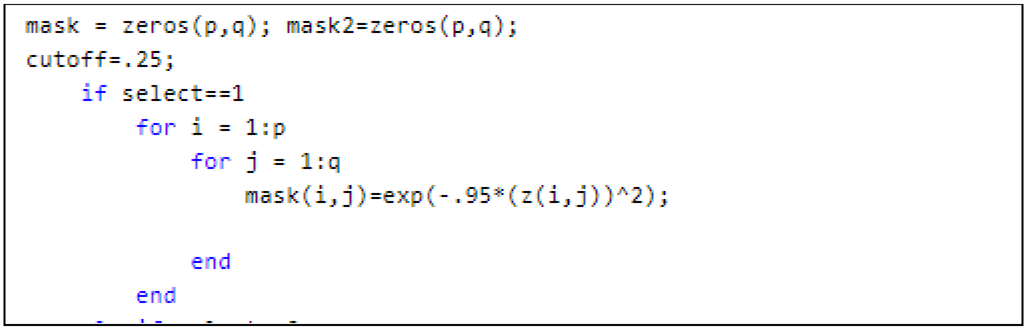


```
for i = 1:p
    for j = 1:q
        z(i,j) = sqrt(x(i,j).^2 + y(i,j).^2);
    end
end
```

Figure 5: Z frequency vector

2.3 Filtering

There are four possible filtering masks to select from. A mask is created with the zeros() command to yield a matrix equal in size to the image being processed. This mask is then filled in with the appropriate mask coefficients on a pixel-by-pixel basis. Due to the design of the filter, the several parameters can easily be manipulated to find an appropriate cutoff frequency and attenuation rate by simply changing one line of code. This allows ease of experimentation to ensure a wide range of ML algorithms can be improved with ease. The proposed ARL mask produced can be seen here:



```
mask = zeros(p,q); mask2=zeros(p,q);
cutoff=.25;
    if select==1
        for i = 1:p
            for j = 1:q
                mask(i,j)=exp(-.95*(z(i,j))^2);

            end
        end
```

Figure 6: ARL Filter Mask

2.4 Image Reconstruction

Once the mask has been constructed, it is overlaid on the original image by matrix multiplication. The image is then once again FFT-shifted (to return to its original form) and followed by an ifft() command in MATLAB to return the image to the time domain. The original padding is then removed, and the filtered image is both displayed and written as a .png image to the computer.

2.5 Changeable Parameters

Since ML algorithms come in all shapes and sizes, it is important to not only have a working filter, but it is also important for it to be easily tuned to optimize whatever algorithm it is being used to improve. As a result there are several parameters that can be changed to optimize results for specific cases. The following parameters are the easiest to change with only one line of code:

- Image Title: By making the image title a variable as opposed to a hard-coded file path, it is easy to filter large quantities of images when calling the filter.
- Cutoff frequency: As proposed in our sponsor's paper, the 95% energy level is used as the cutoff frequency, thus we are using p=.95. Through experimentation, n=2 was the filter order used on the images. It is encouraged for the user to use their optimal values for their particular dataset.
- Pass-band: The default pass-band is a high-pass filter, and can easily be changed to a low-pass filter by changing > to <, or a band pass by adding a logical AND to the mask conditional statement. Remember that an FFT-shift will change your frequency values to be different than typically expected, so make sure to look at the image matrix when selecting an appropriate frequency.

3) Validation

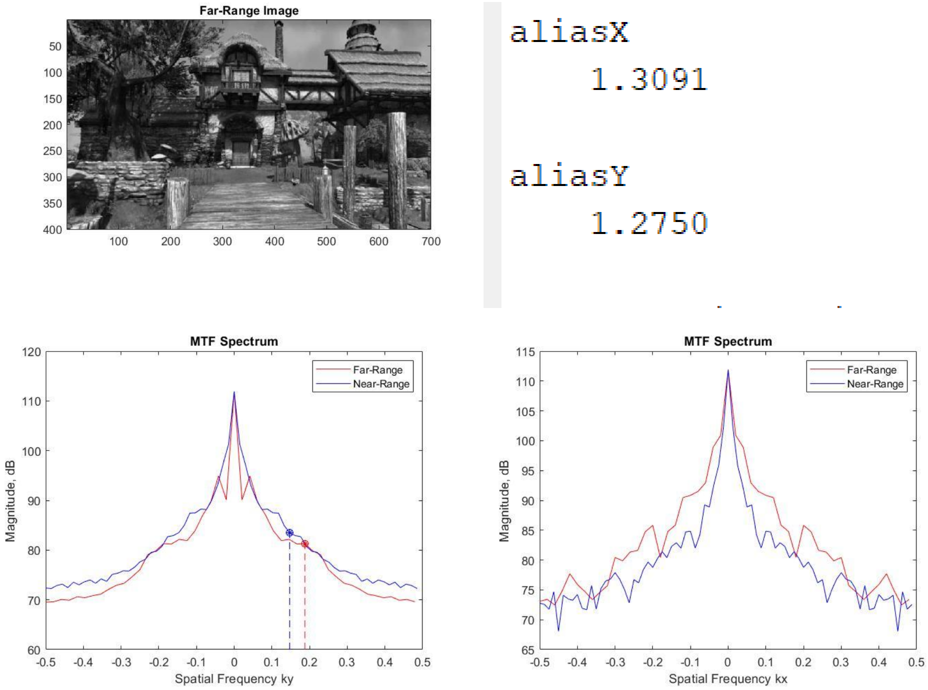
3.1 Validation Plan - Spring 2022

FSR Paragraph #	Test Name	Success Criteria	Methodology	Status
2.2.1	Antialisasing	The program must be able to import .png files and reduce aliasing noticeably (>10%).	Aliasing will be graded via ARL's alias detection program, with 1.00 being a perfectly unaliased image.	SUCCESSFUL
2.2.2	Filtering	The program must be able to take in any image file, grade the image for aliasing, execute the filter as necessary, then produce and regrade the output filtered image and store the image on the hard drive.	Execute the program on test images, and adjust the filter as desired for an appropriate cutoff frequency, due to the natural distortion-filtration tradeoff that exists in image processing	SUCCESSFUL
2.2.3	Unfiltered Recognition	ARL-proposed filter must increase object recognition distance compared to unfiltered images by at least 20%	Execute the testing on ARL-filtered images and compare to unfiltered images	SUCCESSFUL
2.2.4	Filtered Recognition	ARL-proposed filter must increase object recognition distance compared to other filtration types by at least 10%	Execute the testing on ARL-filtered images and compare to images filtered by other means	SUCCESSFUL
2.2.5	Accuracy & Improvement	ARL-proposed filter must not reduce validation accuracy by more than 5%	Train the ML algorithm with a mixture of real and synthetic filtered images and record the validation accuracy.	SUCCESSFUL

Table 1: Validation Timeline

3.2 Proof of Concept

Courtesy of our sponsor, Army Research Labs, we have been provided a “grading” program for how aliased an image is. This is conducted by finding the near-range and far-range of an image, and finding the 95% energy point in the spatial domain, then finding their ratio. A perfectly non-aliased image will achieve a ratio of 1 in the x and y directions, however most real images will be roughly 1 +/-0.15 with a 4K camera. I also altered the sponsor’s program to be easier to use, and to read out aliasing error and ratio. The sponsor’s program to grade images will *not* be provided in the report, however the results will be. Figure 6 will show the aliasing of an input image, Figure 7 will show the filtration of the image, and Figure 8 will show the regraded image, and the results will be commented on:



```
aliasX
     1.3091

aliasY
     1.2750
```

Figure 7: Initial Image and Grading

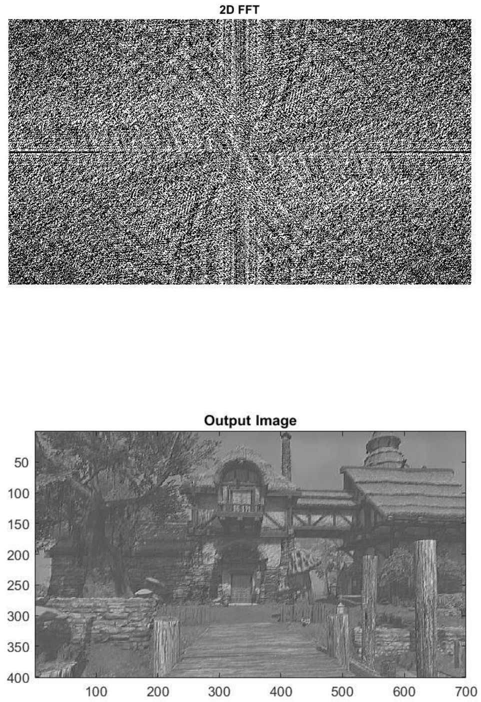
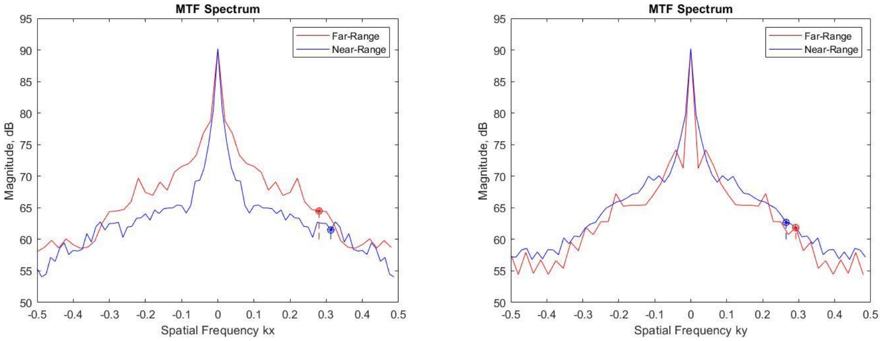
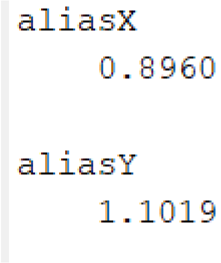


Figure 8: 2-D FFT of Initial Image and Filtered Output Image





```
aliasX
    0.8960

aliasY
    1.1019
```

Figure 9: Grading of Filtered Output Image

As you can see, figure 7 shows an initial image grading of roughly 1.31 and 1.28 aliasing ratios in the x and y directions, respectively. This indicates about 31% and 28% aliasing errors, and clearly have a high degree of spectral noise. Once filtered, the filtered output image was graded to 0.90 and 1.10 x and y aliasing ratios, respectively. This indicates about 10% error from a perfectly unaliased image. In essence, the regraded image is indistinguishable spectrally from a real image. Table 2 shows the results of tests with other images, and these images can be found in Appendix B. X-ratio improvement is found by abs((1-Xratio,before)-(1-Xratio,after)), and the same formula is used for Y-ratio improvement. Table 3 indicates if an image is located in the “real” spectral aliasing band.

Name	X-aliased before filtering	Y-aliased before filtering	X-aliased after filtering	Y-aliased after filtering	X-aliased improvement	Y-aliased improvement
Seyda Neen (Figures 6, 7, 8)	1.3091	1.2750	0.896	1.1019	20.51%	17.31%
Castle Volkihar	0.9143	1.6528	0.8800	0.9208	-3.43%	57.36%
Dragonsreach	0.6857	0.8500	0.7724	1.0054	8.67%	15.54%
Real Car Image	0.8495	0.9721	0.8901	0.9756	4.06%	0.35%

Table 2: Test Results

Name	X-aliased before?	Y-aliased before?	X-aliased after?	Y-aliased after?
Seyda Neen (Figures 6, 7, 8)	No	No	Yes	Yes
Castle Volkihar	Yes	No	Yes	Yes
Dragonsreach	No	Yes	Yes	Yes
Real Car Image	Yes	Yes	Yes	Yes

Table 3: Test Results Cont.

As table 3 shows, synthetic images occasionally can be spectrally similar to real images, and vice versa, depending on many factors (lighting, color intensity, motion blur, etc). However after this filter is applied, all images become indistinguishable from real ones and are well-anti-aliased (<0.85 ratio and 15%< error). All validation involving the ML algorithm will be addressed in the system validation section, as this section is done isolated from access to the ML algorithm.

4) Conclusion

In conclusion, I developed a spectral-based anti-aliasing filter for anti-aliasing synthetic images for ML applications. The primary point of validation was done via ARL's aliasing grading program, which I had altered a bit for ease of use. As a result, my filter made images spectrally indistinguishable from high-quality, real images. Four images of varying types from my testing were included in the appendix, however several more different images were tested during the development and validation processes. The algorithm achieved all goals it set out to accomplish as outlined by the sponsor in Figure 2. I also improved the sponsor's program to be much easier to use by image processing laymen, and made it read out important values that normally must be extracted from a graph.

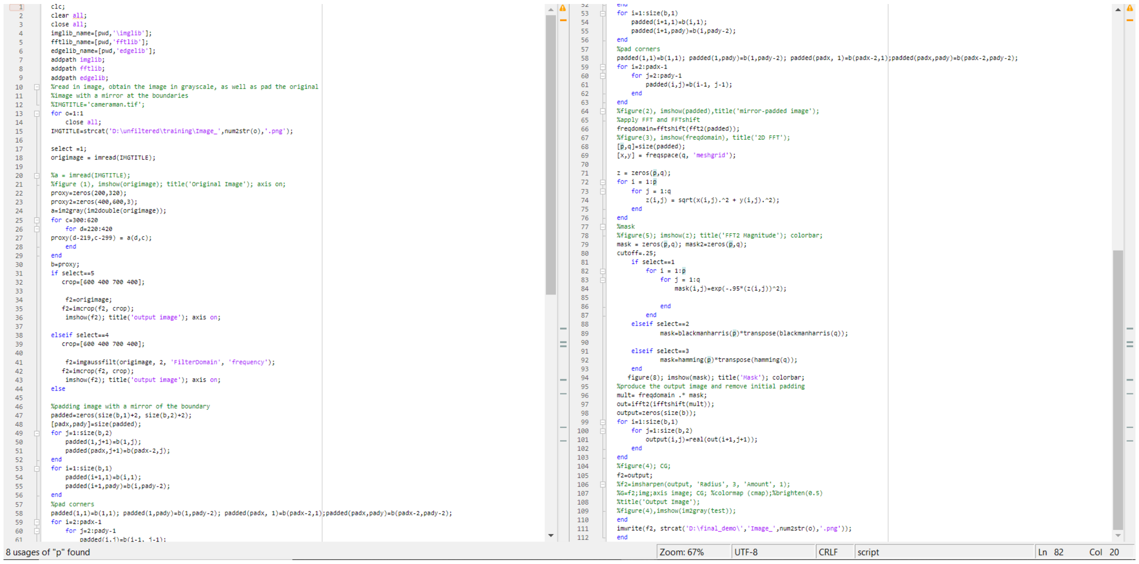
5) Future Development

While this program set out to accomplish its goal, there is always room for improvement. The following are possible improvements that could yield better results and are a good starting place for improving the program:

- Changing the filter cutoff order or energy level. For simplicity, order n=2 was used and several different integer orders were tested, and higher orders caused more distortion. Using non-integer filter orders may yield better results. Also, changing the required energy level to be lower and/or higher based on your ML's capabilities could yield better results. p=.95 was proposed by our sponsor and thus used, but for better algorithms a lower p could work better, and a higher p for worse algorithms.
- Color recovery. Due to the nature of the 2D FFT, color cannot be preserved in images after filtration. Luckily one significant finding of this study is that image color does not impact object recognition, however the program in its current state cannot recover the original color if the user is interested in maintaining the image color for other experiments.

Appendix A: Source Code

The following program was developed as the subsystem used in the project. Please email hfabbott2018@gmail.com for more information or questions.



```
clc;
clear all;
close all;
imglib_name=[pwd,'\imglib'];
fftlib_name=[pwd,'fftlib'];
edgelib_name=[pwd,'edgelib'];
addpath imglib;
addpath fftlib;
addpath edgelib;
%read in image, obtain the image in grayscale, as well as pad the original
%image with a mirror at the boundaries
%IMGTITLE='cameraman.tif';
for o=1:1
    close all;
IMGTITLE=strcat('D:\unfiltered\training\Image_',num2str(o),'.png');

select =1;
origimage = imread(IMGTITLE);

%a = imread(IMGTITLE);
%figure (1), imshow(origimage); title('Original Image'); axis on;
proxy=zeros(200,320);
proxy2=zeros(400,600,3);
a=im2gray(im2double(origimage));
for c=300:620
    for d=220:420
proxy(d-219,c-299) = a(d,c);
    end
end
b=proxy;
if select==5
   crop=[600 400 700 400];

    f2=origimage;
   f2=imcrop(f2, crop);
    imshow(f2); title('output image'); axis on;

elseif select==4
   crop=[600 400 700 400];

    f2=imgaussfilt(origimage, 2, 'FilterDomain', 'frequency');
   f2=imcrop(f2, crop);
    imshow(f2); title('output image'); axis on;
else

%padding image with a mirror of the boundary
padded=zeros(size(b,1)+2, size(b,2)+2);
[padx,pady]=size(padded);
for j=1:size(b,2)
    padded(1,j+1)=b(1,j);
    padded(padx,j+1)=b(padx-2,j);
end
for i=1:size(b,1)
    padded(i+1,1)=b(i,1);
    padded(i+1,pady)=b(i,pady-2);
end
%pad corners
padded(1,1)=b(1,1); padded(1,pady)=b(1,pady-2); padded(padx, 1)=b(padx-2,1);padded(padx,pady)=b(padx-2,pady-2);
for i=2:padx-1
    for j=2:pady-1
        padded(i,j)=b(i-1, j-1);
    end
end
%figure(2), imshow(padded),title('mirror-padded image');
%apply FFT and FFTshift
freqdomain=fftshift(fft2(padded));
%figure(3), imshow(freqdomain), title('2D FFT');
[p,q]=size(padded);
[x,y] = freqspace(q, 'meshgrid');

z = zeros(p,q);
for i = 1:p
    for j = 1:q
        z(i,j) = sqrt(x(i,j).^2 + y(i,j).^2);
    end
end
%mask
%figure(5); imshow(z); title('FFT2 Magnitude'); colorbar;
mask = zeros(p,q); mask2=zeros(p,q);
cutoff=.25;
    if select==1
        for i = 1:p
            for j = 1:q
                mask(i,j)=exp(-.95*(z(i,j))^2);

            end
        end
    elseif select==2
            mask=blackmanharris(p)*transpose(blackmanharris(q));

    elseif select==3
            mask=hamming(p)*transpose(hamming(q));
    end
   figure(8); imshow(mask); title('Mask'); colorbar;
%produce the output image and remove initial padding
mult= freqdomain .* mask;
out=ifft2(ifftshift(mult));
output=zeros(size(b));
for i=1:size(b,1)
    for j=1:size(b,2)
        output(i,j)=real(out(i+1,j+1));
    end
end
%figure(4); CG;
f2=output;
%f2=imsharpen(output, 'Radius', 3, 'Amount', 1);
%G=f2;img;axis image; CG; %colormap (cmap);%brighten(0.5)
%title('Output Image');
%figure(4),imshow(im2gray(test));
end
imwrite(f2, strcat('D:\final_demo\','Image_',num2str(o),'.png'));
end
```

Appendix B: Images Tested in Section 3.2

The following images were tested in section 3.2 as proof of concept for the designed filter in Appendix A.

“Seyda Neen”	
“Castle Volkihar”	
“Dragonsreach”	
“Real Car Image”	

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez

CONVOLUTIONAL NEURAL NETWORK SUBSYSTEM

REVISION – 01
24 April 2022

Change Record

Rev.	Date	Originator	Approvals	Description
00	12/04/2021	Alexia Perez		Initial Draft Release
01	04/24/2022	Alexia Perez		Final Draft

Table of Contents

1) Introduction

1.1 Executive Summary

The convolutional neural network subsystem is designed to classify input images into one of ten total different categories. First, the system will be trained and tested with real images to determine the accuracy of recognition. Then, synthetic images will be tested and filtered appropriately to reach a similar level of recognition accuracy. Finally, the filtered synthetic images will be used to train the model, and the reverse tests will be performed. The object of this subsystem is to test the quality of both the rendered images from Will Cooper’s sub-system, and the filter applied by Harry Abbott in order to ensure their structure is as similar to real images as possible.

1.2 Abbreviations

ML	Machine Learning
CNN	Convolutional Neural Network
HPRC	High-Performance Research Computing

1.3 Definitions

Machine Learning:
A class of artificial intelligence wherein an algorithm is trained on a set of data, learns from it, and can improve itself automatically. In the scope of this project, we are especially interested in machine learning algorithms which can identify objects in images.

Synthetic Image:
A computer-generated image.

Real Image:
An image captured by a real camera.

2) Operating Concept

2.1 Scope

This subsystem was initially developed using the Jupyter Notebook suite. The first CNN algorithm was written in one file, which could be run in order to train the model using a specified dataset of 64x64 images. Once this file was run and the model completed its training, two separate scripts could be run in order to test it. The second CNN version contained code updates that increased the training and validation accuracies of version 1.0. These changes included a new loss function, optimization function, and a custom built learning rate function, which progressively lowered the learning rate to address over/under fitting of the model. I investigated these parameters by generating training vs validation accuracy and training vs. validation loss plots, and adjusting each of the parameters to achieve the best convergence between the two lines in each plot.

The final model was developed in Google Colab, and its architecture differs considerably from that of models 1.0 and 2.0. The reason for changing the model so drastically was that at our mid-semester checkpoint, our synthetic images were still unrecognizable to our CNN, although synthetic images downloaded from the internet were recognized with an accuracy of over 75%. To address this, I built a model that would be trained with 244x244 images rather than 64x64, to allow for more image characteristics to be easily picked up by the algorithm. This new model follows the skeleton of a ResNet50 CNN, and in fact uses transfer learning to load the lower layers, freezes them, and only trains the top layers on our custom data, allowing it to achieve much higher accuracies in less time.

2.2 Constraints

CNN models 1.0 and 2.0 were developed and tested using a “super computer” consisting of 24 cores and 64 GB of RAM. This allowed us to train the models in a reasonable amount of time, but it unfortunately also meant that if this model were to be trained on an average machine, it would take hours or even days. For this reason, training and testing the model requires access to the HPRC computing resources, as well as access to the super computer’s Jupyter suite, for which a new request must be sent before each use.

Since the third model’s architecture is much more complex than that of the first and second models, I had to pay for a pro subscription to Google Collab that allowed me to connect to a GPU node for development and testing. Running this new model in the HPRC took around 6 hours, because the HPRC Jupiter Notebook runs on CPU by default, and I struggled to activate the available GPU nodes within it. Doing this enabled me to run the model in minutes, however, if I tried using this program in an average everyday laptop, it would crash or take days to complete.

Furthermore, the accuracy of the classification depends entirely on the input images used. For this reason, if the images provided to me are not textured or are of very low quality, the algorithm will struggle to recognize the objects represented in them.

2.3 User Interface

All aspects of this subsystem were developed and tested using an HPRC machine (Grace cluster) and Google Colab. The cluster consisted of 24 cores and 64 GB of RAM, and allowed CNN algorithm Ia 1.0 and 2.0 to complete its training in no more than ten minutes. The third model was developed and tested using google colab, as it required the use of a GPU or TPU node. The user must therefore have access to a pro subscription in order to run this model.

3) Equipment

3.1 Software

Both the CNN model and the training scripts *can* be run in any machine with access to Jupyter Notebook, however the performance of each program may be directly affected by the available resources the machine running them has. I used my access to the HPRC Grace cluster and wrote and tested all of my Python scripts in that machine, and Google Colab, which increased performance considerably.

Any user that wishes to access Jupyter Notebook can do so by installing the program from the official website or by using their computer's command line. I chose to use the GUI, as it was already installed in the supercomputer and Google Colab notebooks use the same format.

3.2 Hardware

I requested access to the HPRC computing resources available to all Texas A&M University students, and was able to remotely access these resources to develop and test every part of my sub-system. I used the Grace cluster. Each time I submit an access ticket for my machine, I am able to choose how many cores and GB of RAM I would like to have available, and I have found that 24 and 64 respectively allow me to complete tasks in reasonable times, especially compared to my personal laptop, which has the Intel i7 Processor with 4 cores and 16 GB of RAM. During the second half of the semester, I switched to working on Google colab, which provided me with GPU node access and high-speed RAM.

4) Data

4.1 Real Images

First, real images were used to train and test the CNN algorithm. The images were part of a dataset commonly known as the ImageNet dataset. This dataset includes millions of real images that can be classified into over 1000 different categories. The final model reached an accuracy of 80.53% after being trained and tested using these images.

Figure 1: Real Image of Car

4.2 Rendered Synthetic Images

Once the algorithm had been trained and tested with real images, I then downloaded the synthetically generated images that Will Cooper rendered using Blender. All images were of cars, from different angles and distances, and my first goal was to determine whether the CNN would recognize the object at all, and if so, how far away from the camera could the object be placed before the CNN stopped recognizing it. We would then use the results from these tests to identify which properties could be changed in the synthetic images, both before and during the filtering process, in order to make them more realistic.

4.3 CNN Algorithm Code

4.3.1 Pre-requirements

Before running the test scripts, the CNN model must be trained appropriately. To train the model, the user must first have access to Jupyter Notebook, as all files related to this subsystem are “ipynb” files. Once the Jupyter Notebook has been installed correctly, the user must ensure that they download the appropriate packages using the “pip” command so that all necessary imports may be accessed when running the code.

4.3.2 The Training and Testing Datasets

As I mentioned earlier in the report, 4 classes from the ImageNet dataset will be used to train our CNN with real images. Then, one of the classes (vehicles) will be replaced by the synthetic images.

4.3.3 Pre-Processing Phase

The first thing we must do before we begin training the algorithm is ensure that all images used are in some way “normalized.” To do this, we rescale our images by using “mean-normalization.”

Usually, an image’s pixel values are of a type known as “uint8” which means that they are integers in the range from 0 to 255. The process of mean-normalization will re-scale these values to instead be either a 0 or a 1.

Additionally, in order to be able to determine whether an image belongs to a specific category, we must first generate a probability index for it. This can be done by making a separate “prediction” for each of our 5 categories. This process is known as “binary encoding.”

4.3.4 Machine Learning Model

This could be considered the heart of the CNN algorithm, as this section defines the model itself. Our CNN model follows the same structure as the popular ResNet50 model, and actually loads the weights of a pre-trained ResNet50 into its lower layers, then freezes these layers to avoid overwriting them.

Lastly, the model trains the top layers using the custom library that I built, with MeanLogarithmicNormalization as the loss function, and using a decaying learning rate to avoid over/under fitting. These parameters were determined by observing the changes in the shapes of the plotted training and validation loss as well as the training and validation accuracy.

4.3.5 Model Training

So far, our model has used simple mathematical operations to generate a prediction about an image. In doing this, complex features that an average computer wouldn't usually recognize were extracted and turned into simple numbers that computers do understand.

The next step is to find a few parameters that will increase the accuracy of our CNN's recognition capabilities. These parameters are called "weights" and in order to find ideal ones, we must perform a series of mathematical operations. These mathematical operations will help us find a measure of error between the current result of our prediction and our desired one. If we then average the errors that the model made on a set of images, we can determine how "good" the current weights are and make changes.

As I briefly mentioned in the previous section, I used the Adam optimization algorithm to change these weights during every iteration until the CNN reached an appropriate level of accuracy.

Initially, I set the number of "epochs" to 20 and the batch size to 100. These values were later changed to 10 and 100 in order to increase the accuracy and the performance of the CNN. This means that while it is being trained, the CNN will split every image set into chunks of 100, and it will train on 10 iterations.

4.3.6 Testing the CNN

Finally, once the CNN has been trained and has reached a reasonable accuracy level, we can test it by using a testing script. I wrote two separate ones and they both have the same functionality, the only difference being that one of them tests 3 random images and the other one tests all images from the user's chosen directory.

All code for training and testing can be found in our team's GitHub repository, which I created myself and kept up to date throughout the semester.

5) Analysis

5.1 Functionality

According to my individual goals for the semester, my subsystem was fully completed and performed as expected. The first requirement was to simply train a model with real images, which I was able to do with accuracies much higher than our initial goal of 70%.

Additionally, I tested my subsystem with real and synthetic images. I was able to determine that the first issue to tackle in order to accurately train ML models with synthetic images was improving the realisticness of the objects represented by using more textures and better lighting, and Will Cooper used this knowledge to generate better quality synthetic images. After weeks of testing new batches with different characteristics, we were able to finally create images that would be recognized with over 90% accuracy on the all-real CNN, which we then filtered and used for the synthetic trained CNN.

5.2 Future Implementation

In terms of our current goals and objectives, there is not much more to add to this model to improve it. Its accuracy on all-real training sets was over 90%, recognizing 100% of the 500 real cars that we used to test it. The synthetic batches performed very well too, reaching accuracies of 80.53% and 87.8% when unfiltered vs. filtered.

A new goal that could be pursued by other teams working on a project like this could be to build a different type of ML, and repeat the training and testing process to determine the improvement level when using filter vs. unfiltered images. These results could then be compared to the results our team got, which would give a more accurate representation of how well this technique of filtering synthetic data works.

6) Validation Plan - Spring 2022

FSR Paragraph #	Test Name	Success Criteria	Methodology	Status
2.3.1	Training of CNN	A Convolutional Neural Network should yield a validation accuracy of 70% or above when trained with synthetic images	Build a CNN model and train it using synthetic vehicles (in addition to four other real classes) and plot training and validation accuracies & losses	SUCCESSFUL
2.3.2	Testing Synthetic Images	A pre-trained CNN should be able to identify the synthetic vehicles with an accuracy of at least 70%.	Execute the program using synthetic image library with and without filtering, and the results after filtering should recognize vehicles for at least 70% of the tested library	SUCCESSFUL
2.3.3	Testing Real Images	A pre-trained CNN should be able to identify real vehicles with an accuracy of at least 70% when trained with synthetic images	Execute the program using real image library after training with filtered and unfiltered synthetic images, and the model trained with filtered synthetic cars should recognize 70% or more of the total real cars tested	SUCCESSFUL

Spectral Filtering of Synthetic Images for Machine Learning Applications

Alexia Perez
Harry Abbott
Will Cooper

SYSTEM REPORT

REVISION – 00
24 April 2022

SYSTEM REPORT

SPECTRAL FILTERING OF SYNTHETIC IMAGING FOR MACHINE LEARNING APPLICATIONS

TEAM 08

APPROVED BY:

Project Leader	Date

Prof. Kalafatis	Date

T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
00	04/24/2022	Will Cooper		Final Draft

Table of Contents

List of Figures

1) Objective

1.1 Problem Overview

Machine learning algorithms are a powerful technology that can solve a wide variety of problems. However, ML algorithms are hindered by their dependence on training data to learn their task properly. Faulty training data can result in a failed algorithm, and sometimes accessing necessary training data is not feasible for economic, technological, or environmental reasons. Ultimately, our goal is to develop the means to produce large synthetic image sets that accurately match the spectral properties of real images and can effectively train ML algorithms. Creating such a generator would not only save time and money, but allow society to capture data sets that were once impractical to obtain.

1.2 System Overview

In order to solve this problem, the system was divided into the following three stages, each with their own subset of functional requirements: Synthetic Image Generation, Spectral Filtering, and Machine Learning Validation.

- **Synthetic Image Generation**
  Objective: Quickly and autonomously generate any desired number of randomized synthetic images given a library of 3D models and certain parameters from the user.

- **Spectral Filtering**
  Objective: Develop a filter for the aforementioned synthetic images such that the resulting images mimic the spectral properties of real, camera-captured photographs.

- **Machine Learning Validation**
  Objective: Construct a convolutional neural network (CNN) to help verify whether the filtered images are practically indistinguishable from real photos and explore the capability of these images to train image-recognition machine learning algorithms.

2) Methodology

2.1 Validation Results

FSR Paragraph #	Test Name	Success Criteria	Methodology	Results
2.1.1	System Input	The system must be able to autonomously import .obj/.mtl files cohesively, without error or data loss, given a directory file path.	Run the synthetic image generation subsystem; pause execution after IMPORT_OBJECT() function. Verify if object appears with matching textures/shaders. Also check for any data loss during importation.	Successful, the number of vertices was the same after importation; the files were imported solely from running the python script.
2.1.2	System Output	The system must be able to autonomously export .png images to a specified directory given a file path.	Run the image generation program entirely for one cycle. Verify if new .png image appears in desired directory.	Successful, image appeared in the desired library.
2.1.3	Image Parameters	The system should easily allow the user to adjust different parameters such as camera height, angle, maximum vehicle distance, etc.	Run the program with these different parameters, changing only one at a time, and verify each change results in its intended alteration of the exported image.	User defined constants were added to the script to let the user adjust all randomized aspects of the image, all adjusted the image correspondingly.
2.1.4	Resolution	The system must be able to produce any resolution/dimension of image as desired by the user (up to 8k).	Test the image generation subsystem with four different resolutions and verify exported images follow suit.	Multiple sizes were rendered including 256x256, ½ HD, full HD, 4k, and 8k resolutions
2.1.5	Internal Aliasing	The system must be able to adjust internal aliasing parameters such that the user can control how similar the spectral properties of the synthetic image match that of a real image.	Produce two identical images with one image having aliasing and one without. Run these two images through the aliasing test program to verify their differences.	Successful, our CNN results verify there is a significant difference between filtered and unfiltered images.
2.1.6	Render Time	The system must be able to produce an HD image within a reasonable amount of time given common computer hardware.	Using an intel core i7 processor and an Nvidia 1070 graphics card, the system should produce a single image in less than a minute.	Using the aforementioned hardware, each HD image was rendered in approximately 10 seconds.
2.2.1	Antialisasing	The program must be able to import .png files and reduce aliasing noticeably (>10%).	Aliasing will be graded via ARL's alias detection program, with 1.00 being a perfectly unaliased image.	Aliasing typically reduced by 20%
2.2.2	Filtering	The program must be able to take in any image file, grade the image for aliasing, execute the filter as necessary, then produce and regrade the output filtered image and store the image on the hard drive.	Execute the program on test images, and adjust the filter as desired for an appropriate cutoff frequency, due to the natural distortion-filtration tradeoff that exists in image processing	Successful, all subsets performed as needed.

2.2.3	Unfiltered Recognition	ARL-proposed filter must increase object recognition distance compared to unfiltered images by at least 20%	Execute the testing on ARL-filtered images and compare to unfiltered images	Increased recognition distance by 300%
2.2.4	Filtered Recognition	ARL-proposed filter must increase object recognition distance compared to other filtration types by at least 10%	Execute the testing on ARL-filtered images and compare to images filtered by other means	Increased recognition distance by 90% compared to second strongest filter
2.2.5	Accuracy & Improvement	ARL-proposed filter must not reduce validation accuracy by more than 5%	Train the ML algorithm with a mixture of real and synthetic filtered images and record the validation accuracy.	Validation accuracy improved by about 8%
2.3.1	Training of CNN	A Convolutional Neural Network should yield a validation accuracy of 70% or above when trained with synthetic images	Build a CNN model and train it using synthetic vehicles (in addition to four other real classes) and plot training and validation accuracies & losses	Results Not Provided
2.3.2	Testing Synthetic Images	A pre-trained CNN should be able to identify the synthetic vehicles with an accuracy of at least 70%.	Execute the program using synthetic image library with and without filtering, and the results after filtering should recognize vehicles for at least 70% of the tested library	Results Not Provided
2.3.3	Testing Real Images	A pre-trained CNN should be able to identify real vehicles with an accuracy of at least 70% when trained with synthetic images	Execute the program using real image library after training with filtered and unfiltered synthetic images, and the model trained with filtered synthetic cars should recognize 70% or more of the total real cars tested	Results Not Provided
N/A	Full System Demo	Similar results of the CNN will be yielded when tested with both a training library of all real images and a training library of a mix of real and synthetic images	Each subsystem must be able to demonstrate its required function or present data as proof of performance.	Successful, all systems performed correctly.

2.1 Dynamic Filter Range Test

One of the primary tests to investigate the importance of spectral properties of images was to test the unfiltered images with several different filter masks and compare object recognition distances. This resulted in 5 different filtration methods: unfiltered as a baseline, gaussian filter to see the impact of spectral noise and investigate the impact of color in object recognition, the default Blackman-Harris blender filter to see if time-domain anti-aliasing would have an impact, a Hamming window to test distortion of conventional spectral window filters, and finally the ARL spectral filter specially made for this course. The method to test involved cars being generated 10-100 meters, and the furthest object recognition distance was recorded. The ARL filter would be considered successful if it outperformed the unfiltered image by at least 20%, and the next highest filter type by at least 10%. The results are displayed here:

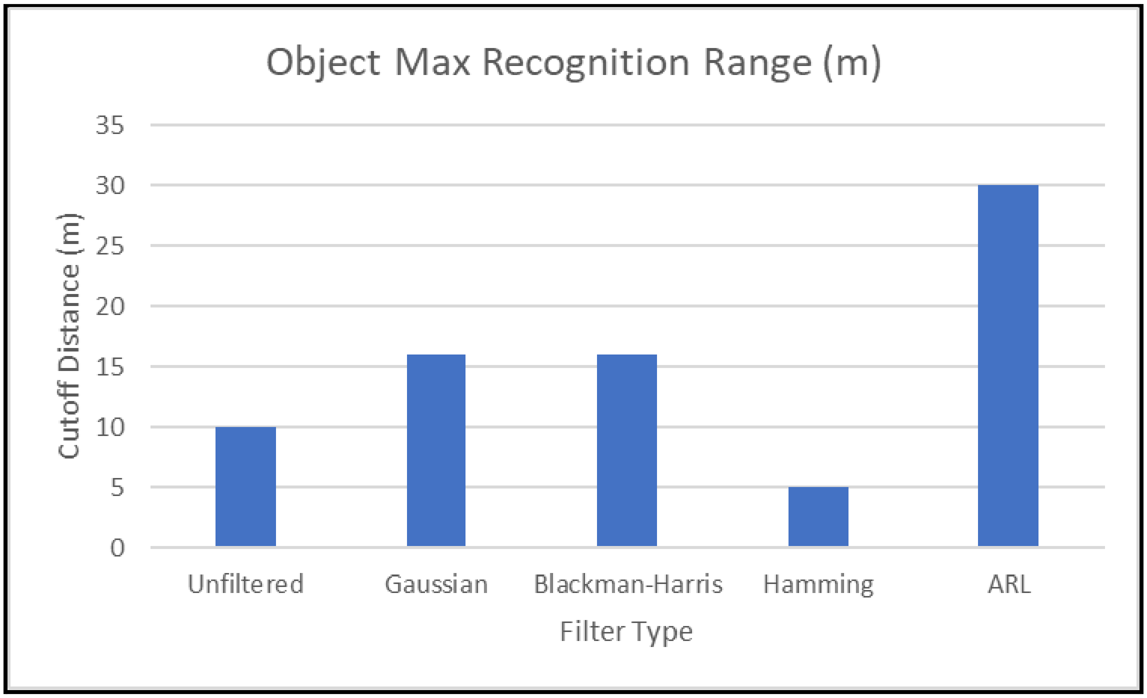


Figure 1: Testing Different Filter Types on Synthetic Images

As the graph shows, the ARL filter outperformed the second highest filtration type by about 100% (30m vs 16m), and the unfiltered by 300% (30m vs 10m). This extreme level of improvement was somewhat surprising, and also tells us a lot about the impact of color and time-domain anti-aliasing, which will be mentioned in the discussion section.

2.2 Real Image Training Set Tests

In order to train our CNN, we had to generate two custom train and validation libraries. For the all-real training and validation libraries, we selected 5 generic classes from the ImageNet dataset and extracted them using two custom python scripts. We used various categories of real vehicles to generate the “vehicle” class, since our synthetic image generation focused primarily on vehicles. The library had a total of 5000 images, 1500 being of vehicles. The validation library was generated following the same procedure, and contained a total of 1500 images between the 5 classes. We then used the dataset containing the training and validation images to train the top layers of a Convolutional Neural Network. The first layers inherited weights from the ResNet50 model, which allowed the training to complete in as fast as 6 minutes, yielding a validation accuracy of 80.53%. After testing 500 real images, we were able to confirm that our all-real model could successfully classify 100% of the tested real images, so we moved on to training it with synthetic images to determine the quality of our filtering technique. The results of these tests can be seen below in Figure 2.

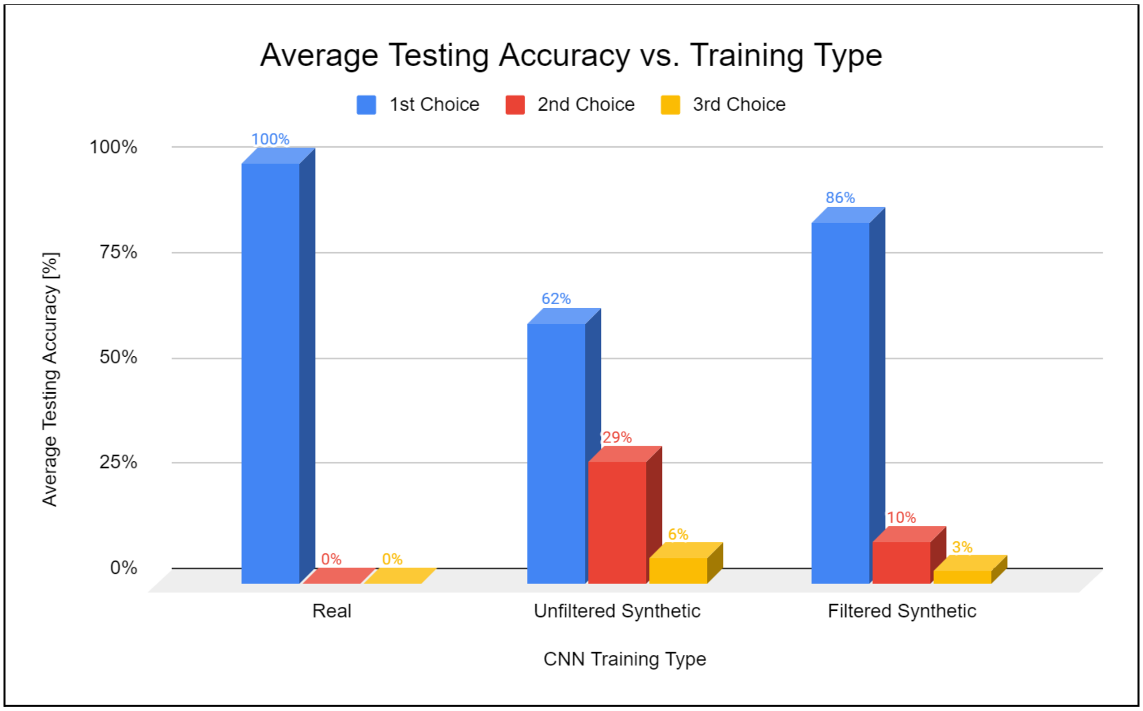


Figure 2: Average Recognition Accuracy Including Top Three Choices

2.3 Synthetic Image Training Set Tests

Two different versions of the synthetic training set were built. One of them using unfiltered images, and the other one using the same synthetic images after the ARL filter was applied. To generate these datasets, we simply made copies of the all-real library, and substituted the vehicle class folder to instead include the synthetic images. We did the same for the validation images in the set, and then used the new libraries to train the model. The train time was longer for both the unfiltered and filtered versions in comparison to the all-real dataset, taking around 20 minutes to complete. We performed the same 500 real image tests on these two new versions of the model, and observed great improvement in recognition accuracy when using the ARL filter. The unfiltered images were more often mis-classified, and in cases where the classification was correct, the confidence was much lower than when we used the filtered data. These confidence disparities can be seen below in Figure 3.

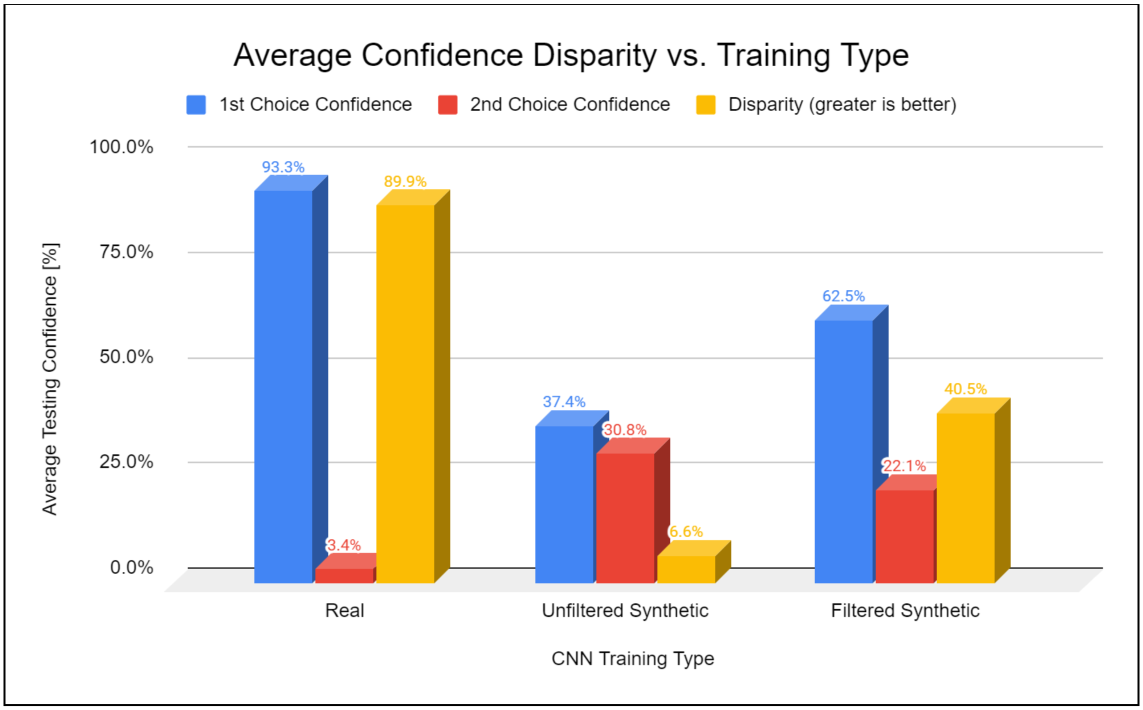


Figure 3: Average Confidence Disparity Between First and Second Choice

3) Analysis & Conclusions

3.1 Discussion

During this research, several very interesting conclusions were made. The dynamic filter range test showed that color had no discernible impact on object recognition, which was counter to our initial hypothesis. We initially believed converting to grayscale would negatively impact object recognition, however that was not the case. In addition, the filter range test verified our hypothesis that spectral properties of images were paramount to object recognition. We also learned that time-domain anti-aliasing via the default Blender filter has a limited effect on object recognition improvement. This is incredibly important, since most conventional image and video anti-aliasing is done in the time domain, since the human eye does not distinguish the difference between spectral and time-domain anti-aliasing, and time-domain anti-aliasing can be conducted using much less memory than spectral domain anti-aliasing. We found the distinction between time and frequency domain anti-aliasing to be surprising, since to the human eye there is little difference, but to a ML algorithm it clearly is significant.

Once the range test was finished, the ARL filter was then used to filter synthetic images for a training and validation library for the CNN. This resulted in an improvement in validation confidence by about 8% when compared to unfiltered images. When tested on real cars, the recognition of the CNN tested with filtered cars was about 25% higher than unfiltered cars, which is not surprising given the results of the range test, and only validated those conclusions further.

3.2 Conclusions

Our team designed a 3-step system that met all validation criteria. The first step involved synthetic image generation that was able to change distance, orientation, model type, and able to dispense as many images as the user wanted. These images then were fed into the spectral filter, which was able to make them spectrally indistinguishable from real images given the same resolution. This was verified in the third step which involved testing 500 real images on 3 different versions of a CNN. All validation criteria set forth by ARL and ourselves were met.

3.3 Future Implementations

When building this project, we never felt it was necessary to connect all the subsystems into one program as we felt it was good for our intents and purposes to keep each portion discrete. However, if desired, each section could be stitched together using a common software language such that there is no required user interaction between each subsystem. In other words, the program would run cohesively and the user would define parameters and provide 3D models, and have the program produce filtered images and results with the click of one button. Combining all three systems however comes at a tradeoff, as it becomes more difficult to do small-scale changes such as changing an object or model or filtration method.

Furthermore, to better match real images, more detailed backgrounds, objects, and textures could be implemented so that the images are both more diverse and more realistic. Additional ML algorithms could also be brought in to verify our results are consistent across ML types. One example would be to test the images with object detection or localization algorithms to see if similar results are produced.

Another future use of this program could be to generate synthetic images of an object that may not exist yet, such as a new style of airplane or vehicle, and then use this new object during the training of the ML, allowing it to then be able to identify something so rare that insufficient real images could be provided. This program, in essence, would allow a user to now have access to an unlimited amount of image data for object recognition software.