The Effect of Training Published Computer Vision Models with Unreal Engine Synthetic Images

An Introduction to the Synergy of Graphic Rendering Software and Machine Learning

John W. Belanger Smutny, Software Engineer, Northrop Grumman Corporation
Bradley Department of Electrical and Computer Engineering, Virginia Tech

***Abstract*— All machine learning (ML) models require large amounts of varied training data to understand the real world. Gathering enough data can be a daunting task if you don't already have it or your task isn't funded well enough to gather it (manually or via web-scraping). The latter of which opens the door to trademark and privacy concerns. Is it possible to generate your own images for your use case in a controlled scalable environment? This project explores this possibility through five experiments to better understand the domain gap between real images and synthetic images created with the Unreal Engine (UE) photo-realistic rendering software. The used synthetic images are collected for two binary classification use cases: First, in a domestic setting (can the model identify if an image is of a cat or a dog). Second, to be used for quality assurance in industrial settings (can the model identify if a metal weld is defective).**

The large VGG16 model showed slight improvements to model performance when synthetic images supplemented real images (real-to-synthetic ratio that is greater than or equal to 50:50). In addition, using synthetic padding to rebalance an imbalance classification dataset is possible, but should be done with caution based on the quality of synthetic images. The on-chip focused MobileNetV3-small architecture struggled to converge during learning for most experiments. Overall, the results from both models provide context to how different types of models react to synthetic images.

***Index Terms*— Computer Vision, Unreal Engine, Real Synthetic Ratio, Synthetic Utilization, Welding.**

1.I. Introduction & Problem Statement

THIS paper investigates how synthetically generated images from the Unreal Engine can be used to improve CV model performance. This paper explores three questions; first, investigate how including synthetically generated images in a Computer Vision (CV) model's training set affects the end model performance (accuracy, loss, and AUCROC). Second, see if making more synthetic images available to be sampled changes model performance metrics. Third, can synthetic images rebalance an imbalanced classification dataset by padding the minority class. In addition, this project provides context on how practical it is to develop one's own synthetic images using Unreal Engine (UE), the potential business cases for it, and possible restrictions.

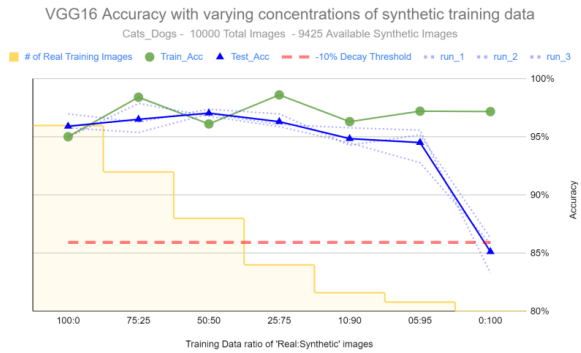


Figure 1.1: Example results from Experiment 1 showing how a transfer learning binary classifier model's accuracy changes as more of the training data is replaced with synthetic images.

To investigate these goals, sets of synthetic images are created for each case: 1) Cat vs Dog and 2) Metal Weld Defect Detection. Free to use .fbx files are used to create the Cat and Dog images, while the author applies their own changes to a purchased metal welding UE Material. Resulting in unique UE Material Instances for three weld defects: porosity, cracks, and incomplete welds. Two differently sized transfer learning models are used for these use cases: VGG16 and MobileNetV3-small. The variation in models used and use cases allows this research to investigate the domain gap between real and synthetic images, as well as how differently sized models are affected by them.

Models	VGG16, MobileNetV3	
Binary Classification use Cases	{Cat, Dog}, {*Defective* weld, *Good* weld}	
Experiments	How does the number of synthetic images in a model's training set affect binary classifier performance?	Can synthetic images be used to rebalance an imbalance classifier dataset?

Figure 1.2: General outline of methods used in this research.

A. Research Outline

This paper covers all design decisions, methods, results, and interpretations on how training CV models on synthetic images can affect model key performance metrics for binary classification tasks. First, the paper covers two concepts relevant to the paper before describing the current literature related to training CV models off of synthetic images. Next, the paper describes the methodology used by the researcher, specifically describing the models used, architecture changes, image augmentations during training, dataset formation, and an outline of performed experiments. Followed by the graphical results, interpretations of said results and final conclusions.

Each section contains high level information meant to be an overview of what is done to understand the next section. At the end of each topic, the paper points the reader to a section in the appendix with more information on that topic. The appendix's "For More Information" section expands upon the high-level knowledge given in the body of the paper. This is so the reader can focus on what they care about the most.

A. Key Terms and Concepts

The findings in this report rely on several background terminology and concepts. The most important terms being *CV model generalization, latent variation* in a set of images, and how metal MIG and TIG welding works. This paper aims to introduce two new concepts: *Real:Synthetic Ratio (RSR)* and *Synthetic Utilization Rate (SUR)*. RSR is the ratio of real images to synthetic images used in a model's training set. SUR is the percentage of synthetic data sampled to create a dataset versus what was available to be sampled.

Introduced in this Research	Real:Synthetic Ratio (RSR) Synthetic Utilization Rate (SUR)
Computer Vision	Model Generalization Latent Variation Transfer learning
UE Graphic Design	Material Texture map

Figure 1.3: Key terms to know to understand this paper's results.

For the definitions of each term and concept, please see the appendix the *A.I. Key Terms and Concepts* section "For More Information".

B. Related Works

Using synthetic images to train ML applications is a relevant pursuit as applications become more diverse and as abundant real data is harder to find. In 2022, Keith Man and Javaan Chahl published a technical summary outlining various types of synthetic image creation, how synthetic data is generated, for what purposes, and provided an outline of how to measure the quality of Generative Adversarial Networks (GANs) images. The common measure of GAN quality is an FID Score (Frechet Inception Distance score). An FID score measures how far apart real and synthetic image feature spaces are from one another, where a score of 0.0 means the pictures are nearly identical. The survey documents that some of the factors to successfully using synthetic images:

1. How the experiment datasets are formed
2. How variation is there in the generated images
3. How to make the images as real as possible. Reducing the domain gap between real and synthetic is paramount [Man22].

The FID scores of synthetic images created for this report were not calculated but could be good for future research.

i. Synthetic Data Substitution

Two papers specifically are used as the basis for this research. Papers published by an anonymous ECCV submission source [ECCV18] and Johan Thornström attempt to use images from the UE and (GANs) to train Convolutional Neural Networks (CNN) for classification. Both papers use the UE to render images with different variations for different binary classification use cases (airplane or bicycle and soldier or civilian respectively) [ECCV18] [Thornström19].

The ECCV paper's experiment trains the AlexNet CV model with different mixtures of real and synthetic images. Images of airplanes and bicycles against different UE full environments, backgrounds, lighting sources, and textures to create image variation. In addition, their experiment involved using GAN models (MrGAN or CycleGAN) to post-process the real and synthetically generated images to look "more real". The GANs were trained to improve the realism features in the image's feature vectors to mostly positive results. Their results indicate that mixing real and synthetic data at a ratio of 1:1 with 2D augmentations yields classifiers with the highest accuracy. In addition, using the GANs to further refine all training images improved test accuracy. The key point

being that adding any amount of synthetic data to a training set of real images improves model performance by introducing new feature variance [ECCV18]. It is believed that this submission is made by Nguyen, Tan, Chen, H, et al of Rice University as the paper was found on their personal blog, but the true author cannot be verified at this time.

Thornström's paper repeatedly trains a ResNet50 model with either a real image dataset or different variations of only synthetic images. Their paper "Domain Adaptation of Unreal Images for Image Classification" uses a python package, UnrealCV, to extract segmented images from Unreal Engine and feed them into a multi-staged GAN to produce unique images used to classify humans as soldiers or civilians. The model struggled to learn due to GAN images not producing enough image variation. After implementing many methods to stabilize the GANs, they ultimately suffered from domain divergence [Thornström19].

Additionally, the paper "*How Low Can you Go?*" published by Zoe Gastelum and Timothy Shead of Sandia National Laboratories, used the SideFx Houdini software to train CV models for industrial safety and compliance (a similar field to the second use case covered in this paper, weld defect detection). Like other papers covered in this section, Gastelum's experiments also varied the ratio of real and synthetic images in the training set, but the training set did not have a lot of images available to them. Only 160 real images were available to them for training, validation, and test sets. When trained on 1000 synthetic images and tested on the 160 real images available, their model classifiers struggled to surpass 60% accuracy and severed from overfitting after only a few epochs of training [Gastelum20].

ii. Rebalancing Techniques

A paper by Gary Weiss, et al. provides the basis for the third goal of this paper; exploring how synthetic images can rebalance an imbalanced dataset. Weiss' research studies how different techniques can mitigate the common pitfalls of training classifiers on a dataset unequal number of samples for each class. They apply oversampling, undersampling, and applying class weighted Cost-Learning techniques to differently sized datasets. In the end, they conclude that for datasets with fewer than 10,000 samples, oversampling (the act of duplicating the minority class samples) provides the greatest benefit since Cost-Learning cannot accurately approximate the value of classes with limited data. Meanwhile for datasets with greater than 10,000 samples, Cost-Learning provides the greatest benefit since it can appropriately approximate class value. The paper mentions synthetic padding, but it is not included in their tests. [Weiss07].

iii. Welding 101

This section of the paper on welding techniques and best practices is provided by professional welder (and volunteer instructor) Brandon Padayao. They provided their experience while instructing the Metal 3 course offered by the OpenWorks makerspace in Baltimore, Maryland.

The second use case in the paper focuses on training a classifier to determine if a metal weld is defective or not. This requires an understanding of what welding is and how defects can occur. In general, metal weld defects occur when one of several factors are mismanaged. Including but not limited to; the operator's speed, set voltage, set filler feed rate, differences in metallic properties between metal plates and metal filler, and contamination of the molten pool during welding. All professional welders are certified by institutions and trained to be the primary inspector to identify surface level defects. After the fact, weld inspectors also perform various tests to examine the quality of an operator's weld.

Please see the appendix *A.I. Key Terms and Concepts* section For More Information on how metal weld defects are formed and current inspection practices.

2.II. Methodology

To better understand the effect of using synthetic images to train CV models, this research uses the Unreal Engine (published by Epic Games) to render images with custom photo-realistic backgrounds and models in a variety of situations. This is done by using the following software versions. For a full list of software packages, please see the *A.II. Installed Python Software Packages* section in the appendix For More Information.

Software Package	Version
Unreal Engine	4.27
TensorFlow	2.12
Keras API	2.12

Figure 2.1: Software versions used for this research.

Images rendered in the UE are then used in the training set of two reduced published transfer learning models for separate binary classification tasks; 1) determine if a picture is of a Cat or Dog and 2) Determine if a picture of a metal weld contains a defect. The Cat_Dog and Welding use cases represent two separate cases that require different types of graphical models with different characteristics. Animals such as cats and dogs have finer features and irregular shapes that could be difficult to capture in a 3D model (hair, whiskers, legs, etc), but provide significant variation. Alternatively, modeling a weld defect occurs on a regular shape, but requires a different type of fine detail (dust, burn marks, holes, and other nuances) to create variance.

A. Model Architectures Used

This project depends on published models already trained on the ImageNet 1000 class dataset: A) VGG16 and B) MobileNetV3-small. The existing ImageNet 1000 weights for each model act as a starting point during training via transfer learning. Both models have a pre-defined architecture, existing research, and documented performance. Approximately 60% of each model's parameters are frozen during training to not tamper with the general feature extraction layers common to image classifiers. Only the last layers of the models were trainable to fit specifically to this research's use cases.

This paper utilizes transfer learning models to take advantage of work already done in other research and keep the paper focused on the effect of synthetic data. Using published models removes uncertainty of model architecture design. In addition, published state-of-the-art CV models have rigorous research as to why the model's architecture exhibits state-of-the-art results compared to any custom architectures. Moreover, using publicly available published models as a starting point is a common development practice. Focusing this research on how models will practically be used will allow any findings to be applied by more developers for their own projects.

i. Originally published Models

VGG16 and MobileNetV3-small are the two models examined in this research. Both represent different ways CV models can be applied. This research focuses on how synthetic data influences the largest (VGG16) and smallest (MobileNetV3-small) models to allow developers to make the most informed design decisions.

The VGG16 model is a model with many parameters that is best ran where there are minor limitations on power consumption and computer memory, such as hosted on an on-site or cloud server. The model has a common convolutional neural network (CNN) architecture involving 16 sets of *Convolution-Pooling-BatchNorm* layers [Simonyan14].

MobileNetV3-small is a variation of the MobileNetV3 model that has the fewest number of trainable parameters. The model is specifically designed for running on mobile devices & on-chip hardware, prioritizing lower power usage, a low memory footprint, and reduced latency from model input to output. MobileNetV3's architecture utilizes Squeeze-Excite layers that advance upon residual neural networks introduced by Dr Kaiming He's ResNet model [He]. After each convolutional layer, MobileNetV3 uses Squeeze-Excite residual network blocks to pool (squeeze the output) the Conv2D layer's output, passes them through fully connected (FC) layers (excite), before scaling the output for the next Conv2D layer. These FC layers apply weights to each of the previous Conv2D layer's filters, thus prioritizing which filters contain the most important features [Howard19].

From now on, any mention of MobileNetV3 will be in reference to MobileNetV3-small.

ii. Modifications to the published Architectures

Due to the differences in classification scope, both transfer learning model's training settings and output architectures are changed to better align with the desired task. Originally, both published models are trained to classify 1000 various classes of objects. Since this research only requires binary classification, the model's architecture and training sets are changed. These changes include changing the number of parameters in the output layers, Learning Rate, training Batch Size, dataset size, and how many parameters are trainable, as well as others. The base architecture layers are unchanged, with the top ~60% of trainable parameters being frozen. The output layers for each model are reduced as well, shrinking the total number of parameters for each model ~85% and ~47% respectively. For a complete list of model architecture values, please see the *A.III. Model Architecture changes from Published Models* section in the appendix For Your Information.

	VGG16	**MobileNetV3**
General Use Case	High Accuracy Adequately available computer resources On-demand power supply	Low power consumption Low prediction latency Low memory footprint
General Architecture	Conv-AvgPool-Dropout	Squeeze & Excite
Total Parameters in the published model for 1000 class classification	138,357,544	2,554,968
Total Parameters in the transfer learning models used in this research for binary classification	21,203,521 (15.3%)	1,365,617 (53.4%)
File size of weights (.h5)	80.9 MB	5.4 MB

Figure 2.2: Magnitude of size reduction in trainable parameters from the original paper to the models used in this report [Simonyan14][Howard19]

B. Model Training

Model hyper parameters such as Learning Rate and Batch Rate are chosen via trial and error when trained on the 100:0 RSR test case. The final values are chosen based on which combinations yielded consistent convergence to the feature space's minima. The learning gradients are calculated using binary cross entropy. Initial Learning and Early Stopping techniques are used for all experiments. The VGG16 and MobileNetV3 models trained for 5 epochs before being evaluated for early stopping. During the Cat_Dog experiment, model training ends if the model does not exhibit a -0.001 change in validation loss for seven consecutive epochs. During the Welding experiment, model training ends if the model does not exhibit a +0.001 change in validation AUC performance within seven epochs. As a note, during all experiments Synthetic images are only used during model training, model validation and testing sets only use real images.

Specifically for the Welding experiment's {Defect, Good} classes; the experiment's classes are assigned so that a *Good* weld is a True Positive while a *Defect* is a True Negative. This decision results in the binary classifier model learning what the features of "a *good* weld". This is done so that model training can be in terms of the three most important confuser matrix cases classifications (True Positive, True Negative, and False Positive). The False Positive classification (a weld *Defect* that is classified as *Good*) carries a much higher cost than a False Negative (a *Good* weld classified as a *Defect*). Therefore, the model is trained to minimize the False Positive classification while maximizing True Positives and True Negatives.

$$AUC = \frac{\#TP + \#TN}{\#FP}$$

Question to the Classifier	*"Is this image a picture of a Good weld?"*
Defect weld	Negative (0)
Good weld	Positive (1)

Figure 2.3: Binary classifier design for the Welding use case to minimize the instances when the models predict an image is of a Good weld but is actually a Defect.

Class weighting for error gradient calculation in the Welding use case is based on the imbalance of the Welding dataset. Since there are more *Defect* welds than *Good* welds in the dataset, error from classifying *Good* welds is greater than classifying *Defects*. This means that the model is more sensitive to misclassifying a weld as Good. Thus, encouraging the model to be more accurate when it says an image is a *Good* weld. However, it leads models to skew towards defaulting every weld to be a *Defect*. Experiment 4 uses class weighting for its results.

C. Image Augmentation

During model training, the training set is expanded using a variety of 2D image augmentation techniques. Each of these techniques has a random chance of being applied and can stack on top of one another (an image could have all, none, or some of the augmentations). These techniques include standard augmentation such as `horizontal & vertical flips`, `rotation`, `shifts`, `brightness change`, `shear`, `zoom`, and `channel shift`. The model batch size and epoch steps are set so that the models train on all available training images every epoch. Meaning that every epoch, the entire training set is randomly augmented and fed into the model to calculate error gradients. Theoretically resulting in a more generalized model since every epoch is a slightly 'brand new' training set. For a list of applied 2D image augmentations, their available ranges, and reasoning for use. Please see the *A.IV. 2D Image Augmentations* section in the appendix For More Information.

Figure 2.4: Example image augmentations for three separate epochs. The left image would be used for training in epoch 1, the middle in epoch 2, and the right in epoch 3. Augmentations: (left) rotate right, shift up, shift right. (middle) zoom, shift up, shift right, brightness up. (right) vertical flip, rotate left.

D. Dataset Formation

The datasets used for all experiments are a combination of real-world images from various public internet sources and synthetic screenshot images from Unreal Engine v4.27. Not every real image gathered is used in this paper's experiments. The process for removing real data and the method of sampling is covered for each individual use case further in this paper. Used real images have varied original dimensions, all synthetic images are captured at the same dimensions based on the chosen camera in UE (782x440 for Cat_Dog images or 853x480 for Welding images). Regardless, all images are resized to 224x224 to match input dimensions of the VGG16 and MobileNetV3 models.

ii. Collecting Synthetic Images via the UE Python API

UE contains a command line that can execute python scripts (if enabled). Executed python scripts can send UE commands and get information from the opened UE Editor that called the script. This leads to two notable computer limitations: installed python packages and threading.

UE runs these scripts in the same environment as the computer programe. Therefore, you must install all needed python packages to the same environment that you run the UE instance. Meaning that you will have package compatibility errors if you develop the scripts in a virtual environment but run UE from your desktop.

Regarding threading, these scripts run on the same software threads as the UE Editor and use the python packages installed on the environment that UE is opened with [DowlingSoka]. Technical Art Engineer, Ryan DowlingSoka, refined a python class that can execute python code every 'tick' of the UE Editor process through a software callback. Every tick, the code in the callback executes. To gather all the needed synthetic image variations for the Cat_Dog and Welding classes, a script based on this python class is created to run a predefined state-machine, that increments after every callback. Every ~30 ticks, the state-machine either A) sets the UE Editor environment (setting model positions, camera+lighting positions, texture properties, lighting intensity, etc) or B) takes a screenshot. At an execution rate of one callback action every ~30 ticks, the script can generate 80 images a minute if not interrupted. This rate is chosen to give each callback enough time to execute all necessary UE commands prior to the prompting of the next callback. (since both the script and UE Editor are on the same thread). To learn more about base python class and writing your own class, please see the *A.V. Collecting Synthetic Images via the UE Python* API section in the appendix For More Information.

iii. Use Case: Cat_Dog

The datasets used in Cat_Dog experiments are sampled from the real Cats_Dogs ImageNet dataset and four different collections of synthetic data using various free to use Cat & Dog .fbx models and UV textures. The binary classes are {Cat, Dog}. The real dataset contains 12500 cats and 12500 dog images. The dataset is referenced directly from the Tensorflow datasets repository, distributed by Microsoft [Elson07]. All 25000 collected real images are available for sampling, none of the images are removed or edited.

iv.1. Synthetic Data

Over the course of two collections (4 runs total), 47864 screenshots of 3D cat and dog models are collected against various backdrops in a variety of camera angles and lighting conditions. The used cat and dog .fbx 3D models are free to download from sites such as Turbosquid and Free3D. It is possible to purchase more advanced models with higher polygon, triangle counts and textures. The backdrop presets are different images and textures put behind the 3D model to represent background variation. Each backdrop preset is operated via a UE parameter that feeds into a software multiplexer in the UE material instance to decide which preset is shown. For all 3D models, authors, and credits, please see the appendix section *A.VI. Cat_Dog – Collecting Synthetic Images* for more information.

	Collection 1		Collection 2	
	Run 1	Run 2	Run 3	Run 4
Collected Image Variations	6672^	2753^	27478	9881
# 3D Models	23	23	23	23
# of Camera Angles	28	10	72*	72*
# of Lighting Settings	1	3	4	1
# of Backdrop presets	4**	4**	6*	6*

Figure 2.5: Summary of variation in screenshots collected for Cat_Dog use case.
^ During collection, some images were lost due to nuances of running the python script through the UE Editor.
**Run 3 & 4 are the same except for the 'unlit' lighting setting.*
***Backdrops for Run 1 & 2 are the same*

Screenshots are collected via a python script that iterates through all possible combinations of these settings, as previously mentioned. During collection, the script moves a cat/dog model to a point in front of a backdrop at a specific lighting (a SunSky light modeled off earth's sun), at hardcoded camera angles. Once all camera angles have been captured, the cat/dog model is moved out of the out of the frame, then the next cat/dog model is moved to the point. The process repeats all set variations (cat and dog models, all backdrop presets, lighting intensities) in the UE test environment until a screenshot is taken of every variation. To learn more about the nuances of this image collection and more details on the different backdrops, please see the *A.VI. Cat_Dog - Collecting Synthetic Images* section in the appendix For More Information.

Figure 2.6: Example screenshots taken at different angles, models, and backdrops that are then possibly used to train the CV models prior to 2D image augmentation.

iv. Use Case: Welding

The real images sampled for the welding experiments come from a variety of public sources. Several public repositories of real weld images are available online, but every dataset has various flaws. Flaws include multiple datasets using the same images, datasets including a mix of original and augmented images, or datasets that contain more than one weld (*Defect* or *Good*) in a single image. Dataset specifics are discussed in the next section. The synthetic data is generated from edits to a purchased UE material package from the Epic Games store. One of these materials in the package is applied to a flat mesh plate where various texture map bitmap files are edited to mimic three weld defects: porosity, crack, and incomplete welds. In general, all real and synthetic images are labeled as {*Defect*, *Good*}. The classifier does discern which defect is present, only that the weld has a defect.

v.1. Real Data

Through five sources, 2221 {624 Good, 1597 Defect} real images of various levels of quality form the real dataset. Out of the 5813 real images found from the five datasets, only 2221 are used. Most of the real images are removed to avoid repeat images, images with irrelevant classes, and images of multiple welds so close together that each weld could not be practically cropped. For more details on how each of the five datasets were used, please see the *A.VII. Welding – Dataset Formation* section in the appendix For More Information

Figure 2.7: Example of an image from dataset 01 that shows two welds in one picture [welddefect]. For some images, the researcher copied the image and cropped both images into separate images. Resulting in two separate image data points.

v.2. Synthetic Data

Images for the Welding synthetic dataset are generated from taking screenshots of points of interest on used UE mesh metal plate actor. Synthetic images are generated from random camera angles, light offsets, metal color, and texture map presets. A 'map preset' is a combination of Metallic, Normal, Roughness, and Ambient Occlusion texture bitmap files specific to a certain welding variation. The sum of these four bitmaps creates the imperfections and details on the metal plate. Weld plate variations include 1) no defects, 2) imperfections but not enough to be defective, 3) porosity defects, 4) crack defects, and 5) incomplete weld defects.

Figure 2.8: (left) Image of the plate in which screenshots are collected (specifically shown is the purchased 'Metal_Weld' material with the UE starter content 'steel' basecolor). (right) Side view of the metal plate used to collect images. Note: all texture map edits are to a 2D plane. Any perceived height is an optical illusion from the Normal and Ambient Occlusion maps.

Like the Cat_Dog synthetic dataset; a total of 8160 synthetic images are collected to provide additional variation to the real dataset. Variation comes from the plate basecolor, texture map changes, camera angle and point light offset at several points. Texture map changes are applied to a base UE material purchased from the Epic Games Unreal Store create the weld imperfections (bitmap and targa files are 4096x4096 pixels) [4k Metal pack]. Six metal colors, 5 map presets and 68 points of interest are cycled through. The defect dimensions are randomly chosen from that dimension's set range of values.

During the execution of the collection python script, the base color and preset texture map variations are set by a switch in a custom UE Node within a UE function for every change (one switch to decide which combination of basecolor, Metallic, Normal, Ambient Occlusion, and Roughness maps are showing). 68 chosen points of interest are specified along the horizontal and vertical weld seams, separated by 400 pixels. No points are withing 15 pixels of the plate edge, and multiple points can be present at seam intersections.

The collection is done by executing a python script from the UE Editor. A camera and point light iterate through 68 points along the weld seam on the metal plate. Four screenshots are taken at different angles and point light locations. The specific camera angle and point light offsets are selected from a random range. To learn more about how the definitions and plate material variations, please see the *A.VII. Welding – Dataset Formation* section in the appendix For More Information.

Plate Qualities	Variations
Plate base color	Aluminum, Steel, Copper, Gold, rusted metal, and a base metal
Plate map presets [*Metallic, Normal, Ambient Occlusion, & Roughness*]	1) no defects, 2) edits but not enough to be defective, 3) porosity defects, 4) crack defects, and 5) incomplete weld defects
Number of Plate weld seam points of interest	68 (spaced 400 pixels apart)
Number of screenshots per plate point of interest (random Camera & Point Light offsets)	4

Figure 2.9: summary of synthetic weld collection variations.

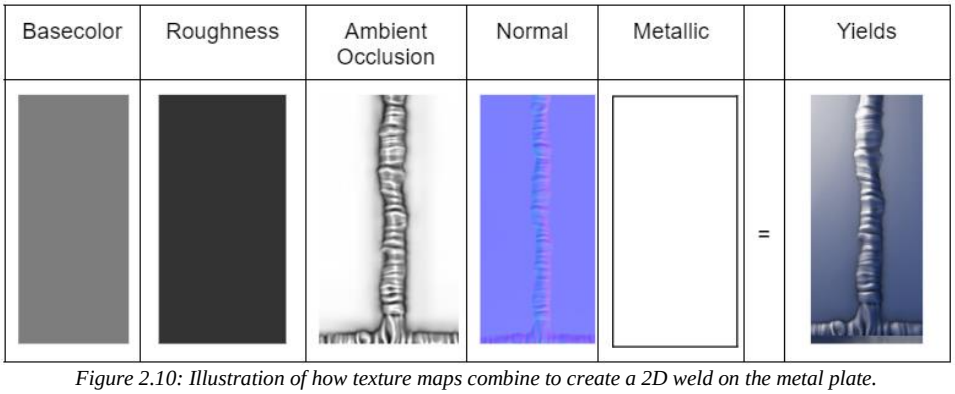


Figure 2.10: Illustration of how texture maps combine to create a 2D weld on the metal plate.

v.3. Weld Imperfection Design

Three welding defects are modeled as a part of this research: Porosity, Crack, and Incomplete Welding. All three defects have different changes to the four main texture maps in a map preset. The following section details the researcher's interpretation of what visually defines each defect, what variables are used to create the UE Texture. In addition, non-defect versions of the three defects are defined for map preset 02. These non-defect versions are labeled as *Good* during model training and considered as imperfections that mimic a defect at a reduced scale or without certain characteristics.

A part of this design is the use of randomness to create a unique weld imperfection (defect or not) at each of the points of interest. Each of the welding defects uses random numbers to subtly change how a defect is designed. Every weld defect is a different size, length, and has different material properties by adding a level of noise to the resulting dimensions or texture map RGB colors. The randomness range for each dimension and RGB color pixel is set for each defect. This randomness helps add variation and make each weld modification unique. To learn more about what is specifically done to create each weld texture, please see the *A.VII. Welding – Dataset Formation* section in the appendix For More Information.

Defect	Visual Description	Real Weld Defect	Synthetic Weld Defect
Porosity	Surface level Elliptical holes caused by trapped gas rising to the surface of the molten pool as a weld cools. Random Variables • xy-radius • xy-offset from weld center. Non-Defect = Smaller ellipse		
Crack	Surface level splitting or cutting of the metal, creating triangular separation compromising weld strength. Random Variables: • Length • Number of angled segments • Offset of each angle segment Non-Defect = a shorter length, lower offset per segment and fewer segments.		
Incomplete Weld	A missing area of filler on a welding line. Revealing the split between metal plates Random Variables: • Width of gap • Depth of gap towards the seam. • Seam size Non-Defect = Smaller gap width and not enough gap depth to reveal the plate seam		

Figure 2.11: Summary table of each of the three weld defects modeled for map presets 02-05 in the synthetic Welding datset.

3.III. Results

Five total experiments are performed to understand how synthetic data affects CV models in different ways. By design, different synthetic datasets in different experiments can reveal nuances to how synthetic images can be used to train CV models. The two types of experiments (RSR & Rebalancing) are documented below. Both experiments train the transfer learning models on different variations of training data, specifically how much and what synthetic data is used. The reported data results are the average of three training runs for each test within an experiment. Three runs are believed to give enough variation in training results to uncover the desired trends.

RSR experiments explore how different proportions of real to synthetic images in the training set effects model performance. The RSRs tested range from 100:0 (all real) to 0:100 (all synthetic). The sole Rebalancing experiment compares model performance when the training dataset is imbalanced to after it is balanced by four methods. This is an expansion of Dr Gary Weiss' 2007 experiment [Weiss07].

Experiment	RSR Experiments	Rebalancing Imbalanced Dataset Experiment
Goal	How do different RSRs affect CV Models	Can synthetic data be used to rebalance an imbalanced dataset
Cat_Dog	3x	Not performed
Welding	1x (Limited)	1x

Figure 3.1: Outline of the two types of experiments performed for Cat_Dog (Experiments 1-3) and Welding (Experiments 4 & 5).

All five experiments and dataset variations use an 80/10/10 training/validation/test split to maximize training samples while attempting to provide enough testing samples (a constraint with small datasets). Furthermore, all images in the validation and test sets are only real images, no synthetic images are used in the validation or test sets. Synthetic images are only used in the training set.

A. Cat_Dog Experiments

Three RSR experiments are performed with a balanced dataset of 10,000 total images. 1) model performance trained on different RSR proportions of real and synthetic images when with only 9425 synthetic images are available to be sampled, 2) model performance with different RSR values with 46784 synthetic images to sample, and 3) baseline performance of models with only real images but decreasing dataset sizes to compare against 1) & 2).

All tests are performed with balanced training/validation/test sets. The training set is a mix of real images and synthetic images from the Cat_Dog synthetic dataset. Model accuracy and loss are the primary focus of the Cat_Dog use case results.

Experiments 1) and 2) investigate how different levels of synthetic data availability can affect model performance. Different Synthetic Utilization Ratios (SURs) could affect the level of variance shown to a model with the chosen synthetic samples. The third test highlights the effect of padding real data with synthetic data. The displayed bar chart showing the differences between training datasets with or without synthetic data highlights if adding synthetic images improved model metrics. The definitions for all experiments & tests are outlined in the appendix. Please see the *A.IX. Results* section in the appendix For More Information.

RSR	Total Images	Number of **Real** Training Images	Number of **Synthetic** Training Images	Number of real **VALIDATION** images	Number of real **TEST** images
100:0	10000	8000	0	1000	1000
75:25	10000	6000	2000	1000	1000
50:50	10000	4000	4000	1000	1000
25:75	10000	2000	6000	1000	1000
10:90	10000	800	7200	1000	1000
05:95	10000	400	7600	1000	1000
0:100	10000	0	8000	1000	1000

Figure 3.2: Outline of tests for Experiment 1-3.

i. Experiment 1 – RSR with 9425 Synthetic Images

When the VGG16 model is trained on various RSRs with only 9425 synthetic images to sample, the test accuracy and test loss performance begins to degrade at RSR 25:75. The model's accuracy increases to its max at RSR 75:25 at 96.5% (+0.6%) before decreasing for every subsequent RSR test. The lowest test accuracy occurs after a steep decrease in performance from RSR 05:95 to 0:100 with 85.9% (-10.8%) binary accuracy. The model's test loss follows a similar pattern to the test accuracy. The performance is consistent before steadily increasing when RSR lowers past 25:75. Meanwhile, the model's training accuracy remains relatively flat throughout the experiment even as RSR lowers to 0:100. These results show that with a 1.4% decrease in test accuracy, a user can use 7600 less real training samples if RSR 05:95 is used.

Also, when RSR is at 75:25, the test accuracy is greater than the training accuracy indicating a positive level of generalization. However, the significant difference between training and test loss implies that the VGG16 model is still overfit.

Regarding test accuracy variance, it showed a relatively positive parabolic shape. This means that the test accuracy variance peaks at RSR 0:100 and 100:0 respectively but is at a minimum during RSR 50:50 and 25:75. This should be inspected further and, in more experiments, to see if the trend holds. At worst, this is noise due to too few data points, at best this is a sign of the VGG1 model learning the latent features of the synthetic data and applying them to provide a more stable test accuracy performance.

Please see table *A.IX.R4* in the appendix section *A.IX. Results* For More Information.

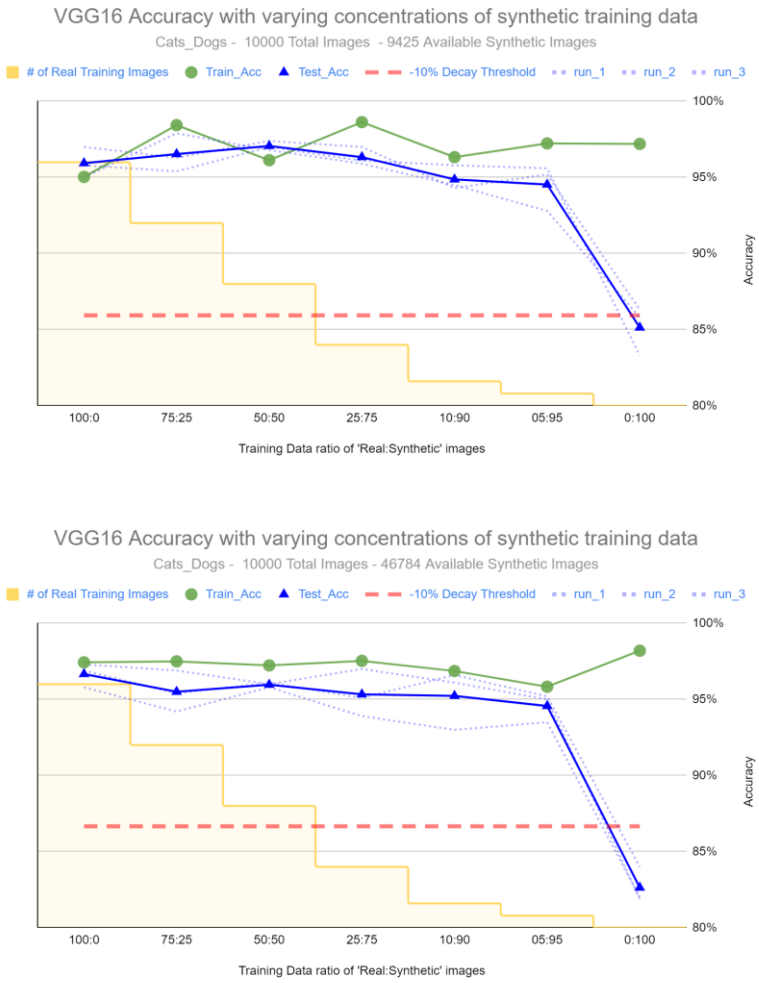


Figure 3.3: (Top) accuracy and (bottom) loss graphs of VGG16's performance over different RSR settings with the Cat_Dog use case.

The MobileNet model's performance is slightly different from VGG16 when sampling from 9425 synthetic images. The model's test accuracy decreases consistently until 05:95 where it steeply decreases to ~50%. The training accuracy remains flat after an initial decrease, diverging from test accuracy. The model's test loss appears slightly lower at RSR 75:25 before increasing for the remaining tests.

The combination of flat training accuracy while decreasing test accuracy and widening training & test loss gap signals that the MobileNetV3 model is very overfit to the training data. This is significant because at lower RSRs, the model is overfitting to the synthetic data and cannot apply the variations to the real-world test samples. In addition, unlike the VGG16 model, the MobileNet model's test accuracy degrades past 10% from training on only real images before the 0:100 RSR.

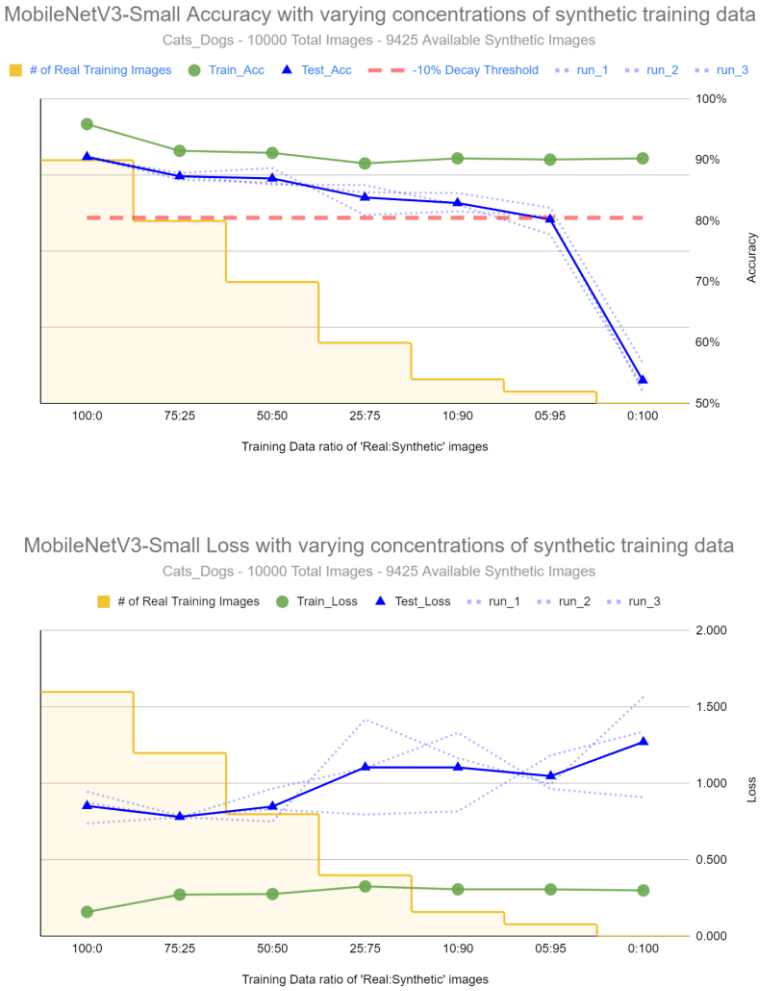


Figure 3.4: (top) accuracy and (bottom) loss graphs of MobileNetV3's performance over different RSR settings with the Cat_Dog use case.

ii. Experiment 2 – RSR with 46784 Synthetic Images

Compared to Experiment 1 above, increasing the number of synthetic images for sampling does not appear to significantly change model performance or observations expressed in Experiment 1 above. This implies that having image latent variation does not scale with the number of camera angles taken. Other changes like backdrops, models, and lighting require just as many variations. This is because the Cat_Dog gathered dataset primarily used varying camera angles around a center location during the collection. The number of backdrops and lighting changes are limited.

Some slight differences include the VGG16 model not being greater than training accuracy and its test accuracy dropping much more significantly from RSR 05:95 to 0:100. MobileNetV3 shows a slightly lower training accuracy to test accuracy gap, but the test Loss is significantly higher when there are 46784 available synthetic samples compared to only 9425.

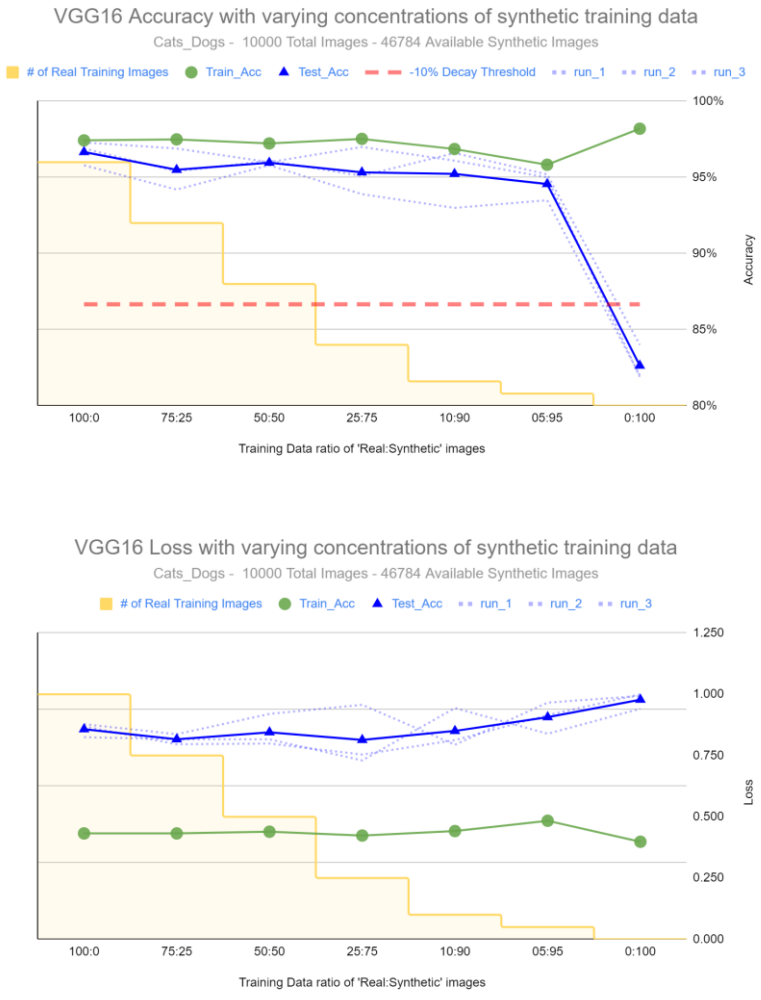


Figure 3.5: (top) accuracy and (bottom) loss graphs of VGG16's performance over different RSR settings with the Cat_Dog use case when there are 46784 synthetic images to sample. Shows that accuracy behaves approximately the same to when 9425 synthetic images are available. Implies that there is no performance gain by having more samples when those samples are collected in a ridged way.

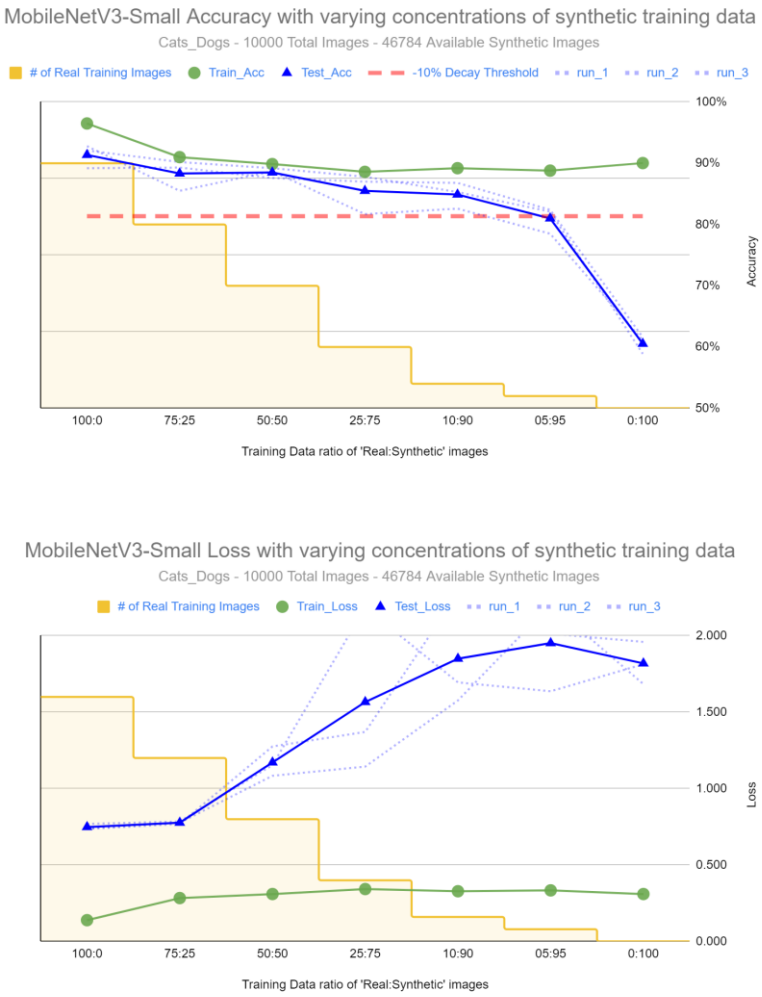


Figure 3.6: (top) accuracy and (bottom) loss graphs of MobileNetV3's performance over different RSR settings with the Cat_Dog use case when there are 46784 synthetic images to sample. Plots show that accuracy degrades exponentially as more of the training data is synthetic.

iii. Experiment 3 – RSR. Mixed vs Smaller Datasets

When comparing results of models trained solely off datasets with only real images versus datasets with a mix of real and synthetic images, the results below show different trends for test accuracy and test loss depending on the model.

For the VGG16 model, the results do not show a meaningful change in test accuracy regardless of synthetic availability. However, having a mixed dataset of real and synthetic images lowered the test loss for nearly all RSR tests. For the 05:95 RSR test (5% in Figure 3.7), these trends highlight how synthetic image quality can be judged. In the 05:95 RSR test, the results show that padding 400 real images with 7600 synthetic images of cats and dogs yielded a test accuracy increase of only ~0.6% and a test loss decrease of approximately 0.1.

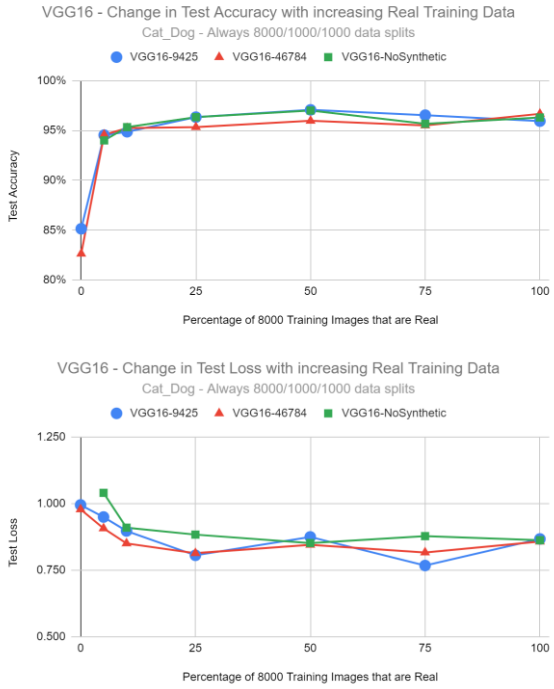


Figure 3.7: VGG16 comparison of three different training sets with their (top) test Accuracy and (bottom) test Loss. 'VGG16-NoSynthetic' uses a training set made of only real images but ranges from 400 images (5%) to 8000 (100%) images. The plots show that, compared to using a smaller training set (green), using additional synthetic images reliably lowers test Loss without a decrease in test Accuracy.

The MobileNetV3 model did not perform comparably when trained on only real images versus a mix of real and synthetic images. Training on only real images yielded the best test accuracy and test loss metrics for all RSR test cases. The results show that accuracy degraded somewhat exponentially as the percentage of synthetic images in the training set increased. In the big picture, this experiment describes that at RSR 05:95, datasets of only 400 real images yielded higher test accuracy and comparable loss to a dataset with those 400 real images plus an additional 7600 synthetic images.

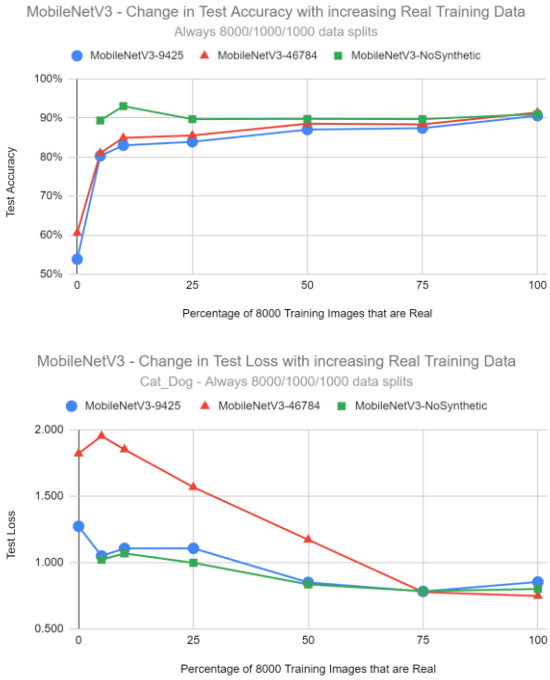


Figure 3.8: MobileNetV3-small comparison of three different training sets with their (top) test Accuracy and (bottom) test Loss. 'MobileNetV3-NoSynthetic' uses a training set made of only real images but ranges from 400 images (5%) to 8000 (100%) images. The plots show that, compared to using a smaller training set (green), using additional synthetic images has a negligible effect on test Loss, but a negative effect on test Accuracy.

A. Welding Experiments

Experiments 4 & 5 are performed with limited amounts of real weld data and the collected synthetic Weld dataset; 4) model performance trained on different RSR proportions of real and synthetic images while using the entire imbalance weld dataset, and 5) model performance changes based on different rebalancing methods. Model ROC AUC (receiver operating characteristics area under the curve), loss, and classifier confusion matrix percentages are the primary focus of the Welding use case results. As a reminder, the *Defect* class is considered Negative, and the *Good* weld class is considered Positive for these experiments. This is done so the models can use Test AUC as an early stopping metric during training.

Question to the Classifier	"Is this image a picture of a Good weld?"
Defect weld	Negative (0)
Good weld	Positive (1)

Figure 3.9: repeat of figure 2.3

Experiment 4 is like Experiments 1-3 for the Cat_Dog use case. The major difference is that the entire real weld dataset (1400 Defect & 600 Good weld samples) is used for every test and the class imbalance from the real weld dataset is maintained for every test. Also, both use case experiments have the same goals; how the welding use case is affected by different RSRs and how that differs from the Cat_Dog use case. However, there are a few key differences.

The first difference is that the RSR experiment is more limited due to the lack of available *Good* weld samples. Second, the training set is imbalanced at 7:3 Defect-to-*Good* class, but the test set is always balanced. Since the training set is imbalanced, the model class weights during training are inversely proportionally weighted, even if the test set is balanced. Therefore, the error gradient for *Defect* weld samples are weighted at 0.34 while the *Good* weld sample error gradients are weighted at 0.66. The imbalance class weighting encourages the model to not overlearn the majority class in the dataset (the potential consequence being that the model thinks all images are defective just because it trained on mostly *Defect* images). Finally, as the RSR decreases (skewing the training set towards mostly synthetic images), the number of real samples in the validation and test set increases. Addign Synthetic data allows for real samples to be moved to the validation and test sets to maintain the 80/10/10 training-validation-test data splits.

Data split	Balanced?	Type of Images
Training set	Imbalanced	Mix
Validation set	Balanced	Only Real
Test set	Balanced	Only Real

Figure 3.10: Summary of dataset splits for Experiment 4.

Experiment 5 expands upon research done by Weiss, et al investigating different methods of sampling on decision trees with imbalance classification data. Weiss concluded that when rebalancing 8 chosen datasets with three types of rebalancing methods (cost-sensitive learning, upsampling, and undersampling); cost-sensitive learning yielded the lowest cost for most datasets with over 10,000 samples, while oversampling yielded the lowest cost for datasets with less than 10,000 samples [Weiss07]. This second experiment expands Weiss' research to rebalance datasets with less than 10,000 samples with synthetic data. Specifically compare how padding the minority class with synthetic data compares against all their previous methods. In particular, the 'Padding with Synthetic' case balances the used dataset by adding 600 synthetic *Good* Weld samples to yield a dataset with 2400 samples (1200 samples per class). All 600 *Good* class synthetic images are screenshots from the Unreal Engine using map preset 01 (no changes to the UE material) & 02 (non-defect imperfections). This experiment focuses on metrics related to confusion matrix performance and model training (such as accuracy & loss).

Balancing Method	Total Number of Images	TP Cost	TN Cost	FP Cost	FN Cost
No Change	1800	0	0	0.5	0.5
Cost Learning	1800	0	0	0.667	0.334
Undersample	1200	0	0	0.5	0.5
Oversample	2400	0	0	0.5	0.5
Synthetic Padding	2400	0	0	0.5	0.5

Figure 3.11: Summary of rebalancing methods used in Experiment 5.

$$Total\ Cost\ per\ Sample\ = \frac{TP * Cost_{TP} + TN * Cost_{TN} + FP * Cost_{FP} + FN * Cost_{FN}}{\#\ of\ Test\ Samples}$$

Figure 3.12: Equation used to calculate the cost of every model classification.

i. Experiment 4 – RSR with 8160 Synthetic Samples

The VGG16 model's test accuracy improved during RSR 75:25, 58:42, and 37:63 tests compared to 100:0. The maximum test accuracy was 84.3% (+4.8%) compared to the 100:0 test accuracy of 79.6%. Test Loss also decreased for every RSR test compared to 100:0. The minimum test loss was measured at RSR 75:25 at 0.987 (-0.100). The training accuracy decreases as the RSR lowers towards 0:100, trending more in line with test accuracy. This is similar performance to Experiments 1 & 2.

Based on the training-test gaps of the loss and AUC metrics, the VGG16 model is most likely still overfit, even if introducing synthetic images reduces the gap slightly. However, the VGG16 test AUC metric decreased as the amount of synthetic data in the training set increased.

When looking at the confusion matrix performance, as more synthetic images were added to the training set, the model's True Negative (predicting a *Defect* is a *Defect*) and False Negative (predicting a *Defect* when actually a *Good* weld) metrics started to equalize. This is problematic since as this trend continues, the model is predicting more images are Defect welds as a default. In terms of cost, this trend is better than the alternative since the number of False Positives decreases as more synthetic data is introduced. The bar chart below shows that even if TN & FN classification percentages on the balanced test set are equalizing, the FN percentage and numeric Cost per Test Sample is decreasing. Along with the test accuracy and test loss trends above, this implies that the synthetic weld data is providing enough extra variation to improve performance (especially at RSR 75:25 and 50:50 where test accuracy, test loss, and the combination of confusion matrix metrics are at their best).

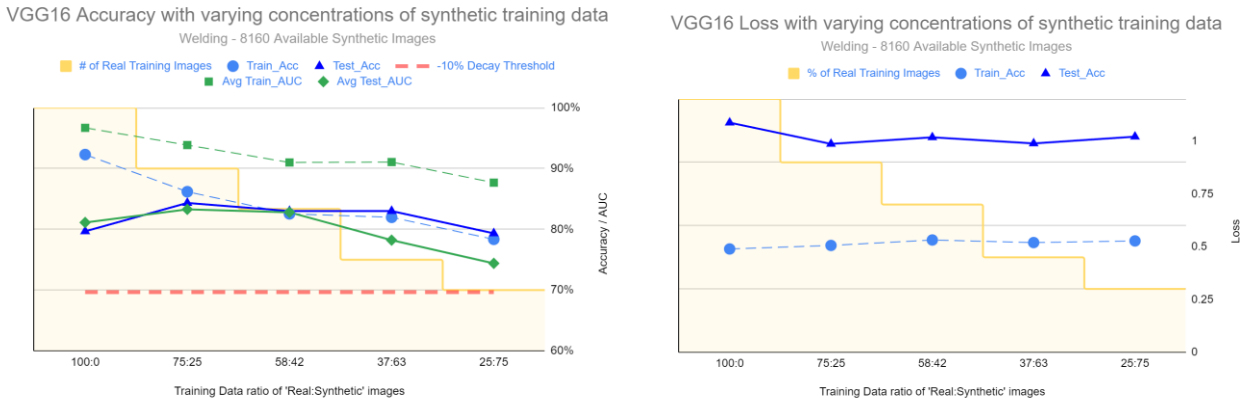


Figure 3.13: Absolute training plots for VGG16 (left) Accuracy and (right) Loss. Shows max accuracy, max AUC, and min Loss when a training set is 75% real images and 25% synthetic. Accuracy and AUC decline for RSRs lower than 58:42, but Loss stays relatively unchanged when at least some synthetic images are used in the training set.

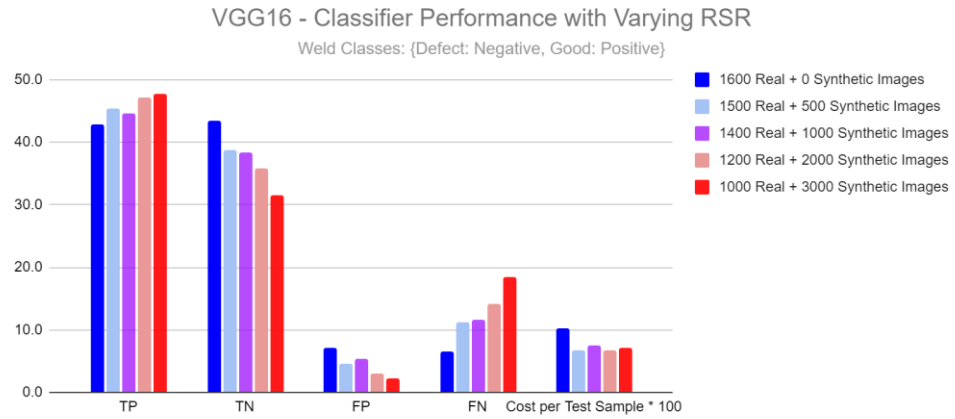


Figure 3.14: Bar chart showing how VGG16 models trained on different sets of training data performed on confusion matrix focused metrics. {Defect weld: Positive, God weld: Negative}. As the number of synthetic samples increased, the number of TN and FP classifications decreased. The number of FNs increased. As such the Cost per Test Sample decreased as the number since a FP has a much higher cost associated with it.

MobileNet3's performance has much more worrying trends. The test accuracy metric shows similar trends to the performance in Experiments 1 & 2 with the Cat_Dog synthetic dataset. Test accuracy does steadily decrease with more synthetic data, but the test loss appears to fluctuate. One of the more problematic trends is the MobileNetV3 test AUC metric hovering at approximately 50%. This signifies that the model has not decoupled what distinguishes a True Positive from a False Positive. These results and the bar chart further below send mixed signals as to how the classifier is learning from the imbalance training set and performing on the balanced test set.

One interesting observation to note is that the test accuracy for both VGG16 and MobileNetV3 are only a few percentage points different until RSR 50:50. This is less of a reflection on the quality of performance between the two models, but of how an imbalance dataset can skew a metric like accuracy and why other metrics are important to analyze. Figure 3.15 below shows that on average the MobileNetV3 model classified more and more samples as Positive (aka *Good*) as more synthetic data is introduced. This could be for many reasons; possibly that the MobileNetV3 model is more sensitive to the increased weight that error from *Good* samples created.

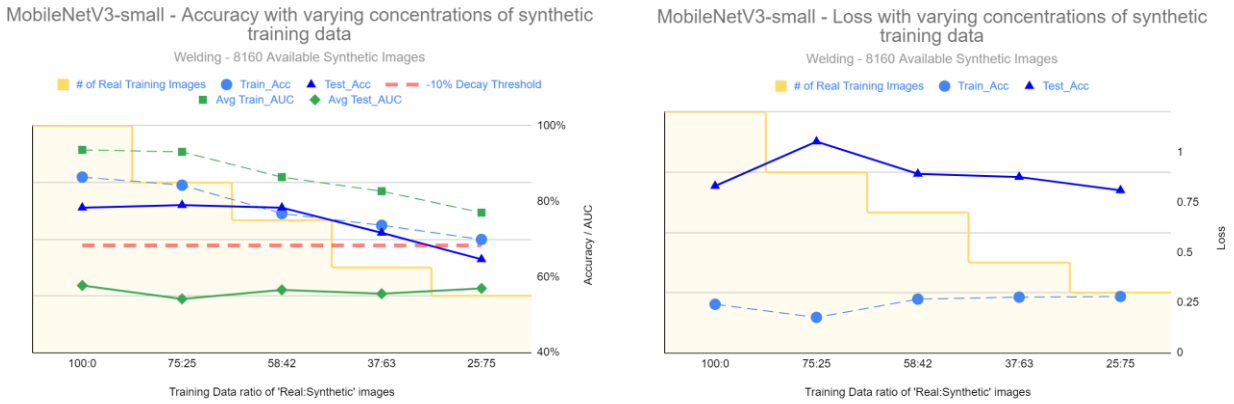


Figure 3.14: Absolute training plots for MobileNetV3-small (left) Accuracy and (right) Loss.

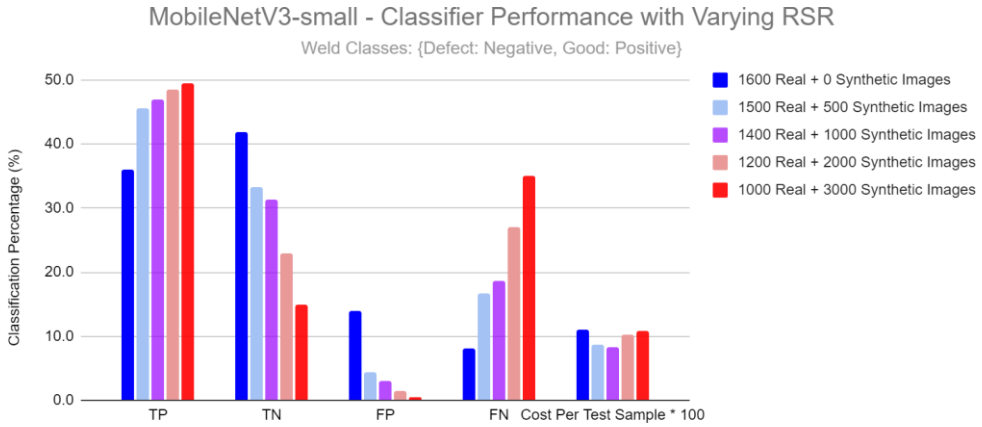


Figure 3.15: Bar chart showing how MobileNetV3 models trained on different sets of training data performed on confusion matrix focused metrics. {Defect weld: Positive, Good weld: Negative}. Highlights how more synthetic samples in the dataset skewed the MobileNetV3 models to classify most samples as Positives (aka. Good welds)

ii. Experiment 5 - Rebalancing

The VGG16 model's results show that in terms of Cost per Test Sample, the oversampling method showed the greatest decline in cost (-18%) followed by synthetic padding (-12%). This is in line with Weiss' research) since the dataset used is below 10,000 samples. Compared to an imbalanced dataset, all methods decreased the Cost per Test Sample and increased test accuracy, but decreased test AUC and increased test Loss. A mix of desired and undesirable reactions. The synthetic padding results for loss are surprising. Dr Man's survey of the literature suggests that using synthetic images tends to reduce loss, not increase it [Man07].

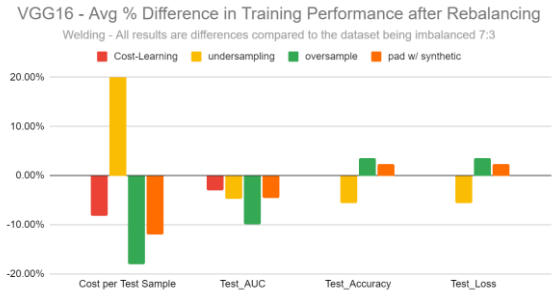


Figure 3.16: Bar chart showing the difference in performance metric between A) a VGG16 model that was trained on a sampled version of the original 2000 image real weld images, no synthetic (1200 Defect & 600 Good images are sampled), vs B) a VGG16 model that was trained a balanced dataset by one of the four methods shown.

In terms of Confusion Matrix classifications, all methods yielded a slight decrease in the cost per test sample and a mix of other reactions if each type is broken down individually. Overall, the oversample method showed a significant improvement of classifying defective welds (labeling TN instead of FN). Synthetic padding showed slight improvements over the imbalanced dataset, but not enough to draw any relevant conclusions. In addition, Cost-Learning closely replicated the results of the original dataset. More VGG16 results should be gathered for this experiment.

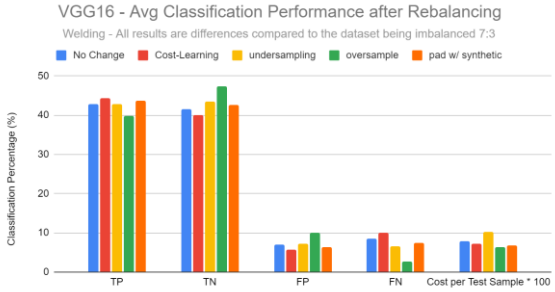


Figure 3.17: Confusion Matrix classifications for each rebalancing method. Only undersampling increased Cost per Test Sample.

For the MobileNetV3 model, the Cost-Learning method had the largest cost decrease, but the smallest change in loss. undersampling had the second largest decrease in

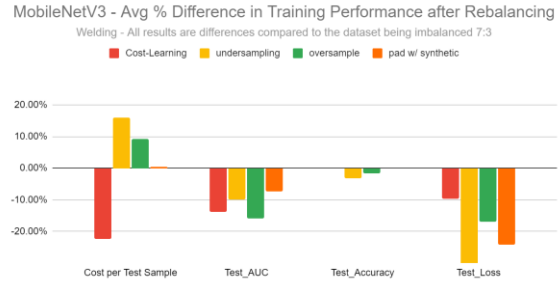


Figure 3.18: Bar chart showing the difference in performance metric between A) a MobileNetV3-small model that was trained on a sampled version of the original 2000 image real weld images and no synthetic (where 1200 Defect & 600 Good images were sampled), vs B) a MobileNetV3-small model that was trained a balanced dataset by one of the four methods shown.

The MobileNetV3 Confusion Matrix classification results show that Cost-Learning had smallest Cost per Test Sample value, followed by synthetic padding and oversampling. This can be misleading since Cost-Learning is the only method with imbalanced Cost classifications ({0, 0, 0.667, 0.334}). Even though the method performed nearly identically to the unchanged imbalanced dataset. For MobileNetV3, these results do not provide a clear 'best rebalancing' method compared to using the dataset as is without class weights during training.

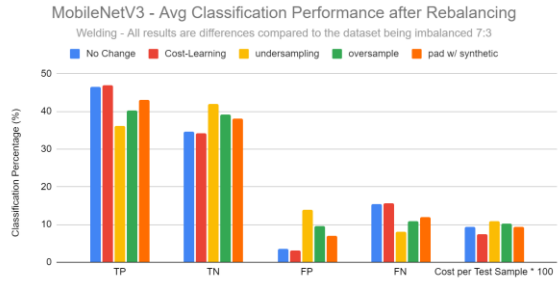


Figure 3.19: Confusion Matrix classification for each rebalancing method. Only oversampling decreased Cost per Test Sample.

Overall, it is not clear for the welding use case if any method provides a clear benefit for rebalancing the dataset. For VGG16, oversampling or synthetic padding may yield slightly better FP performance. For both models, undersampling negatively impacts performance. For this use case, removing samples for the sake of balance is not a good idea.

4.IV. Discussion

Through two use cases, five experiments and 56024 generated synthetic images from UE, this report explored how synthetic images can be used to train published CV models. These results touch on 1) how synthetic images affect CV models, 2) if having more synthetic images to sample improved outcomes, 3) can padding imbalance datasets with synthetic images provide better outcomes than other popular rebalancing methods.

A. Effects of using Synthetic Training Data

Based on Experiments 1, 2, 3, and 4; the VGG16 model showed the greatest test accuracy and test loss benefits when synthetic images are a minor supplement to real data (RSR 75:25 or 50:50). In both use cases, RSRs of 50:50 yielded performance improvements in binary classification. This 1:1 ratio of real-to-synthetic images matches what the ECCV submission reported as their best performing training set mixture [ECCV18].

The largest benefit that these experiments do support is that using some synthetic images may not increase performance, but it also will not harm a transfer learning CV model. In use cases where real images are scarce like, being able to reallocate 25-50% of your training set samples to the validation and test sets is extremely valuable. Every experiment for VGG16 and all but two for MobileNetV3 (Experiment 1 & 2) show that RSR the model performs similarly with 25-50% synthetic training data as it does 0% synthetic. Several of the papers cited in this report use as low as 200 real images in its paper. With an 80/10/10 data split, that leaves only 20 samples for each validation and test sets. If 50% of their training set was replaced with synthetic images, the validation and test set sizes increase to 60 each (since 80 synthetic samples are added. (80+80)/60/60=280).

Notes on Overfitting

One challenge of these results is that the first test of RSR 100:0 (before synthetic data was introduced), the VGG16 model was overfit to the training set. Looking back at Experiment 1 and all other Experiments, nearly all experiments show a large gap between training and test metrics. This is problematic because it implies that the models are overfitting to the training data (real, synthetic, or mixed). Adding synthetic data to provide more latent variation in the image feature space marginally narrowed the training to test gap in model metrics.

The cause of overfitting could be for different reasons; the 80/10/10 dataset split may be too restrictive and a 70/10/20 might be more appropriate. Or the VGG16 could be too large to use for these binary classification tasks. (even with a reduced number of trainable parameters). It is possible that the VGG16 model had enough parameters to overfit both the real and the synthetic image sets. Thus, providing multiple paths for neurons to best match what image it was given. More research is needed to see if there is a connection between the number of trainable parameters in a model and classification task generalization.

i. Notes on MobileNetV3-small Training

Unlike VGG16, the MobileNetV3-small model struggled to provide a clear connection of its own between synthetic images and model performance. MobileNetV3 reacted in similar ways to VGG16 like narrowing the training-test gap until after RSR 50:50, but the model was significantly more sensitive to the synthetic image datasets. It is possible that there were internal issues to how MobileNetV3 was being trained. During the training process, the best epoch of training was on average the first epoch to be judged for early stopping. In addition, while trying on the Cat_Dog synthetic dataset for RSR 25:75 and lower, the test loss of the model could not converge. The validation loss diverged after 2-3 epochs like what happened in Gastelum's results, but with a training set with up to 2000 real images [Gastelum20]. However, in Experiment 4, VGG16's validation accuracy was slower to respond to every epoch's error gradient. As having more synthetic data created two separate feature space minimas (see Figure A.VII.3 in the appendix).

For various accuracy and loss model training graphs showing converge and divergence, please see *A.VIII. Model Training Charts* section in the appendix For More Information.

B. Synthetic Availability's Effect on Performance

Based on Experiment 3's results and these two binary use cases (Cat_Dog & Weld Defect), there is no statistically relevant change in model performance if there is more synthetic data to be sampled. These results could be different if there was more varied synthetic data (such as being collected in full UE environments) or in different use cases. It is possible that having more synthetic samples can allow for more variation in the training set, but the way that the training set is sampled would also have to be more advanced. Furthermore, as the number of generated samples increases, each sample needs to be increasing

variation amongst the images, not repeating the feature space. The variations collected from the python state-machine could not be varied enough to expand the feature space. One interesting theory of expanding this need for variation to an infinitely large synthetic dataset is about becoming a simulation. If you require a sufficiently large synthetic dataset, conceptually the model might become the simulation that generated the images by being able to account for every little variation of 'the real world'.

Please consider the following recommendations to improve collected synthetic data. First, utilize randomness in every facet of 'setting up' a screenshot (like the Welding collection), instead of collecting images on a set track with hardcoded values (like the Cat_Dog collection). Second, in addition to sampling the synthetic collections proportionally (like what was done in this paper), a developer should proportionally sample the collection's meta-data (model, lighting, etc). Proportionally sampling based on the collection's meta-data can help prevent the oversampling of one 3D model over another to maintain image variation. Lastly, try incorporating more expansive UE environments during image collection. Only the UE test environment is used in this experiment, there are fully fleshed out landscapes, cities, and environments ready to insert the desired 3D models into.

C. Efficacy of Rebalancing datasets with Synthetic data

Yes. Synthetic data can be used to rebalance an imbalance dataset if the synthetic data has enough variation. Synthetic Padding can be used to balance the minority class so more can be reallocated to the validation and test sets, if developers confirm the performance of a model does not change with synthetic padding. In terms of performance, Experiment 5 details how synthetic padding does not provide a clear advantage over other common sampling methods. It is possible that by mixing Cost-learning (class weighting) and synthetic padding that there would be a greater benefit. More research on more use cases will need to be done to make a clear determination.

Despite how rebalancing though synthetic padding performed, this is a reminder that undersampling without enough remaining data can harm a CV model more than it can benefit.

D. Quantifying the Domain Gap between Real and Synthetic images

One way to quantify the domain gap between real and synthetic images is to use the model's Validation Accuracy and Loss training graphs. As designed in Experiment 4, Figure 3.7 provides a way to visually represent the differences between a training set with and without synthetic data. The key is that the number of real images in the dataset needs to stay the same. With the number of real images changing from test to test (like in Experiments 1, 2, and 3), it is unclear if the change in performance is due to the added synthetic data or the loss of the real data.

E. Commentary on created Image quality

The 3D models used in the Cat_Dog and Welding use cases were fine representations of real life. They shared the shapes of those in real life and were quick to implement in the UE Editor. Dozens of models and weld defect variations can be created for little cost ($12 for the base metal pack, all cats and dogs were free) and little startup time. After creating the python script, the Unreal Engine could output 80 image variations per minute. However, it is the author's opinion that the generated images struggled from two critical flaws: resolution and feature variation. Resolution is the factor that determines how detailed the models or textures can be while zoomed in. Feature variation being how different the images are from one another.

The 3D Cat_Dog model's resolution is controlled by the 1) pixel resolution of their textures and 2) the polygon count that outlines the smoothness of their body. After reviewing the model's descriptions, the authors report that most of the models are made from 10,000 Polygons. According to 3D Ace, 10,000 Polygons is considered a low-detailed character where the most complex environments can reach 60,000 polygons [3D Ace]. Therefore, the cat and dog models used in this report are of lower quality. In addition, the models lacked a 'skeleton' component, so it was impossible to adjust the limbs of said models to place them in different poses to increase image variation in the dataset.

The weld textures are constrained for a different type of resolution, pixel count. All of the welds provided in the Samur Art 4K Metal Panel Material Pack are described to be "4K". It is true that the bitmap files that contain the groups of weld seams are 4096x4096 pixels, but the thickness of the weld seams in those 4096x4096 bitmaps are only ~35 pixels wide. Therefore, any created weld defects have less than 35 pixels to create dynamic ranges

of color and light refraction. One alternative would be to diffuse the weld seam into a larger bitmap to get more pixels to work with. Regarding the defect design, the weld defect textures did provide significant feature differences since the size and shape of the defects are randomized, but the range of that randomization was limited. Also, more can be done to additional variation, such as using more than 6 noise texture maps to merge with the 'base color' textures.

One idea to improve image variation is to collect the images in more dense environments (as mentioned at the beginning of the Discussion). All of the created synthetic images for this research are taken in the UE project default test environment, a place without external shapes and objects. A blank canvas. There are maps pre-made in the UE marketplace such as a city block, a park, a spaceship, Italian riverways, etc. Having more complex objects in the background and foreground of the image could improve training results.

Regarding the idea of using a more recent version of Unreal Engine (v5.3 has been released as of writing). It is the author's belief that placing these low-polygon-count objects in Unreal Engine v5.3 will not yield significantly different results since the limiting factor is the model resolution and not the UE's lighting engine.

F. Practicality of generating Synthetic Images

UE created screenshots of cats, dogs and welds that are relevant representations of what can be done to gather a dataset quickly. With prior knowledge of a graphical engine and python, a user can generate a screenshot in a fraction of a second. They can perfectly control how the images are made, what environments to use, and what noise to add. If your application requires regular shapes and not a lot of detail (such as an airplane vs bicycle classifier like what the ECCV submission created) then generating your own synthetic data can save you time, money, and risk of errors should the data you do collect be incorrect, corrupted, or lost [ECCV18]. The ability to regenerate your entire dataset is a powerful tool to have in case the unexpected happens or if there needs to be a change in project scope. Alternatively, any use cases that requires complex shapes or finer details (like fur), generating your own data might not be practical unless there is a skilled graphic designer creating the models and is able to overestimate the required model variations to train a successful model.

G. Commentary on using GANs for Image Production

Regarding the use of GANs in synthetic image creation. As of the date of publication, open-source tools can produce highly detailed images with user provided text prompts. These images can be saved and stored for training. This is an alternative process compared to the process done in this report of creating a UE environment and setting up the pipeline to generate synthetic images.

There are several engineering and business considerations to using GANs as your image pipeline. First, GANs produce images by selecting from learned distributions of features. Meaning that each image is unique but also not re-producible unless you know the random seed that created it. From a practical sense, if you store GAN images and lose the images, they are gone. Also, you do not control what variations each image will emphasize. Establishing a UE pipeline allows users to reproduce images using the exact variations that they want and offers a nimble way of generating new images as customer requirements change.

Second, if you are using an open-source tool to produce your synthetic images, you do not own the tool or control how you access it. Providers of open-source tools could limit access to these models such as when Reddit and Twitter began to charge for access to their API in 2023. What is free today may not be free tomorrow. In-house developed GANs can be trained to provide slight variations, such as the ECCV paper's use to improve image 'realism' of created synthetic images. Regardless, developers should consider all the risks and complexities before relying on them exclusively.

5.V. Contributions

This work contributes to research related to using synthetic images in model training. First, having low quality 3D models (<10,000 polygons) can replicate parts of the feature space of complex shapes (animals) and minor alterations (welding defects). It is possible to train CV models on only low-quality 3D model images and produce significant classifier accuracy (as shown in Figure 4.1 when training with 75% real samples). Second, CV models with millions of parameters can train on small amounts of real images and yield significant classifier accuracy. This supports an alternative for needing thousands of images to train an adequate classifier. Lastly, if large amounts of real images are available, this research also shows that low-quality 3D synthetic images can reduce model loss and possibly increase model accuracy.

6.VI. Future Research

Future research can expand upon several areas. First, how to reduce the domain gap between real and synthetic images, such as the relationship between classifier accuracy and factors such as model resolution and using UE pre-built environments. Second, these tests can be repeated to explore how synthetic images can introduce object variations unknown to your model (reducing the Curse of Dimensionality). Finally, work can be done to improve the image pipeline software to create images with bounding boxes that can train more advanced CV models.

Specific to this research, further information can be gathered by 1) repeating the RSR and Rebalancing experiments for both use cases more than three times, 2) repeating each experiment with a simplified CNN (made from three decreasing kernel Convolution layers & two Fully-Connected layers for instance) to better understand the level that the published models are overfit, under parameterized or the synthetic images poison the model, 3) repeat the experiments with less trainable parameters as proposed by other literature (that suggest transfer learning models to be ~10%-30% trainable) [Pan10], and 4) repeat the Welding experiments with even more variation provided by other weld textures in the 4K Metal Materials pack or in UE environments other than the test environment. Regardless of use case, experiments like these should be repeated to better understand the current domain gap between simulation and the real world.

7.VII. Conclusion

Variation. The key to a successful CV classifier is having a dataset with a diverse feature space to learn from. Through the documented RSR experiments, this research can conclude that there are benefits to supplementing a CV training set with synthetic Unreal Engine images. For particular RSRs, replacing training set real images with synthetic images can decrease your CV model's loss and narrow the training-test accuracy gap. This also allows developers to move valuable real samples from the training set to the validation & test sets. However, choosing a model architecture requires experimentation to balance model size with the application's hardware limitations. The 'server based' VGG16 model benefitted from the injection of synthetic training images, while the on-chip focused MobileNetV3-small model was hampered. Investing in processes to produce synthetic data can reduce the work, cost, and risk of collecting real images manually.

Source Code

Please see the following Git repository for all source code related to this report. The repository includes 1) the python base class to implement a UE python callback, 2) proof of concept python scripts to edit a UE environment for any reader's reference, 3) python scripts to train the VGG16 and MobileNetV3-small model and 4) any other assisting scripts.

Link:

https://github.com/smutnyjw/training_cv_models_on_ue_images_jws

APPENDIX

A.I. Key Terms and Concepts

Paper Specific Terms

Real:Synthetic Ratio	(RSR) The ratio of real images to synthetic images used in a model's training set. This is heavily used to describe all results in this report.
Synthetic Utilization Rate	(SUR) The percentage of synthetic data included in a dataset versus what was available to be sampled. Used primarily in Experiment 3 of this report

Computer Vision Terms

Model Generalization	How well a Machine Learning model performs on samples it has not trained on. Characterized by the difference in a model's performance metrics during training and during testing.
Latent Variation	The amount of variance in features and/or the amount of variety in the images that make up a dataset. Examples include differences in camera perspective, backgrounds, foregrounds, color, shapes, positions, etc.
Transfer Learning	During model training, only apply loss gradients to specific layers of a model.

Graphic Design in Unreal Engine Terms

UE Editor	The central place in which a 3D simulated environment is viewed. This is where you can control the actors and their parameters. Python scripts can be executed from a UE Editor's command line if the Python API is enabled. As explained in the Generating Synthetic Images from the Unreal Engine 4.27, an executed python script through the python API will run on the same thread as the editor. Therefore, you must provide a frame buffer to your actions
Actor Mesh	The polygons that make up an actor's (object's) shape.
Material	The surface characteristics of an object (or mesh) in a 3D environment. A resulting output of Textures, Layering, UV Mapping, shading, and other techniques are what we see as the 'skin' of an object.
Map	A collection of pixels represented as a vector of one to four dimensions. [Red, Green, Blue, Alpha]
Texture map	A characteristic of an object that can be simulated over the course of all pixels in a map. Most common: • Basecolor: The RGB color • Metallic: How similar a surface is to metal. • Roughness: Amount of light reflection on a surface • Ambient Occlusion: Definition of how light should reflect. • Normal: Definition of when light should reflect. Simulate height.
Texture Parameter	Common ways to describe a particular texture, such as its metallic, roughness, etc qualities. This affects the shininess and light reflections of the object. Normally represented as a float between 0 and 1.
Texture mask	A way to selectively mix a texture with another in a material
Rendering	The act of converting a 3D simulation into a 2D image of a particular resolution.
UV Mapping	Flattening a 3D model into a 2D image. 'U' and 'V' reference the resulting 2D image's axis, not the x,y,z axis of the 3D model.
SunSky Lighting	A light ray generator to act like the sun
Individual Light	A light source of light rays that acts like a light bulb or spotlight
Camera Location	The geographic coordinates of a CineCameraActor
Camera Tracking	Boolean setting to force a camera to always follow an actor's center point

Welding 101

Terms

A Metal Weld	Using heat to break the molecular bonds of different metals and then reforming bonds to another material (filler or another piece of metal). Non-ionizing gas is used to create an environment for welding to occur and prevent weld contamination.
Weld Filler	Another alloy is added between two metal plates that you wish to weld. The filler bonds to the plates when melted and results in the weld when cooled.
MIG Welding	Metal Inert Gas. Welding is where a filling alloy is extruded through a welding gun into a gas mixture to facilitate an arc. Heating the metal plates and filler material. Bonding them when cooled [Openworks].
TIG Welding	Tungsten Inert Gas. Welding where an electrode is at the end of a welding gun. Gas is emitted from the gun to facilitate an arc between the electrode and desired metal. Adding filler material via a separate filler rod is optional [Openworks].

How weld defects form?

The quality of the weld is determined by human technique (speed), heat of the metals, coverage of shielding gas, and metallic properties of the destination metal & filler. Defects in a welding material can occur when there is a mismanagement in any of these. The biggest concern with a weld is balancing a welder's output voltage (temperature) and feed rate (how much filler material to output) to ensure that the weld has sufficient penetration (the filler material bonds to the entire depth of the weld [Padayao].

Shielding gas such as Argon or Argon Carbon Dioxide mix is emitted from the arcing tool that heats up the metal. This gas is meant to protect the molten pool of metal that eventually creates the weld when cooled. Defects can occur when particles or gasses mix get past the shielding gas and mix with the molten pool before it cools. Potentially causing splatter, gas bubbles to form and cause porosity or internal holes [Openworks].

Significant differences in metallic properties of the base and the filler material such as heat diffusion and thermal conductivity can make forming a good weld difficult. Thermal conductivity describes how quickly a metal heats up while heat diffusion describes how quickly heat spreads throughout a metal. If a base and filler materials have different properties it is possible that one can get too hot and melt through material while the other is still heating up. Or inversely, one is ready while the other is too cold for the weld to adhere too. The wrong choice of filler for your base material can cause holes in your material or incomplete welds [Padayao].

Finally, operator speed, material output rate, and chosen heat output control the molten pool's penetration into the weld and completeness of the weld as just discussed with metallic properties. Think of managing the molten pool like using a hot glue gun. Move too fast or output not enough material and your weld will not penetrate through the entire weld (resulting in a weak weld and potential cracking). Moving too slow and outputting too much material results in splatter, contamination, and excess [Padayao].

Current Weld Defect detection practices

According to professional welder Brandon Padayao, there are several checks to uncover the integrity of welds involving different parties. The most common defect detector is the welder. All professional welders are certified by the American Welder Society to do their job safely and effectively. Becoming certified signals, the skills to visually identify at the point of creation if a weld is sufficient or not. The second check is other professional inspectors that perform various tests to ensure the weld meets specifications, such as being liquid tight. Tests involve flashlight inspection, applying dyes to find holes, applying water to find leaks, x-rays to check weld penetration, or even cutting the weld open to visually inspect the operator's performance (the welder then re-welds the cut. The last check comes from post-work visual inspections [Padayao]

A.II. Installed Python Software Packages

Package	*Version*
absl-py	*2.1.0*
archspec	*0.2.1*
astunparse	*1.6.3*
boltons	*23.0.0*
Brotli	*1.0.9*
cachetools	*5.3.2*
certifi	*2024.2.2*
cffi	*1.16.0*
charset-normalizer	*2.0.4*
colorama	*0.4.6*
conda-package-handling	*2.2.0*
conda_package_streaming	*0.9.0*
cryptography	*41.0.7*
distro	*1.8.0*
dm-tree	*0.1.8*
flatbuffers	*23.5.26*
gast	*0.4.0*
google-auth	*2.27.0*
google-auth-oauthlib	*0.4.6*
google-pasta	*0.2.0*
grpcio	*1.60.1*
h5py	*3.10.0*
idna	*3.4*
jsonpatch	*1.32*
jsonpointer	*2.1*
keras	*2.10.0*
Keras-Preprocessing	*1.1.2*
libclang	*16.0.6*
libmambapy	*1.5.6*
Markdown	*3.5.2*
markdown-it-py	*3.0.0*
MarkupSafe	*2.1.5*
mdurl	*0.1.2*
menuinst	*2.0.2*
namex	*0.0.7*
numpy	*1.26.4*
oauthlib	*3.2.2*
opt-einsum	*3.3.0*
packaging	*23.1*
pip	*23.3.1*
platformdirs	*3.10.0*
pluggy	*1.0.0*
protobuf	*3.19.6*
pyasn1	*0.5.1*
pyasn1-modules	*0.3.0*
pycosat	*0.6.6*
pycparser	*2.21*
Pygments	*2.17.2*
pyOpenSSL	*23.2.0*
PySocks	*1.7.1*
requests	*2.31.0*
requests-oauthlib	*1.3.1*
rich	*13.7.1*
rsa	*4.9*
ruamel.yaml	*0.17.21*
ruamel.yaml.clib	*0.2.6*
setuptools	*68.2.2*
six	*1.16.0*
tensorboard	*2.10.1*
tensorboard-data-server	*0.6.1*
tensorboard-plugin-wit	*1.8.1*
tensorflow	*2.10.1*
tensorflow-estimator	*2.10.0*
tensorflow-io-gcs-filesystem	*0.31.0*
termcolor	*2.4.0*
tqdm	*4.65.0*
truststore	*0.8.0*
typing_extensions	*4.9.0*
urllib3	*1.26.18*
Werkzeug	*3.0.1*
wheel	*0.41.2*
win-inet-pton	*1.1.0*
wrapt	*1.16.0*
zstandard	*0.19.0*

A.III. Model Architecture changes from Published Models

Output Layers

Specifically for the model's output layers, minimal changes are made to the output layers of both VGG16 and MobileNetV3 models. This is done to maintain the architecture of the original models and the author's thought process behind their perspective research. However, changes are made to reduce the size and complexity of the models so that they are better aligned with the change from the 1000 class classification task to only a binary classification task. The reduced output layers result in an approximate 84.7% and 46.6% reduction in model size for the VGG16 & MobileNetV3 models respectively.

	VGG16	**MobileNetV3**
Total Parameters in the published model	138,357,544	2,554,968
Total Parameters in the transfer learning model	21,203,521 (15.3%)	1,365,617 (53.4%)

Figure A.III.1: Summary of model size reduction from the 1000 class case to binary classification [Simonyan14][Howard19].

In conjunction with reducing model size, only a fraction of the model's parameters are trainable (thus transfer learning). In this research only 41.7% and 39.4% of parameters are affected by learning gradients for layers in the output and non-output layers. The primary cause for this reduction is the reduction in output layer size for both models. The VGG16 output layers are a pair of Fully-Connected (FC) Dense layers which are reduced from 4096 nodes each to 256 nodes. The MobileNetV3's pair of Convolutional output layers are also reduced from 1024 filters each to 512 filters and 256 filters respectively. Both reductions are chosen to yield a total trainable parameter percentage of approximately 40% for each model. 40% is the target percentage of trainable parameters due to the disproportionate size of the model output layers even after the previously stated reductions above and a desire to retrain at least one convolutional layer in each respective model. More review should be done to determine what is the best approach to setting the number of trainable parameters.

	VGG16	**MobileNetV3**
Total Parameters in the transfer learning model	21,203,521	1,365,617
Total Trainable Parameters	8,848, 641 (41.7%)	538,433 (39.4%)
Further breakdown		
Number of parameters in the non-output layers	14,714,688 (69.4%)	939,120 (68.8%)
Percentage of parameters in the non-output layers that are trainable	26.7%	20.8%
Number of parameters in the output layer	6, 488, 833 (30.6%)	426,497 (31.2%)
Percentage of Parameters in the output layers that are trainable	100%	100%
Total Conv Layers	13	28
Trainable Conv Layers	1 trainable conv layers (3 trainable layers total)	11 trainable conv layers (24 trainable layers total)

Figure A.III.2: Summary of the parameters in the VGG16 and MobileNetV3 transfer learning models used in this report. This documents the location of parameters for the reduced models to binary classification.

The tables below present more specific architectural changes between the published paper and the models used in the paper. Details include output layer size, amount of layer regularization, additional learning techniques, etc. Any row in **BOLD** represents a setting that is different from the published model's settings.

VGG16		
Settings	Original Paper	Transfer Learning Model
Optimizer	Mini-batch SGD	Batch SGD
Optimizer settings	Momentum=0.9	beta_1=0.9 (default) beta_2=0.999 (default) epsilon=1e-7 (default)
Initial Learning Rate	**0.01**	**0.0001**
Batch Size	**256**	**16**
Learning Rate Decay	**0.1 when Val_accuracy stopped improving**	**0.95 every epoch**
Dataset Used	**ILSVRC 2012-2014 (1.45 million images)**	**Various (Cats_Dogs & Welding)**
Dataset Split	**1.45M Total (1.3M/50k/100k)**	**Various (up to 10k total) (80%/10%/10%)**
Input Image Size	(224, 224, 3)	(224,224,3)
Output Layer Information		
Number of Nodes per FC Dense Layer	**4096**	**256**
Dropout Rate in Output Layers	None	None
Last Layer Activation Function	**Softmax, 1000**	**Sigmoid, 1**
Kernel Regularization	L2=5e-4	L2=5e-4
Kernel Weight Initialization	Glorot & Bengio (2010) o Normal Distribution	GlorotNormal (see documentation)
Early Stopping		
Metric	**Val_Accuracy**	**Val_Loss (Cat_Dog) Val_AUC (Welding)**
Minimum Delta	**0.0**	**0.0005**
Patience (# epochs)	**unknown**	**7**
Initial Training		
Number of Epochs before evaluation	**Unknown**	**5**

Figure A.III.3: Summary of the training differences between the originally published VGG16 model and what is used in this report [Simonyan14].

Deployed VGG16 Model Architecture

The following output layer information is taken from the Keras API's implementation of the VGG16 model. Only the number of parameters in each Fully-Connected layer is different between the implementation below and the published VGG16 model. The published model uses 4096 nodes in the *fc1* and *fc2* Dense layers.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Trainable
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688  Y

 flatten (Flatten)           (None, 25088)             0         Y

 fc1 (Dense)                 (None, 256)               6422784   Y

 fc2 (Dense)                 (None, 256)               65792     Y

 Prediction (Dense)          (None, 1)                 257       Y

=================================================================
Total params: 21,203,521
Trainable params: 8,848,641
Non-trainable params: 12,354,880
_________________________________________________________________
```

Figure A.III.4: Model architecture of the VGG16 transfer learning model. 'vgg16' is the keras pre-built VGG16 model [Simonyan14].

The tables below present more specific architectural changes between the published paper and the models used in the paper. Details include output layer size, amount of layer regularization, additional learning techniques, etc. Any row in **BOLD** represents a setting that is different from the published model's settings.

MobileNetV3-Small		
Settings	Original Paper	Transfer Learning Model
Optimizer	Tensorflow RMSProp	Tensorflow RMSProp
Optimizer settings	Momentum=0.9	Momentum=0.9 (default)
Initial Learning Rate	**0.1**	**0.0001**
Learning Rate Decay	**0.99 every 3 epochs**	**0.95 every epoch**
Batch Size	**4096**	**100**
Input Image Size	(224, 224, 3)	(224,224,3)
Dataset Used	**ImageNet**	**Various (Cats_Dogs & Welding)**
Dataset Split	**Unknown**	**Various (up to 10k total) (80%/10%/10%)**
Output Layer Information		
Dropout rate in Output Layers	0.8	0.8
Last Layer Activation Function	**Softmax, 1000**	**Sigmoid, 1**
Number of Filters per Convolutional Layer in the Output Layers	**{1024, 1024}**	**{512, 256}**
Kernel Regularization	**L2=1e-5**	**L2=1e-4**
Kernel Weight Initialization	**Unknown**	**GlorotNormal (see documentation)**
Early Stopping		
Metric	**Unknown**	**Val_Accuracy (Cat_Dog) Val_AUC (Welding)**
Minimum Delta	**Unknown**	**0.0005**
Patience (# epochs)	**Unknown**	**7**
Initial Training		
Number of Epochs before evaluation	**Unknown**	**5**

Figure A.III.5: Summary of the training differences between the originally published MobileNetV3-small model and what is used in this report [Howard19]

Deployed MobileNetV3-small Architecture

The output layers for the MobileNetV3-small architecture were recreated based on the published paper's description. The main architecture changes are 1) the use of AvgPooling instead of GlobalPooling, 2) the number of filters for convolutional layers *Conv_2* and *logits*. Instead of 1024 filters for each Conv2D layer, 512 and 256 filters are used. Howard, et al. uses a new activation layer called hard-swish in order to complete its set of convolutional layers. Hard-Swish uses a residual network split, starting from the *Conv_2* layer and ending at the *multiply_18* layer. One of the splits is not processed further, but the other goes through several steps summarized by the equation below [Howard19].

x	The output of the pool_7x7 layer. The split starting the residual network
y	The merging of the residual network. Input into the multiply_18 layer

$$y = x * \frac{Relu(x+3)}{6}$$

Layer (type)	Output Shape	Param #	Connected to	Trainable
input_2 (InputLayer)	[(None, 224, 224, 3)]	0	[]	Y
MobilenetV3small (Functional)	(None, 7, 7, 576)	939120	['input_2[0][0]']	Y
pool_7x7 (AveragePooling2D)	(None, 1, 1, 576)	0	['MobilenetV3small[0][0]']	Y
Conv_2 (Conv2D)	(None, 1, 1, 512)	294912	['pool_7x7[0][0]']	Y
h_swish-tf_TFOpLambda_add_3 (Lambda)	(None, 1, 1, 512)	0	['Conv_2[0][0]']	Y
h_swish-re_lu (ReLU)	(None, 1, 1, 512)	0	['h_swish-tf_TFOpLambda_add_3[0][0]']	Y
h_swish-tf_TFOpLambda_multiply_6th (Lambda)	(None, 1, 1, 512)	0	['h_swish-re_lu[0][0]']	Y
multiply_18 (Multiply)	(None, 1, 1, 512)	0	['Conv_2[0][0]', 'h_swish-tf_TFOpLambda_multiply_6th[0][0]']	Y
dropout (Dropout)	(None, 1, 1, 512)	0	['multiply_18[0][0]']	Y
logits (Conv2D)	(None, 1, 1, 256)	131328	['dropout[0][0]']	Y
flatten (Flatten)	(None, 256)	0	['logits[0][0]']	Y
dense (Dense)	(None, 1)	257	['flatten[0][0]']	Y

Total params: 1,365,617

Figure A.III.4: Model architecture of the VGG16 transfer learning model. 'vgg16' is the keras pre-built VGG16 model.

A.IV. 2D Image Augmentations

The following list of augmentations are available to be applied to every image used in a training epoch. All augmentations have a uniform random change of being applied and a uniform random sampling of the value range. These are all implemented through the Tensorflow `ImageDataGenerator` class.

Augmentation	Value	Reason
Rotation range	+/-15 deg	Desire to give enough variation especially when combined with horizontal and vertical flipping. 15 seemed about right. No elaborate testing
width_shift_range	+/-10%	Since the camera is always centering the object, I wanted at least a +/- 10% shift from a (224, 224, 3) image. See the *Fill_mode* row for information on the resulting fill algorithm.
height_shift_range	+/-10%	Since the camera is always centering the object, I wanted at least a +/- 10% shift from a (224, 224, 3) image. See the *Fill_mode* row for information on the resulting fill algorithm.
Fill_mode	nearest	Between nearest and constant. Chosen "nearest" so there is continuity in the image. Other options result in hard dividing lines if combined with augmentations such as rotate or shear.
brightness_range	[0.1, 1.0]	The pictures are already bright, so more brightness was not needed. However, the range was put to near 0 to prevent any BLACK images but still try to lower the brightness. Upon inspection, it doesn't look like this setting did much.
shear_range	+/-10 deg	Desire to give enough variation. 10 seemed about right. No elaborate testing
zoom_range	0.15	Through trial and error, 15% zoom was enough to give valid variation. Anymore and there were too many close ups, and the models were too small when zoomed out.
channel_shift	30	Gave enough rgb alterations ('halo' effect) to believe that 30 is enough. Too much channel_shift led to wild color contrast that took away from the image
horizontal_flip	True	General augmentation.
vertical_flip	True	General augmentation

Figure A.IV.1: List of 2D Augmentations and values that could be applied to an image in the training set every epoch. Augmentations stack so none, all, or some may be applied.

A.V. Collecting Synthetic Images via the UE Python API

Synthetic images are collected by taking screenshots of the current UE Editor being viewed from a `CineCameraActor` and in the Editor's `Cinematic` mode. It is possible to write a python script that imports the unreal python package and then the .py file be executed in a currently active UE Environment through its terminal. The python script sends commands to the UE Editor and can manipulate numerous parts of the Editor such as an object's geospatial position, lighting intensity, displayed materials on a mesh, and texture parameters on an object's material instance. For this research, all Editor variations are done through a single python script instead of experimenting with UE Blueprint functions or other associated timetables due to the researcher's introductory knowledge of the UE.

The developed python script requires that the UE Editor make precise changes in a structured sequence (like a state-machine). The script takes a defined sequence and iterates through all model/background/lighting variations required to set the Editor in unique ways for a unique screenshot. This is possible by creating a python class to facilitate a user created software callback that is tied to the UE's internal frame counter. This means that every 'tick' of the UE software process, the python callback is executed, and UE commands are processed before the next process 'tick'. The python scripts created for this research is based on the class structure outlined by the Technical Art Engineer, Ryan DowlingSoka, to control the UE through asynchronous loops to accomplish this [DowlingSoka].

```python
class MyClass(object):
    def __init__(self) -> None:
        self.frame_count = 0
        self.max_count = 1000

    def start(self) -> None:
        self.slate_post_tick_handle = unreal.register_slate_post_tick_callback(self.tick)
        self.frame_count = 0

    def tick(self, delta_time: float) -> None:
        print(self.frame_count)
        self.frame_count += 1
        if self.frame_count >= self.max_count:
            unreal.unregister_slate_post_tick_callback(self.slate_post_tick_handle)

test = MyClass()
test.start()
```

Figure A.V.1: Base python class DowlingSoka used from UE documentation to create a 'tick' callback.

Operationally, the developed python scripts that gather all the synthetic Cat_Dog and Weld screenshots operate like a state-machine iterated via a counter. Every number of UE Editor frame 'ticks' increments the python scripts sequence counter. Based on this counter, the material parameters (like what background is showing or what defect is present), lighting position, camera position, and focused model changes. The script ends when the sequence is completed, the frame count exceeds the user defined limit, or if the script takes too many actions. These error checks are put in place to prevent the python executable from running an infinite loop. The author has not identified how to stop a python script executed from the UE Editor command line once it starts other than terminating the entire UE program.

Based on the author's research on online forums, Unreal Engine help threads, and Ryan DowlingSoka's documentation, it is the author's belief that there is one large limitation of controlling the UE Editor via tick callback. This limitation is that the python script is executed on the same thread as the actual UE Editor. The UE Editor does not appear to create a separate thread to execute the python script. Therefore, it is the developer's responsibility to not execute too many commands in a single UE frame tick. If a script callback has too many actions, then the callback won't be completed by the time the next tick executes the callback. Thus, the script drops the remaining commands. This can be mitigated by changing when the code in the `::tick()` class method is executed via the modulus operator (%). Creating a `frame_buffer` will slow down the rate at which callbacks can be executed but it guarantees that a single callback does all the actions that the user wants it to.

```python
class MyClass(object):
    def __init__(self, frames_buffer: int) -> None:
        self.frame_count = 0
        self.max_count = 1000
        self.frame_buffer = frames_buffer

    def start(self) -> None:
        self.slate_post_tick_handle = unreal.register_slate_post_tick_callback(self.tick)
        self.frame_count = 0

    def tick(self, delta_time: float) -> None:
        print(self.frame_count)
        self.frame_count += 1
        if self.frame_count >= self.max_count:
            unreal.unregister_slate_post_tick_callback(
                self.slate_post_tick_handle)
        elif self.frame_count % self.frame_buffer == 0:
            print("Do things for one callback")


frames_bw_ticks = 10
test = MyClass(frames_bw_ticks)
test.start()
```

Figure A.V.1: Appended python class from DowlingSoka to use a class paramenter 'frames_bw_ticks' to specify how many ticks are needed to fully execute each callbacak. The user should customize this value, so it is 1) large enough to fully complete the expected actions and 2) be small enough to generate images as fast as possible.

A.VI. Cat_Dog - Collecting Synthetic Images

The generated synthetic Cat_Dog dataset contains 46784 images of 3D models & accompanying textures of cats and dogs with a variety of camera angles, backgrounds, and lighting. A custom python script connects to the active UE project via the UE Python API and sends commands to the UE Editor to change the Editor's camera and environment to collect a unique screenshot. The following attributes are changed throughout the collection: camera angle (spherical position centered around a model), the 3D model in focus, the sky lighting, the background in the screenshot.

Figure A.VI.1: Examples of the UE test environment backdrop during Cat_Dog synthetic image collection.

The 3D models are free to download from websites such as Turbosquid and Free3D. Each model has different levels of quality, polygon count, texture variation, and anatomic position. Purchased models can have animations, skeletal positions, and more texture color palettes that could increase variation. In total 13 unique cat models & variations and 10 unique dog models, are set in front of 10 separate backgrounds to generate all synthetic data. Please see the References section for all 3D models used for authors and credits.

Figure A.VI.2: Examples of a cat and dog screenshot during the Cat_Dog synthetic image collection.

The used backgrounds consist of three of three main components, each with their own texture; Center Scene (directly behind the model), Setting (additional surrounding background affiliated with the Center Scene), and floor (underneath the model). During each collection, the python script flips switches in the loaded UE materials for all three of these mesh objects to change which texture is shown. Across the two collections, ten unique combinations are used. Please see the References section for image credit for all images used.

Figure A.VI.3: Description of the backdrop used during Cat_Dog synthetic image collection. A unique Center_scene, setting, and floor are used for each variation during image collection.

Collection Method

46784 images were collected over the course of two collections. The first set of images (9424) contained 4 different scenes while the second set of images (37360). The table on the following page outlines the iterated through settings to yield the total amount of images. Due to how the python script sends commands over the UE Python API and how the UE operates on the computer's threads, it is possible to lose images during collection. It is the researcher's belief that this is due to the UE operating on a set number of computer threads but not being a multithreaded software process. Thus, if the user clicks from the UE during collection, commands sent by the python script can be missed or skipped, losing that screenshot.

Screenshots are collected by moving a cat/dog model to a center point in front of the set background, in set lighting, at hardcoded camera angles. Once all camera angles have been captured, the cat/dog model is moved out of the center point out of the frame and the next cat/dog model is moved to the center. The process repeats for all cat and dog models, then the background preset rotates to the next. Once all background presets are completed then the lighting in the test environment changes. The lighting element in these data collections is the 'sun'. The lighting applied to the model and environment is simulated off earth's sun. The process of iterating through cat/dog models, background presets, and sun time variations continues until a screenshot is taken of every variation.

Please see the tables below for what specific camera angles, backdrops and light intensities were used to collect the synthetic data.

	Collection 1	
	Run 1	Run 2
Total Images	6672	2753
Total Variations	7728	2760
Image Loss (no screenshot taken)	13.7%* (6672 from expected 7728)	0.03% (2753 from expected 2760)
Variations		
Models	23	23
Angles (sets of angles that are unique to each run. Not shared)	Radi = [175, 300] Alpha = [-180, -135, -105, -90, -75, -45, 0] Phi = [55, 75]	Radi = [220] Alpha = [-120, -110, -92, -70, -60] Phi = [60, 80]
Total Lighting settings	150	[90, 140, 190]
Total available Presents	4	4

Figure A.VI.4: Description of the first Cat_Dog synthetic image collection. Note that in Run 1, 13.7% of samples were lost. This was due to the user clicking off of the UE window during python callback execution. Moving off of the UE window caused the process to slow down enough that the python script needed more frame_buffer (more ticks) to complete.

	Collection 2	
	Run 3	Run 4
Total Images	27478	9881
Total Variations	29808	9936
Image Loss (no screenshot taken)	0.078% (27478 from expected 29808)	0.006% (9881 from expected 9936)
Variations		
Models	23	23
Angles (sets of angles that are unique to each run. Not shared)	Radi = [200, 240, 290] Alpha = [-140, -105, -95, -85, -75, -55] Phi = [45, 55, 65, 80]	Radi = [200, 240, 290] Alpha = [-140, -105, -95, -85, -75, -55] Phi = [45, 55, 65, 80]
Total Lighting settings	[75, 90, 140, 170]	[unlit]
Total available Presents	6	6

Figure A.VI.5: Description of the second Cat_Dog synthetic image collection.

Cats - 10 Models + 3 model repeat with a different texture

Please see the References section for 3D model credit.

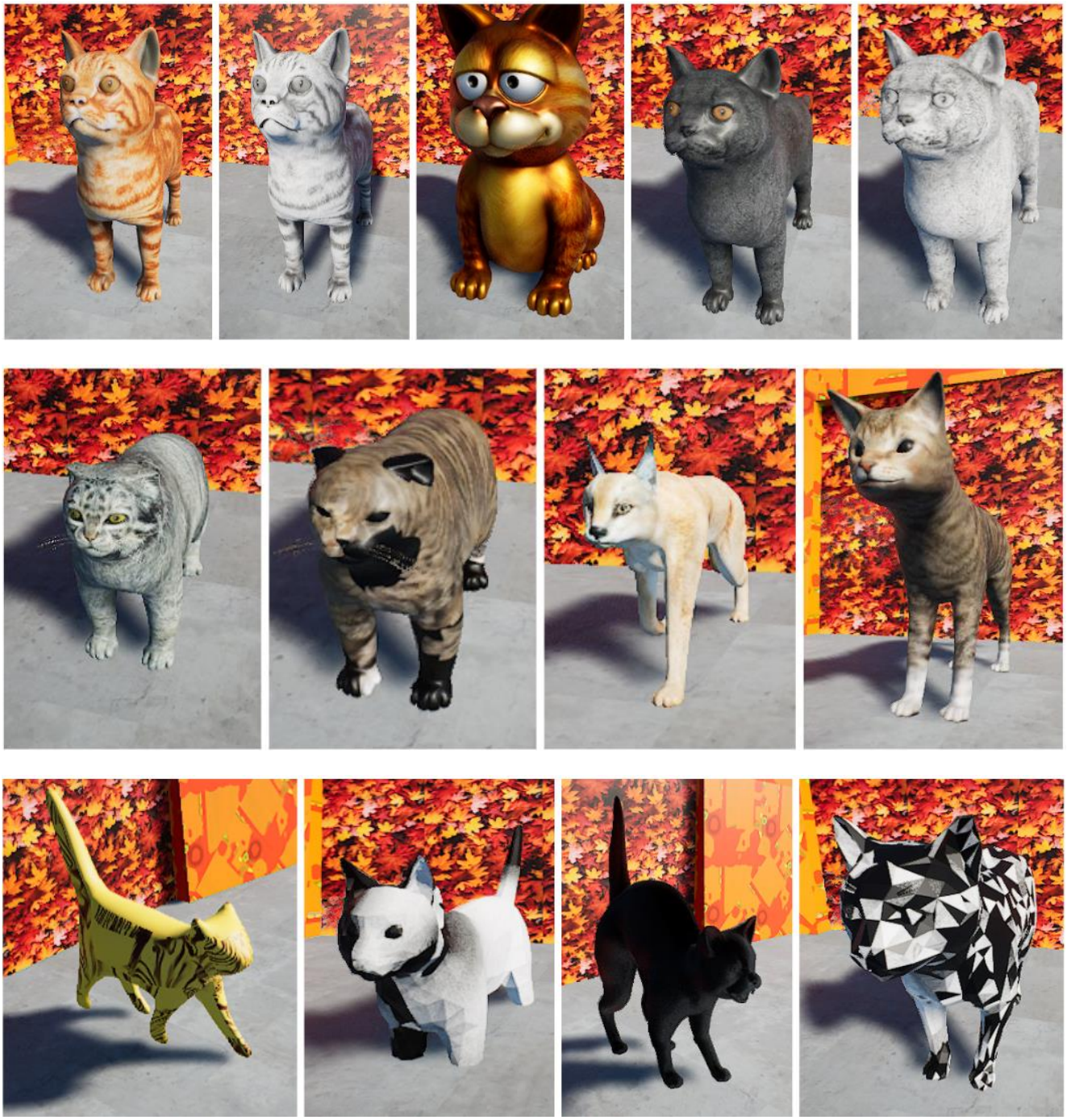


Figure A.VI.6: List of 3D model cats used in the Cat_Dog synthetic image collection.

Dogs - 9 Models + 1 model repeat with a different texture.

Please see the References section for 3D model credit.

Figure A.VI.7: List of 3D model dogs used in the Cat_Dog synthetic image collection.

Pictures of Cat_Dog Backgrounds

Set 01 (4 total)

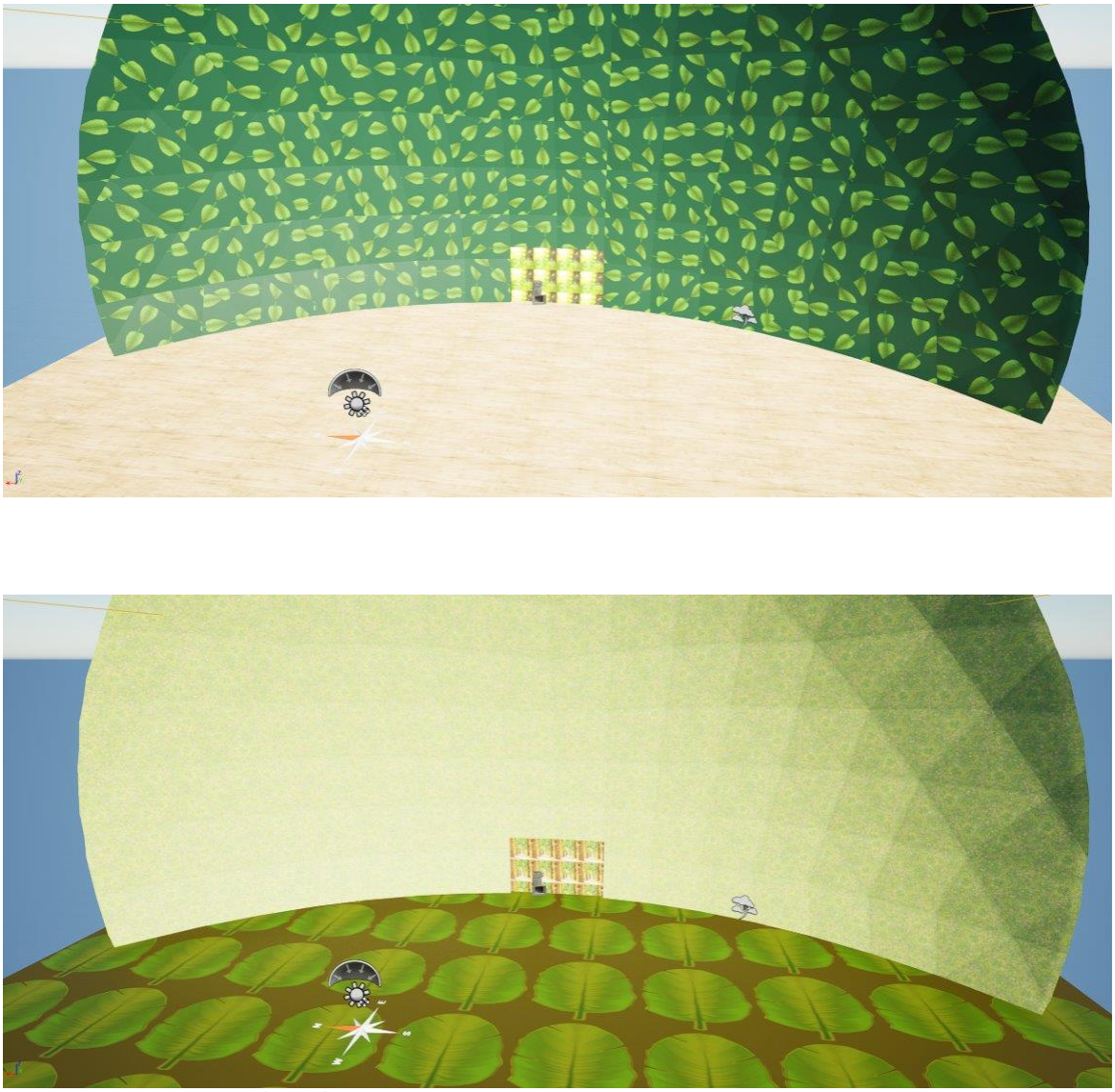


Figure A.VI.8: Backdrops for the first Cat_Dog synthetic image collection. (Top) Flowering garden, (bottom) woodland path.

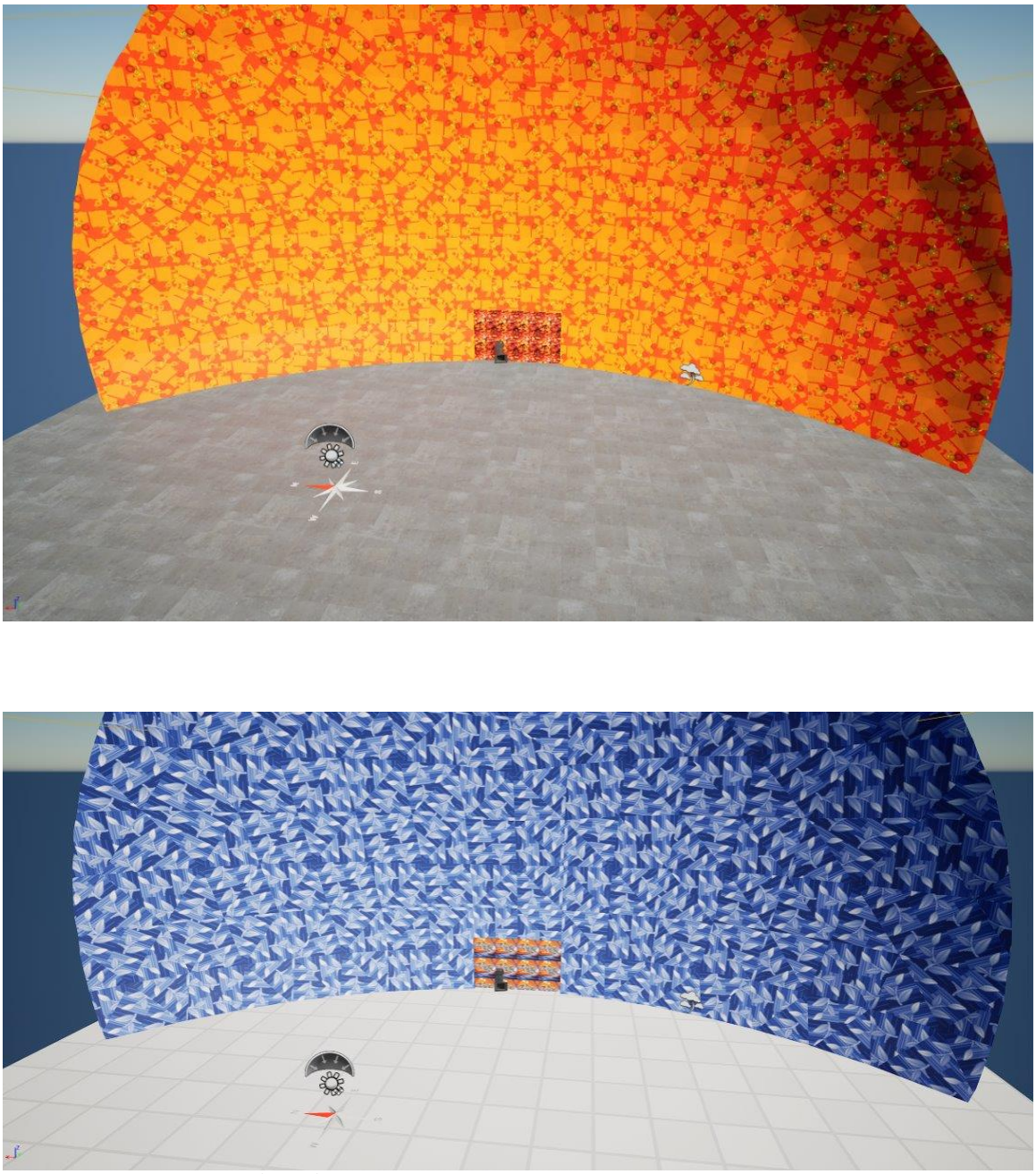


Figure A.VI.9: Backdrops for the first Cat_Dog synthetic image collection. (Top) Fall leaves, (bottom) Futuristic interior.

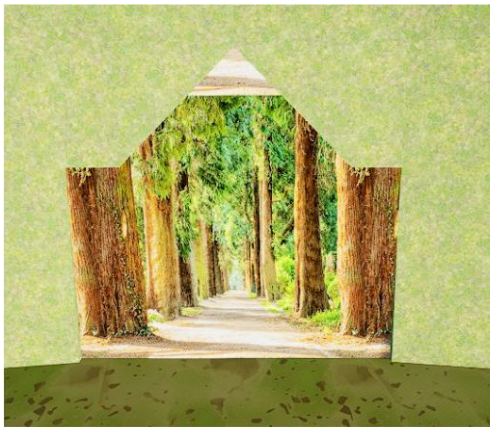
Set 02 (6 total)

[PARKS & OPEN SPACE STAFF]

[KOSHKINA]

[PRITCHARD]

[WIRESTOCK]



[HERRMANN]

[SmartFurnature]

Figure A.VI.10: Backdrops for the second Cat_Dog synthetic image collection.

A.VII. Welding - Real Dataset collection

Below is a table showing how many images are found in each dataset, how many images are used in this experiment and reasons for why images were not used.

Id	Source	Number of Used Images	Number of Images in the Original dataset	% Used	Reasons for Removals and Edits
01	[welddefect]	104	408	25.5%	Repeat images with augmentations. Images with multiple welds that could not be cropped.
02	[Celebal Workspace]	510	1027	49.7%	Repeat images with augmentations.
03	[15DEM20F]	173	1184	14.6%	Repeated images from other datasets
04	[afiys]	1279	3038	42.1%	Repeat images with augmentations. Repeated images from other datasets
05	[Weld 1]	156	156	100	Cropped images to improve quality.

Figure A.VII.1: Descriptions of the found real welding datasets that form the 2221 real images.

A map preset refers to the combination of various graphical maps to yield a resulting image. In practice, a map is an image represented by [256, 256, 256] RGB values for each pixel in a bitmap file. A material being the result of these different images being used interpreted in different ways by the graphics engine. In this use case, each map preset is the combination of the Metallic, Roughness, Ambient Occlusion and Normal texture maps. By intentionally editing each of these maps, weld defects can be simulated in the UE. Please see the appendix for more detailed information on what effect each texture map has on the resulting material.

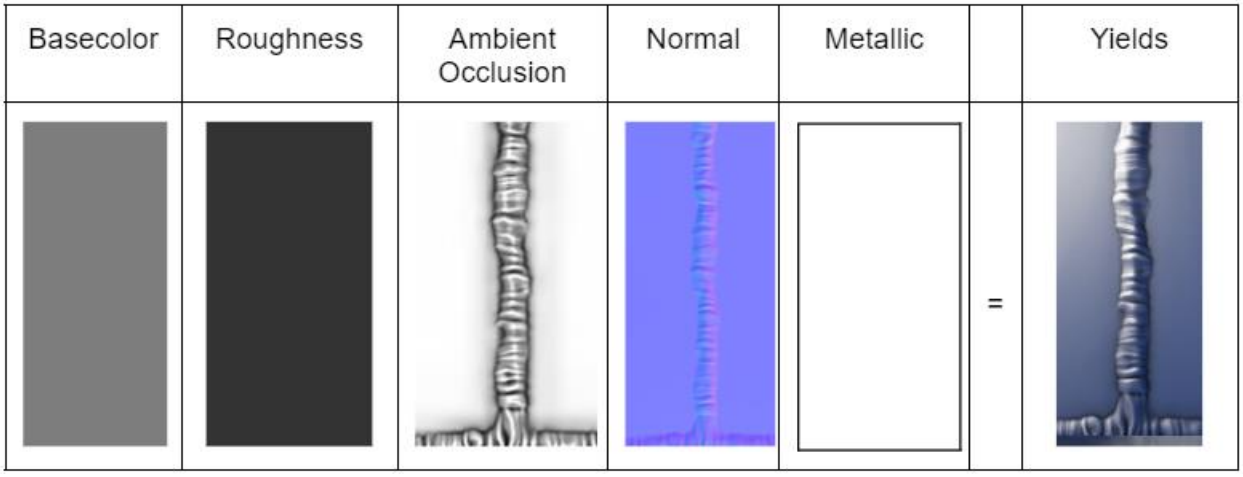


Figure A.VII.2: Demonstration of how different texture maps yield a UE Material.

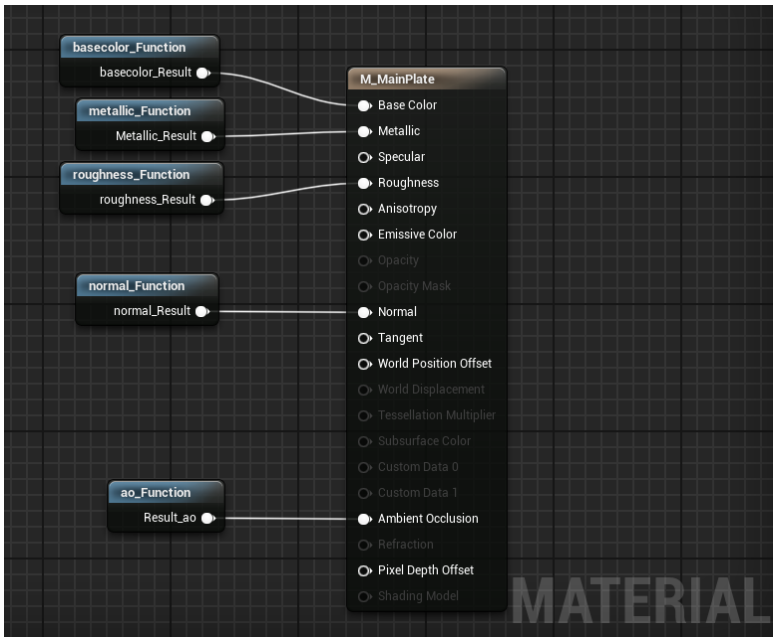


Figure A.VII.3: UE Material node view on how to create a map preset. This is repeated for all map presets.

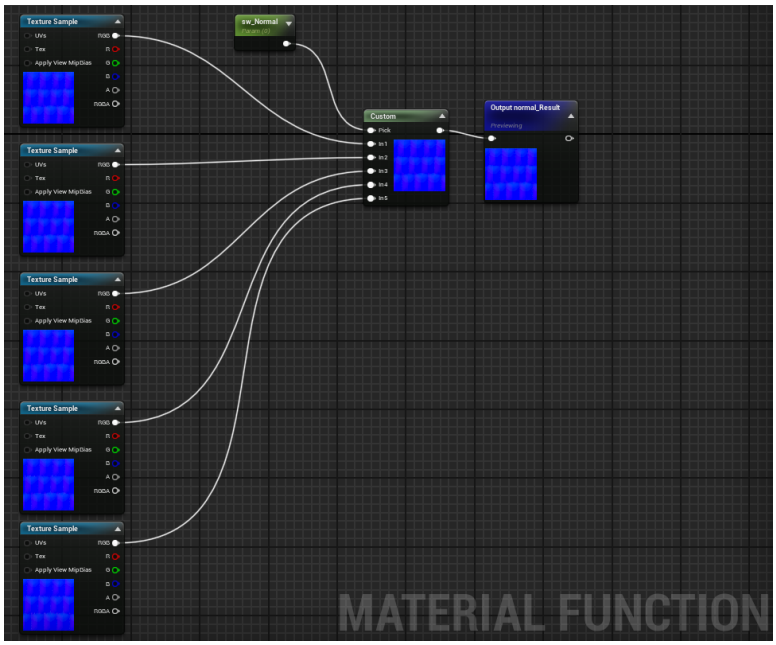


Figure A.VII.4:Example of a material function used in Figure A.VII.3 to create a map preset. The switch decides which texture map to use.

Collection Method

To collect all 8160 synthetic images, a camera and point light focused on points spread out on a flat mesh object. The purchased "Metal_Weld' material is applied to a flat rectangular plate in the UE test environment [4k metal pack]. Screenshots are taken along a set of points of interest where the weld seam is present in the original base material. The purchased material base texture size is defined from texture map bitmap files that are 4096x4096 pixels. No points of interest are defined beyond the 4096x4096 bitmap or within ~15 pixels of the image border (resulting in a 4066x4066 pixel area in which screenshots are focused). Starting at these defined borders, points of interest are recorded every 400 pixels along each horizontal and vertical weld seam. Resulting in 68 specific points in which defects are applied and the camera focuses on.

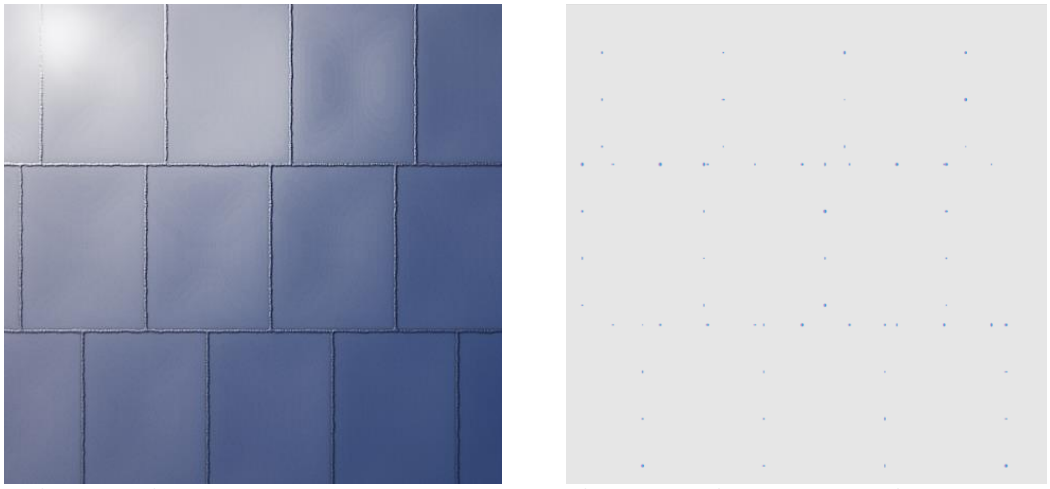


Figure A.VII.5: (left) top view of the base welding material. (right) dot locations of all 68 points of interest in which defects are created and screenshots taken.

Once the points of interest are defined, a custom python script sends commands to the UE Editor over the UE Python API each process frame tick. The process for creating the python scripts is documented earlier in this Dataset Formation section. The python script moves the UE Editor CineCamera object to each point of interest, where one of many actions in the collection sequence occurs; a screenshot is taken, the camera & light angle changes, the displayed metal plate changes color or the map preset changes. For every point of interest, four screenshots are taken from a camera and point light source that randomly is shifted in the x, y, and z directions. This allows for the displayed plate's point of interest to have different camera angles and light behavior. This creates a unique screenshot for every plate variation. Across the entire collection process, screenshots are taken for each of the defined variations in plate colors, plate map sets, plate points of interest and point of interest camera angles.

Plate Qualities	Variations
Plate base color	Aluminum, Steel, Copper, Gold, rusted metal, and a base metal
Plate map presets [Metallic, Normal, Ambient Occlusion, & Roughness]	1) no defects, 2) edits but not enough to be defective, 3) porosity defects, 4) crack defects, and 5) incomplete weld defects
Plate weld seam points of interest	68 (spaced 400 pixels apart)
Number of screenshots per plate point of interest (random Camera & Point Light offsets)	4

Figure A.VII.6: Overview of the variations used in the Weld Synthetic image collection.

Synthetic Weld Variation

Six base colors and five map presets are the basis for generating variation in the training set's synthetic images. Screenshots are taken when the weld plate is displaying each of these base colors and map presets. Collectively, the base color and map preset create unique defects at each of the 68 assigned points of interest (as shown above in the gray dotted square image in the previous section).

The various base colors highlight the different metals that could be practically (or impractically) welded with FILLER, Aluminum, Steel, Copper, Gold, rusted metal, and a base metal color. In addition, macro variations from the UE starter content are mixed with some colors to provide more variations. Additionally, the five map presets provide finer detailed changes to the welding material. Each map preset changes how the welded area appears at each point of interest. All defects have some element of randomness, so every point for every map preset has a variation of the defect, further increasing variation.

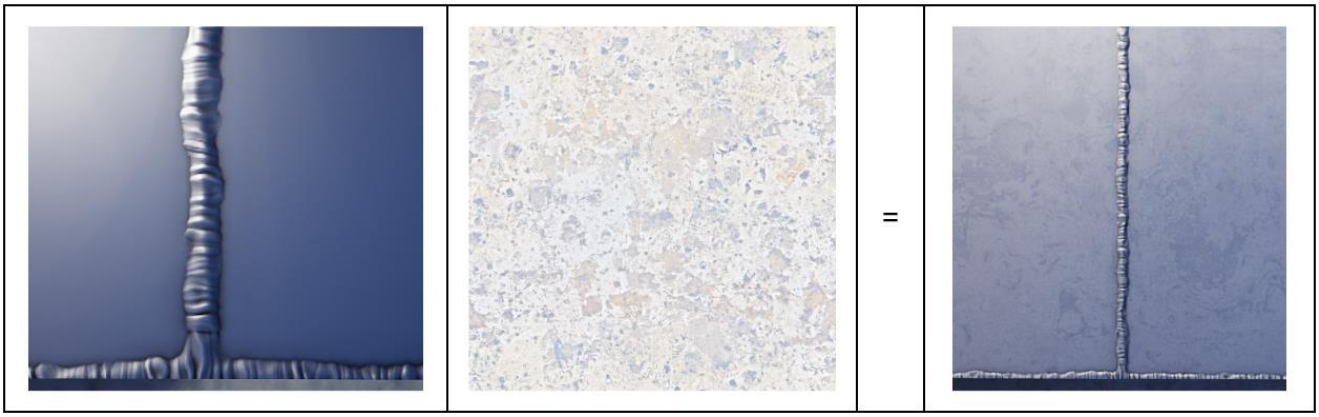


Figure A.VII.7: Example of how using a LERP (linear interpolation) node to combine base colors can yield image variation.

Defect Specific Texture Modifications

On the following pages are descriptions on how each weld defect model is created. Descriptions include which texture maps are edited and how to reach the shown effect.

Porosity

In the real weld dataset, a porosity defect is defined by elliptically shaped holes in the surface of a weld. The created weld UE defect textures use edits to the Ambient Occlusion and Normal maps to outline a feeling of depth to an elliptical shape.

Ellipse logic to fill

For pixel point (x′,y′) for ellipse with center (x, y) and radius (r_x, r_y)

$$d = \left(\frac{x'-x}{r_x}\right) + \left(\frac{y'-y}{r_y}\right)$$

If d <= 1 then fill with color

Figure A.VII.8: Overview of how a pixel is determined to be inside of the designed ellipse to be in the porosity defect.

To add a difference in height to the hole, the RGB of the ellipse fill area is set as a radial gradient where the color in the Normal map is determined by the distance a pixel in the hole is to the ellipse center. light reflects and scatters as the pixel intensity changes around the ellipse center mimicking a hole. At every applied plate point of interest, a single ellipse is created with a random x-radius, y-radius, and xy-offset from the point of interest coordinate. Unfortunately, the UE defect does not replicate the trend in real images to create multiple porosity imperfections in a cluster.

Interpolation from Blue to Pink

For pixel point (x′,y′) for ellipse center (x, y)

$$d = \sqrt{(x' - x)^2 + (y' - y)^2}$$

If d > 4 then color = PINK
Elif 2 < d <= 4 then color = LIGHT BLUE
Elif d <= 2 then color = BLUE

Key
- PINK = [242, 129, 189]
- LIGHT BLUE = [220, 203, 83]
- BLUE = [220, 158, 44]

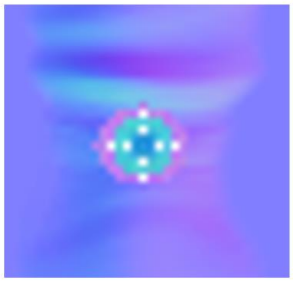


Figure A.VII.9: Overview of how the Normal map's RGB intensity is selected for the Porosity Defect

Ellipse Characteristic	Range of the Defective Texture	Range of the Non-Defective Texture
Center x	[-3, 3]	[-3, 3]
Center y	[-3, 3]	[-3, 3]
Radius x	[2, 6]	[1, 2]
Radius y	[2, 6]	[1, 2]
Texture Map Pixel Color Noise	[-10, 10]	[-10, 10]

Figure A.VII.10: Outline of the design variables to create a Porosity defect and non-defect imperfection.

Crack

A crack defect is characterized by a continuous jagged line along a weld. The UE model defect attempts to replicate this by splitting a line, several pixels thick, into multiple points then moving each of those points off the original line in the positive or negative direction. The line is multiple pixels thick to create the illusion of depth. Each split line segment starts and end points are moved positively or negatively (depending on the direction of the weld seam) to create the jagged shape. All pixels inside of the line thickness are filled with the same value. The Ambient Occlusion and Normal texture maps create the shape of the crack while the Metallic and Roughness maps define how reflective the inside of the crack is.

Figure A.VII.11: Example of the crack defect. (left) 6 segments, (right) 8 segments

Crack Characteristic	Range of the Defective Texture	Range of the Non-Defective Texture
Length	[30, 80] pixels	[30, 80] Pixels
Number of Segments	[3, 10]	[3, 10]
Segment Offset	[4, 10]	[1, 1]
Line Thickness	[2, 4]	[1, 1]
Texture Map Pixel Color Noise	[-15, 15]	[-15, 15]

Figure A.VII.12: Outline of the design variables to create a crack defect and non-defect imperfection

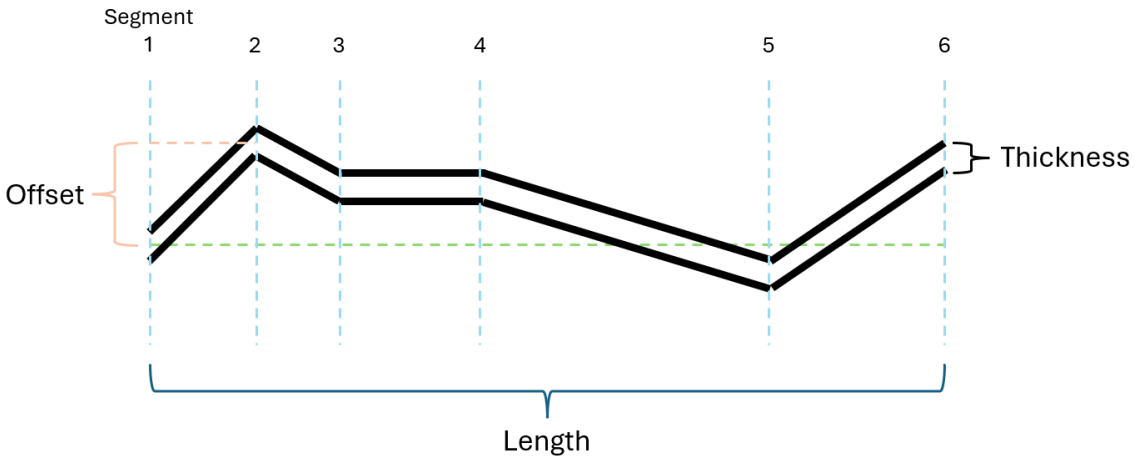


Figure A.VII.13: Visual representation of how a weld crack is defined by pixel. The shown crack is made up of a line with 6 segments with slight thickness variation and have positive offsets for segment point 2, & 6. Segment points 3 & 5 have negative offsets. Segment point 4 has an offset of zero.

Incomplete Weld

Incomplete welds are instances where the filler material is not completely continuous along a weld line. If enough filler is missing, the weld seam that you are attempting to weld could show. The UE models attempt to replicate this by inserting a rectangular area perpendicular to the weld seam. If the cut area extends past the weld midpoint, then a 'seam' is created in the Normal, Metallic, and Roughness texture maps. The seam is 1 pixel wide with 3 additional pixels on either side that is slightly more metallic than the center seam. These two areas represent the edge of the plate and the free space between the plates. The pixels in the rectangular region resample the pixel intensity of the pixel from the direction of the cut. As seen in the picture below, the cut is beyond the weld midpoint. The seam is showing and there is slight discoloration in the rectangular area that was cut out. This is the mismatching of pixels between what was resampled and what is around the overwritten pixel.

Figure A.VII.14: Example of the incomplete weld defect with the seam showing. (left) Partial incomplete weld, (right) missing weld.

Incomplete Weld Characteristic	Range of the Defective Texture	Range of the Non-Defective Texture
Gap Width	[3, 30]	[0, 30]
Gap Depth	[~40%, 100%]	[0%, 30%]
Texture Map Pixel Color Noise	[-10, 10]	[-10, 10]

Figure A.VII.15: Outline of the design variables to create an incomplete weld defect and non-defect imperfection

Non-Defect Versions of UE Defect Models

In the table below are examples of synthetic weld imperfections that are labeled during data collection as {*Defect*} or {*Good*}. Defects modeled as *Good* welds add to the variation of the *Good* weld feature space. By adding examples of weld imperfections being non-defects, this will help prevent false negatives (model think a minor distortion is a defect).

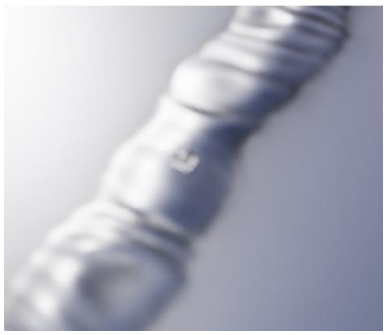
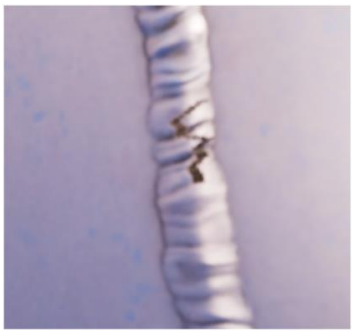
Imperfection	Defect UE Model {*Defect*}	Non-Defect UE Model {*Good*}
Porosity		
Crack		
Incomplete Weld		

Figure A.VII.16: Differences in when created weld variations are labeled as Defect or Good in the training data.

A.VIII. Model Training Charts

The following are examples of the Accuracy and Loss training charts for the various experiments.

Experiment 1 – Cat_Dog - 9425 Synthetic Images available

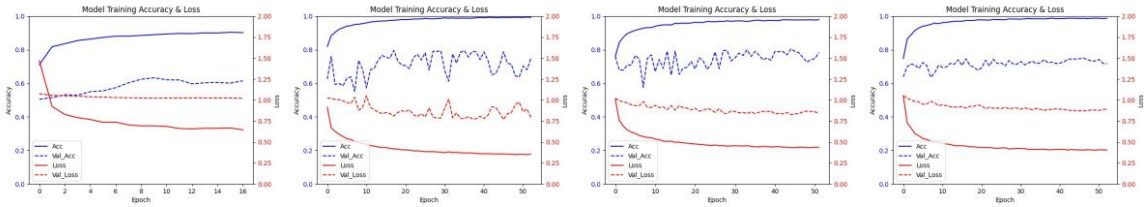


Figure A.VII.1: Accuracy and Loss training graphs for VGG16, RSR (left) 0:100, (middle left) 75:25, (middle right) 50:50, (right) 05:95

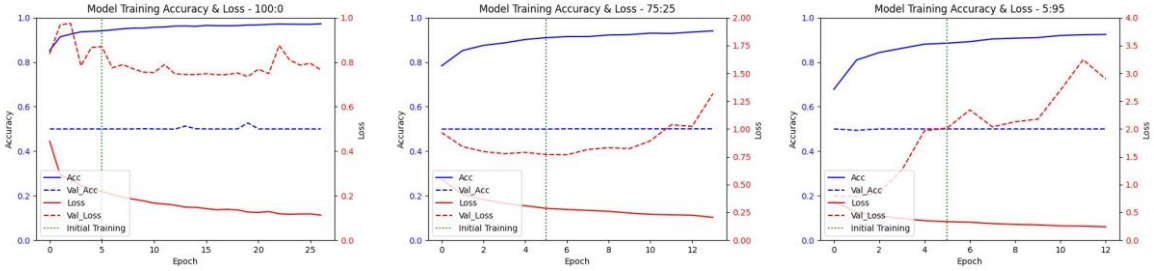


Figure A.VII.2: Accuracy and Loss training graphs for MobileNet, RSR (left) 0:100, (middle) 75:25, (right) 05:95. As discussed in the Discussion section of the paper, the MobileNet model struggled with training in several aspects. Although the training and test gap for loss was about the same as VGG16, the MobileNet model's absolute minima was so sensitive that the model diverged continuously during training. This is why the five epochs of initial training were put in place (to give the model more time to find a local minima). Experimenting with different learning rate and batch size combinations did not resolve this. In addition, the Kerasv2.12 implementation of the MobileNet model did not respond with any change to Validation Accuracy (blue dashed line). This is most irregular and, during this project, was present in MobileNetV3 variations (small, large, large-minimal).

Experiment 4 – Welding - 8160 Synthetic Images available

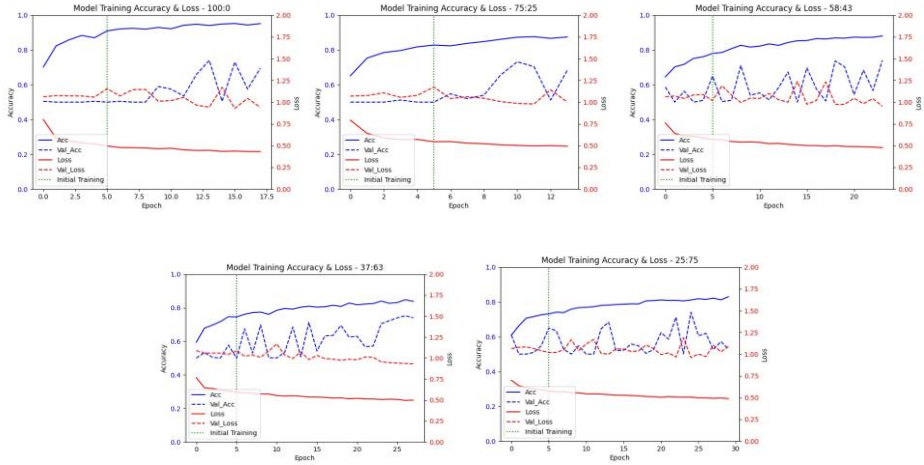


Figure A.VII.3: Accuracy and Loss training graphs for VGG16. All show a struggle to reduce Validation Loss, especially as the RSR decreases.

A.IX. Results

Definition of Experiments 1 & 2

The tables below detail how many images of each type are used for Cat_Dog Experiments 1 & 2 analyzing how using synthetic images and synthetic data availability affect model performance.

Real:Synth Ratio	Total Images	Required Number of Real Images	Data Split	Number of real **TRAINING** images	Number of synthetic **TRAINING** images	Number of real **VALIDATION** images	Number of real **TEST** images
100:0	10000	10000	80/10/10	8000	0	1000	1000
75:25	10000	8000	80/10/10	6000	2000	1000	1000
50:50	10000	6000	80/10/10	4000	4000	1000	1000
25:75	10000	4000	80/10/10	2000	6000	1000	1000
10:90	10000	3000	80/10/10	800	7200	1000	1000
05:95	10000	2800	80/10/10	400	7600	1000	1000
0:100	10000	2000	80/10/10	0	8000	1000	1000

Figure A.IX.1: Definition of Experiment 1. Mix of real and synthetic data in the training set. Always 1000 real images in the validation and test sets.

Real:Synth Ratio	Total Images	Number of **Real** Training Images	Number of **Synthetic** Training Images	Synthetic Utilization Rate (9425 Synthetic Images Available)	Synthetic Utilization Rate (46784 Synthetic Images Available)
100:0	10000	8000	0	0%	0%
75:25	10000	6000	2000	21.2%	4.2%
50:50	10000	4000	4000	42.4%	8.4%
25:75	10000	2000	6000	63.6%	12.6%
10:90	10000	800	7200	76.4%	15.0%
05:95	10000	400	7600	80.6%	15.9%
0:100	10000	0	8000	84.8%	16.8%

Figure A.IX.2: Definition of Experiment 1 expanded to Experiment 2. Details of the SUV for each test in Figure A.IX.1.

Raw Data - Experiments 1 & 2

Experiment 1 - 9425 Synthetic Images Available

VGG16							
RSR	Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.950	0.00178	0.959	0.00010	-	-
75:25	8000	0.984	0.00009	0.965	0.00016	0.0060	58.2%
50:50	6000	0.961	0.00061	0.971	0.00001	0.0113	-90.8%
25:75	4000	0.986	0.00009	0.963	0.00003	0.0040	-66.1%
10:90	3000	0.963	0.00038	0.949	0.00007	-0.0107	-34.5%
05:95	2800	0.972	0.00035	0.945	0.00023	-0.0140	126.3%
0:100	2000	0.972	0.00027	0.851	0.00026	-0.1080	154.9%

Figure A.IX.3: Raw data

VGG16							
RSR	Required Number of Real Images	Avg Train_Loss	Variance	Avg Test_Loss	Variance	Loss change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.504	0.01584	0.867	0.01621	-	-
75:25	8000	0.387	0.00109	0.767	0.00079	-0.0997	-95.2%
50:50	6000	0.478	0.00481	0.875	0.00397	0.0077	-75.5%
25:75	4000	0.386	0.00175	0.806	0.00120	-0.0613	-92.6%
10:90	3000	0.472	0.00449	0.897	0.00406	0.0300	-75.0%
05:95	2800	0.454	0.00375	0.949	0.00858	0.0823	-47.1%
0:100	2000	0.445	0.00368	0.995	0.00089	0.1276	-94.5%

Figure A.IX.4: Raw data

MobileNetV3-small							
RSR	Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.959	0.00002	0.905	0.00000	-	-
75:25	8000	0.915	0.00007	0.874	0.00003	-0.032	1200.00%
50:50	6000	0.912	0.00044	0.870	0.00022	-0.035	621.98%
25:75	4000	0.895	0.00000	0.839	0.00065	-0.067	197.87%
10:90	3000	0.903	0.00006	0.830	0.00023	-0.076	-64.69%
05:95	2800	0.901	0.00000	0.803	0.00051	-0.103	119.39%
0:100	2000	0.903	0.00004	0.538	0.00063	-0.367	25.13%

Figure A.IX.5: Raw data

MobileNetV3-small							
RSR	Required Number of Real Images	Avg Train_Loss	Variance	Avg Test_Loss	Variance	Loss change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.160	0.00040	0.854	0.01116	-	-
75:25	8000	0.273	0.00065	0.782	0.00005	-0.0713	-99.6%
50:50	6000	0.277	0.00313	0.850	0.01196	-0.0037	7.2%
25:75	4000	0.327	0.00002	1.107	0.09706	0.2530	769.9%
10:90	3000	0.308	0.00043	1.106	0.06910	0.2523	519.3%
05:95	2800	0.308	0.00007	1.049	0.01413	0.1953	26.6%
0:100	2000	0.301	0.00008	1.272	0.11135	0.4187	898.0%

Figure A.IX.6: Raw data

Experiment 2 - 46784 Synthetic Images Available

VGG16							
RSR	Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.974	0.00011	0.967	0.00006	-	-
75:25	8000	0.975	0.00005	0.955	0.00018	-0.0117	203.3%
50:50	6000	0.972	0.00003	0.960	0.00000	-0.0070	-96.1%
25:75	4000	0.975	0.00045	0.953	0.00024	-0.0133	305.0%
10:90	3000	0.969	0.00071	0.952	0.00038	-0.0143	530.4%
05:95	2800	0.958	0.00078	0.946	0.00009	-0.0210	43.1%
0:100	2000	0.982	0.00007	0.826	0.00014	-0.1403	132.6%

Figure A.IX.7: Raw data

VGG16							
RSR	Required Number of Real Images	Avg Train_Loss	Variance	Avg Test_Loss	Variance	Loss change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.432	0.00190	0.857	0.00080	-	-
75:25	8000	0.432	0.00023	0.816	0.00042	-0.0410	-47.7%
50:50	6000	0.439	0.00056	0.845	0.00429	-0.0123	433.5%
25:75	4000	0.423	0.00656	0.814	0.01555	-0.0437	1833.6%
10:90	3000	0.442	0.00761	0.850	0.00667	-0.0070	729.3%
05:95	2800	0.484	0.00586	0.907	0.00408	0.0493	407.8%
0:100	2000	0.398	0.00075	0.978	0.00104	0.1207	29.2%

Figure A.IX.8: Raw data

MobileNetV3-small							
RSR	Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.965	0.00005	0.914	0.00036	-	-
75:25	8000	0.910	0.00002	0.883	0.00062	-0.030	70.81%
50:50	6000	0.899	0.00009	0.885	0.00007	-0.029	-89.23%
25:75	4000	0.886	0.00004	0.855	0.00110	-0.059	1540.30%
10:90	3000	0.892	0.00004	0.849	0.00045	-0.065	-58.78%
05:95	2800	0.888	0.00001	0.810	0.00047	-0.104	3.97%
0:100	2000	0.900	0.00006	0.605	0.00021	-0.308	-56.19%

Figure A.IX.9: Raw data

MobileNetV3-small							
RSR	Required Number of Real Images	Avg Train_Loss	Variance	Avg Test_Loss	Variance	Loss change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	10000	0.139	0.00073	0.748	0.00032	-	-
75:25	8000	0.284	0.00009	0.777	0.00006	0.0287	-80.3%
50:50	6000	0.310	0.00033	1.171	0.00944	0.4230	2811.0%
25:75	4000	0.343	0.00009	1.567	0.29963	0.8183	92284.7%
10:90	3000	0.328	0.00013	1.850	0.14130	1.1020	43465.5%
05:95	2800	0.334	0.00000	1.952	0.08354	1.2037	25657.1%
0:100	2000	0.310	0.00026	1.819	0.01907	1.0710	5778.3%

Figure A.IX.10: Raw data

Definition of Experiment 3

Cat_Dog Experiment 3 is a baseline measurement of how the models perform if no synthetic images are used, but the dataset of real images decreases.

Total Images	Data Split	Percentage of TRAINING images are Real	Number of synthetic **TRAINING** images	Number of real **TRAINING** images	Number of real **VALIDATION** images	Number of real **TEST** images
7500	80/10/10	100	0	6000	750	750
5000	80/10/10	100	0	4000	500	500
2500	80/10/10	100	0	2000	250	250
1000	80/10/10	100	0	800	100	100
500	80/10/10	100	0	400	50	50

Figure A.IX.11: Definition of Experiment 3 comparing mixed datasets with smaller datasets but only real images. The mixed dataset test results are collected from Experiments 1 & 2

Raw Data - Experiments 3

VGG16								
Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Avg Train_Loss	Variance	Avg Test_Loss	Variance
10000	0.967	0.00005	0.957	0.00023	0.462	0.00065	0.878	0.00293
8000	0.978	0.00014	0.970	0.00010	0.432	0.00172	0.852	0.00235
6000	0.984	0.00002	0.963	0.00043	0.421	0.00011	0.883	0.01554
4000	0.982	0.00001	0.953	0.00013	0.438	0.00035	0.909	0.00130
3000	0.933	0.00162	0.940	0.00480	0.575	0.01213	1.040	0.00430

Figure A.IX.12: Raw data

MobileNetV3-small								
Required Number of Real Images	Avg Train_Acc	Variance	Avg Test_Acc	Variance	Avg Train_Loss	Variance	Avg Test_Loss	Variance
10000	0.954	0.00006	0.897	0.00023	0.184	0.00065	0.784	0.00307
8000	0.950	0.00000	0.897	0.00056	0.196	0.00008	0.835	0.01024
6000	0.926	0.00008	0.897	0.00103	0.272	0.00038	0.998	0.03269
4000	0.928	0.00006	0.930	0.00010	0.263	0.00068	1.069	0.04207
3000	0.937	0.00126	0.893	0.00053	0.266	0.00771	1.021	0.09085

Figure A.IX.13: Raw data

Welding

Definition of Experiment 4 - Change in RSR values

For every test in Experiment 4, all available 2000 real images are used. As such, to maintain the 80/10/10 split, real images are transferred into the Validation and Test sets. The RSR is only reduced to 25:75 due to the results of 0:100 tests in Experiments 1-3 and that it is desired to always have real images for this test. It is possible to have a test of 0:100 where all real images are in the validation and test set with synthetic images still imbalanced at 7:3.

Real:Synth Ratio	Total Number of Images	# of GOOD	# of DEFECT	Data Split	# of real TRAINING	# of real VALIDATION	# of real TESTING
100:0	2000	600	1400	80/10/10	1600	200	200
75:25	2500	750	1750	80/10/10	1500	250	250
58:42	3000	900	2100	80/10/10	1400	300	300
37:63	4000	1200	2800	80/10/10	1200	400	400
25:75	5000	1500	3500	80/10/10	1000	500	500

Figure A.IX.14: Definition of Experiment 4. As the RSR decreases the number of real images in the validation and test sets increase.

Real:Rynth Ratio	Total Number of Images	Train Images: REAL	# of Training Real GOOD	# of Training Real Defect	Train Images: Synthetic	# of Added Synthetic GOOD	# of Added Synthetic DEFECT
100:0	2000	1600	400	1200	0	0	0
75:25	2500	1500	350	1150	500	250	250
58:42	3000	1400	300	1100	1000	420	580
37:63	4000	1200	200	1000	2000	760	1240
25:75	5000	1000	100	900	3000	900	2100

Figure A.IX.15: Definition of Experiment 4 continued. Shows the class breakdown for each class in the imbalanced dataset.

Raw Data

VGG16						
Real:Rynth Ratio	Total Number of Images	Avg Train_Acc	Avg Test_Acc	Variance Test_Acc	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.9229	0.7967	0.00823	-	-
75:25	2500	0.8618	0.8433	0.00223	0.0467	-72.9%
58:42	3000	0.8254	0.8300	0.00000	0.0333	-100.0%
37:63	4000	0.8197	0.8300	0.00030	0.0333	-96.4%
25:75	5000	0.7835	0.7933	0.00003	-0.0033	-99.6%

Figure A.IX.16: Raw data

VGG16						
Real:Rynth Ratio	Total Number of Images	Avg Train_Loss	Avg Test_Loss	Variance Test_Loss	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.4895	1.0880	0.00462	-	-
75:25	2500	0.5062	0.9873	0.00311	-0.1008	-32.6%
58:42	3000	0.5320	1.0186	0.00051	-0.0695	-89.0%
37:63	4000	0.5198	0.9896	0.00193	-0.0985	-58.1%
25:75	5000	0.5275	1.0217	0.00201	-0.0664	-56.4%

Figure A.IX.17: Raw data

VGG16						
Real:Rynth Ratio	Total Number of Images	Avg Train_AUC	Avg Test_AUC	Variance Test_AUC	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.9669	0.8112	0.00026	-	-
75:25	2500	0.9386	0.8328	0.00036	0.0216	38.7%
58:42	3000	0.9100	0.8279	0.00093	0.0167	253.7%
37:63	4000	0.9106	0.7821	0.00025	-0.0291	-4.3%
25:75	5000	0.8767	0.7439	0.00009	-0.0673	-65.7%

Figure A.IX.18: Raw data

VGG16						
Real:Rynth Ratio	Total Number of Images	Avg TP (%)	Avg TN (%)	Avg FP (%)	Avg FN (%)	Cost per Test Sample
100:0	2000	42.8	43.5	7.2	6.5	10.3
75:25	2500	45.5	38.8	4.5	11.2	6.8
58:42	3000	44.7	38.4	5.3	11.6	7.5
37:63	4000	47.1	35.8	2.9	14.2	6.8
25:75	5000	47.7	31.6	2.3	18.4	7.1

Figure A.IX.19: Raw data

MobileNetV3-small						
Real:Rynth Ratio	Total Number of Images	Avg Train_Acc	Avg Test_Acc	Test_Acc Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.8642	0.7833	0.01003	-	-
75:25	2500	0.8428	0.7900	0.00190	0.0067	-81.06%
58:42	3000	0.7678	0.7833	0.00263	0.0000	-73.75%
37:63	4000	0.7367	0.7167	0.00203	-0.0667	-79.73%
25:75	5000	0.6991	0.6467	0.00053	-0.1367	-94.68%

Figure A.IX.20: Raw data

MobileNetV3-small						
Real:Rynth Ratio	Total Number of Images	Avg Train_Loss	Avg Test_Loss	Test_Loss Variance	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.2418	0.8312	0.00037	-	-
75:25	2500	0.1765	1.0525	0.08723	0.2213	23569.36%
58:42	3000	0.2675	0.8916	0.01107	0.0603	2904.19%
37:63	4000	0.2774	0.8760	0.01654	0.0447	4387.65%
25:75	5000	0.2806	0.8096	0.00192	-0.0216	421.92%

Figure A.IX.21: Raw data

MobileNetV3-small						
Real:Rynth Ratio	Total Number of Images	Avg Train_AUC	Avg Test_AUC	Variance Test_AUC	Accuracy change compared to RSR 100:0	Variance change compared to RSR 100:0
100:0	2000	0.9363	0.5773	0.00116	-	-
75:25	2500	0.9309	0.5417	0.00305	-0.0356	162.07%
58:42	3000	0.8641	0.5657	0.00181	-0.0117	55.59%
37:63	4000	0.8269	0.5556	0.00009	-0.0217	-92.23%
25:75	5000	0.7701	0.5696	0.00010	-0.0077	-91.18%

Figure A.IX.22: Raw data

MobileNetV3-small						
Real:Rynth Ratio	Total Number of Images	Avg TP (%)	Avg TN (%)	Avg FP (%)	Avg FN (%)	Cost per Sample
100:0	2000	36.1	42.0	13.9	8.0	10.933
75:25	2500	45.6	33.3	4.4	16.7	8.600
58:42	3000	47.0	31.3	3.0	18.7	8.333
37:63	4000	48.5	23.0	1.5	27.0	10.167
25:75	5000	49.5	15.0	0.5	35.0	10.833

Figure A.IX.23: Raw data

Definition of Experiment 5 - Rebalancing Imbalanced Datasets with Synthetic data

case	Total Number of Images	# of GOOD	# of DEFECT	# of TRAINING	# of VALIDATION	# of TESTING
No Change	1800	600	1400	1600	200	200
Cost-Learning	1800	600	1400	1600	200	200
Undersample	1200	600	600	960	120	120
Oversample	2400	1200	1200	1920	240	240
Synthetic Padding	2400	1200	1200	1920	240	240

Figure A.IX.24: Definition of Experiment 5. General description of what images are used.

case	Total Number of Images	Train Images: REAL	# of Training Real GOOD	# of Training Real Defect	Train Images: Synthetic	# of Added Synthetic GOOD	# of Added Synthetic DEFECT
No Change	1800	1600	400	1200	0	0	0
Cost-Learning	1800	1600	400	1200	0	0	0
Undersample	1200	960	480	480	0	0	0
Oversample	2400	1920	960*	960	0	0	0
Synthetic Padding	2400	1320	360	960	600	600	0

Figure A.IX.25: Definition of Experiment 5. Further breakdown of images used in each rebalancing method.
**In the oversample method, all 480 available Good images not used in the other sets are duplicated to yield 960 images.*

Raw Data

VGG16					
case	Avg Test_Acc	Avg Train_AUC	Avg Test_AUC	Test_AUC Variance	Effect of Sampling Method compared to the imbalanced dataset
No Change	0.8433	0.9847	0.8519	0.0001	-
Cost-Learning	0.8433	0.9843	0.8257	0.0009	-0.0263
Undersample	0.8633	0.9660	0.8234	0.0002	-0.0285
Oversample	0.8733	0.9721	0.7676	0.0017	-0.0844
Synthetic Padding	0.8633	0.9925	0.8137	0.0050	-0.0382

Figure A.IX.26: Raw data

VGG16				
case	Avg Train_Loss	Avg Test_Loss	Test_Loss Variance	Effect of Sampling Method compared to the imbalanced dataset
No Change	0.4369	0.8433	0.0000	-
Cost-Learning	0.4385	0.8433	0.0082	0.0000
Undersample	0.4701	0.8633	0.0000	0.0200
Oversample	0.4841	0.8733	0.0006	0.0300
Synthetic Padding	0.4151	0.8633	0.0037	0.0200

Figure A.IX.27: Raw data

VGG16				
case	Avg Cost per Test Sample	Avg TP+TN (%)	Avg FP (%)	Avg FN (%)
No Change	0.0780	0.844	0.0707	0.0850
Cost-Learning	0.0717	0.842	0.0573	0.1003
Undersample	0.0697	0.795	0.1247	0.0807
Oversample	0.0640	0.872	0.1010	0.0267
Synthetic Padding	0.0687	0.862	0.0640	0.0737

Figure A.IX.28: Raw data

MobileNetV3-small					
case	Avg Test_Acc	Avg Train_AUC	Avg Test_AUC	Test_AUC Variance	Effect of Sampling Method compared to the imbalanced dataset
No Change	0.8100	0.9847	0.8519	0.0001	-
Cost-Learning	0.8100	0.9843	0.8257	0.0009	-0.0263
Undersample	0.7967	0.9660	0.8234	0.0002	-0.0285
Oversample	0.7967	0.9721	0.7676	0.0017	-0.0844
Synthetic Padding	0.8100	0.9925	0.8137	0.0050	-0.0382

Figure A.IX.29: Raw data

MobileNetV3-small				
case	Avg Train_Loss	Avg Test_Loss	Test_Loss Variance	Effect of Sampling Method compared to the imbalanced dataset
No Change	0.4369	0.8433	0.0000	-
Cost-Learning	0.4385	0.8433	0.0082	0.0000
Undersample	0.4701	0.8633	0.0000	0.0200
Oversample	0.4841	0.8733	0.0006	0.0300
Synthetic Padding	0.4151	0.8633	0.0037	0.0200

Figure A.IX.30: Raw data

MobileNetV3-small				
case	Cost per Test Sample	Avg TP+TN (%)	Avg FP (%)	Avg FN (%)
No Change	0.0780	0.844	0.0707	0.0850
Cost-Learning	0.0717	0.842	0.0573	0.1003
Undersample	0.0697	0.795	0.1247	0.0807
oversample	0.0640	0.872	0.1010	0.0267
pad w/ synthetic	0.0687	0.862	0.0640	0.0737

Figure A.IX.31: Raw data

Acknowledgments

Thank you to Dr Creed Jones, Dr Ryan Williams, and Dr Jason Xuan for being a part of my committee, especially to Dr Jones for giving me their ear and guidance throughout this project.

Additional thank yous to …

Northrop Grumman Corporation for funding my Master's Degree at Virginia Tech University.

To the numerous teachers that I consulted with throughout this project. To Brandon Padayao with Baltimore OpenWorks for allowing me to sit in on a welding course and learn from you about metal welding. To Riley Van Etten and Deion Waddell for their expertise in 3D graphic design. To numerous others helping me brainstorm through ideas and barriers.

This project gave me the chance to learn about one of the most powerful graphics engines - one of many that fueled my imagination as a child.

Page intentionally left blank

References

I. Papers and Articles

Chuanqui Tan, Fuchun Sun, Tao,Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. “A Survey on Deep Transfer Learning”. State Key Laboratory of Intelligent Technology and Systems. Tsingua National Laboratory for Information Science and Technology, Tsingua University. Distributed by Cornell University. arXiv:1808.0197. Aug 6th, 2018

ECCV-18 Unknown author submission ID 2903. “Learning image classifiers from (limited) real and (abundant) synthetic data.” Semantic Scholar. 2018. Corpus ID: 53472409. https://www.semanticscholar.org/paper/and-(abundant)-synthetic-data/1f759dbd74ad3907457dcbf50d4be54b470e33f4. Citation possibly for Nguyen, Tan, Chen, H, et al. “Learning image classifiers from (limited) real and (abundant) synthetic data.” Published by Rice University. https://tannguyen.blogs.rice.edu/current-research/. 2018.

Gastelum, Zoe Nellie, and Shead, Timothy. How Low Can You Go? Using Synthetic 3D Imagery to Drastically Reduce Real-World Training Data for Object Detection. United States: N. p., 2020. Web. doi:10.2172/1670874.

Howard, Andrew, et al. "Searching for mobilenetv3." Proceedings of the IEEE/CVF international conference on computer vision. 2019.

Man, Keith, and Javaan Chahl. “A Review of Synthetic Image Data and Its Use in Computer Vision.” Journal of imaging vol. 8,11 310. 21 Nov. 2022, doi:10.3390/jimaging8110310

Thornström, J. “Domain Adaptation of Unreal Images for Image Classification (Dissertation).” Linköping University Department of Electrical Engineering, Computer Vision (Sweden). 2019. URN: 165758. https://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A1431281&dswid=6387

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

S. J. Pan and Q. Yang, "A Survey on Transfer Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010, doi: 10.1109/TKDE.2009.191. keywords: {Machine learning;Training data;Data mining;Knowledge transfer;Space technology;Knowledge engineering;Machine learning algorithms;Labeling;Learning systems;Testing;Transfer learning;survey;machine learning;data mining.}

Weiss, Gary M, McCarthy, Kate, and Zabar, Bibi. “Cost-Sensitive Learning vs. Sampling: Which is Best for Handling Unbalanced Classes with Unequal Error Costs?”. Department of Computer and Information Science, Fordham University. 2007. https://storm.cis.fordham.edu/~gweiss/papers/dmin07-weiss.pdf

II. Additional Resources

3D Ace. “Does Polygon Count matter in 3D Modeling for Game Assets?”. Nicosia, Cyprus. Jan 3rd, 2024. https://3d-ace.com/blog/polygon-count-in-3d-modeling-for-game-assets/

DowlingSoka, Ryan. “Python in Unreal Tips”. Published on Mar 18th, 2024. https://ryandowlingsoka.com/unreal/python-in-unreal/

“Keras Documentation: Mobilenet, Mobilenetv2, and MobileNetV3.” Keras, Google, keras.io/api/applications/mobilenet/. Accessed 25 Apr. 2024.

OpenWorks, Baltimore City. Mar 16th, 2024. Metal 3 with Brandon Padayao. Articles: “Metal 3: MIG Welding” and “Metal 3: TIG Welding”. 1400 Greenmount Ave, Baltimore, MD 21202

Padayao, Brandon. “Metal 3”. Baltimore Openworks. Mar 16th, 2024.

III. Datasets

15DEM20F. “Welding Images Dataset”. Dataset 01. Publisher Roboflow. Roboflow Universe. Jul 2022. https://universe.roboflow.com/15dem20f-ab0x1/welding-images

afiys. “Proyek Akhir - OFA5 Dataset”. Dataset 01. Publisher Roboflow. Roboflow Universe. Oct 2023. https://universe.roboflow.com/afiys-oxbaf/proyek-akhir-ofa5

Celebal Workspace. “Welding_Detection_v2 Dataset”. Publisher Roboflow. Roboflow Universe. Dec 2022. https://universe.roboflow.com/celebal-workspace/welding_detection_v2

Elson, Jeremy and Douceur, John (JD) and Howell, Jon and Saul, Jared. “Cats vs Dogs”. Proceedings of 14th ACM Conference on Computer and Communications Security (CCS). October 2007. Published by Association for Computing Machinery, Inc. Distributed by Microsoft. https://www.microsoft.com/en-us/download/details.aspx?id=54765

Welddefect. “Weld_detections Dataset”. Publisher Roboflow. Roboflow Universe. Jul 2023. https://universe.roboflow.com/welddefect-4pvic/weld_detections/dataset/1

Weld 1. “Final Dataset Dataset”. Dataset 01. Publisher Roboflow. Roboflow Universe. April 2023. https://universe.roboflow.com/weld-1-mba6s/final-dataset-rc2vm

IV. Photos

Herrmann, Peter. “Avenue Trees Tree Lined”. Pixabay. April 10, 2017. https://pixabay.com/photos/avenue-trees-tree-lined-path-trail-2215317/. Accessed February 22, 2024.

Koshkina, Larisa. “Flowers Meadow Sunlight”. Pixabay. https://pixabay.com/photos/flowers-meadow-sunlight-summer-276014/. Accessed February 22, 2024.

Parks & Open Space Staff. “Red Rocks Spur Rocks”. BoulderColorado. https://bouldercolorado.gov/trail/red-rocks. Accessed March 1, 2024.

Pritchard, Neal. “Autumn Leaves on Forest Floor”. Stocksy. October 27, 2017. https://www.stocksy.com/1672804/autumn-leaves-on-forest-floor. Accessed February 22, 2024.

SmartFurnature. “Living-sets-thumbnail.jpg”. SmartFurnature. https://smartfurniture.com.eg/wp-content/uploads/2023/12/Living-sets-thumbnail.jpg. Accessed March 1, 2024.

Wirestock. “Beautiful light interior design of a modern living room in blue and yellow colors”. FreeP!k. https://www.freepik.com/free-photo/beautiful-light-interior-design-modern-living-room-blue-yellow-colors_30561045.htm#fromView=search&page=1&position=14&uuid=ab0dba17-f662-4c0b-9692-767da974e1a2. Accessed March 1, 2024.

V. 3D Models – Cats

Andedali. Cat. Turbosquid. FBX. Product ID: 1353041.”Its a cat)+rig+face”. https://www.turbosquid.com/3d-models/cat-1353041. December 4, 2018

Idris_1986. “Cat”. Turbosquid. ID: 1099111. “cat with blender create .basic bones and paint fur and bacis hair fur .eye textures etc features .others format ojb 3ds”. https://www.turbosquid.com/3d-models/cat-1099111. December 03, 2016

Printable_Models. “Cat V1”. Free3D. .obj. ID: 123c6a1c5523. https://free3d.com/3d-model/cat-v1--579827.html. September 6, 2018. Last updated November 20, 2019

Printable_Models. “Pallas Cat V1”. Free3D. .obj. ID: 123c73c389b. https://free3d.com/3d-model/pallas-cat-v1--576987.html. September 27, 2018. Last updated November 20, 2019

Printable_Models. “Cat V1”. Free3D. .obj. ID: 123cb1b1943a.. “Cat v1 printable, low poly model.”. https://free3d.com/3d-model/cat-v1--522281.html. September 12, 2018. Last updated November 20, 2019
Printable_Models. “Cat V1”. Free3D. .obj. ID: 123c3567cbad.. “Cat v1 printable, low poly model.”. https://free3d.com/3d-model/cat-v1--220685.html. October 10, 2018. Last updated November 20, 2019
Printable_Models. “Cat V3”. Free3D. .obj. ID: 123c86a1fcd3.. “Cat V3 printable, low poly model”. https://free3d.com/3d-model/cat-v3--351220.html. October 10, 2018. Last updated November 20, 2019
Printable_Models. “Black Cat Back Arched V2”. Free3D. .obj. ID: 123cdfd0222e.. “Black Cat Back Arched V2 printable, low poly model”. https://free3d.com/3d-model/black-cat-back-arched-v2--916864.html. October 10, 2018. Last updated November 20, 2019
Snippysnappets. “Low Poly Cat”. Free3D. .obj. “Slightly low poly cat with whiskers”. https://free3d.com/3d-model/low-poly-cat-46138.html. February 25, 2017. Last updated November 20, 2019
Studio King. “Base mash Cat”. Turbosquid. Product ID: 2080612. “Base Mesh Cat”. https://www.turbosquid.com/3d-models/base-mash-cat-2080612. June 7, 2022

VI. 3D Models – Dogs

3d_logoman. “Golden Retriever”. Turbosquid. ID: 1845760. “Low poly model, created for 3D printing. You can also use it as a base model for your projects. This model can be used as a decoration in your interior. We make models for individual orders. See my other models”. https://www.turbosquid.com/3d-models/golden-retriever-1850419. February 05, 2022
Lima, Elvair. “White Dog”. Turbosquid. ID: 1845760. “Low poly. Game ready.Textures included an materials applied. All formats tested and working. Textures PBR 2048x2048. I's not rigged.”. https://www.turbosquid.com/3d-models/white-dog-1845760. January 26, 2022
Iashkoalex. “English Bulldog Real-Time Free Sample”. Turbosquid. ID: 1155164. “FREE Sample of a Realistic English Bulldog Real-Time Game-Ready 3D Model. FULL version of this model is available for purchase under ID 1111972”. https://www.turbosquid.com/3d-models/english-bulldog-real-time-free-sample-1155164. May 9, 2017
Printable_Models. “Australian Cattle Dog V1”. Free3D. .obj. ID: 123c86a1fcd3.. “Australian Cattle Dog v1 printable, low poly model.”. https://free3d.com/3d-model/australian-cattle-dog-v1--993323.html. September 27, 2018. Last updated November 20, 2019
Printable_Models. “Canaan Dog V1”. Free3D. .obj. ID: 123c82cb7c11. “Canaan Dog v1 printable, low poly model.”. https://free3d.com/3d-model/canaan-dog-v1--72376.html. September 27, 2018. Last updated November 20, 2019
Printable_Models. “Cardigan Welsh Corgi V1”. Free3D. .obj. ID: 123ce3e4b2fe.. “Cardigan Welsh Corgi v1 printable, low poly model.”. https://free3d.com/3d-model/cardigan-welsh-corgi-v1--784056.html. September 27, 2018. Last updated November 20, 2019
Printable_Models. “ConcreteDogStatue V1”. Free3D. .obj. ID: 123c476a785d. “ConcreteDogStatue v1 printable, low poly model.”. https://free3d.com/3d-model/concretedogstatue-v1--25932.html. September 27, 2018. Last updated November 20, 2019
Printable_Models. “Dog V2”. Free3D. .obj. ID: 123ca88417bb. “Dog v2 printable, low poly model.”. https://free3d.com/3d-model/dog-v2--703191.html. October 10, 2018. Last updated November 20, 2019
Printable_Models. “Farm Dog V2”. Free3D. .obj. ID: 123c96195bae. “Farm Dog v2 printable, low poly model.”. https://free3d.com/3d-model/farm-dog-v2--499533.html. October 10, 2018. Last updated November 20, 2019
Rocazella, Sam. “Aleu”. Free3D. .obj. ID: 9v371wmvoc.. “This model was edited solo by me (edited by: Sam Rocazella) from the original balto model. big thanks to the original creator of the balto model letting this be possible..”. https://free3d.com/3d-model/aleu-44920.html. August 8, 2013. Last updated November 20, 2019
umar6419. “BASSET Dog”. Free3D. .obj. “Low POLY AND hIGH TEXTURES”. “https://free3d.com/3d-model/basset-dog-20248.html.”. https://free3d.com/3d-model/aleu-44920.html. August 8, 2013. Last updated November 20, 2019

VI. 3D Models - Welding

Samur Art. “4K Metal Panel Material Pack”. Distributed by Epic Games Unreal Engine Marketplace. samurart00@gmail.com. “High quality 15 pieces of metal panel material. All textures are in 4k resolution (4096x4096)”