DEVCOM DAC-TR-2022-089
September 2022

Analysis Methodology of Image Classifiers in Stressed Environments

by Patrick S. Debroux

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so specified by other official documentation.

WARNING

Information and data contained in this document are based on the input available at the time of preparation.

TRADE NAMES

The use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial hardware or software. The report may not be cited for purposes of advertisement.

DEVCOM DAC-TR-2022-089
September 2022

Analysis Methodology of Image Classifiers in Stressed Environments

by Patrick S. Debroux

DEVCOM Analysis Center

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2022	Technical Report	1 October 2021 – 30 September 2022

4. TITLE AND SUBTITLE
Analysis Methodology of Image Classifiers in Stressed Environments

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Patrick S. Debroux

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Director
DEVCOM Analysis Center
1624 Headquarters Ave.
White Sands Missile Range, NM 88002

8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM DAC-TR-2022-089

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report defines the methodology used to analyze 2D (image) classifiers in the form of convolutional neural network models in the presence of battlefield stresses. Battlefield stresses are defined in this context as those that alter the sensor data relative to the images used to train the neural networks. Battlefield stresses can be intentional, such as camouflage or distortion, or unintentional, such as obscurants or excessive background (or foreground) clutter. Parts needed to perform the analysis are described first, followed by the methodology steps used to rank and rate the neural network classifiers, and the report concludes with a practical analysis case.

15. SUBJECT TERMS
CNN, stressed environments, ROC curves, image classifier evaluation, convolutional neural network

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Patrick S. Debroux
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	111	19b. TELEPHONE NUMBER (include area code) (575) 678-5238

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

1. INTRODUCTION

In the last decade, pushed by the discovery of deep neural networks and convolutional methods of classification, 2D classification systems have made huge progress. This progress was funded by the push to develop autonomously driving vehicles. Autonomous vehicles have been deployed in certain driving environments but have been kept away from densely populated environments because of the severe consequences of classification error.

Once aware of these classification errors, researchers have analyzed their causes and have even researched intentional attacks on classification systems. One such attack is called an adversarial attack, where key pixels in images to be classified are altered to intentionally mislead the classification systems. Mitigation techniques for these attacks have been proposed. Adversarial attacks work on a “micro” (pixel) level and are meant to be unnoticeable to the casual human eye.

Instead of analyzing this type of micro attack, this research looks at the effect of “macro” attacks on the neural network model classifier. These macro attacks are very noticeable, such as target camouflage, disruptive coloration, intentional distortion, and hull defilade. These are classification system attacks that would be found in battlefield environments and are referred to as battlefield stresses. Because of the complexity and variability of macro attacks, a “black box” analysis is undertaken to determine the robustness of convolutional neural networks (CNNs) exposed to battlefield stresses. The neural networks used to develop this analysis framework are published, pre-trained, neural network models fined-tuned to the classification classes of interest using transfer-learning. The same neural networks are also trained from neutral weights and biases and their robustness is tested with test images that contain battlefield stresses.

This report is organized as follows: Section 2 describes the image database used to train and test the convolutional neural network models under test. Section 3 describes the different battlefield stresses used, how they are synthesized, and how they are incrementally added to the images used to test the neural networks. Section 4 describes the choice of pretrained CNN models used for the analysis described in this report. Section 5 describes the metrics used to rate the different neural networks against battlefield stresses. Sections 6 and 7 provide a methodology and an analysis example on the fine-tuned pretrained neural networks downloaded from the web. The report ends with a conclusion of the work done followed by appendices of all MATLAB scripts written to perform the analyses.

2. DATABASE CREATION AND TRAINING FILE SETUP

Neural network classification requires that the CNN under study be trained or refined to classify the images to the classes of interest. Without large military vehicle image databases available, a database of images in the visible range (and training set formation methodology) had to be planned and created. Images of common military vehicles were downloaded from the web, and 3D model renditions were used to augment the database. Because the availability of military vehicle images is limited on the web, further training images were obtained by “ripping” frames from web-based video. Although not perfectly curated, images obtained from video clips are perhaps more representative of actual dynamic sensor data expected in working conditions. The database and MATLAB scripts developed to establish the training sets from the database in prescribed quantities are also described in this section.

2.1. Introduction

Because specialized military vehicle image databases large enough to train CNN models are not yet readily available for research, a new database had to be created. Without access to large amounts of military vehicle imagery in the visible spectrum (VIS), it was decided that a database had to be created from common images found on the Internet. The use of the Internet dictated that the military vehicle classes chosen should be common so that sufficient images could be found to make the database useful in training CNN models. Because the images were uploaded from the Internet, the database remains unclassified.*

Fine-tuning previously trained CNN models to target classes of interest (called transfer learning) involves a few hundred images, while training the CNN models from neutral weights and biases requires many thousands of images per class. Because the CNN classification failure (CF) analysis may require custom training, and because existing concerns that pretrained CNN models may eventually be built to intentionally ignore targets of interest have already been raised, a database large enough to train CNN models from neutral weights and biases was developed.

To complement the limited images available, video clips of the military vehicles of interest were uploaded and image frames were extracted from these videos. Video clips, usually recorded at 24 frames per second, yielded an abundant number of frames that were stored as database images.

* All images used in this report were obtained from websites within the public domain or are not subject to copyright restrictions.

Model renderings on neutral backgrounds were also added into the database to provide canonical class forms at incremental azimuth and polar angles. Previous studies have shown that using model renditions in conjunction with images to train or fine-tune CNNs is very powerful.

The images derived from video are not considered to be as good quality as the web images because they generally are not framed as well as photographs, and at 24 frames per second, the aspect and background of the vehicles as seen by the video recorder changes very little between frames. In another sense, images derived from video frames are expected to be closer to what is encountered in realistic fielded system sensors.

2.2. Methodology

Images were collected for six military vehicle classes. These classes were the mixed M1A1s and M1A2s, the Bradley Fighting Vehicle, the high-mobility multipurpose wheeled vehicle (HMMWV), the 6 × 6 Cougar mine-resistant ambush-protected (MRAP) vehicle, the infantry carrier vehicle Stryker, and the Russian T-72 tank.

These vehicle classes were found on the web and their images uploaded. Even though images were sought to represent the vehicles in action, that is, in practical configurations and with various background and foreground clutter, care was taken to try to isolate a single target to one image (e.g., not to have more than one vehicle per image). Duplicate images were removed, even if the images were cropped or resized, to the best of ability. Figure 1 is an example of an uploaded web image.

Figure 1. An example of a web image upload

With these limitations, approximately 300 unique images for each class were found and downloaded. This amount was found adequate for transfer-learning, but too small to train the CNN models considered from neutral weights and biases.

Rendering of 3D models were also used to augment the image database. The Army developed BRL-CAD program was used to develop these renderings, since high-resolution models for the vehicle classes collected already existed. The 3D model renderings were used in gray scale against a homogenous background. Each model was rendered at 90° and 60° polar angles, with 5° increments in azimuth angle. Figure 2 is an example of the BRL-CAD 3D rendition of the M1A2.

Figure 2. A BRL-CAD rendition of an Abrams tank model

Finally, to fill in the database with more images (needed to train the CNNs from neutral weights and biases), video clips of the military vehicle classes were collected from the Internet. Since the free-access YouTube platform does not allow download of its video clips, NCH's Debut Video Capture and Prism Video File Converter software was purchased and used to upload YouTube video clips and to rip frame images out of the videos. This process yielded many images as video is normally recorded at 24 frames per second.

The quality of the video images is lower than the web images uploaded because they lack the careful composition usually put toward taking a web image photograph. Also, because of the relatively rapid frame rate of the video stream, many adjacent frames will have almost identical images. Thus, less information is inherent in adjacent video frames than there would be in completely independent images of the objects of interest. Figure 3 is an example of a video-frame image of an Abrams.

Figure 3. A video-frame image of an Abrams tank

There was a large variation in the number of video images harvested in this manner, depending on the number of independent images found on the web for each class, and the amount and duration of relevant video clips found. For example, a total of 32,650 images were collected for the Bradley, while only 7,710 images were found for the 6 × 6 Cougar MRAP. The images collected for the six classes of military vehicles amounted to a total of 117,770 images.

A MATLAB script, `CHOOSE_TRAINING_IMAGES_FROM_WEB_AND_VIDEO.m` found in Appendix A, was written to extract images from the database in a prescribed manner that tries to equalize the uneven number of images collected for each class. User definable in the script, a number greater than the maximum amount of web image (thus greater than 300) is chosen so that all the web images for each class are copied from the database to the training folders. The number of web images is equalized for each class by choosing random web image duplicates from the database until a specified total (in this case 500) of web images are chosen for each class. The script allows the user to specify whether the model renderings are used in the training set. If so, all 852 model images are uploaded to the training set. The training files are rounded up to a specified overall number of images (in this case 7,500 per class) by randomly choosing video-frame images from the video database for each class. Different random frame images are chosen until all frame images are used, then random duplicates are chosen until the overall number training images per class is copied to the training set.

Because the MATLAB training algorithm is specified to randomly flip training images horizontally, and randomly translate them by up to 30 pixels to reduce CNN model training memorization, it is thought that some additional information is provided by repeated training on a single image. The written script will custom-populate the training files with whatever quantities of web, model, and video images are requested, thereby allowing experimentation on the effect of training file set size on CNN classification

success in stressed environments. The training sets in this example thus consist of 45,000 images that, according to convergence curves shown during training, are overabundant for transfer learning (training converges to near 100% success very rapidly) and seem to be marginally adequate when training the CNNs from neutral weights and biases.

The training images of the six classes are placed in six folders, and MATLAB is instructed to label the images of each folder by the folder name in which they are found. The six folders are thus named *abrams*, *bradley*, *hmmwv*, *mrap, stryker,* and *t72*.

3. SYNTHESIS OF COMPUTER-GENERATED BATTLEFIELD STRESSES ON DIGITAL IMAGES

To measure and characterize the effects of battlefield clutter on image classification using artificial intelligence (AI) CNN models, methods have been developed to synthesize the effects of camouflage, disruptive coloration, distortion, and defilade on targets of interest found in sensor images, while leaving the background as undisturbed as possible. Also described is the synthetic application of sensor jamming or scene obscuration caused by battlefield dust and smoke. Methodologies and algorithms to create and apply incremental synthetic battlefield-type stresses to the AI classification system test images are described and presented in this section.

3.1. Introduction

To characterize the classification performance of the CNN models considered, a separate set of images (20 test images per class in this study) are used to test the trained CNN models independently of the MATLAB validation procedures. These images are incrementally stressed with the five stresses described, and the classification successes of the CNN models considered are measured as the stress levels are increased.

To test CNN models that are used in image target classification, targets of interest must be isolated from background before applying the stresses. Different stress types are then applied on the image area defined as the target to simulated battlefield stresses.

The first battlefield stress type relevant to AI CNN model classification is target camouflage, where the target found in the test image is covered with patches of random background pixels. Thus, the number of patches used, and the overall size of the patches relative to the target, are the incremental variables of this stress. Figure 4 is an example of simulated camouflage on a test image.

Figure 4. An Abrams M1A1 target image covered with background pixels to simulate camouflage

The second stress type implemented is that of disruptive coloration. Examples of disruptive coloration found in nature or previously used to hide objects are shown in Figures 5–7. Disruptive coloration in a herd of zebra makes it difficult for a predator to isolate a single zebra, by breaking up its outline (Figure 5); disruptive coloration of a patrol boat makes it difficult to identify the patrol boat (Figure 6); and the coloration of a moth's wing suggests fierce eyes to a predator (Figure 7). Of these, the randomly sized zebra stripes were synthesized to cover the targets with disruptive coloration. Figure 8 is an example of the applied disruptive coloring on a test target image. The incremental variable of this stress is the opacity of the applied stripes.

Figure 5. A herd of zebra showing the merits of disruptive coloration

Figure 6. A patrol boat demonstrating effective disruptive coloration

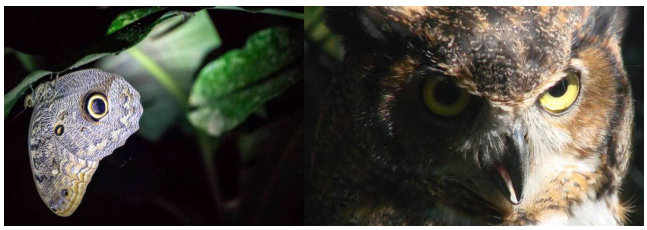

Figure 7. A moth increasing its fierceness with disruptive coloration

Figure 8. A target image overlain with random-width vertical stripes to simulate disruptive coloring

The third stress applied tries to simulate intentional target distortion by pixelating the target. Pixelation in this context means choosing random target pixels and enlarging their color and intensity to neighboring pixels in incremental fashion. Care is taken to expand the pixels beyond the target outline to eventually change the outline of the target. Target distortion is brought about with the loss of target detail that pixel dilution causes. The incremental variable of distortion stress is the size increase of the pixelation. Figure 9 is the distortion stress applied to a test target image.

Figure 9. A pixelated target image to simulate distortion

The fourth stress applied is caused by battlefield hull-down defilade. Defilade is the act of mostly burying the military target to make its silhouette less recognizable. Thus, starting at the bottom, the target is incrementally filled-in with background pixels. Figure 10 is the synthesized defilade stress applied to a test image target. The incremental variable is the percentage of the target covered with background pixels.

Figure 10. A target image covered with background pixels from the bottom of the target to simulate hull defilade

Finally, the fifth stress applied the test images is adding noise to the whole image to simulate either sensor jamming or smoke and dust obscuration. This stress can be intentional (as in obscuration smoke) or incidental (as in munitions smoke or dust). Figure 11 is an example of a test target imaged obscured by “salt and pepper” noise. The incremental variable is the percentage of image pixels randomly changed to black or white.

Figure 11. A target image covered with “salt and pepper” noise to simulate sensor jamming or scene obscuration

3.2. Methodology

To create most of these stresses (scene obscuration notwithstanding), the target of interest must first be determined from the background in the test images. This target identification, called data labeling, must be performed manually, and is facilitated by the MATLAB creation of “masks” that are assemblages of image regions that are similar in color and intensity. Creation of masks is performed with the internal MATLAB function `imsegkmeans`. A script was written (`DEFINE_MASKS.m`) that displays the masks of the k-mean assemblage divided into five groups, and lets the user decide which mask(s) make up the target of interest. A snapshot of the display is shown in Figure 12, and the script is shown in the Appendix B.

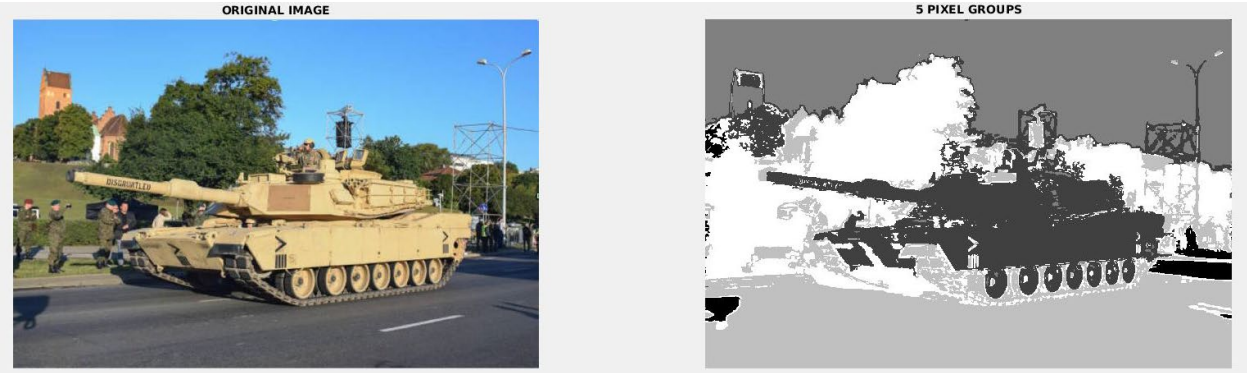

Figure 12. Display of DEFINE_MASK.m script showing the k-mean pixel groupings

The user is prompted to choose the pixel group(s) that represents the target (in the case of Figure 12, light gray), the group(s) that represents the background from which background pixels are to be used (in this case, white and medium gray), and the pixel groups that are background but should not be used in adding stress to the target (in this case, black). The script then shows the target defined by the chosen mask(s) as seen in Figure 13. Different processing parameter options, such as image resizing, and different color mappings can be changed in the script to help enhance k-mean pixel grouping to isolate the target from the background.

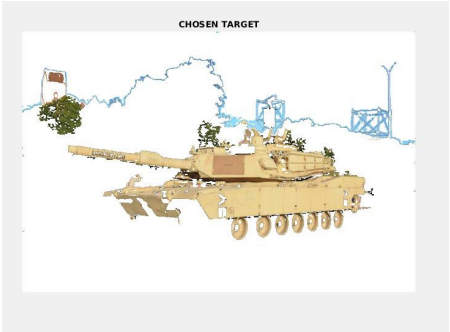

Figure 13. The target defined by the chosen mask(s)

Figure 13 represents the part of the image that will get incrementally changed as different stresses are applied. As can be seen, this masking is imperfect, and in some of the test images parts of the background is included (or part of the target is excluded), but if a majority of the target that includes its salient features is represented in the target mask, the target should be adequately represented by the mask to effectively apply battlefield stresses to the test images.

To apply camouflage stress, the number of camouflage patches placed on the target is specified. For this work, analyses of 5, 10, 15, 20, 25, and 30 camouflage patches were performed concurrently. The script CREATE_TEST_STRESS_SUITE.m uses the function camouflage.m, both found in Appendix B, and places the seeds for these patches at random locations on the defined target mask(s). The script then chooses random pixels from the image defined by the background mask(s) and places them adjacent to the seed locations to form patches. This process is repeated to grow the patches until the specified percentage of the target is covered. Because the seed locations are randomly placed on the target and may or may not cover a salient feature used in classification, 25 sets of incrementally stressed camouflage test images are created, each with different, random, seed locations. The test results of the trials can then be averaged, and statistical classification differences due to patch placement calculated. The camouflage stressed images are stored in the TEST_MILITARY_VEHICLES/cammo_data/ directory.

To apply disruptive coloration stress to the test images, the part of the image overlain by the target mask(s) is covered with alternating black and white stripes of random width. The script CREATE_TEST_STRESS_SUITE.m uses the function coloration.m, both found in Appendix B, to apply disruptive coloration stress. The stripes in this algorithm can be chosen to be vertical or horizontal. Because of the randomized variable width of the stripes superimposed on the target, 25 sets of disruptive coloration stress test images were created, yielding the necessary test data for 25 trials. The test results of the trials can then be averaged, and statistical classification differences due to stripe-width calculated. The transparency of the disruptive coloration is decreased incrementally to provide a range of disruptive coloration stress test images. The vertical disruptive coloration stressed images are stored in the TEST_MILITARY_VEHICLES/coloration1_data/ directory and the horizontal disruptive coloration images are stores in the TEST_MILITARY_VEHICLES/coloration2_data/ directory.

To apply distortion stress to the targets found in the test images, the MATLAB internal function imdilate increases the effective pixel size of the image covered by the target mask(s). The script CREATE_TEST_STRESS_SUITE.m uses the function distortion.m, both found in Appendix B, to pixelate the target while leaving the background intact. An extra step is taken to adjust the target mask(s) to take into the account its increased

area due to dilation along the edge of the target mask(s). This allows the target profile to change with increased dilation, which leads to a better target distortion approximation. Because of the inherent variability of the pixelation, test image data for 25 trials were created applying distortion stress. The test results of the trials can then be averaged and statistical classification differences due to different pixelation calculated. The size of the dilation (pixelation) is controlled by the `strel` function, and its variation sets up an incremental range of distortion stresses needed to characterize CNN CF. The resulting distortion stressed suite of test images is stored in the `TEST_MILITARY_VEHICLES/distortion_data/` directory.

Defilade stress is applied to test images in a similar way than the camouflage stress, but instead of overlaying random background pixels around random patch seeds placed on the target, the random background pixels fill in the bottommost position of the image covered by the target mask(s). The script `CREATE_TEST_STRESS_SUITE.m` uses the function `defilade.m`, both found in Appendix B, to simulate the partial burial of the target. This process is iterated until a specified percentage of the target is covered with random background pixels. The resulting defilade suite of test images is stored in the `TEST_MILITARY_VEHICLES/defilade data/` directory.

Sensor jamming or battlefield obscuration stress is applied by overlaying salt and pepper noise over the whole image using the `imnoise` internal MATLAB command. This is the only battlefield stress type that is applied over the whole image. Salt and pepper noise presents itself as sparsely occurring white and black pixels over the image. The script `CREATE_TEST_STRESS_SUITE.m` uses the function `jam.m`, both found in Appendix B, to simulate scene obscuration of jamming of the sensor with Gaussian noise. The intensity of the stress is controlled by the density of the salt and pepper corrupted pixel density in the image. The resulting jam or obscuration suite of test images is stored in the `TEST_MILITARY_VEHICLES/jam_data/` directory.

To summarize the stress types, Table 1 recaps the 11 battlefield stress synthesis scripts that were written for this study. Five percent stress increments were chosen for this study, ranging for 0% to 100%. The calibration of different stress increments is difficult to perform. Some, like camouflage and defilade stresses, can be calibrated by the percentage of the target that the placed background pixels cover, while other stresses, such as disruptive coloration opacity or distortion pixelation size increments are more difficult to cross-calibrate to other stresses.

Table 1. Summary of Battlefield Stressor Simulations

Stress	Explanation	Test Suites Used (trials)
Camouflage: 5 patches	Camouflage grown in 5 patches	25
Camouflage: 10 patches	Camouflage grown in 10 patches	25
Camouflage: 15 patches	Camouflage grown in 15 patches	25
Camouflage: 20 patches	Camouflage grown in 20 patches	25
Camouflage: 25 patches	Camouflage grown in 25 patches	25
Camouflage: 30 patches	Camouflage grown in 30 patches	25
Disruptive Coloration I	Black and white vertical stripes	25
Disruptive Coloration II	Black and white horizontal stripes	25
Distortion	Target and edge pixelation	25
Hull Defilade	Camouflage starting at target bottom	1
Obscuration/Jamming	Salt and Pepper noise over whole image	1

The fined-tuned, pretrained CNN (or the trained-from-neutral) then classify the targets in these 120 test images as the battlefield stressors are incrementally increased in the images. The CF and receiver operating characteristic (ROC) curves yield information on CNN model robustness against the synthesized battlefield stresses applied to the test images.

For some classes, test images were chosen over others for their ability to mask the target without any of the background. These images generally had targets with greater color or intensity contrast relative their background and this may bias the outcome of their initial classification success rate. The characteristic of CF with target stress should not be affected by this test image selection process.

4. CNN MODELS CHOSEN FOR ANALYSIS

To establish an AI classification CNN model testing methodology, representative neural network models have been chosen and downloaded for analysis. Because of training constraint, pretrained CNN models were originally imported into MATLAB and transfer-learning was used to fine-tune the models to classify six classes of military vehicles. Nine award-winning models that define the efficiency (size and speed) versus effectiveness (accuracy) envelope of neural network models available were used, and both training from neutral and transfer-learning techniques were used to prepare the neural network models to classify military vehicle classes. With a large image database, a subset of the neural network models used can be trained-from-neutral weights and biases, allowing analysis of the training requirements. Porting of CNN models written in other languages is also discussed.

4.1. Introduction

Because this research deals with the analysis of CNN models in stressed environments, CNN models were not designed. The design of CNN models is quite artful and requires long and expensive iterative training/testing to optimize. Instead, CNN models that have won national competitions and are available to download from MATLAB are used to develop the methodology to test their behavior, robustness, and failure with respect to battlefield stresses.

Many pretrained CNN models are available through the MathWorks (the developers of MATLAB) site, and it was decided to work with multiple models to mitigate any anomalies due to a particular model. Figure 14 shows the classification accuracy of the pretrained CNN models available in MATLAB versus the time it takes to make their classification predictions. The dashed line between the CNN models that define the envelope defines the Pareto frontier. It was decided to use the CNN models that make up the Pareto frontier to develop the analysis.

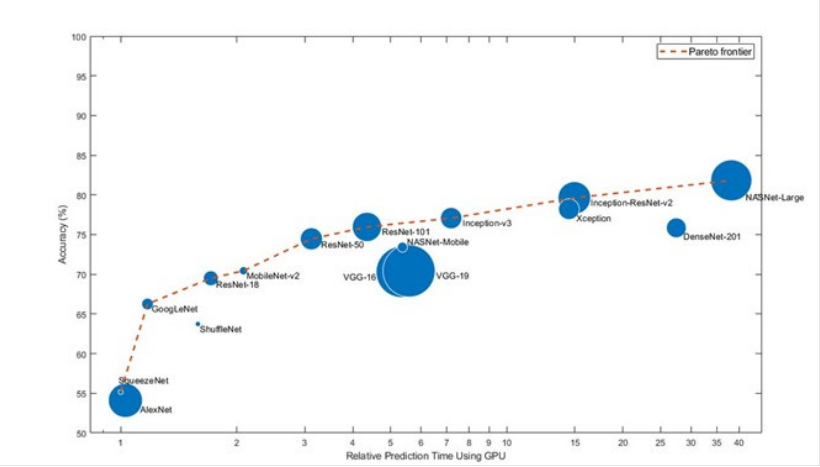

Figure 14. Pretrained CNN models showing the Pareto Front [https://www.mathworks.com/solutions/deep-learning/models.html]

Thus, the `Squeezenet`, `Googlenet`, `Resnet18`, `Mobilenetv2`, `Resnet50`, `Resnet101`, `Inceptionv3`, `Inceptionresnetv2`, and the `Nasnetlarge` CNN models were used for this analysis.

4.2. Training and Fine-Tuning

The pretrained CNN models available and downloaded for analysis are well trained to classify general classes and can be fined-tuned with transfer-learning. Most can also have their internal weights and biases reset and then trained-from-neutral on images other than those used in their original pretraining. With the inclusion of abundant video frames as training images and the ability to translate and mirror training images that give some additional use of duplicate training images, the database has become large enough to train the CNN models from neutral. The six CNN models that are easily able to have their weights and biases reset to neutral are: `Squeezenet`, `Googlenet`, `Resnet18`, `Mobilenet`, `Resnet50`, and `Resnet101`.

The chosen pretrained CNN models have been fine-tuned to classify the six classes of military vehicles using transfer-learning techniques, and some have been trained-from-neutral (starting with neutral weights and biases). To compare classification robustness of pretrained and trained-from-neutral CNN models, both pretrained and neutral CNN models are trained using the same training sets consisting of 7,500 images derived from web images, 3D model renderings, and video frame images. The pretrained CNN models were trained with less iterations (called epochs) than the neutral CNN models. In both cases, 30% of the images were separated in a random fashion from the training set and were kept for training validation.

Both transfer-learning and training-from-neutral methods yield accuracies close to 100% by the end of the training iterations. Figure 15 shows the training and validation trends for the `Googlenet` network fine-tuned with transfer-learning, and Figure 16 shows the training and validation trends of the `Googlenet` model trained-from-neutral on the same training images.

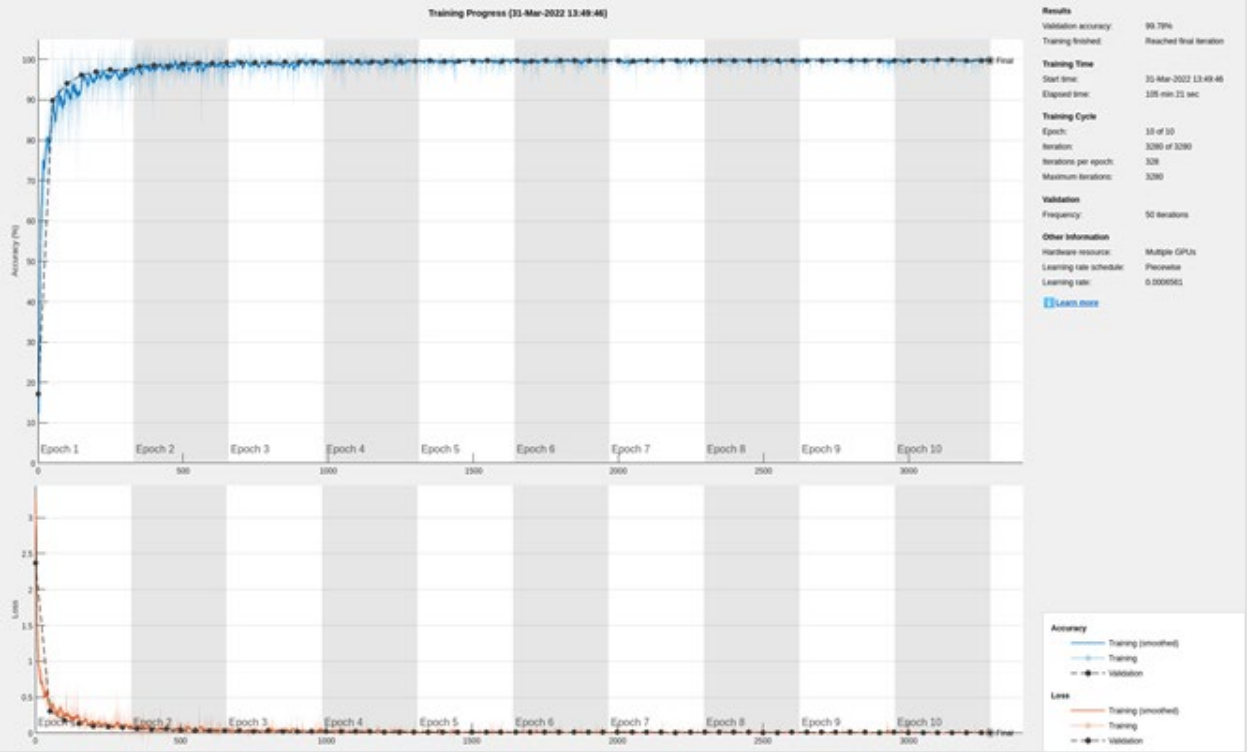

Figure 15. Transfer-learning of the Googlenet CNN model

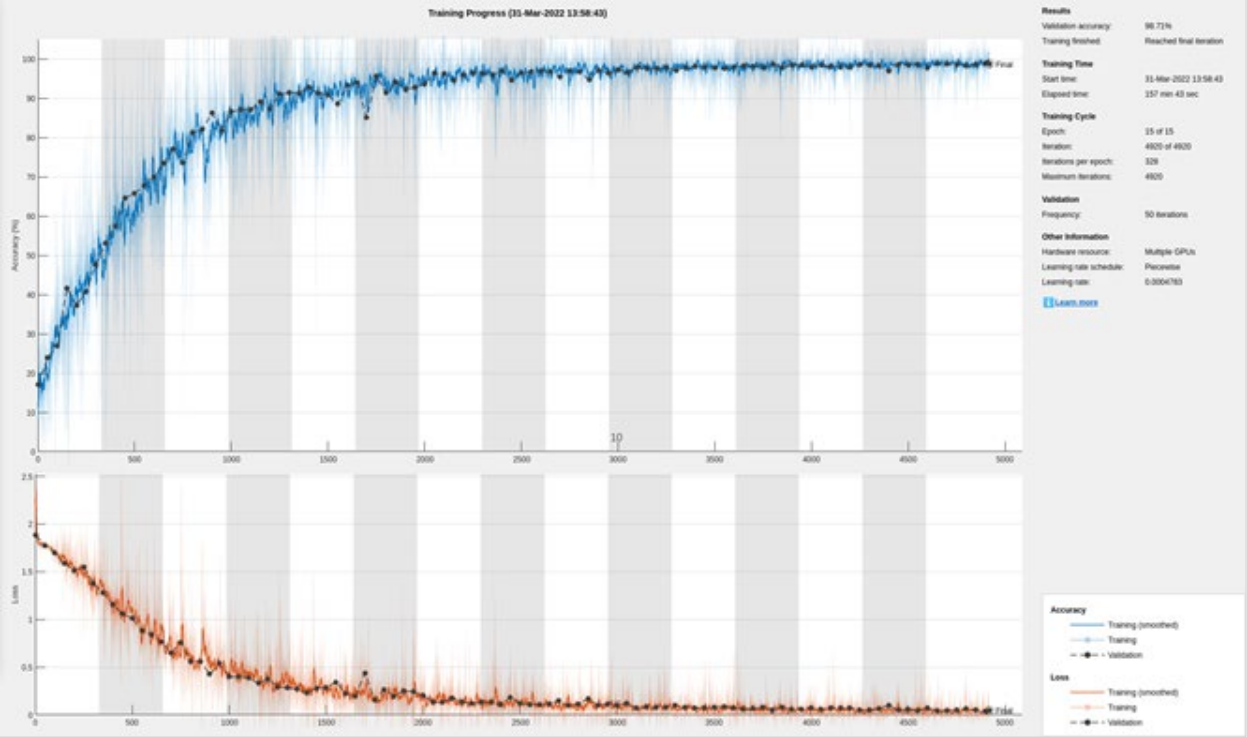

Figure 16. Training Googlenet CNN model from neutral

Figures 15 and 16 show that pretrained CNN models fine-tuned to classify a specialized set of images require fewer iterations than the CNN trained-from-neutral. This was expected from previously published information. Because of this, the transfer-learning fine-tuning is allowed to train for only 10 epochs, while the CNN models trained-from-neutral are allowed to train for 15 epochs.

When all CNN models were trained on the same data sets so that a performance comparison was possible, it was noted that the classification accuracy levels obtained by different CNN models depend on the structure and the size of the training sets. This was indicated by the fact that, in broad terms, the `Mobilenet` and `Resnet50` CNN models were more successful in the classification of military vehicles when trained with this data set, even though the Pareto curve shown in Figure 14 indicates that the `Nasnetlarge` should have been more successful. This is undoubtedly due to the limited size of the training sets. Perhaps a larger fine-tuning training set would have made the `Nasnetlarge` a better classifier than `Mobilenet` and `Resnet50`.

4.3. Importing CNN Models from Other Environments

Even though MATLAB remains a strong environment for analysis, the cutting-edge CNN model development has coalesced to Python in the PyTorch or TensorFlow environment. Fortunately, PyTorch and TensorFlow models and their pretraining information can be imported into MATLAB where pre- and post-analysis can be

performed on the data before classification. The importation of TensorFlow files is different than that of PyTorch.

There are two versions of TensorFlow (TensorFlow 1, that uses the Keras neural network library, and the more modern TensorFlow 2). TensorFlow 1 uses the HDF5 format and TensorFlow 2 uses either the HDF5 or SavedModel formats. The more modern SavedModel format is recommended due to its translation robustness, and the ability to import complex models with layers not supported for conversion into built-in MATLAB layers.

TensorFlow models saved in HDF5 format can also be imported into MATLAB, but more manual adjustments will be necessary for complicated or unsupported model features. The internal commands used to import models, namely `importTensorFlowNetwork` and `importKerasNetwork` can be downloaded using MATLAB's `Add-On Explorer`.

To import models developed in the PyTorch environment, they must first be exported to the open neural network exchange (ONNX) format. The ONNX format models are then imported using the `importONNXNetwork` internal command. If CNN models are created in the MATLAB environment, for example using the `Deep Network Designer` GUI, the model can be exported in the ONNX environment using the `exportONNXNetwork` command. For information on how to incorporate these commands in the MATLAB environment using the Deep Learning Toolbox, simply type the command on the command line.

5. METRICS USED IN THE CNN MODEL EVALUATION

Different metrics that measure and compare CNN model classification success in stressed environments are developed, discussed, and compared. Averaged curves and decile bar graphs of classification success as a function of stress imposed on test target images help smooth out the curves enough to show their shape and may help in designing target stress for AI classification evasion. For more compact figures of merit that importantly take account of the system threshold level, the ROC curve has been developed and is used. Ultimately, single figures of merit for a CNN model are developed by calculating the area under both the CF curves and the ROC curves.

5.1. Introduction

Analysis of CNN models not only requires large training and testing image databases, but also metrics by which to determine the success of CNN classification. Quantitative figures-of-merit of their robustness to stresses in sensor data, implemented by synthesized stresses on the test image targets, are sought in this study. To obtain these metrics, sensor stress is introduced incrementally on the targets found in the test images. Even though some “natural” stresses exist in the training data, the stresses synthesized on the test image targets are not included in the training images and therefore are new to the CNN models during testing.

5.2. CF Metric

CF curves versus test image stress are developed at a specific probability threshold. As the stresses are incrementally increased on the test image targets, the classification success of the CNN models of the six military vehicle classes considered is measured and curves of CF versus test image stress are developed. Because the classification success of CNN models depends on many, sometimes subtle, features of the test image target, a lot of CF variation exists as stresses on the test images increase. By averaging the CF curves over the variables that are not being compared, smoother CF curves that are dependent on fewer variables are possible to analyze the variables of interest.

The MATLAB script `TOTAL_analysis.m` gathers the test classification results, as well as the true positive rate (TPR) and false positive rate (FPR) of all the CNN models results, as a function of test set trials, stresses, and classes, and forms two multidimensional matrices. The first matrix contains all the CF data, and the second matrix contains the TPRs and FPRs as a function of system threshold needed to form ROC curves. These matrices are sliced, averaged, and otherwise processed to yield different visualization data of the CNN models under test. Figures 17–19 are examples of highly averaged CF

curves. These curves are smoothed by the averaging of the families of parameters that are not being considered and show a comparison of CNN model performance.

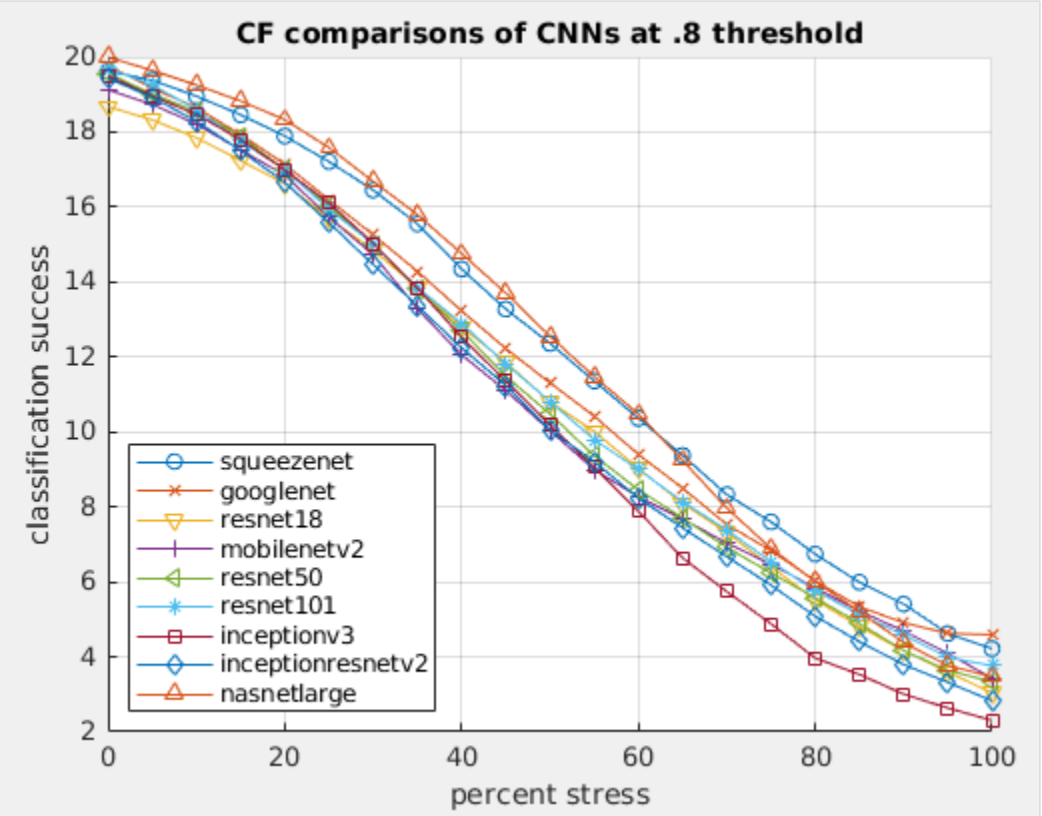

Figure 17. Comparison of CNN performance with stress types and classes averaged

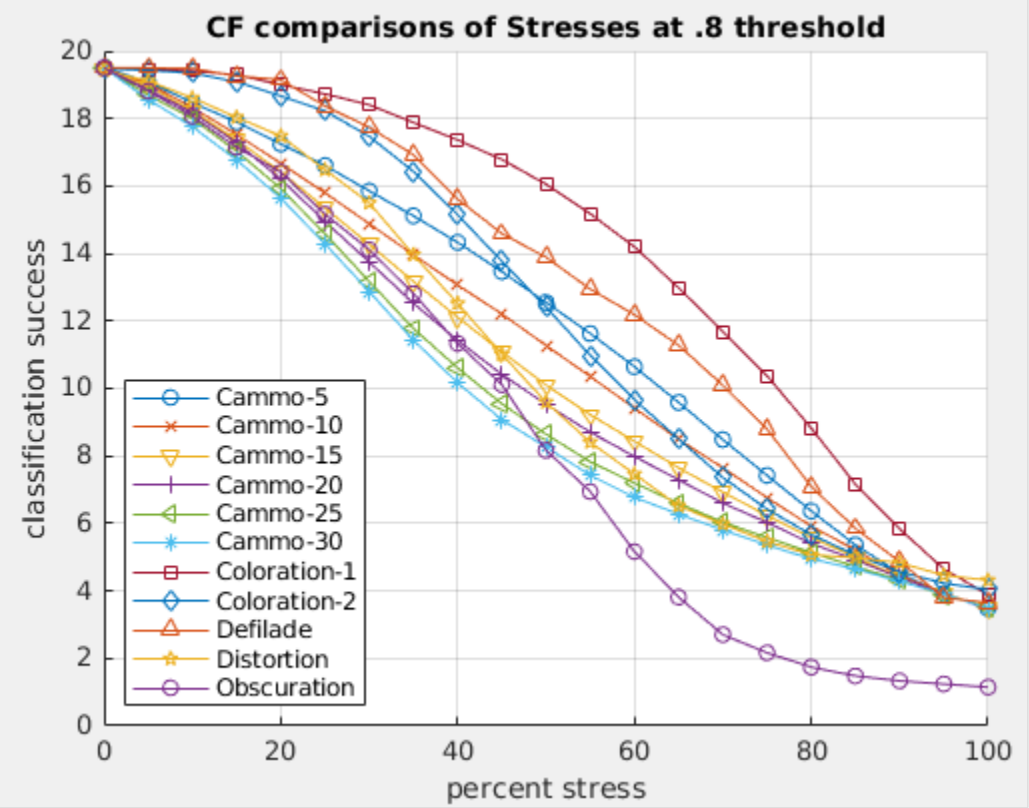

Figure 18 **Comparison of stress performance with CNN and classes averaged**

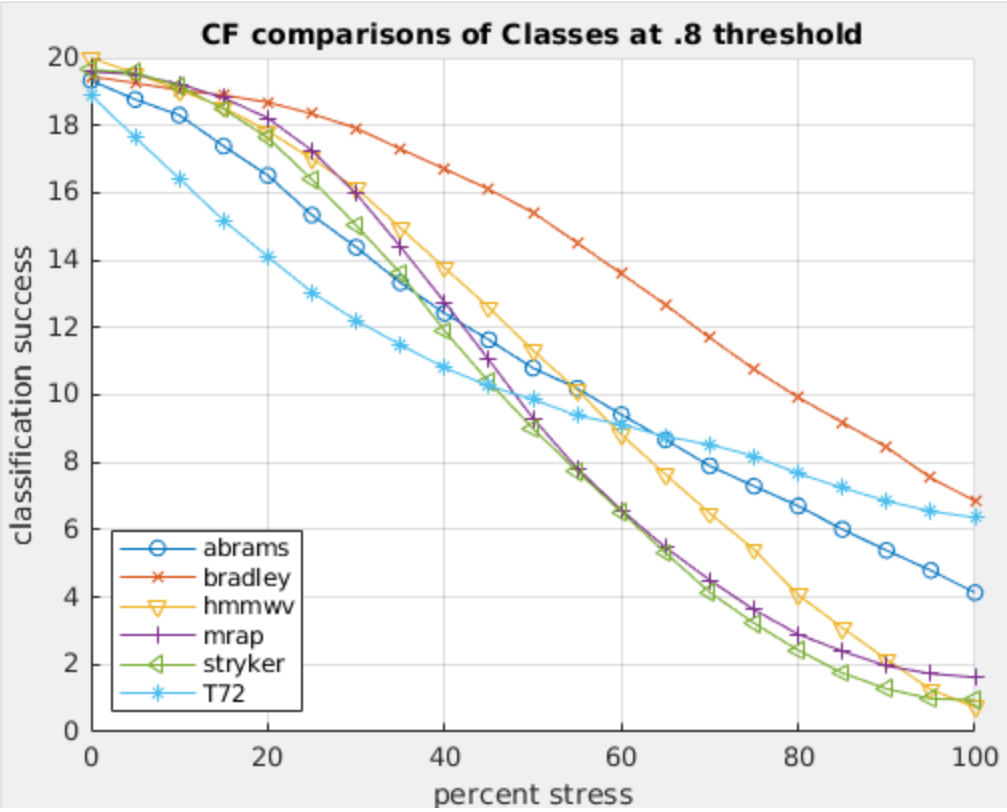

Figure 19. Comparison of class performance with CNN models and stress types averaged

5.3. Sectional CF-AUC Metrics

Another visualization that is sometimes found in the literature is the data presented in bar graphs, whose values represent the area under a section of the CF curves. Figures 20–22 show the data in bar graph form, with the curves' area divided in deciles of 10%. Bar graph visualization, which further smooths the data, is also dependent on a single, constant, threshold value.

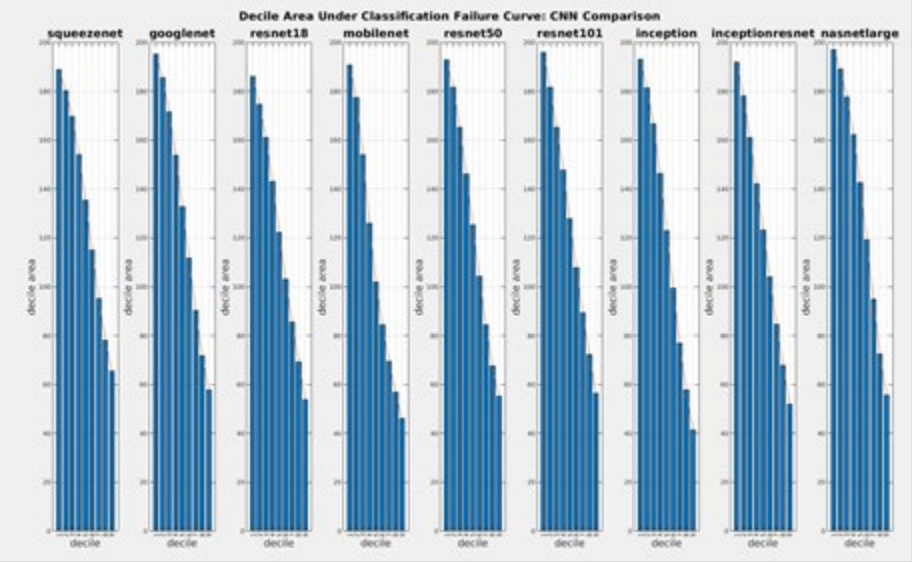

Figure 20. Bar curve representation of classification success to compare CNN models

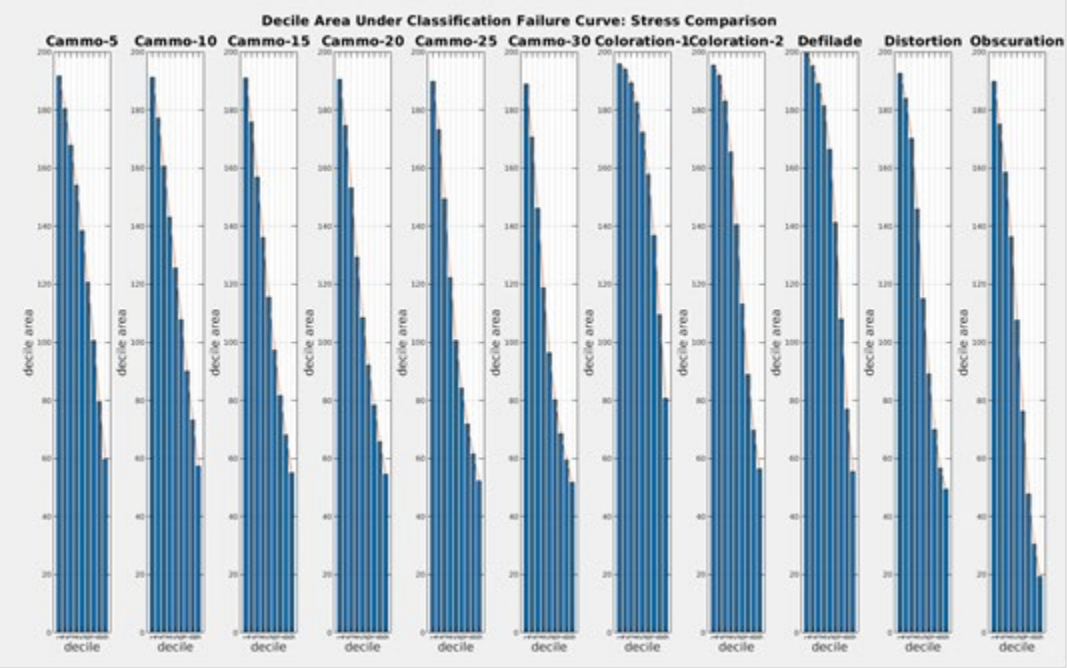

Figure 21. Bar curve representation of classification success to compare stress

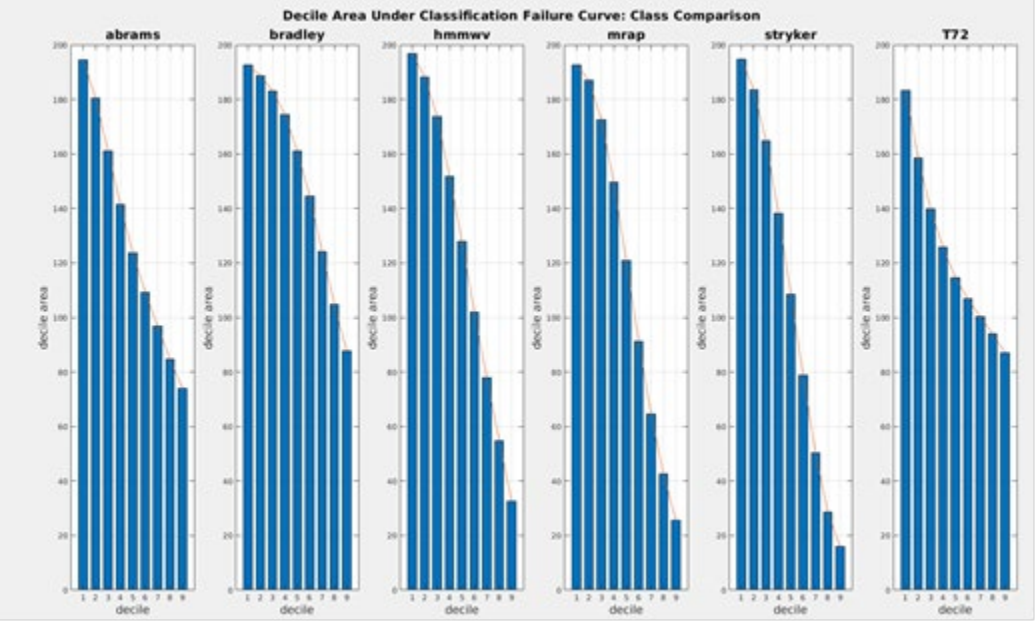

Figure 22. Bar curve representation of classification success to compare classes

Bar graph representations such as Figure 22 are good to get the smoothed characteristics of the CF curve. It is interesting that the different vehicle classes considered have different drop-off characteristics. For example, the HMMWV, MRAP, and Stryker classes drop off almost linearly, while the Abrams and T72 curves have their slope decrease with stress (concave) and the Bradley curve has the slope increase with stress (convex).

5.4. ROC Curve Metric

The other type of performance metric that is widely used in AI deep learning analysis is the ROC curve, because it mitigates the threshold value variable. The points that make up the ROC curves are dependent on the threshold value of the classification system, and therefore the threshold value does not need to be considered as an explicit parameter.

The ROC curve is the TPR plotted against the FPR for various threshold values. These are defined as

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

where *TP* is the number of true positives, *FN* is the number of false negatives, *FP* is the number of false positives and *TN* is the number of true negatives of the classification set.

The ROC curves for the same data as found in the bar curves shown in Figures 20–22 are shown in Figures 23–25 for different test image stress levels. In these ROC curve plots, the best classification occurs when the curves quickly rise to coordinate (0, 1), and stay at an ordinate (TPR) value of 1 for subsequent values of the abscissa (FPR). This is seen at ROC curves calculated in the first plot of the series (0% stress). The ROC curve diagonal (from coordinates (0, 0) to (1, 1)), implies completely random classification. Figures 23–25 show that the CNN models considered lose the ability to classify with greater stress added to the test images.

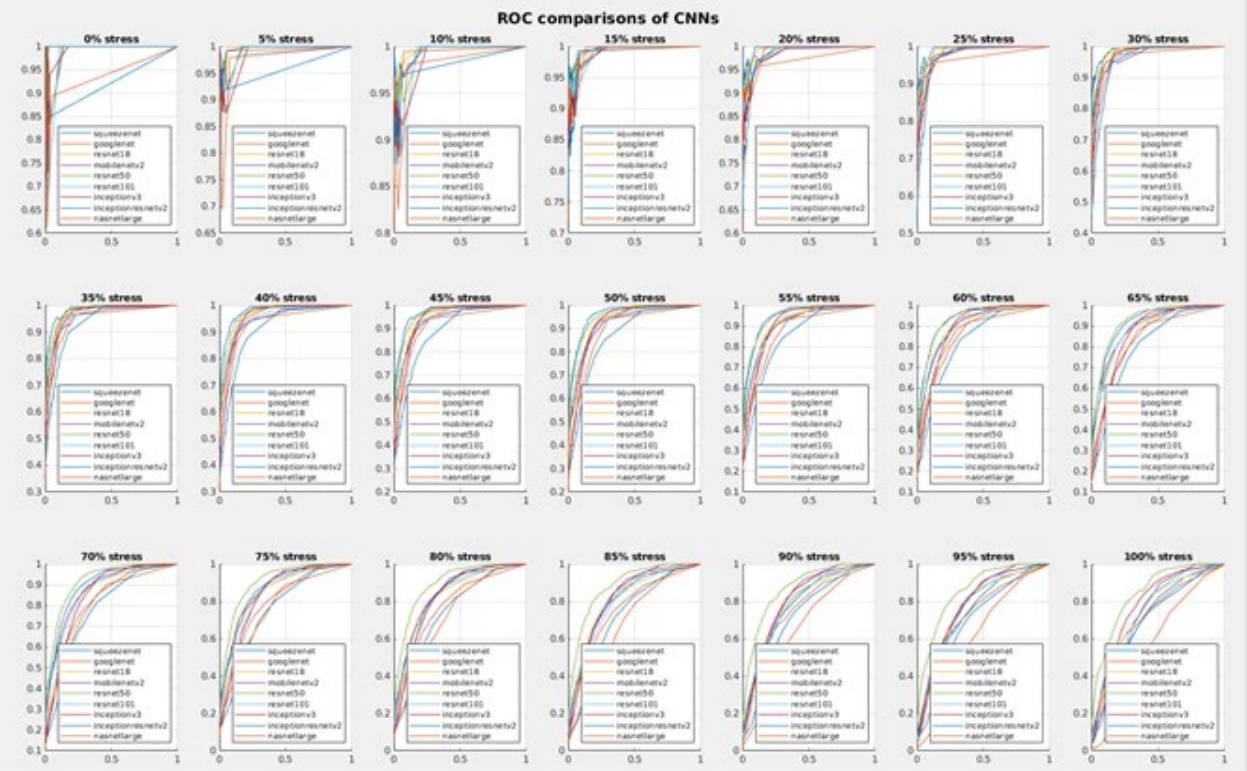

Figure 23. ROC curves for various CNNs for incrementally increasing test image stress

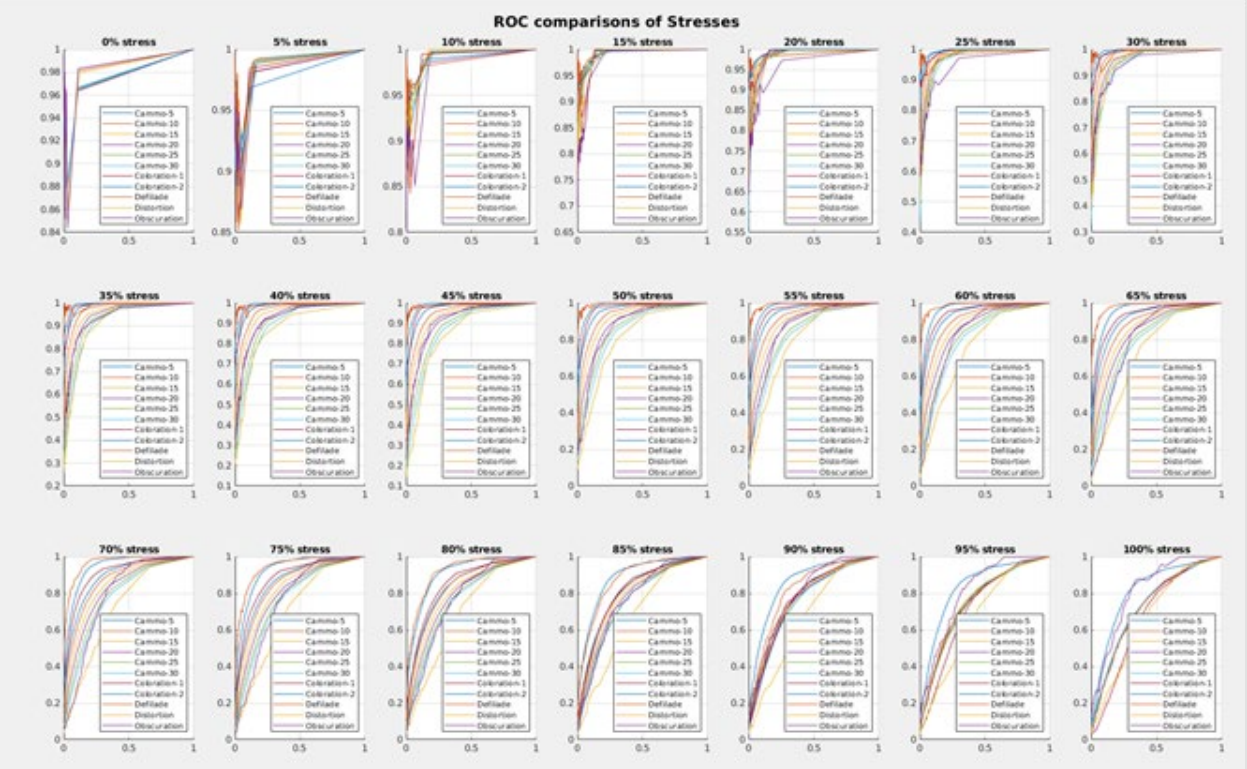

Figure 24. ROC curves for various stresses for incrementally increasing test image stress

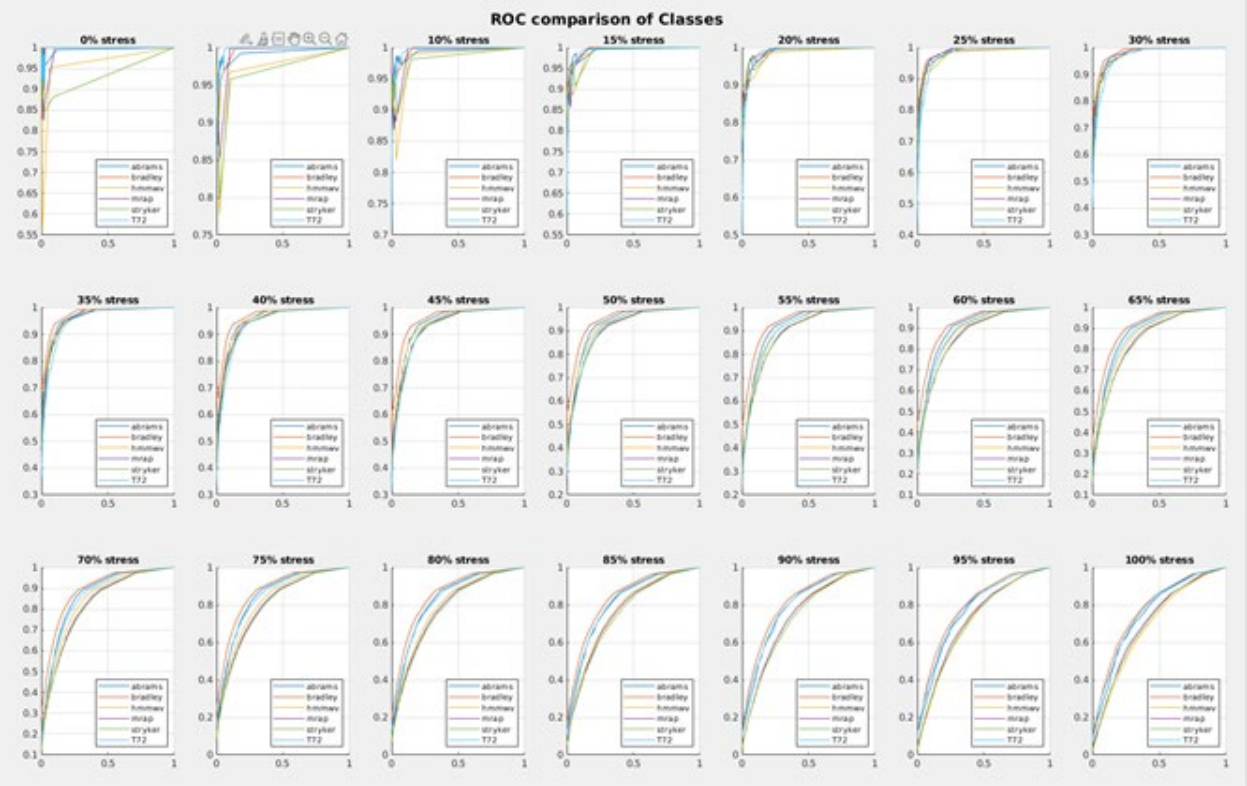

Figure 25. ROC curves for various classes for incrementally increasing test image stress

The MATLAB internal `ROCplot` function does not calculate enough points to yield good ROC curves for this analysis. The calculation of acceptable ROC curves was forced by rewriting the function to force the calculation of TPR and FPR at enough threshold levels.

5.5. Continuous CF-AUC and ROC-AUC Metrics

The many plots in Figures 23–25 contain a considerable amount of data to interpret and tend to be redundant and perhaps confusing. An even more compact metric to measure the ability of CNN models to correctly classify test images is to take the area under the curve (AUC) of the CF shown in Figures 17–19 and the ROC curves shown in Figures 23–25. The CF-AUC values are plotted against the threshold level, while the ROC-AUC values are plotted as a function of stress levels. Figures 26–31 show this compact way of presenting the CF and ROC data, respectively. Calculating the ROC and AUC-ROC of averaged data is symmetric in the sense that the averaged ROC curves of particular data are the same as the ROC curve of the averaged data.

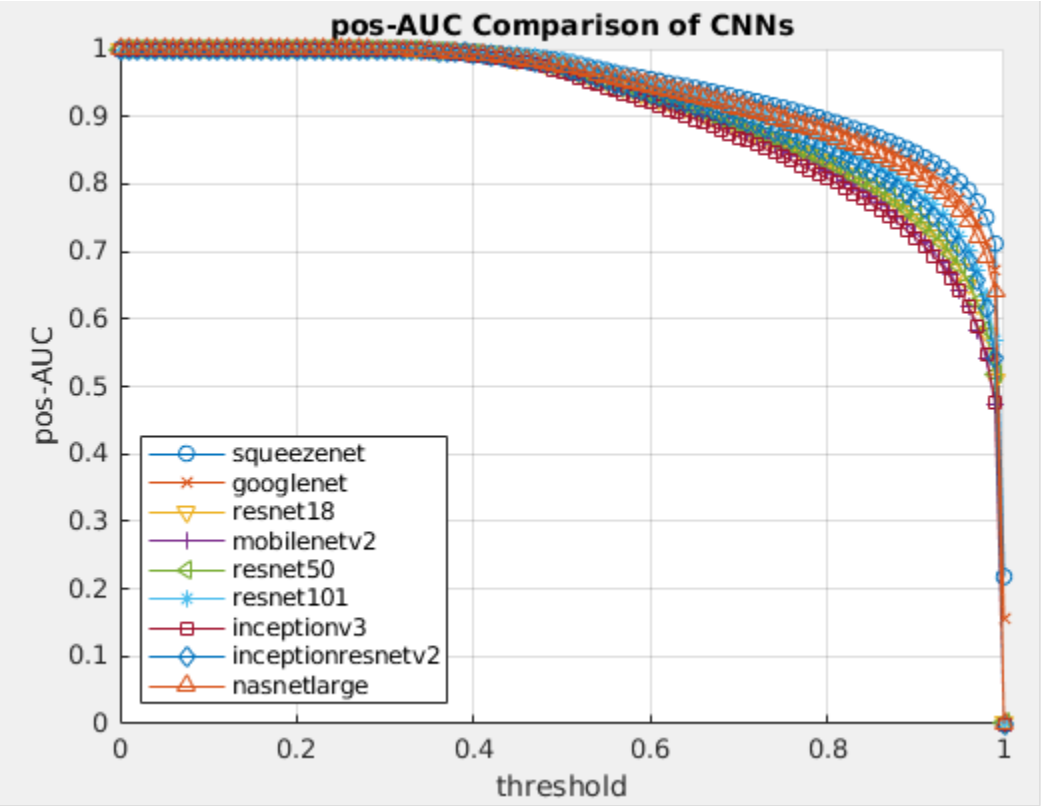

Figure 26. CF-AUC curve comparing the different CNNs analyzed

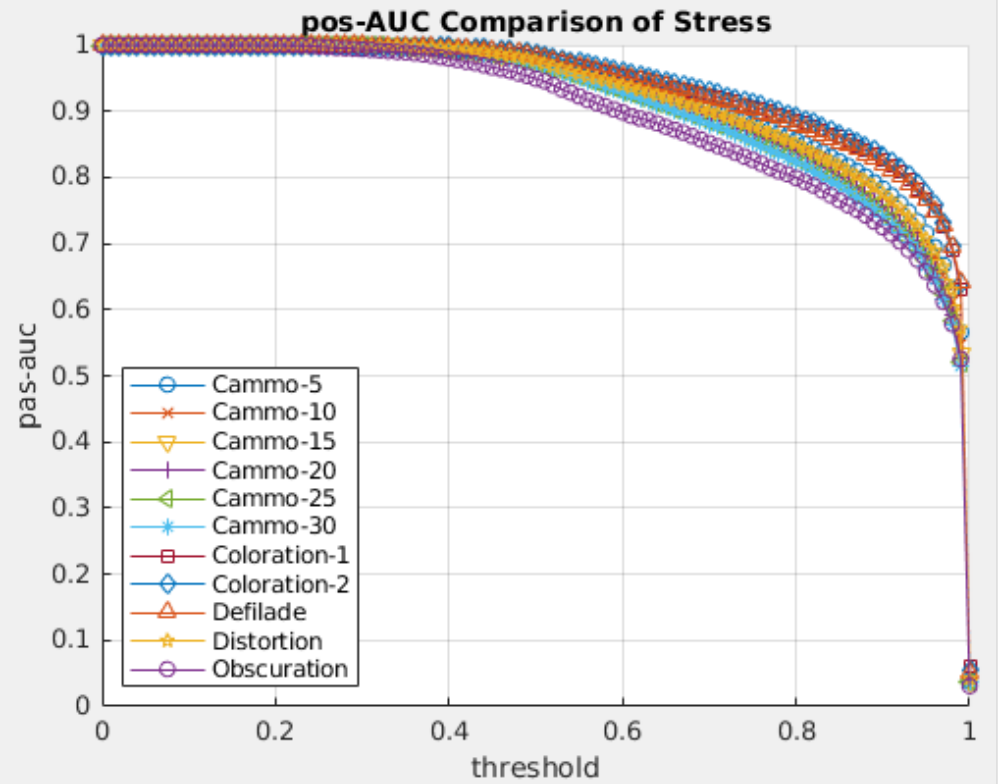

Figure 27. CF-AUC curves comparing the different stresses applied

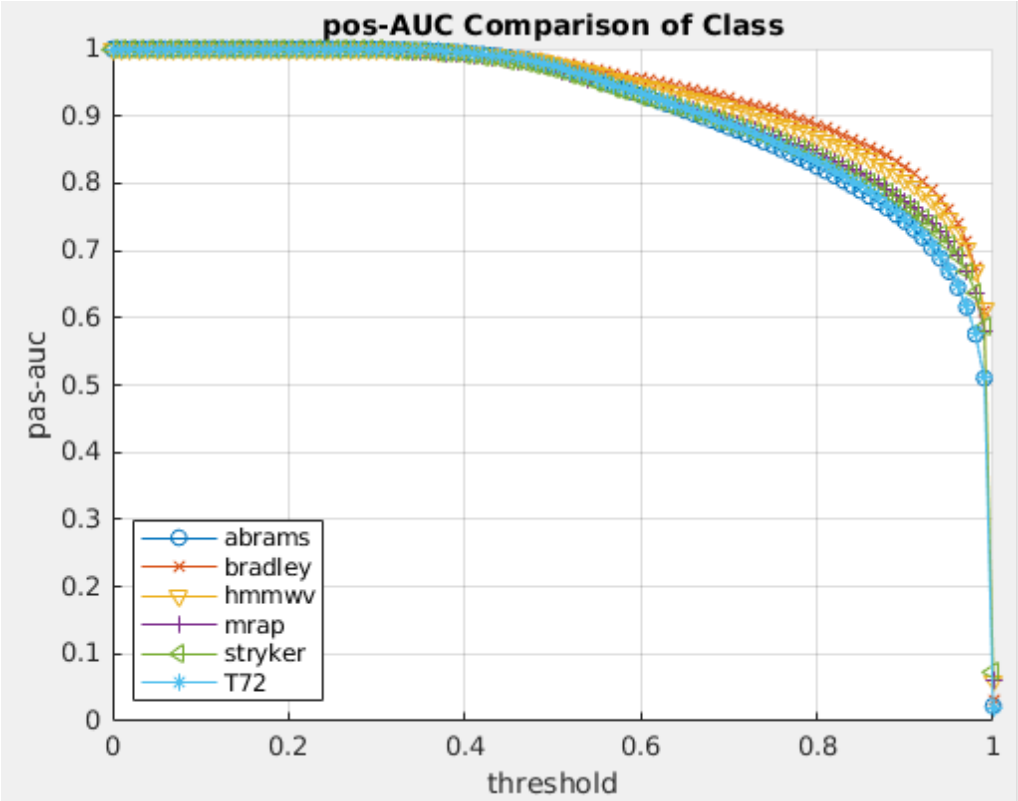

Figure 28. CF-AUC curves comparing the different classes analyzed

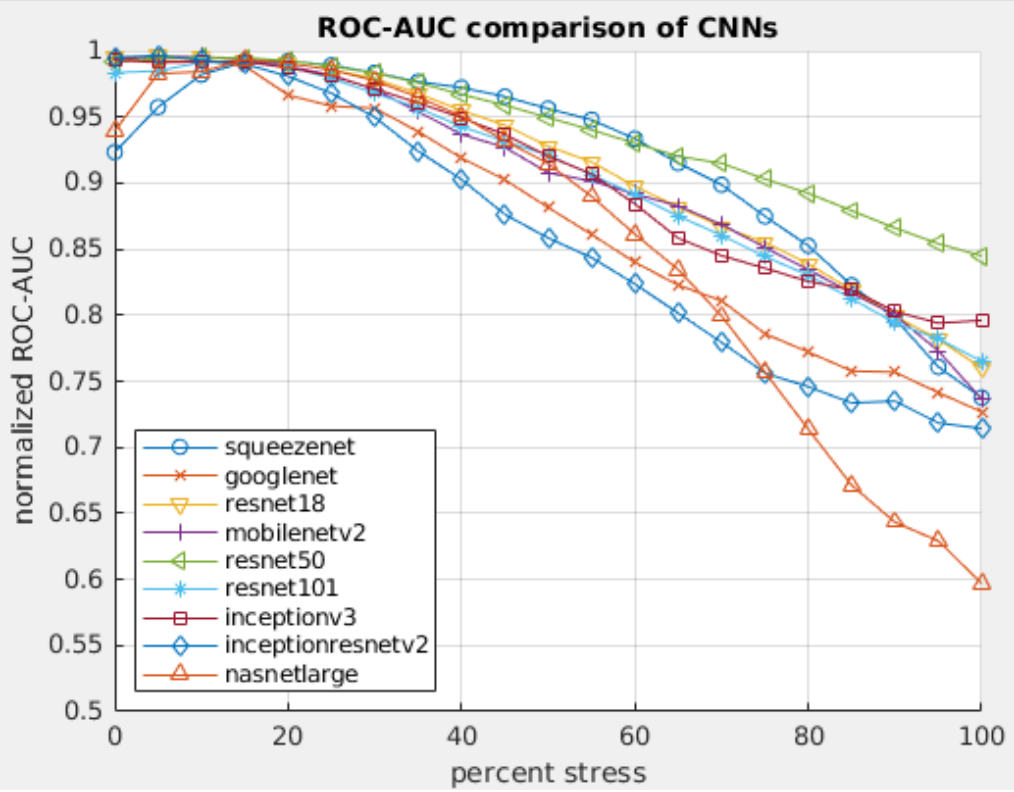

Figure 29. ROC-AUC curves comparing the CNNs tested

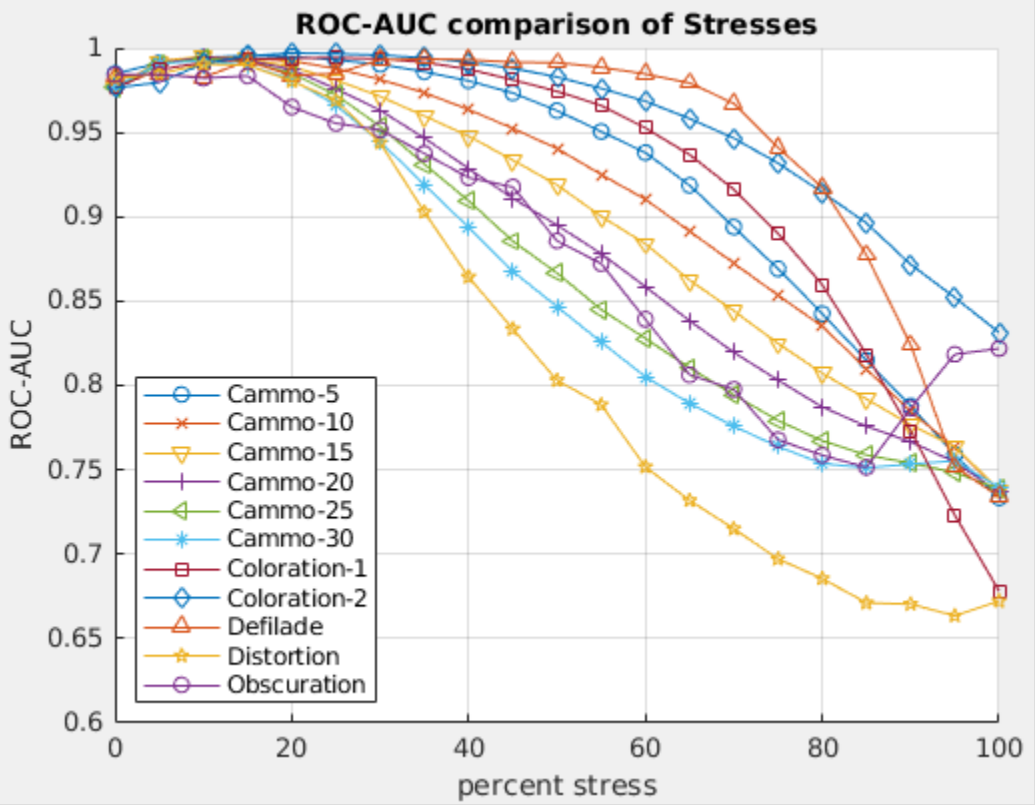

Figure 30. ROC-AUC curves comparing the stresses tested

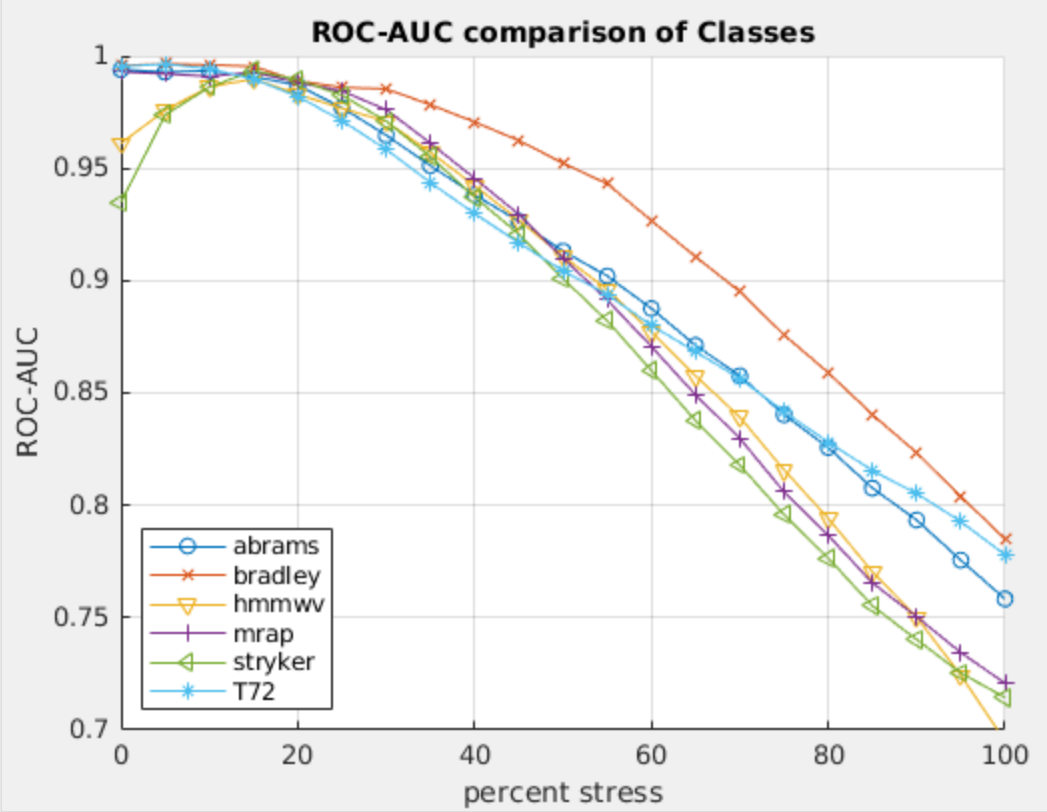

Figure 31. ROC-AUC curves comparing the classes tested

An intermediate step taken to calculate the ROC-AUC needs further explanation. Because the ROC curves do not have equally spaced (or sorted) false alarm rates of the CNN classification for different thresholds, different ROC curves do not necessarily cover the same ordinate length. Figure 32 shows the ROC curves for different CNN models at the 65% stress level.

To complete the curve so that the AUC can be calculated, the curves were extended to a TPR and FPR of 1. Figure 33 shows this ROC curve manipulation. By extending the ROC curves to a TPR and FPR of 1, it is possible to calculate and compare AUC curves for different parameters, even though an error is undoubtedly introduced by the extrapolation. Similarly, the MATLAB `ROC` and `plotroc` internal functions do not calculate the ROC curve with enough data points to yield accurate ROC-AUC results.

To summarize the ROC and AUC-ROC metric, the largest advantage of using these is that the system threshold level (i.e., how certain the CNN classification system needs to be before it is accepted as a successful classification) is inherent in the calculation. Aside from the inclusion of threshold, the ROC and AUC-ROC offer potentially precise measurements of CNN classification performance in comparative terms.

AUC can also be calculated for the CF curve. AUC represents a single value, and since the CF curves are plotted against incremental stress levels added to the test images, the AUC values obtained from the CF curves are plotted against the threshold value used in the calculation.

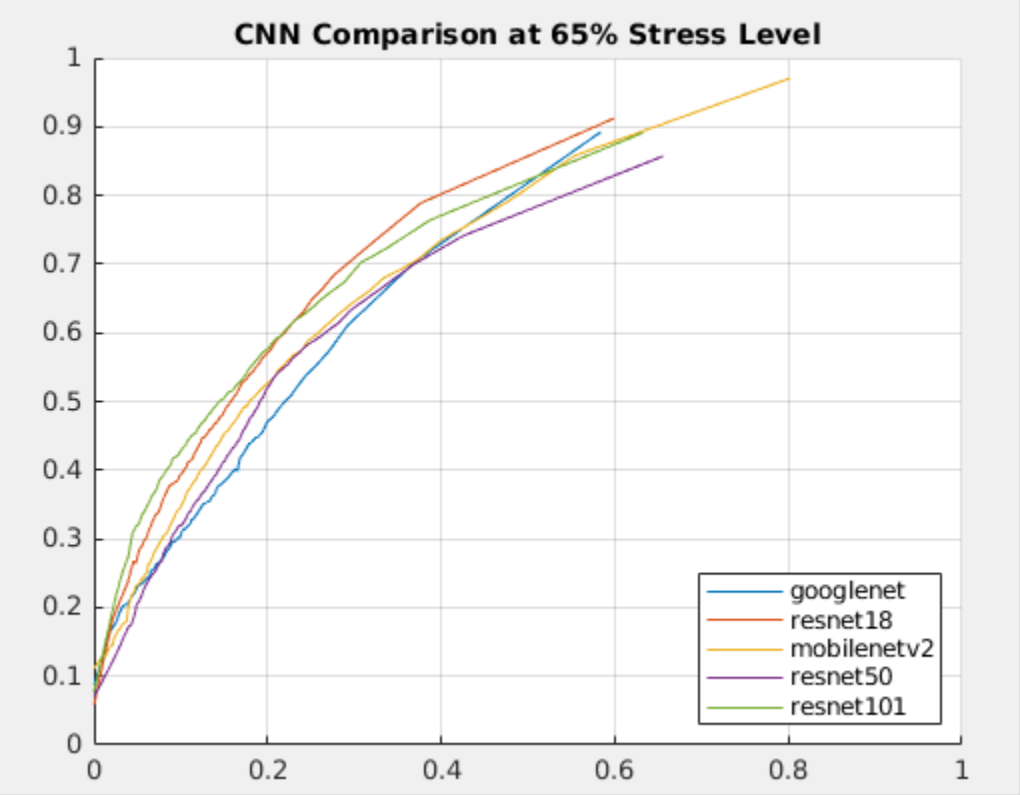

Figure 32. The ROC curves of different CNNs at 65% image stress

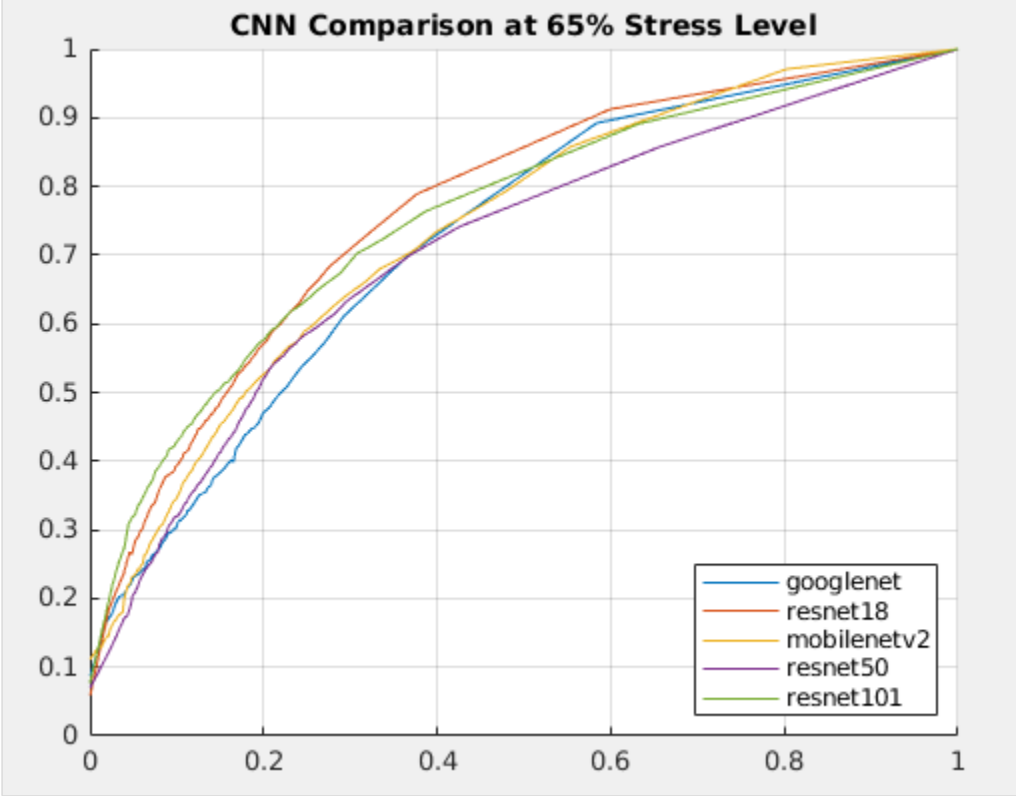

Figure 33. The ROC curves extrapolated to an FPR and TPR value of 1

Figure 34 is a comparison of CNN performance using the CF-AUC metric. By this metric `NASnetlarge` is the better performing CNN, while the `Mobilenet` CNN performs the worst. Figure 35 is a comparison of averaged CNN performance as a function of stress. Figure 35 shows that obscuration degrades the classification ability of the CNNs the most, while the vertical disruptive coloration affects the CNNs the least. Because it is difficult to calibrate between the obscuration and disruptive coloration, perhaps a more meaningful observation is that there exists a substantial difference between CNN performances stressed with vertical disruptive coloring versus horizontal disruptive coloring. These two stress types are equivalent and do not need calibration. Finally, Figure 36 shows the CF-AUC curves comparing the classification success as a function of threshold for the different classes. Figure 36 shows the classification of the Bradley to be the least susceptible to battlefield stress and the Stryker to be the most susceptible. All classes seem to be approximately equally susceptible to threshold value.

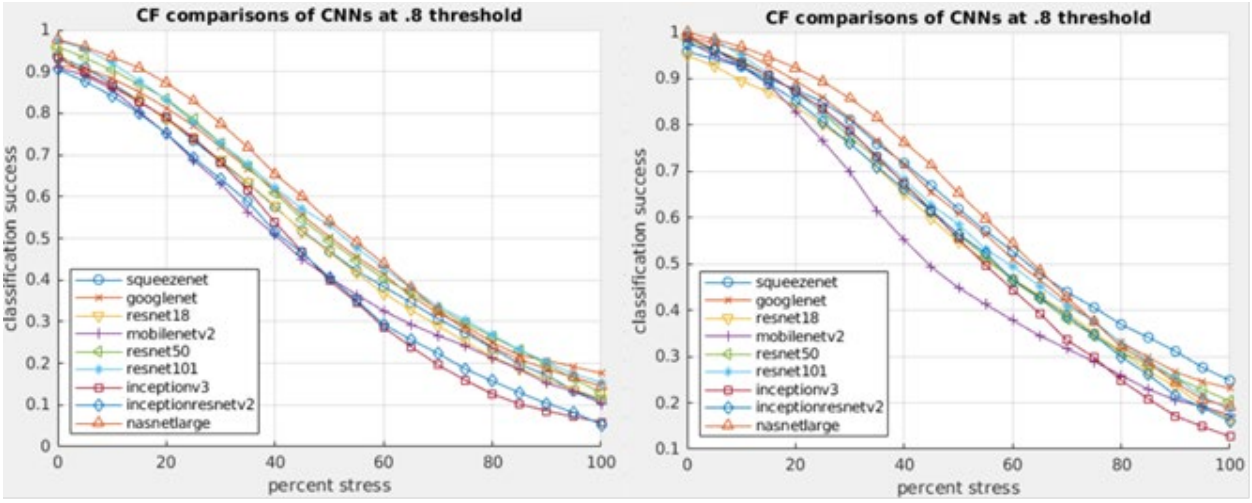

Figure 34. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different CNNs

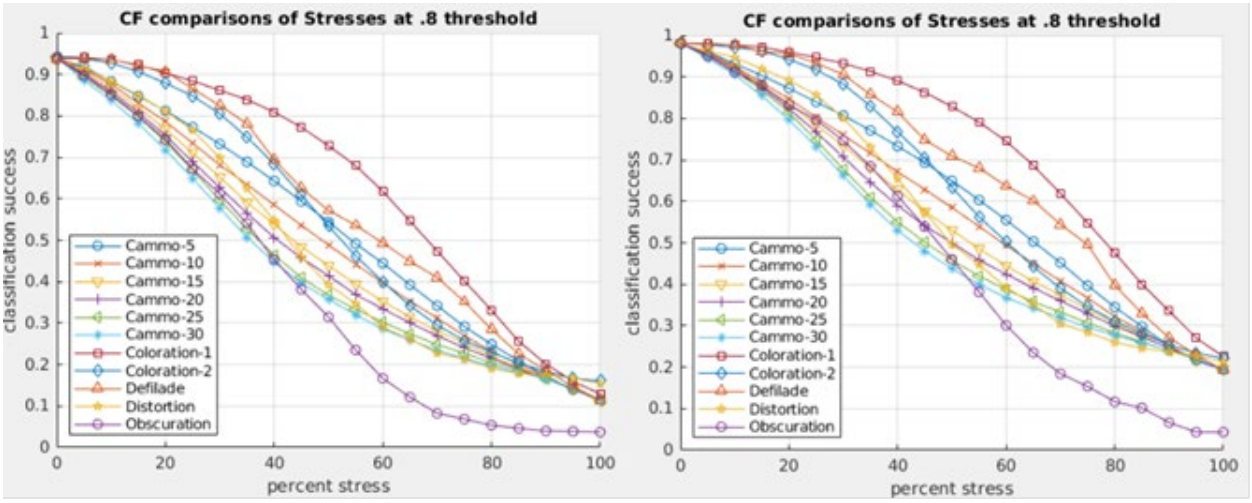

Figure 35. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different stresses

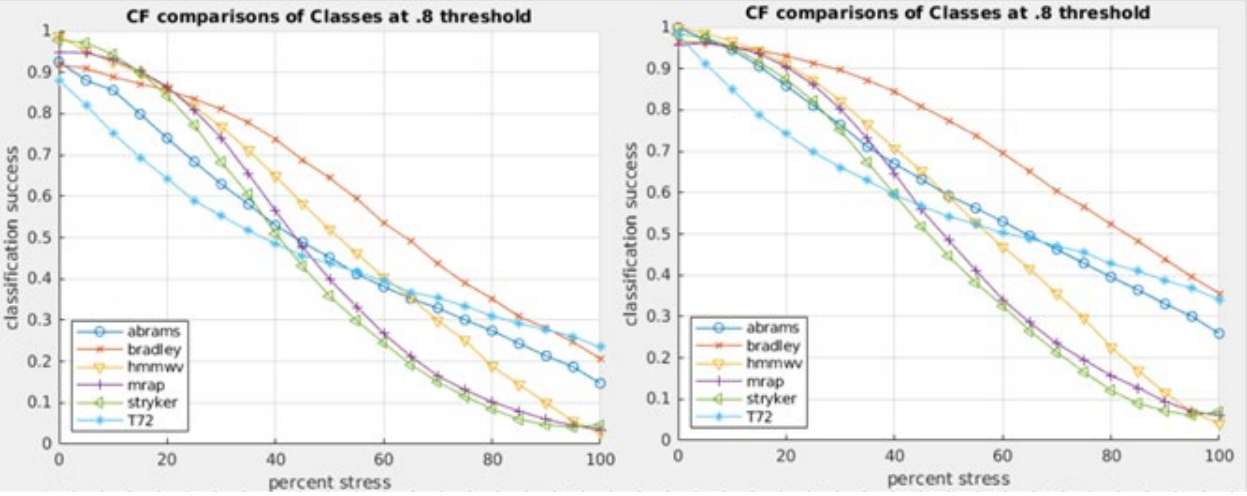

Figure 36. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different classes

To summarize, two metrics have been developed for this study. The first, CF versus incremental test image stress (CF) curves can be used to compare the classification performance when different CNN models, stresses, and classes are considered. CF curves are set to a particular probability threshold so are better suited to analyze a particular system with known system parameters.

Further smoothing of the CF curves can be achieved by visualizing the data in bar graphs of decile area under the curve. This is basically the same information as the CF curves and can be used in similar circumstances.

Another way to present the data and one that is very touted in the literature, is to develop ROC curves, which are created with different values of probability threshold, and therefore removes that parameter from analysis. The ROC curve calculation had to be rewritten to include enough data points to be able to calculate fairly accurate AUCs for comparison. Also, most ROC curves had to be extrapolated to be able to compare different curves since the curves independent variable, the FPR, rarely reaches a value of 1. This may lead to some error in the AUC calculated from these curves, but the error is expected to be acceptable.

The most compact and useful metrics found were to calculate the CF-AUC and the ROC-AUC curves. The ROC-AUC metric has been found in the literature and provides a single figure of merit for the CNN classification success, regardless of system threshold value. The CF-AUC follows the same construction as the ROC-AUC and therefore is just as compact, but yields different information as it considers classification success over threshold instead of using the TPR and FPR ratios.

The ROC-AUC values, calculated with system threshold, are plotted as a function of image stress to allow comparative rating and ranking of CNN classification systems in stressed environments. The CF-AUC values, calculated with percent stress applied to the test images, can be plotted as a function of system threshold to allow comparative rating and ranking of CNN classification systems in stressed environments. The ROC-AUC and CF-AUC curves thus utilize the same sets of independent variables, namely the incremental stress put on the test images and the threshold levels used in the analysis.

The CF-AUC curves are not the same as the ROC-AUC curves and do not always convey the same information. For example, comparing the CF-AUC curves shown in Figure 26 for different CNN models with the corresponding CF-ROC curves shown in Figure 29, the least robust CNN model (the curve with the fastest falloff) predicted from the ROC-AUC curves belongs to the `InceptionResnet` model, while the CF-AUC curves predict the `Inception` model to be the least robust.

The error introduced by closing the ROC curve off (forcing the curve to reach the FPR/TPR coordinates of (1, 1) may contribute to the difference in results of the ROC-AUC and CF-AUC curves. Even though the independent variables for both sets of curves are the system threshold and percentage stress applied to the test images, the values calculated for the curves are fundamentally different.

The CF, CF-AUC, and the ROC-AUC curves have their uses. The CF curves are created with a constant threshold value but are easier to interpret than the ROC-AUC while the ROC-AUC curve metric does not need to have a known threshold value. Because the ROC-AUC does not need a system threshold value, the it is a popular metric in the CNN community, but the ROC and the ROC-AUC curves are less intuitive than the CF curves or bar graphs.

6. METHODOLOGY

Using the developed database, the stressed test images, pretrained CNNs, and the CNN performance metrics components described in Sections 2–5, the methodology used to characterize the different CNN models is explained in this section.

6.1. Introduction

With the description of the parts needed to analyze CNNs described in Sections 2–5, a careful description of the steps needed to perform an analysis is warranted. The MATLAB toolboxes used to train and test CNN seem to be overparameterized in Mathworks' attempt to be inclusive of all analysis possibilities. The relevant input parameters of the MATLAB scripts used (that are included in the Appendices A–E) are discussed as the analysis proceeds. Not all MATLAB parameters were exercised during this project, but there is high confidence that most of the salient parameters were recognized, exercised, and optimized.

6.2. Training Database

As described in Section 2, the image database of military vehicles is assembled from open sources and is made up of six vehicle classes. The images are individual images downloaded from the web, 3D model renderings, and video frames obtained from the web. The individual images downloaded from the web are the most curated and most unique. The 3D model renderings provide the essential form and relative dimensions of the vehicle, with no background or foreground clutter. The model renderings are taken at two polar angles (90° and 60°) and incremental (5°) azimuthal angles. The video clips downloaded from the web offer the most image frames but are the least curated and are the most nonunique as there exists little difference between the images of adjacent frames.

All images in the database are stored in folders that indicate class and origin of the images. A script (`CHOOSE_TRAINING_IMAGES_FROM_WEB_AND_VIDEO`) was written that copies these images to training folders according to user input. The user specifies the number of web images used, whether the model renderings are used, and the total number of images used in the training set.

To set up the training sets, more than the maximum number of web images found for a particular class is normally specified. The script copies whatever web images it finds for each class to the training folders and duplicates random web images until the specified number of web images is met. This ensures that each class has the same amount of web images. Reflection and translation will help the training CNN model not to "memorize" duplicate images. It next copies over the model rendering to the training

folders if specified to do so. Finally, it obtains the specified total image in the training set folders by filling in the training set with random video frame images. If not enough video frame images exist, random duplicates are used to obtain the total training set number specified. Using this script, the training sets for each class will have an equal amount of web images, model renderings, and video frame images, which implies an equal number of total training images.

The result of running this script is a training folder with each class represented as a subfolder. Because of some randomness in which images the script chooses to copy, two training sets created with the same script may not contain the same image and therefore may train the CNN in slightly different manner, but these differences should be negligible for large training sets.

6.3. Training the CNNs Using Transfer-Learning

As described in Section 4, nine pretrained, MATLAB available CNN models were used for this study. The rationale for using existing CNNs is that the iterative effort invested to design, train, and test the CNNs is substantial. Award-winning CNN models that are already optimized exist and are available. With all the research and optimization that has occurred in CNN model design, it is doubtful that designing new CNN models will lead to better classification systems.

The nine CNN models explored were downloaded as MATLAB add-on applications. MATLAB scripts that define the CNN model training parameters, manipulate the training data (resize and place in minibatches for fast delivery), and feed them to the CNN model that is being trained were adapted from MATLAB examples provided online (https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html). The nine scripts, called `train_squeezenet.m`, `train_googlenet.m`, `train_resnet18.m`, `train_mobilenetv2.m`, `train_resnet50.m`, `train_resnet101.m`, `train_inceptionv3.m`, `train_inceptionresnetv3.m`, `train_nasnetlarge.m`, are found in Appendix C.

These scripts all establish a random 70%–30% split of the training images to train the CNN and to validate the training, respectively. All scripts also find the size of the training image (in pixels) that the particular CNN model needs. The training images are resized to fit the CNN's input needs.

Next, the last few layers of the CNN need to be changed to adapt to transfer-learning. This is done by removing the last layer of the CNN and specifying its replacement. Finally, the first replaced layer needs to be reconnected to the last model layer. The

method to adapt pretrained networks to classify specific classes through transfer-learning is referenced in MATLAB online pages (https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html).

As mentioned previously, random translation and horizontal reflection of the training image is specified to advert “target memorization” by the CNN. The maximum random translation is left at the default 30 pixels and because the classes considered are all land vehicles, only the random horizontal image reflection (`RandXreflection`) command is activated. This introduced randomness would seem to help the CNN learn from accidental and intentional duplication of training images.

Next, the images are resized to the pixel size required by the CNN under test. In this step, the `gray2rgb` command is run to convert any possible gray scale images downloaded from the web to the RGB structure that is needed by the CNN. The `gray2rgb` command does not alter RGB images. With the steps described previously, the training data are inserted into the MATLAB `datastore` structure for fast delivery and are ready to be used.

The training options, which are numerous, are now specified. The germane options used are found in Table 2. Much iteration through these variables were made and optimized as their effect became understood. The rest were left at their default values.

Table 2. Training Variables and Values

Variable	Value	Explanation
`Training method`	sgdm	Stochastic gradient descent with momentum
`MiniBatchSize`	128	Size of image set for each training iteration
`MaxEpochs`	10, 15	Number of training cycles
`InitialLearnRate`	1e-3	Initial learning rate
`Shuffle`	Every Epoch	When to shuffle training images
`ValidationFrequency`	50	Number of iterations before validation
`DispatchInBackground`	true	Asynchronous prefetch queuing
`LearnRateSchedule`	piecewise	Gradually reduce learning rate
`LearnrateDropFactor`	0.9	Factor for dropping the learning rate
`LearnRateDropPeriod`	2	Number of epochs for dropping the learning rate
`Plots`	training-progress	Display training progress
`ExecutionEnvironment`	multi-gpu	Use multiple GPUs

All the networks were then trained with the same data and input parameters. The CNN model synaptic weights and biases determined by this training are then stored for later use. Depending on the number of epochs, minibatch size, and number of hidden layers and complexity of the CNN model specified, each CNN model took approximately 0.5 to 6 h to fine-tune.

The specialized CNN weights and biases are saved to a hard drive and reloaded before performing CNN testing. This allows the fine-tuned CNN models to be used for different analyses and comparisons at later dates. The specialized weights and bias values obtained from the transfer-learning process can also be exported to other workstations for testing, so that hardware and software configuration specialization can be accomplished between CNN model training and CNN model testing.

6.4. Testing the CNN Model Performance with Battlefield Stressors

During transfer-learning and training of the CNN models, the progress of transfer-learning and training is checked periodically against 30% of the training images saved for validation. This internal training validation does not explicitly and quantitatively test the performance of the CNN in stressed environment even though many of the training images are of vehicles in action that may contain stress.

Independent from the training/validation training sets, 20 images per class were chosen as test images. These test images contain targets that include various amounts of background and foreground clutter, but most were chosen because the targets were well isolated and the class target of interest was easily recognizable within the image. These 120 images were stressed by 11 synthesized stress types in increments of 5%.

For a particular applied stress and level of stress, the stressed test images of all six classes are grouped together, and the nine pretrained and fine-tuned CNN models are made to classify the 120 images into their six classes. True positive, false positive, true negative, and false negative of each classification for a particular threshold level are recorded. Control of which stresses, classes to use, and CNN models to analyze is found in the master script `COMPLETE_TRAIN_AND_TEST_SUITE.m` provided in Appendix C.

The MATLAB script `DATA_REDUCTION.m`, found in Appendix C, is run by `COMPLETE_TRAIN_AND_TEST_SUITE.m` and gathers all the testing results and creates a multidimension matrix of all the test results. The visualization calculations are taken from slices and averages in different dimensions of this matrix.

The data can be compared and visualized using the CF/CF-AUC or the ROC/ROC-AUC curves. The MATLAB script `TOTAL_analysis.m`, loads the result matrices, requests what parameters are to be compared (and averages the other parameters), and calculates CF curves, decile bar graphs, ROC curves, CF-AUC, and ROC-AUC curves.

7. EXAMPLE OF A CNN MODEL ANALYSIS

The performance of pretrained CNN models fine-tuned on two different-sized training sets is compared. The network models are fine-tuned to classify six classes of military vehicles through transfer-learning, and the performance of the network models is tested on incrementally increasing stressed images. A comparison of particular parameters of interest is achieved by averaging the remaining parameters. The different steps of the analysis are recapped to demonstrate the flow the analysis, and the comparative results are presented fully to show the visualization possibilities in detail.

7.1. Introduction

At this stage, all the major components used in the analysis of CNN classification systems have been described in some detail. It would be informative to run through the analysis process of some pretrained CNN models, fine-tuned on two different-sized training image sets, from beginning to end to show a potential user the methodology flow employed in the process. Since the pretrained CNN models used in this study are all award-winning models, large performance differences between the CNN models are not expected. The demonstrative analysis is instead performed on the size of the training sets used in the fine-tuning (transfer-learning). Two fine-tunings are performed on the nine pretrained CNN models, and the metrics derived from the ROC curves and the CF curves are compared for the two cases.

In this study, the performance of the CNN models defined by the Pareto boundary (see Figure 14), is analyzed in a stressed environment. The test images used to evaluate the CNN models have multiple categories, including multiple trials for the stresses that have a random component to them, class type, and stress type. These 9 CNN models, 25 trials, 6 classes, and 11 stresses, establish a 4D black box analysis problem, with the incremental percentage of stress applied to the test image and the system threshold levels treated as the independent variables.

As discussed previously, the dimensions other than the parameter of interest are averaged. It is realized that some granularity of the analysis is lost by averaging the unconsidered dimensions, but the problem is overparameterized, and the results due to parameters outside the range of interest must be averaged so that useful results can be visualized.

To demonstrate capability, the four developed visualization metrics are used to compare the performance of the CNNs fine-tuned with different-sized transfer-learning image sets. Comparison plots of CNN performance under stress are presented for CNN type, stress type, and class.

7.2. Choosing Training Sets

The first step in the analysis was to specify the size and composition of the training sets to be used to fine-tune the CNN models under consideration. To contrast CNN model behavior due to training image set size difference, two largely different-sized training sets were used.

Because transfer-learning was used to fine-tune the downloaded pretrained networks, only 830 images per class were used in the first training set, composed of 300 web images per class, all (142) 3D model images per class, and 388 random video-frame images per class. The total number of images used in the first set is thus 4980.

The second training set was made much larger, this time having a total of 7500 images per class, consisting of 500 web images, all 142 model images, and 6858 random video-frame images per class. The second training image set thus had a total of 45000 images.

The `CHOOSE_TRAINING_IMAGES_FROM_WEB_AND_VIDEO.m` script is used to delete old training sets and generate the wanted training set, by specifying the amount of web images versus video-frame images, and whether the model images are used.

7.3. Fine-Tuning (Training) the CNN Models

The nine pretrained CNNs were fine-tuned with transfer-learning using the two training sets. The training sets were separated into two groups: the default 70% of the images were used for training and the remainder 30% used for training validation. As described previously, the training images were randomly horizontally flipped, and randomly translated as much as 30 pixels during training, to prevent “training memorization” by the CNNs. The relevant parameters used in training are presented in Table 2.

As the CNN training script is run, the accuracy of each training/validation set is displayed (as well as the training loss, which in the case of this study, is the cross-entropy loss). An example of training progress is shown in Figure 2. Training of each CNN model considered takes between 0.5 and 6 h depending on the size of the training sets and the complexity of the CNN model.

For this example, the nine CNNs were fine-tuned from their original pretrained status twice, the first with the 4980-image data set and then again with the 45,000-image data set. The differently fine-tuned CNN model weights and biases were written to file separately for the two training sets.

7.4. Testing the CNN as a Function of Test Image Stress

As described earlier, 20 test images per class were chosen from the images downloaded from the web to test the CNN model robustness to synthesized battlefield stresses. These 120 test images were incrementally altered with 11 different simulated stresses in 5% increments, yielding 27,720 test images.

For a certain stress type and increment of stress, the MATLAB test script outputs the TPR and FPR values as a function of set threshold. The script also outputs the number of correctly classified test images (decided by maximum probability) per class, to file. As mentioned previously, a custom script was written that evaluates TPR and FPR for various threshold levels instead of using the internal MATLAB function.

The differently trained CNNs were tested with these test images, and the TPR, FPR, and classification successes of each CNN were written to file separately for the two training sets.

7.5. Visualization of the CF, ROC-AUC, and CF-AUC Curves

The CNN model testing results can now be plotted and compared to allow their analysis depending on the input parameters considered. The `TOTAL_FAILURE.m` script requests what input parameter (CNNs, stresses, classes) are to be compared. After averaging the parameters that are not under consideration, the CF, ROC-AUC, and CF-AUC curves are calculated and visualized. First, the CF curves are compared for the CNNs trained, the stresses applied to the test images, and the different classes classified by the CNNs. The system threshold value was set at 0.8. These comparisons are presented in Figures 37–39, respectively.

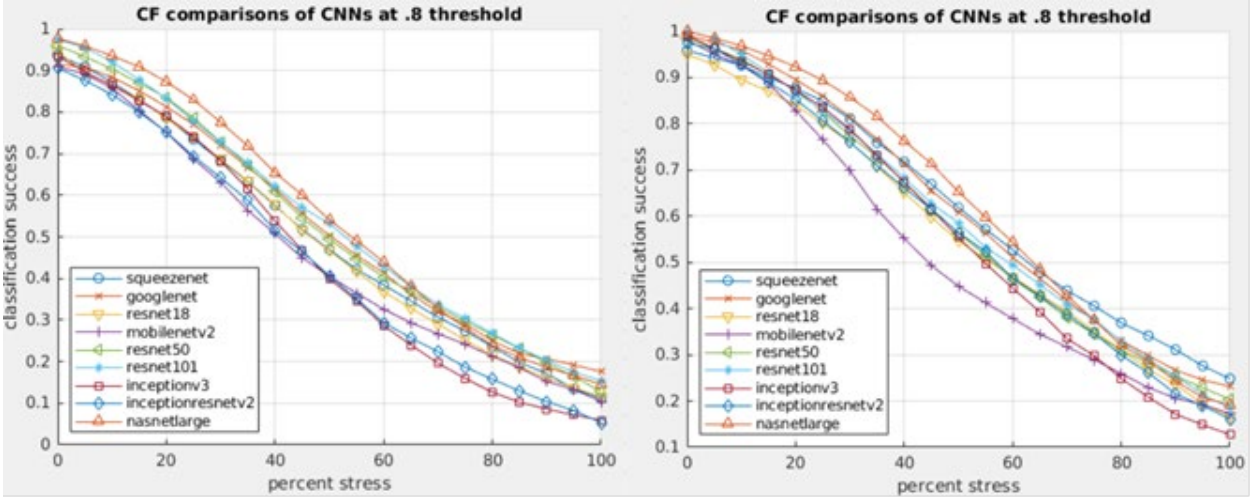

Figure 37. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different CNNs

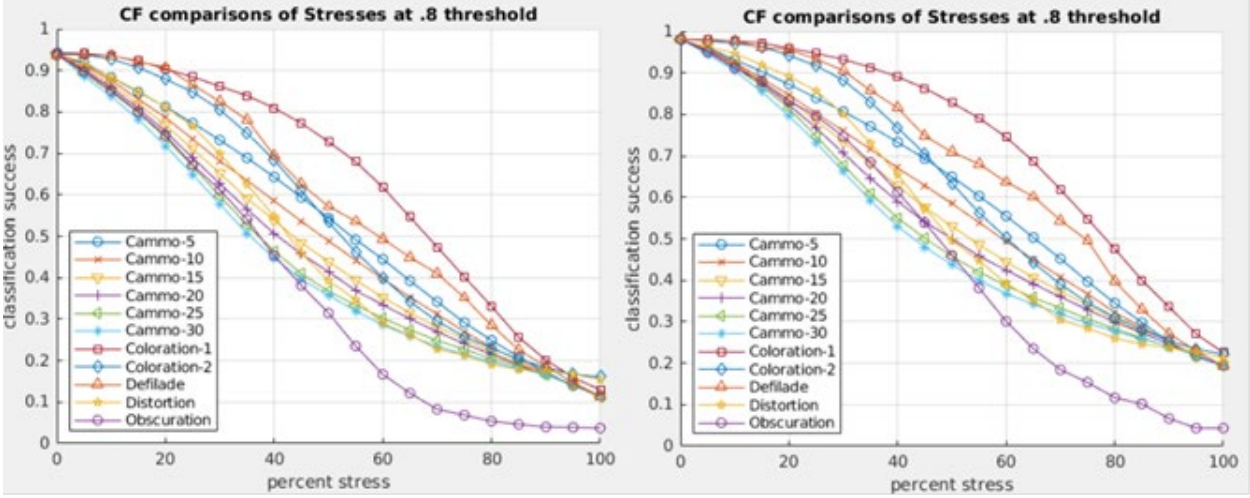

Figure 38. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different stresses

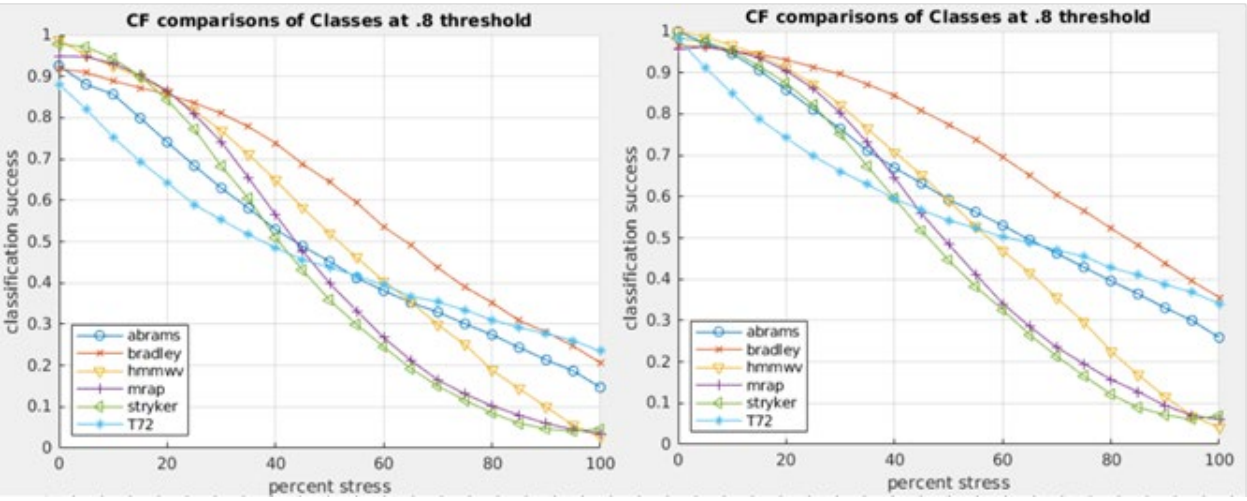

Figure 39 Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF curves for different classes

From Figures 37–39, the resilience of the CNNs increases as they are trained with more images. This increase in resilience is not uniform between consideration of CNN, stresses, and classes, nor for all in the family of curves plotted in each plot.

Next, the three parameters are compared with the CF-AUC curves, which as a reminder, are the areas under the CF curves. The CF curves are a function of test image stress and successive AUC are plotted as a function of system threshold. Figures 40–42 use CF-AUC curves to compare the parameters of interest between the two fine-tuned CNN model sets.

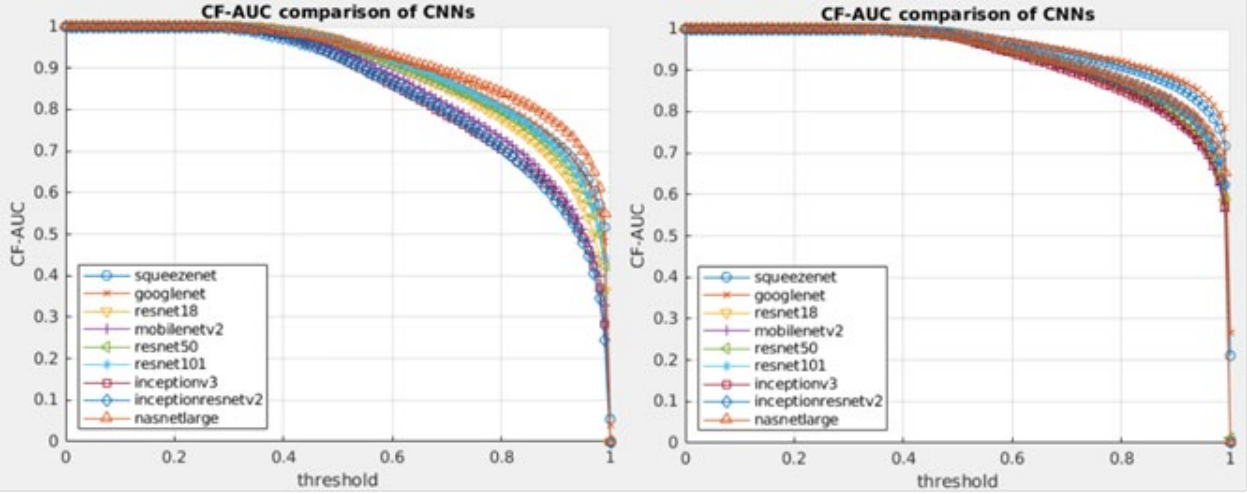

Figure 40. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF-AUC curves for different CNN models

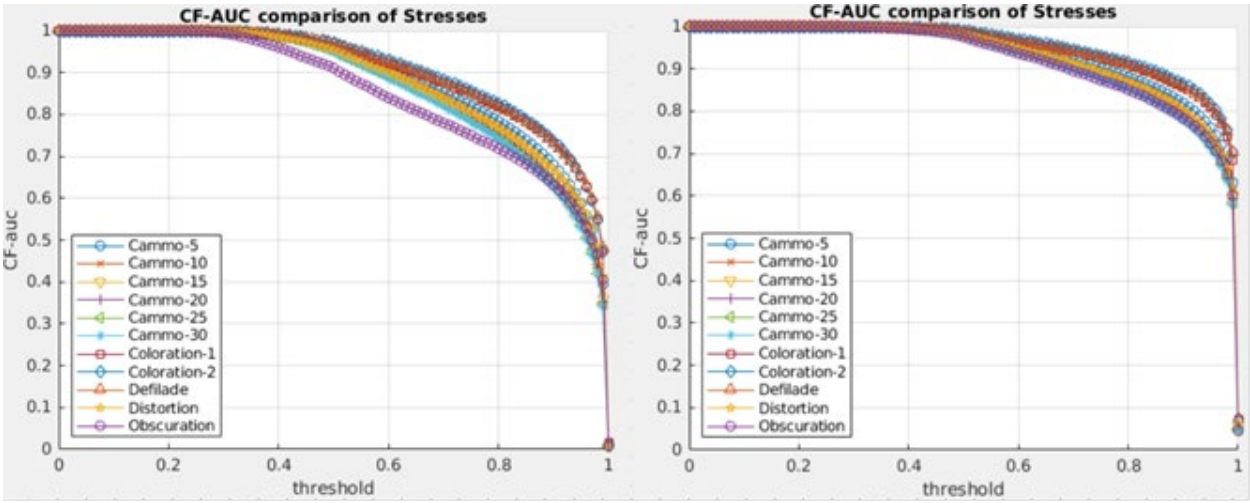

Figure 41. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF-AUC curves for different stresses

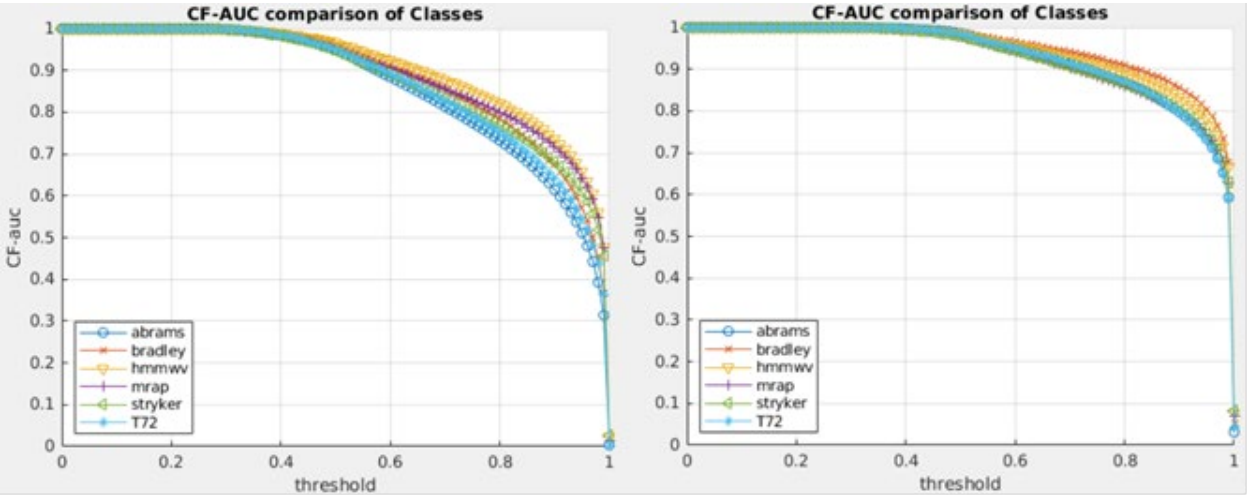

Figure 42. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using CF-AUC curves for different classes

Using the CF-AUC curves to compare performance as a function of fine-tuning shows a substantial difference in classification robustness when a range of thresholds are considered. In general, the outlying curves in the CF curves remain the same outliers in the CF-AUC curves. For example, the high and low curves of Figure 37 are the same high and low curves of Figure 40.

Finally, the ROC-AUC curves are used to compare the parameters considered. Since the ROC curve use thresholds as their independent variable, the ROC-AUC curves are plotted over percent stress added to the test images. Figures 43–45 show the ROC-AUC curve comparisons of the three parameters considered.

Comparison of the ROC-AUC curves for CNN models that have been fine-tuned with small training sets versus those fine-tuned with larger training sets show that the larger training sets make the CNN models more robust when comparing different CNN models and different stresses but remain approximately the same when comparing different classes.

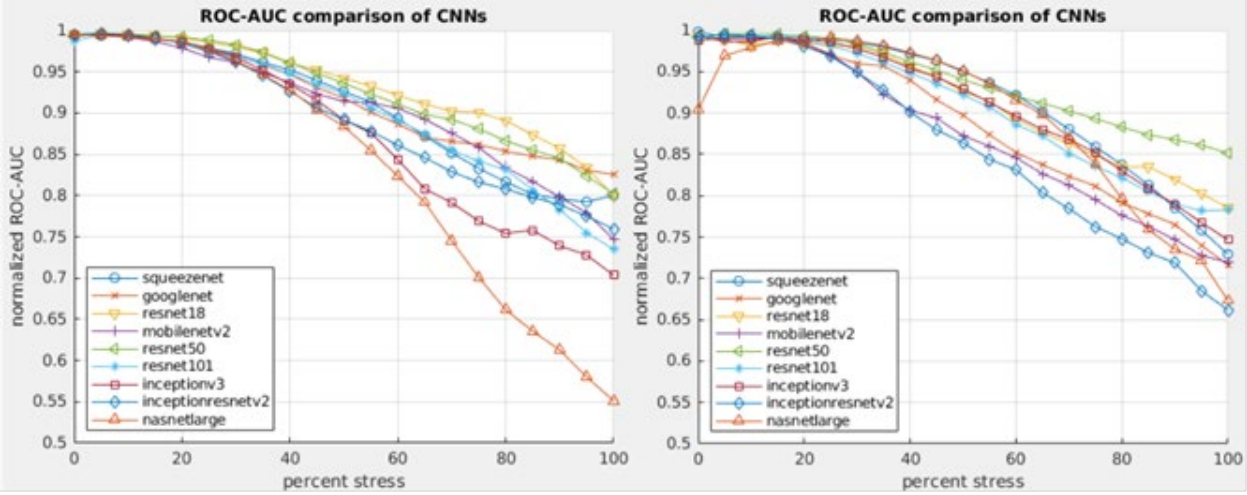

Figure 43. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using ROC-AUC curves for different CNN models

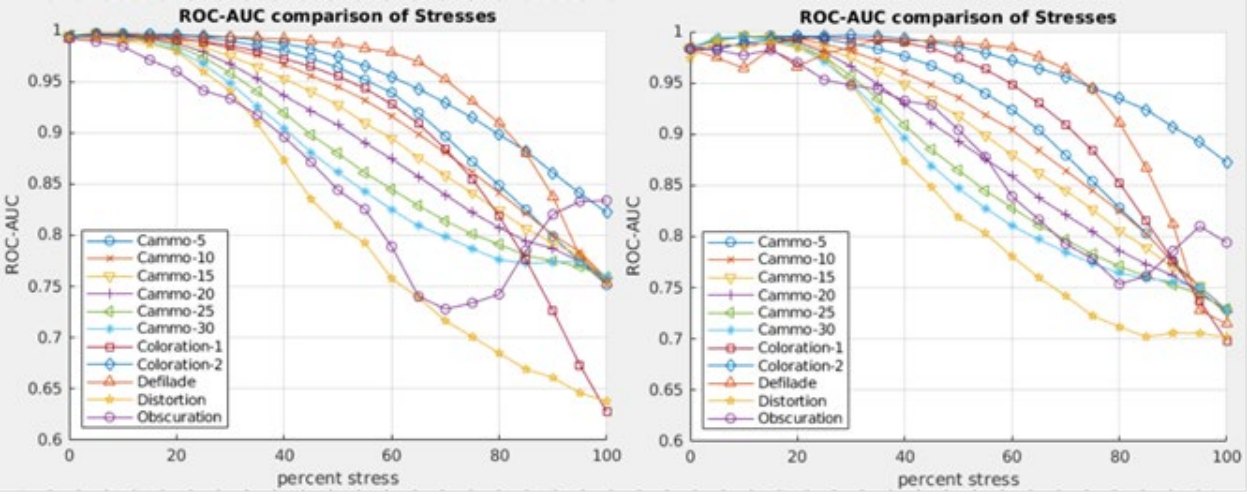

Figure 44. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using ROC-AUC curves for different stresses

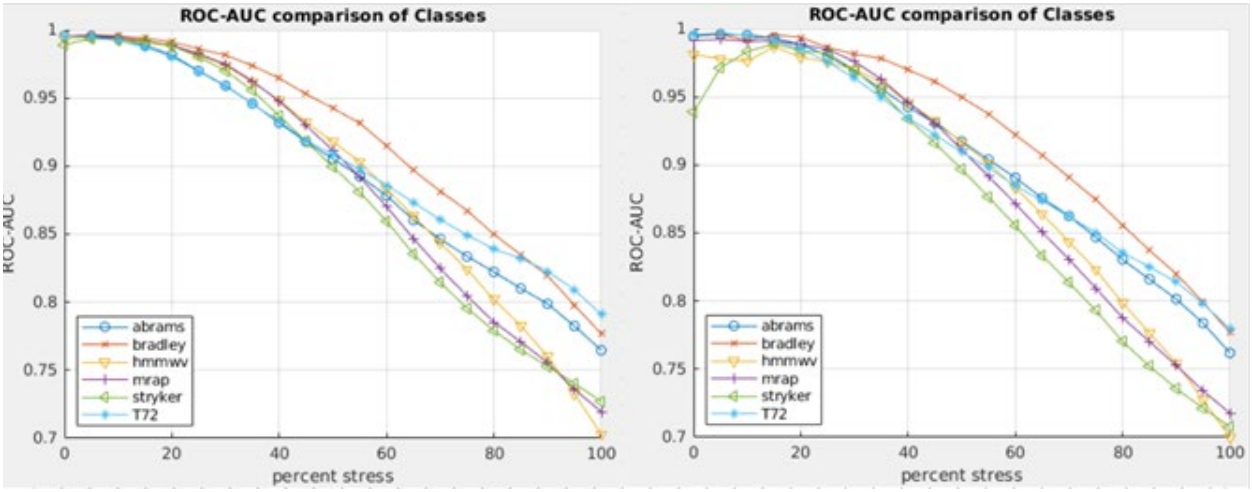

Figure 45. Comparison of classification performance fine-tuned with 830 (left) and 7500 (right) images per class using ROC-AUC curves for different classes

8. CONCLUSIONS

Much investment and advancement has been made in image classification systems due to the push to develop autonomous vehicles. The heart of this effort has been in the development of neural network image classification (recognition) systems whose present state is found in convolutional (sliding averages of pixel values) and deep (containing many intermediate, or hidden, layers) neural networks.

All neural network classification systems need to be trained with data and data labels that indicate the desired classification. Much advancement has been made in neural network training methods, such as rapid image feeding algorithms, automated image offsets and reflections that mitigate “image memorization,” and interim training validation that incrementally evaluates the training progress.

With all this classification progress, some of it not well understood by the developing researchers, it was found that the classification systems were sometimes substantially vulnerable to small alterations to the test images. This early discovery indicated that a potential problem existed if the classification systems were not trained on exactly the same data as it is classifying, and of course the image data input the classification system is rarely exactly the same as the training data.

To investigate the effect of the difference in training and classification data, especially on the battlefield, a method has been developed to synthetically alter digital images to mimic different types of battlefield stress so that the effect of these stresses on image classification system can be quantified. The five battlefield stresses synthesized over the test images are camouflage patches over the target, target disruptive coloration, target distortion, target hull defilade, and image obscuration.

Next, an extensive development of classification system failure under test image stress evaluation was undertaken. Four metrics were developed, namely CF curves as a function of stress, CF-AUC as a function of system threshold level, ROC as a function of system threshold, and the ROC-AUC as a function of incremental stress.

Even though the CF-AUC and ROC-AUC curves encompass the same independent variable ranges (system threshold, image stress levels applied), they do not offer the same information because the CF-AUC is calculated against stress increment and plotted against system threshold while the ROC-AUC is calculated against system threshold and plotted against stress increment. Moreover, the ROC and ROC-AUC curves use nonbinary information; they differentiate between true and false positives, and true and false negatives, and thus carry more information.

When the different metrics are compared against each other, the CF and CF-AUC curves show the same gross features (i.e., which curve is least and most affected by the independent variable). This can be seen when the same parameter comparisons are compared in Figures 37–45. The same cannot be said of the ROC-AUC curves, which do not track the gross features from the CF and the CF-AUC curves. This may be due to the use of different plotting parameters (TPR vs. FPR instead of CF) or is perhaps due to the approximation made when extrapolating the ROC curve to its TPR = 1, and FPR = 1 coordinates (See the Section 4.2 discussion associated with Figures 32 and 33).

Because ROC curves are not compact enough (they are a function of system threshold and are not explicitly dependent on test image stress levels), and because of the unknown effects of the extrapolations used to calculate the ROC-AUC curves, it is thought that the CF-AUC curves, which are a function of both test image stress levels and system threshold levels, are the best metric to use for the ranking and rating the robustness of CNN classification systems exposed to battlefield stresses.

Because the pretrained CNNs used defined the Pareto frontier (see Figure 14), and all are competition winners, not much classification success difference is expected between them. This would not necessarily be the case when rating and ranking CNNs models designed and pretrained especially for Army systems, and therefore the attention would predominantly be on the comparison of CNN performance.

To go through an analysis example, different-sized image data sets were used to fine-tune the pretrained CNN used, and their robustness to battlefield stresses were compared with the different metrics. In general, the broad features were found in both cases: 1) the Nasnetlarge CNN model was more robust to stresses than the rest of the CNN considered; 2) vertical coloration was the least effective, and image obscuration was the most effective in causing CF due to test image stress; and 3) the Bradley Fighting Vehicle was the most robust class of military vehicle (the Stryker being the least robust). Also apparent in the metric result comparisons is that the CNNs fined-tuned with the larger data set of 7500 stayed more robust with increasing stress levels.

Again, the ROC-AUC curve metric did not display these trends as clearly as the CF and CF-AUC curves, and moreover, the magnitude of failure of the two fined-tuned sets, as seen in the ROC-AUC curves, were approximately the same. In contrast, the CNN models fined-tuned with the larger data set were more robust when compared with the CF and CF-AUC metrics. This again shows that the ROC-AUC curve measures different parameters than the CNN model failure versus stress curves by comparing the TPR to the FPR ratios.

It is emphasized that the comparison between stressors is not very meaningful because the incremental stress added to the test images are sometimes hard to calibrate. It is, for example, difficult to quantify how disruptive coloration incremental opacity relates to target percentage covered with camouflage patches. When considering this, the more meaningful information may be found in how the classification success curves drop with increased test image stress.

Two quantities were calculated, the CF and the ROC curve. The first is simply the CF of the test images divided by the number of images tested in each class as a function of stress type and stress level. This value is binary (it either identified the target correctly or it did not) and is useful to determine fall-off of CF as a function of stress included in the test images. Because MATLAB chooses the class with the highest probability for its classification prediction, set threshold levels may, but do not have to be included in this calculation.

Without the use of set threshold levels, this metric is not very useful in the analysis of AI systems that do operate with threshold levels. The superposition of a specific threshold on the classification success data would change the shape of the failure curves and importantly, would negatively narrow the analysis of the various CNNs considered to a particular threshold value.

The second metric developed is the ROC curve. The data points that are used to create this curve depend on different thresholds of detection, and therefore remove the system threshold variable from the analysis.

The AUC can be taken from both the success curve and ROC curve metrics for a single figure of merit for the CNN models and their training. Because the independent variable in the ROC curve is the FPR, the ROC-AUC figures of merit can then be plotted as a function of stress levels to yield ROC as a function of applied stress.

The CF curves are already dependent on stress levels and therefore their AUC cannot be plotted as a function of image stress. The CF-AUC can, however, be plotted against threshold level to reveal robustness of the CNN models as a function of threshold. The created ROC-AUC curve as a function of stress levels and the CF-AUC curve as a function of threshold level are both representation of CNN model behavior under stress but display different information.

The development of this analysis methodology shows how to quantify the failure of the CNN’s ability to classify 2D images. Even though award-winning CNN models were used to develop this methodology, it was developed to analyze future CNN model classifiers of unknown provenance developed specifically for Army systems. Since so

much of these emerging systems will depend on the correct classification of sensor images in real time, the ability to rank and rate CNN models is expected to become an important part of the analysis of emerging Army AI system.

Appendix A – MATLAB Scripts for Database Management

The following script deletes the train directory that contains all training images labeled by their directory names and repopulates the directory with the prescribed number of web image, model rendering images, and video frame images. The class labels are the subdirectory names.

CHOOSE_TRAINING_IMAGES_FROM_WEB_AND_VIDEO.m

```
% choose the amount of web images to be put into training set
% choose the total amount of images to be put into training set (the rest
% will be filled in with video frames until amount specified by total). If
% not enough web or video files exist, random images in the category will
% be used.
% class = ["abrams","bradley","commercial","hmmwv","mrap","stryker","t72"];
class = ["abrams","bradley","hmmwv","mrap","stryker","t72"];
%%% user inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% approx 300 web and model images exist per class
total_web = 300;       % total web images needed
use_model_data = 1;        % use_model_data = 1 uses model renderings
total = 830;         % total number of images used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% remove existing training sets and re-create folders %%%%%%%%%%%%%%%%
   if exist(strcat('../TRAIN_MILITARY_VEHICLES/'),'dir')
      rmdir(strcat('../TRAIN_MILITARY_VEHICLES/'),'s');
   end
for ii = 1:length(class)
   mkdir('../TRAIN_MILITARY_VEHICLES/',class(ii));  % clean folders
   disp(class(ii))
    %%% 1) add model renderings if use_model_data = 1 %%%%%%%%%%%%%%%%%%
   model_image_count = 0;
   if use_model_data == 1
      model_images = dir(strcat(pwd,'/TRAIN_IMAGES/',class(ii),'_model/*.jpg'));
      model_image_count = length(model_images);
      for tt = 1:model_image_count
         model_source = strcat(pwd,'/TRAIN_IMAGES/',class(ii),'_model/',char(model_images(tt).name));
         model_dest = strcat('../TRAIN_MILITARY_VEHICLES/',class(ii),'/m_',char(model_images(tt).name));
         copyfile(model_source,model_dest)
      end
   end
   %%% 2) copy web images up to specified number = total_web %%%%%%%%%%%%%
   web_images = dir(strcat(pwd,'/TRAIN_IMAGES/',class(ii),'_web/*.jpg'));
   web_image_count = length(web_images);
   web_random_index = 1:web_image_count;
   while length(web_random_index) < total_web
      web_random_index = [web_random_index  randperm(web_image_count)];
   end
   tt = 1;
   while tt  <= total_web
      web_source = strcat('./TRAIN_IMAGES/',class(ii),'_web/',...
         web_images(web_random_index(tt)).name);
      web_dest = strcat('../TRAIN_MILITARY_VEHICLES/',class(ii),...
         '/w_',num2str(tt),'.jpg');
```

```
45          copyfile(web_source,web_dest);
46          tt = tt + 1;
47       end
48       %%% 3) fill-in with random video images until total %%%%%%%%%%%%%%%%%%%
49       total_video = total - total_web - model_image_count;
50       video_images = dir(strcat(pwd,'/TRAIN_IMAGES/',class(ii),'_video/*.jpg'));
51       video_image_count = length(video_images);
52       video_random_index = randperm(video_image_count);
53       while length(video_random_index) < total_video
54          video_random_index = [video_random_index randperm(video_image_count)];
55       end
56       tt = 1;
57       while tt <= total_video
58          video_source = strcat('./TRAIN_IMAGES/',class(ii),...
59             '_video/',video_images(video_random_index(tt)).name);
60          video_dest = strcat('../TRAIN_MILITARY_VEHICLES/',class(ii),...
61             '/v_',num2str(tt),'.jpg');
62          copyfile(video_source,video_dest);
63          tt = tt + 1;
64       end
65    end
```

Appendix B – MATLAB Scripts That Control Incremental Stress Synthesis

Twenty images per class were chosen from the training image set to test the resulting fine-tuned convolutional neural networks (CNNs). These images were chosen as good representations of the target class they represent. Clutter was not minimized, but the targets were isolated and not overly obscured. It is important to move, not copy, these images from the training set so that the CNNs cannot train on these particular images.

Target masks are created for each image using the MATLAB script DEFINE_MASK.m. The mask information is stored and used when applying stress to the images. It may be necessary to choose a test image whose target can be masked without including too much of the background so that stress can be applied to the target only.

Once an adequate target mask is defined, the CREATE_TEST_SUITE.m script is run to generate test images with incremental stresses applied to them.

DEFINE_MASKS.m

```
close all; clc; clear;
% script to isolate target in an image by forming k-mean pixel assemblages.
% user is prompted to chose masks that define target, those that define
% background colors, and those that are not to be used for stressing. more
% than one mask can be used for each category.
% mask information will be stored in a subfolder for later reference.
%%% user input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
class = 'mrap';
image_name = 'new_test_20';    % input image name
enhancement =  0;            % 1 = hsv for greater contrast:
resize = 512;               % specify edge size, if 0, no resize
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% read in original image
image = imread(['./',class,'/',image_name,'.jpg']);
% resize close to specified while holding aspect ratio
[a,b,~]=size(image);
c = ceil(resize^2/(a*b));
if resize ~= 0; image = imresize(image,c); end
% break out object from background into 5 color groups
[object_mask, ground_mask, sky_mask] = find_mask(image, enhancement);
object_image = image .* uint8(object_mask);
object_image(object_image==0) = 255;
imshow(object_image,[]);title('CHOSEN TARGET');
% store enhancement, resize and mask information
if ~exist(['./',class,'/','mask_info'],'dir')
   mkdir(['./',class,'/','mask_info']);
end
saved_file = strcat('./',class,'/mask_info/',image_name);
save(saved_file,'enhancement','resize','object_mask','ground_mask','sky_mask');
 end
```

find.mask.m

```
function [object_mask, ground_mask, sky_mask] = find_mask(image, enhancement)
% function to interactively define target, ground and sky in image
figure('units','normalize','outerposition',[.1 ,.1 .9 .9]);
subplot(1,2,1); imshow(image); title('ORIGINAL IMAGE');
if enhancement ==1
Z = rgb2hsv(image);
else
Z = image;
end
ab = Z(:,:,2:3);
ab = im2single(ab);
ncolors = 5;
pixel_labels = imsegkmeans(ab,ncolors,'NumAttempts',12);
pixel_labels = imfill(pixel_labels);    % fill in regions
subplot(1,2,2); imshow(pixel_labels,[]);title('5 PIXEL GROUPS');
object_group = 1;
object_mask = zeros(size(image));
while object_group ~= 0
   prompt = 'What are target shades? (black=1, dk.gray=2, gray=3, lt.gray=4, white=5, end=0)';
   object_group = input(prompt);
   if object_group ~= 0
      object_mask_t = pixel_labels == object_group;
      object_mask = object_mask + object_mask_t;
   end
end
ground_group = 1;
ground_mask = zeros(size(image));
while ground_group ~= 0
   prompt = 'What are ground shades? (black=1, dk.gray=2, gray=3, lt.gray=4, white=5, end=0)';
   ground_group = input(prompt);
   if ground_group ~= 0
      ground_mask_t = pixel_labels == ground_group;
      ground_mask = ground_mask + ground_mask_t;
   end
end
sky_group = 1;
sky_mask = zeros(size(image));
while sky_group ~= 0
   prompt = 'What are sky shades? (black=1, dk.gray=2, gray=3, lt.gray=4, white=5, end=0)';
   sky_group = input(prompt);
   if sky_group ~= 0
      sky_mask_t = pixel_labels == sky_group;
      sky_mask = sky_mask + sky_mask_t;
   end
```

CREATE_TEST_STRESS_SUITE.m

```
close all; clc; clearvars;
% script to take the test images and their mask information and develop
% incremented stressed images
class_type = ["abrams","bradley","hmmwv","mrap","stryker","t72"];
for aa = 1:length(class_type)   % loop through classes
   imdir = dir(strcat(pwd,'/TEST_IMAGES/',class_type(aa),'/*.jpg'));
   for bb = 1:length(imdir)         % loop through test images
      image = imread(strcat(pwd,'/TEST_IMAGES/',char(class_type(aa)),'/',imdir(bb).name));
      [~,image_name,~] = fileparts(imdir(bb).name); % strip extension
      mask_info = strcat(pwd,'/TEST_IMAGES/',char(class_type(aa)),'/mask_info/',image_name);
      load(mask_info);      % load mask information
      % resize to size that masks were defined on
      [a,b,~] = size(image);
      c = ceil(resize^2/(a*b));
      if resize ~= 0; image = imresize(image,c); end
      parfor cc = 1:21        % loop through percent coverage
         per = (cc-1)*5;
         class_type_par = class_type;
         for dd = 1:25              % loop through trials (camouflage,coloration, distortion only)
            for ee = 1:6                % loop through number of patches (camouflage only)
               patches = ee * 5;
               disp(strcat(num2str(aa),":",num2str(bb),":",num2str(dd),":",num2str(ee)));
               %%% camouflage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               cammoed_image = camouflage(image,object_mask,ground_mask,patches,per);
               % write out inside patches loop
               result_dir = strcat(pwd,'/../TEST_MILITARY_VEHICLES/cammo_data/', ...
                  num2str(patches),'/',num2str(dd),'/',num2str((cc-1)*5),'/',class_type_par(aa));
               if ~exist(result_dir,'dir')
                  mkdir(result_dir);
               end
               saved_image = strcat(result_dir,'/',image_name,'.jpg');
               imwrite(cammoed_image,saved_image);
            end    % end of patch loop
            %%% disruptive coloration 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % user input
            stripe_orientation = 1;    % vertical stipes = 1
            stripe_width = 10;         % stripe_width
            colored_image1 = coloration(image,object_mask,cc,stripe_orientation,stripe_width);
            %%% disruptive coloration 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % user input
            stripe_orientation = 0;    % vertical stipes = 1
            stripe_width = 10;         % stripe_width
            colored_image2 = coloration(image,object_mask,cc,stripe_orientation,stripe_width);
            %%% distortion %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            distorted_image = distortion(image,object_mask,cc);
            disp([num2str(aa),':',num2str(bb),':',num2str(cc),':',num2str(dd)]);
            % write out to disk inside trial loop
            result_dir = strcat(pwd,'/../TEST_MILITARY_VEHICLES/coloration1_data/',...
               num2str(dd),'/',num2str((cc-1)*5),'/',class_type_par(aa));
            if ~exist(result_dir,'dir')
               mkdir(result_dir);
```

```
            end
            saved_image = strcat(result_dir,'/',image_name,'.jpg');
            imwrite(colored_image1,saved_image);
            % write out to disk inside trial loop
            result_dir = strcat(pwd,'/../TEST_MILITARY_VEHICLES/coloration2_data/',...
               num2str(dd),'/',num2str((cc-1)*5),'/',class_type_par(aa));
            if ~exist(result_dir,'dir')
               mkdir(result_dir);
            end
            saved_image = strcat(result_dir,'/',image_name,'.jpg');
            imwrite(colored_image2,saved_image);
            result_dir = strcat(pwd,'/../TEST_MILITARY_VEHICLES/distortion_data/', ...
               num2str(patches),'/',num2str(dd),'/',num2str((cc-1)*5),'/',class_type_par(aa));
            if ~exist(result_dir,'dir')
               mkdir(result_dir);
            end
            saved_image = strcat(result_dir,'/',image_name,'.jpg');
            imwrite(distorted_image,saved_image);
         end    % end of trials loop
         %%% defilade %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         defilade_image = defilade(image,object_mask,ground_mask,per);
         %%% jam %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         jammed_image = jam(image,per);
         %%% outline %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         outline_image = outline(image,object_mask);
         % wtite out to disk inside percent stress loop
         if ~exist(strcat(pwd,'/../TEST_MILITARY_VEHICLES/defilade_data/', ...
               num2str((cc-1)*5),'/',char(class_type_par(aa))),'dir')
            mkdir(strcat(pwd,'/../TEST_MILITARY_VEHICLES/defilade_data/', ...
               num2str((cc-1)*5),'/',char(class_type_par(aa))));
         end
         saved_image = strcat(pwd,'/../TEST_MILITARY_VEHICLES/defilade_data/', ...
            num2str((cc-1)*5),'/',char(class_type_par(aa)),'/', ...
            image_name,'.jpg');
         imwrite(defilade_image,saved_image);
         if ~exist(strcat(pwd,'/../TEST_MILITARY_VEHICLES/jam_data/', ...
               num2str((cc-1)*5),'/',char(class_type_par(aa))),'dir')
            mkdir(strcat(pwd,'/../TEST_MILITARY_VEHICLES/jam_data/', ...
               num2str((cc-1)*5),'/',char(class_type_par(aa))));
         end
         saved_image = strcat(pwd,'/../TEST_MILITARY_VEHICLES/jam_data/', ...
            num2str((cc-1)*5),'/',char(class_type_par(aa)),'/', ...
            image_name,'.jpg');
         imwrite(jammed_image,saved_image);
         if ~exist(strcat(pwd,'/../TEST_MILITARY_VEHICLES/outline_data/',char(class_type_par(aa))),'dir')
            mkdir(strcat(pwd,'/../TEST_MILITARY_VEHICLES/outline_data/',char(class_type_par(aa))));
         end
         saved_image = strcat(pwd,'/../TEST_MILITARY_VEHICLES/outline_data/',char(class_type_par(aa)),'/', ...
            image_name,'.jpg');
         imwrite(outline_image,saved_image);
      end
   end
end
```

camouflage.m

```
function[cammoed_image] = camouflage(image,object_mask,ground_mask,patches,per)
% choose random seeds on object for cammo patches
[r,c,~] = size(object_mask);
idr = randperm(r);     % randomize row index
idc = randperm(c);     % randomize column index
% find random patch seeds on object
count = 1;             % initialize
x = 1;                 % initialize
rseed = zeros(1,patches);   % initialize
cseed = zeros(1,patches);   % initialize
while count <= patches
   if object_mask(idr(x),idc(x)) == 1     % is seed on object?
      rseed(count) = idr(x);              % save seed coordinate
      cseed(count) = idc(x);
      count = count + 1;
   end
   x = x+1;
end
% set up cammo mask and place patch seeds
cammo_mask = zeros(r,c);    % initialize
for i = 1:patches   % place seeds
   cammo_mask(rseed(i),cseed(i)) = 1;
end
% loop over percentage (inceremetal) cover
sum_object_mask = sum(sum(object_mask(:,:,3)));       % initialize
sum_ground_mask = sum(sum(ground_mask(:,:,3)));    % initialize
color = zeros(sum_ground_mask,3);                          % initialize
 [gr,gc] = find(ground_mask(:,:,1));
if per == 100; per = 99; end    % remove the 100% trap
cover  = (per)/100;
% expand (dilate) patch seeds until cammo cover ratio is met
while sum(sum(cammo_mask .* object_mask(:,:,3)))...
      /sum_object_mask <= cover
   se = strel('sphere',6);
   cammo_mask = imdilate(cammo_mask,se);
   % cammo_mask_par = imdilate(cammo_mask_par,true(3));    % 30% faster
end
ground = image .* uint8(ground_mask);   % create ground image
% color in image with random pixels from ground
count = 1;     % inititialize
for dd = 1:length(gr)    % loop through all pixels in picture
   color(count,:) = ground(gr(dd),gc(dd),:);   % save color
   count = count + 1;
end
% randomize ground color
yy=length(color);
idx = randperm(yy);
% copy image to cammoed_image
cammoed_image = image;
 [cr,ccc] = find(cammo_mask(:,:,1));
% apply ground color to camoed_image according to cammo mask
```

```
count = 1;     % initialize
for dd = 1:length(cr)    % loop through all pixels in picture
   if cammo_mask(cr(dd),ccc(dd)) * object_mask(cr(dd),ccc(dd)) == 1
      if count == length(idx)  % cycle if not enough
         count = 1;
      end
      cammoed_image(cr(dd),ccc(dd),:) = color(idx(count),:);
      count = count + 1;
   end
end
```

coloration.m

```
function[colored_image] = coloration(image,object_mask,cc,stripe_orientation,stripe_width)
[r,c,~] = size(object_mask);
% loop over percentage (inceremetal) distortion
% cammo_mask = zeros(r,c);                    % initialize
% [g_r,g_c] = find(ground_mask(:,:,1)); % find size of ground mask
hh = abs(cc-21);
stripe_contrast = hh/20;
if stripe_contrast==0; stripe_contrast=.001; end
% create striped image
if stripe_orientation == 1  % (vertical)
   one_line = [255*ones(1,stripe_width), zeros(1,stripe_width)];
   for ii=1:floor(c/stripe_width)
      rr = round(rand(1,1) * stripe_width); % randomize bands
      ss = stripe_width - rr;
      one_cycle = [255*ones(1,rr), zeros(1,ss)];
      one_line = cat(2,one_line, one_cycle);
      striped_image = repmat(one_line,[r,1]);
   end
   striped_image = striped_image(1:r,1:c); % crop
end
if stripe_orientation == 0 % (horizonral)
   one_line = [255*ones(stripe_width,1); zeros(stripe_width,1)];
   for ii=1:floor(r/stripe_width)
      rr = round(rand(1,1) * stripe_width);
      ss = stripe_width - rr;
      one_cycle = [255*ones(rr,1); zeros(ss,1)];
      one_line = cat(1,one_line,one_cycle);
      striped_image = repmat(one_line,[1,c]);
   end
   striped_image = striped_image(1:r,1:c); % crop
end
% isolate object
% object_mask = pixel_labels == object_group;
object_image = image .* uint8(object_mask);
%  object_image = rgb2gray(object_image);     % convert to grayscale
% overlay striped image onto object image
f = inline(['imadjust(x,[],[],',num2str(stripe_contrast),')']);
alt_object_image_1 = roifilt2(object_image(:,:,1),striped_image,f);
alt_object_image_2 = roifilt2(object_image(:,:,2),striped_image,f);
alt_object_image_3 = roifilt2(object_image(:,:,3),striped_image,f);
```

```
alt_object_image = cat(3,alt_object_image_1,alt_object_image_2,alt_object_image_3);
% create group masks (legacy, but works)
background_mask = alt_object_image(:,:,3) == 0;
alt_object_mask =alt_object_image(:,:,3) ~= 0;
colored_image = image .* uint8(background_mask) + alt_object_image .* uint8(alt_object_mask);
end
```

defilade.m

```
function[defilade_image] = defilade(image,object_mask,ground_mask,per)
[r,c,~] = size(object_mask);
[tr,~] =  find(object_mask(:,:,1)); % max row index of mask
tr_max = max(tr);
% loop over percentage (inceremetal) cover
sum_ground_mask = sum(sum(ground_mask(:,:,3)));    % initialize
color = zeros(sum_ground_mask,3);                  % initialize
defilade_mask = zeros(r,c);                        % initialize
if per == 100; per = 99; end   % remove the 100% trap
   cover =(per)/100;
    while sum(sum(defilade_mask .* object_mask))/sum(sum(object_mask(:,:,3))) <= cover
       defilade_mask(tr_max,:) = 1;
       tr_max = tr_max - 1;
    end
    ground = image .* uint8(ground_mask);   % create ground image
    % color in image with random pixels from ground
     count = 1;     % inititialize
     for dd = 1:r    % loop through all pixels in picture
        for ee = 1:c
           if ground_mask(dd,ee,3) == 1    % pixel in ground mask?
              color(count,:) = ground(dd,ee,:);   % save color
              count = count + 1;
           end
        end
     end
      % randomize ground color
      yy=length(color);
       idx = randperm(yy);
       % copy image to cammoed_image
       defilade_image = image;
        % apply ground color to camoed_image according to cammo mask
        count = 1;     % initialize
        for dd = 1:r    % loop through all pixels in picture
             for ee = 1:c
                if defilade_mask(dd,ee) * object_mask(dd,ee) == 1
                   if count == length(idx)  % cycle if not enough
                      count = 1;
                   end
                   defilade_image(dd,ee,:) = color(idx(count),:);
                   count = count + 1;
                end
             end
          end
       end
```

distortion.m

```
1  function[distorted_image] = distortion(image,object_mask,cc)
2  % dilate object
3  object_image= image .* uint8(object_mask);
4  se = strel(rand(cc*2,cc*2));
5  distorted_image= imdilate(object_image,se);
6  % re-create group masks after distortion to allow object mask to shift with
7  % distorion
8  background_mask = distorted_image(:,:,3) == 0;
9  distorted_mask = distorted_image(:,:,3) ~= 0;
10 % recreate distorted image
11 distorted_image = image .* uint8(background_mask) + distorted_image .* uint8(distorted_mask);
12 end
```

jam.m

```
1  function[jammed_image] = jam(image,per)
2  cover = per / 100;
3  jammed_image = imnoise(image,'salt & pepper',cover);
4  end
```

Appendix C – MATLAB Scripts That Control Convolutional Neural Network Training

The master script that is used to train and test the convolutional neural networks (CNNs) is called COMPLETE_TRAIN_AND_TEST_SUITE.m. This script determines whether the CNNs specified are fine-tuned using transfer learning or are trained from neutral. The name of the training/fine-tuning data sets (usually the numerical date) also includes a flag indicating if transfer learning or learning-from-neutral was used. This name will also be extended to the testing results stored on disk. The CNNs, stresses, and CNNs used in the analysis are also specified in the header of this script.

The script loops through the CNNs specified for training/fine-tuning, then tests the resulting CNNs against the suite of incrementally stressed test images to determine true positive (TP), true negative (TN), false positive (FP), and false negative (FN) classification results.

The training loop and the testing calls are commented or un-commented depending on demand.

COMPLETE_TRAIN_AND_TEST_SUITE.m

```
clearvars; clc;
% this is the master file that controls the training (either from neutral
% or transfer learning pre-trained CNN models, and the testing of these
% models against battlefield stressed test images.
tic    % set job timer
global now CNN_training_date CNNtype train_class_type test_class_type ...
   weights_used;
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CNN_training_date = '20220427'; % training date of CNN model used
now = '20220427';            % date of training or testing run
weights_used = 1;
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% define input parameters of training/testing
train_class_type = ["abrams","bradley","hmmwv","mrap","stryker",...
   "t72"]; % alphabetical order must be maintained
test_class_type = ["abrams","bradley","hmmwv","mrap","stryker",...
   "t72"]; % alphabetical order must be maintained
% define CNNs to be tested
CNNtype = ["SQUEEZENET","GOOGLENET","RESNET18","MOBILENETV2","RESNET50",...
   "RESNET101","INCEPTIONV3","INCEPTIONRESNETV2","NASNETLARGE"];
% change output file names if trained-from-neutral
if weights_used ~= 1
   CNN_training_date = strcat(CNN_training_date,'_from_neutral');
   now = strcat(now,'_from_neutral');
end
% % reset GPUs
delete(gcp('nocreate'))
parpool(23);   % define 23 workers for parallelization
%%% train networks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for ii = 1:length(CNNtype)
   run(strcat(pwd,'/STAGING-TRAIN_MILITARY_VEHICLES/TRAINING/TRAIN_',...
```

```
        CNNtype(ii)));
    close all
end
%%% train networks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% test networks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% test against camouflage stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/CAMOUFLAGE_ANALYSIS',...
    '/TEST_CNN_ON_CAMMO/RUN_CAMMO_BATCH'));
% test against coloration1 stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/COLORATION1_ANALYSIS',...
    '/TEST_CNN_ON_COLORATION1/PARALLEL_COLORATION1_TEST'));
% test against coloration2 stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/COLORATION2_ANALYSIS',...
    '/TEST_CNN_ON_COLORATION2/PARALLEL_COLORATION2_TEST'));
% test against distortion stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/DISTORTION_ANALYSIS',...
    '/TEST_CNN_ON_DISTORTION/PARALLEL_DISTORTION_TEST'));
% test against hull defilade stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/DEFILADE_ANALYSIS',...
    '/TEST_CNN_ON_DEFILADE/PARALLEL_DEFILADE_TEST'));
% test against jamming stressors
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/JAM_ANALYSIS',...
    '/TEST_CNN_ON_JAMMING/PARALLEL_JAM_TEST'));
% gather all results into a mega-matrix
run(strcat(pwd,'/STAGING-TEST_MILITARY_VEHICLES/DATA_REDUCTION'));
%%% test networks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
toc    % stop job time
```

Nine convolutional neural networks were analyzed in this work. Because the last layers of each network had to be adjusted to the classes of objects classified, the script for each CNN used is included even though there may be substantial repetition.

`FreezeWeights.m` is a function that allows the weights and biases of most of the models to be frozen so that they are not changed during fine-tuning of the models. This function is called in most of the larger CNN models when using transfer-learning to fine-tune the CNN models to the classes of interest.

1. TRAIN_SQUEEZENET.m

```
global CNN_training_date weights_used;
tic;
disp('training SQUEEZENET');
storage_folder = strcat(pwd,'/../TRAINED_CNN/squeezenet_',CNN_training_date);
plots = 0;
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
```

```
net = squeezenet;
% assign weights and biases from previous training
if weights_used == 1
   lgraph = layerGraph(net);      % use pre-trained weights
   epochs = 10;
else
   lgraph = squeezenet('Weights','none'); % train from neutral
   epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% REPLACE FINAL LAYERS: to retrain squeezenet to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'pool10','prob','ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
   fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
   'BiasLearnRateFactor',10)
   softmaxLayer('name','softmax');
   classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'relu_conv10','fc');
 %%% unique to squeezenet %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if isa(net,'SeriesNetwork')
   lgraph = layerGraph(net.Layers);
   if weights_used ~= 1
      lgraph = squeezenet('Weights','none');
   end
else
   lgraph = layerGraph(net);
   if weights_used ~= 1
      lgraph = squeezenet('Weights','none');
   end
end
[learnableLayer,classLayer] = findLayersToReplace(lgraph);
numClasses = numel(categories(imdsTrain.Labels));
newLearnableLayer = convolution2dLayer(1,numClasses, ...
      'Name','new_conv', ...
      'WeightLearnRateFactor',10, ...
      'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);
%%% unique to squeezenet %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
% layers(1:10) = freezeWeights(layers(1:10));
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
   'RandXReflection',true, ...
```

```
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
    mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

2. TRAIN_GOOGLENET.m

```
global CNN_training_date weights_used;
tic;
disp('training GOOGLENET');
storage_folder = strcat(pwd,'/../TRAINED_CNN/googlenet_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load convolutional neural network
net = googlenet();
% assign weights and biases from previous training (or not)
if weights_used == 1
    lgraph = layerGraph(net);      % use pre-trained weights
```

```
    epochs = 10;
else
    lgraph = googlenet('Weights','none'); % train from neutral
    epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain GoogLeNet to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'loss3-classifier','prob','output'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
    fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',10)
    softmaxLayer('name','softmax');
    classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'pool5-drop_7x7_s1','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
lif weights_used == 1; ayers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
```

```
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
   mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

3. TRAIN_RESNET18.m

```
global CNN_training_date weights_used;
tic;
disp('training RESNET18');
storage_folder = strcat(pwd,'/../TRAINED_CNN/resnet18_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
   'IncludeSubfolders',true, ...
   'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = resnet18();
% assign weights and biases from previous training (or not)
if weights_used == 1
   lgraph = layerGraph(net);      % use pre-trained weights
   epochs = 10;
else
   lgraph = resnet18('Weights','none'); % train from neutral
   epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain resnet18 to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'fc1000','prob','ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
   fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
   'BiasLearnRateFactor',10)
   softmaxLayer('name','softmax');
   classificationLayer('Name','classoutput')];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'pool5','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
% layers = lgraph.Layers;
% layers(1:110) = freezeWeights(layers(1:110));
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
```

```
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
    mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

4. TRAIN_MOBILENETV2.m

```
global CNN_training_date weights_used;
tic;
disp('training MOBILENET');
storage_folder = strcat(pwd,'/../TRAINED_CNN/mobilenetv2_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = mobilenetv2();
% assign weights and biases from previous training
if weights_used == 1
    lgraph = layerGraph(net);      % use pre-trained weights
```

```
    epochs = 10;
else
    lgraph = mobilenetv2('Weights','none'); % train from neutral
    epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain mobilenet to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'Logits','Logits_softmax','ClassificationLayer_Logits'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
    fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',10)
    softmaxLayer('name','softmax');
    classificationLayer('Name','classoutput')];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'global_average_pooling2d_1','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
```

```
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
   mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

5. TRAIN_RESNET50.m

```
global CNN_training_date weights_used;
tic;
disp('training RESNET50');
storage_folder = strcat(pwd,'/../TRAINED_CNN/resnet50_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
   'IncludeSubfolders',true, ...
   'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = resnet50();
% assign weights and biases from previous training
if weights_used == 1
   lgraph = layerGraph(net);      % use pre-trained weights
   epochs = 10;
else
   lgraph = resnet50('Weights','none'); % train from neutral
   epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% REPLACE FINAL LAYERS: to retrain resnet50 to clasify new images, replace
% the last three layers of the network.
lgraph = removeLayers(lgraph, {'fc1000','fc1000_softmax',...
   'ClassificationLayer_fc1000'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
   fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
   'BiasLearnRateFactor',15)
   softmaxLayer('Name','softmax');
   classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'avg_pool','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
```

```
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify the training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
    mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

6. TRAIN_RESNET101.m

```
global CNN_training_date weights_used;
tic;
disp('training RESNET101');
storage_folder = strcat(pwd,'/../TRAINED_CNN/resnet101_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = resnet101();
% assign weights and biases from previous training (or not)
if weights_used == 1
    lgraph = layerGraph(net);      % use pre-trained weights
```

```
    epochs = 10;
else
    lgraph = resnet101('Weights','none'); % train from neutral
    epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain resnet101 to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'fc1000','prob','ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
    fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',10)
    softmaxLayer('Name','softmax');
    classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'pool5','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify the training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
```

```
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
   mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

7. TRAIN_INCEPTIONV3.m

```
global CNN_training_date weights_used;
tic;
disp('training INCEPTION');
storage_folder = strcat(pwd,'/../TRAINED_CNN/inceptionv3_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
   'IncludeSubfolders',true, ...
   'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = inceptionv3();
% assign weights and biases from previous training
if weights_used == 1
   lgraph = layerGraph(net);      % use pre-trained weights
   epochs = 10;
else
   lgraph = inceptionv3('Weights','none'); % train from neutral
   epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain GoogLeNet to clasify new images, replace
% the last three layers of the network.
lgraph = removeLayers(lgraph, {'predictions','predictions_softmax','ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
   fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
   'BiasLearnRateFactor',10)
   softmaxLayer('Name','softmax');
   classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers.
lgraph = connectLayers(lgraph,'avg_pool','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero.
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
   'RandXReflection',true, ...
```

```
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify the training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data.
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
    mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

8. TRAIN_INCEPTIONRESNETV2.m

```
global CNN_training_date weights_used;
tic;
disp('training INCEPTIONRESNET');
storage_folder = strcat(pwd,'/../TRAINED_CNN/inceptionresnetv2_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load convolutional neural network
net = inceptionresnetv2();
% assign weights and biases from previous training (or not)
if weights_used == 1
    lgraph = layerGraph(net);      % use pre-trained weights
    epochs = 10;
```

```
else
    lgraph = inceptionresnetv2('Weights','none'); % train from neutral
    epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain inceptionresnet to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'predictions','predictions_softmax',...
    'ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
    fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
    'BiasLearnRateFactor',10)
    softmaxLayer('name','softmax');
    classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers.
lgraph = connectLayers(lgraph,'avg_pool','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero.
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify the training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',128, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
```

```
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
   mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

9. TRAIN_NASNETLARGE.m

```
global CNN_training_date weights_used;
tic;
disp('training NASNETLARGE');
storage_folder = strcat(pwd,'/../TRAINED_CNN/nasnetlarge_',CNN_training_date);
% load new images as an image datastore
imds = imageDatastore(strcat(pwd,'/../../TRAIN_MILITARY_VEHICLES'), ...
   'IncludeSubfolders',true, ...
   'LabelSource','foldernames');
% divide the data into training (70%) and validation (30%) sets
[imdsTrain, imdsValidation] = splitEachLabel(imds,0.7,'randomized');
% load pretrained network
net = nasnetlarge();
% assign weights and biases from previous training (or not)
if weights_used == 1
   lgraph = layerGraph(net);      % use pre-trained weights
   epochs = 10;
else
   lgraph = nasnetlarge('Weights','none'); % train from neutral
   epochs = 15;
end
% determine needed size for input layer
inputSize = net.Layers(1).InputSize;
% to retrain GoogLeNet to clasify new images, replace
% the last three layers of the network
lgraph = removeLayers(lgraph, {'predictions','predictions_softmax',...
   'ClassificationLayer_predictions'});
numClasses = numel(categories(imdsTrain.Labels));
newLayers = [
   fullyConnectedLayer(numClasses,'Name','fc','WeightLearnRateFactor',10,...
   'BiasLearnRateFactor',10)
   softmaxLayer('name','softmax');
   classificationLayer('Name','classoutput') ];
lgraph = addLayers(lgraph,newLayers);
% connect the last transferred layer remaining in the network to the new layers
lgraph = connectLayers(lgraph,'global_average_pooling2d_2','fc');
% optionally, you can "freeze" the weights of the earlier layers in the
% network by setting the learning rates in those layers to zero
layers = lgraph.Layers;
if weights_used == 1; layers(1:110) = freezeWeights(layers(1:110)); end
% set training parameters
pixelRange = [-30,30];
imageAugmenter = imageDataAugmenter( ...
```

```
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
% augment and resize training set
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% augment and resize validation set
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation, ...
    'DataAugmentation',imageAugmenter,'ColorPreprocessing','gray2rgb', ...
    'DispatchInBackground',true);
% specify training options
options = trainingOptions('sgdm', ...
    'MiniBatchSize',32, ...
    'MaxEpochs',epochs, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augimdsValidation, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',50, ...
    'DispatchInBackground',true, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.9, ...
    'LearnRateDropPeriod', 2, ...
    'Plots','training-progress', ...
     'Verbose', false, ...
     'ExecutionEnvironment','multi-gpu');
% train the network using the training data
net = trainNetwork(augimdsTrain,lgraph,options);
% store trained network, labels, and numclasses
if ~exist(storage_folder,'dir')
    mkdir(storage_folder);
end
save(strcat(storage_folder,'/1_testnet'),'net');
save(strcat(storage_folder,'/1_numclasses'),'numClasses');
toc;
```

freezeWeights.m

```
% layers = freezeWeights(layers) sets the learning rates of all the
% parameters of the layers in the layer array |layers| to zero.
function layers = freezeWeights(layers)
for ii = 1:size(layers,1)
    props = properties(layers(ii));
    for p = 1:numel(props)
        propName = props{p};
        if ~isempty(regexp(propName, 'LearnRateFactor$', 'once'))
            layers(ii).(propName) = 0;
        end
    end
end
end
```

Appendix D – Scripts to Test the Fined-Tuned Convolutional Neural Networks on Battlefield Stressed Images

For completeness, the test script files are all presented, even though there is substantial repetition between them. The first six stresses are for camouflage over the target in 5, 10, 15, 20, 25, and 30 patches. The seventh and eighth stresses are the disruptive coloration simulated as vertical and horizontal lines. The ninth stress is dilation, which pixelates the detail out of the target. The tenth stress is defilade, which simulates partial burial of the target. The eleventh stress is obscuration of the whole image simulated by adding noise to random pixels. All the test scripts use the `test_tuned_cnn.m` function that gathers classification failure (CF) and receiver operating characteristic (ROC) information of the CNN model classification of the test images for different stress types as stress levels are increased.

RUN_CAMMO_BATCH.m

```
function RUN_BATCH_test
global now CNN_training_date;
PARALLEL_CAMMO_5_PATCH_TEST
PARALLEL_CAMMO_10_PATCH_TEST
PARALLEL_CAMMO_15_PATCH_TEST
PARALLEL_CAMMO_20_PATCH_TEST
PARALLEL_CAMMO_25_PATCH_TEST
PARALLEL_CAMMO_30_PATCH_TEST
end
```

PARALLEL_CAMMO_5_PATCH_TEST.m

```
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 5 SEEDS');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;   % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   cammo_suite = '5';
   train_class_type_par = train_class_type;
   fixed_threshold_par = fixed_threshold;
   CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/',...
      CNN,'_',CNN_training_date);
   for bb = 1:num_trials
      test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/',...
         cammo_suite,'/',num2str(bb),'/');
      parfor cc=1:21
         percent = (cc-1)*5;
         local_test_dir = strcat(test_dir,num2str(percent));
         disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
         [TPR_local,FPR_local,pos_local] = ...
            test_tuned_cnn(CNN_dir,local_test_dir, ...
            train_class_type_par,fixed_threshold_par);
         TPR(bb,cc,:,:) = TPR_local;
         FPR(bb,cc,:,:) = FPR_local;
         pos(bb,cc,:,:) = pos_local;
      end
```

```
    end
    % write out results
    if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
       mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'));
    end
    save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
       '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_CAMMO_10_PATCH_TEST.m

```
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 10 seeds');
 %%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;    % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
    CNN = CNNtype(aa);
    cammo_suite = '10';
     train_class_type_par = train_class_type;
    fixed_threshold_par = fixed_threshold;
    CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/',...
       CNN,'_',CNN_training_date);
    for bb = 1:num_trials
       test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/',...
          cammo_suite,'/',num2str(bb),'/');
       parfor cc=1:21
          percent = (cc-1)*5;
          local_test_dir = strcat(test_dir,num2str(percent));
          disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
          [TPR_local,FPR_local,pos_local] = ...
             test_tuned_cnn(CNN_dir,local_test_dir, ...
             train_class_type_par,fixed_threshold_par);
          TPR(bb,cc,:,:) = TPR_local;
          FPR(bb,cc,:,:) = FPR_local;
          pos(bb,cc,:,:) = pos_local;
       end
    end
    % write out results
    if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
       mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'))
    end
    save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
       '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_CAMMO_15_PATCH_TEST.m

```
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 15 SEEDS')
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;    % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
```

```
    CNN = CNNtype(aa);
    cammo_suite = '15';
    train_class_type_par = train_class_type;
    fixed_threshold_par = fixed_threshold;
    CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/', ...
       CNN,'_',CNN_training_date);
    for bb = 1:num_trials
       test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/', ...
          cammo_suite,'/',num2str(bb),'/');
       parfor cc=1:21
          percent = (cc-1)*5;
          local_test_dir = strcat(test_dir,num2str(percent));
          disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
          [TPR_local,FPR_local,pos_local] = ...
             test_tuned_cnn(CNN_dir,local_test_dir, ...
             train_class_type_par,fixed_threshold_par);
          TPR(bb,cc,:,:) = TPR_local;
          FPR(bb,cc,:,:) = FPR_local;
          pos(bb,cc,:,:) = pos_local;
       end
     end
    % write out results
    if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
       mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'));
    end
    save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
       '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_CAMMO_20_PATCH_TEST.m

```
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 20 seeds');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;    % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
    CNN = CNNtype(aa);
    cammo_suite = '20';
    train_class_type_par = train_class_type;
    fixed_theshold_par = fixed_threshold;
    CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/',...
       CNN,'_',CNN_training_date);
    for bb = 1:num_trials
       test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/',...
          cammo_suite,'/',num2str(bb),'/');
       parfor cc=1:21
          percent = (cc-1)*5;
          local_test_dir = strcat(test_dir,num2str(percent));
          disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
          [TPR_local,FPR_local,pos_local] = ...
             test_tuned_cnn(CNN_dir,local_test_dir, ...
             train_class_type_par, fixed_theshold_par);
```

```
            TPR(bb,cc,:,:) = TPR_local;
            FPR(bb,cc,:,:) = FPR_local;
            pos(bb,cc,:,:) = pos_local;
        end
    end
    % write out results
    if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
        mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'));
    end
    save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
        '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_CAMMO_25_PATCH_TEST.m

```
global now  CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 25 SEEDS');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;    % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
    CNN = CNNtype(aa);
    cammo_suite = '25';
    train_class_type_par = train_class_type;
    fixed_threshold_par = fixed_threshold;
    CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/',...
        CNN,'_',CNN_training_date);
    for bb = 1:num_trials
        test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/',...
            cammo_suite,'/',num2str(bb),'/');
        parfor cc=1:21
            percent = (cc-1)*5;
            local_test_dir = strcat(test_dir,num2str(percent));
            disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
            [TPR_local,FPR_local,pos_local] = ...
                test_tuned_cnn(CNN_dir,local_test_dir, ...
                train_class_type_par,fixed_threshold_par);
            TPR(bb,cc,:,:) = TPR_local;
            FPR(bb,cc,:,:) = FPR_local;
            pos(bb,cc,:,:) = pos_local;
        end
    end
    % write out results
    if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
        mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'));
    end
    save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
        '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_CAMMO_30_PATCH_TEST.m

```
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing CAMMO 30 SEEDS');
```

```
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;    % number of incremental cammo sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   cammo_suite = '30';
   train_class_type_par = train_class_type;
   fixed_threshold_par = fixed_threshold;
   CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/',...
      CNN,'_',CNN_training_date);
   for bb = 1:num_trials
      test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/cammo_data/',...
         cammo_suite,'/',num2str(bb),'/');
      parfor cc=1:21
         percent = (cc-1)*5;
         local_test_dir = strcat(test_dir,num2str(percent));
         disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)));
         [TPR_local,FPR_local, pos_local] = ...
            test_tuned_cnn(CNN_dir,local_test_dir, ...
            train_class_type_par,fixed_threshold_par);
         TPR(bb,cc,:,:) = TPR_local;
         FPR(bb,cc,:,:) = FPR_local;
         pos(bb,cc,:,:) = pos_local;
      end
   end
   % write out results
   if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'),'dir')
      mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis'));
   end
   save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_cammo_analysis/',cammo_suite,...
      '_',string(CNN),'_cammo_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_COLORATION1_TEST.m

```
clearvars;
global now  CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing COLORATION1');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/');
test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/coloration1_data');
train_class_type_par = train_class_type;
fixed_threshold_par = fixed_threshold;
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   trained_CNN_dir = strcat(string(CNN_dir),'/',CNNtype(aa),'_',CNN_training_date);
   for bb = 1:num_trials   % loop over statistical trial
      test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/coloration1_data/', ...
         num2str(bb),'/');
      parfor cc=1:21  % loop over percent
         percent = (cc-1)*5;
```

```
            local_test_dir = strcat(test_dir,num2str(percent));
            disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)))
            [TPR_local,FPR_local,pos_local] = ...
               test_tuned_cnn(trained_CNN_dir,local_test_dir, ...
               train_class_type_par,fixed_threshold_par);
            TPR(bb,cc,:,:) = TPR_local;
            FPR(bb,cc,:,:) = FPR_local;
            pos(bb,cc,:,:) = pos_local;
         end
      end
      % write out results
      if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration1_analysis'),'dir')
         mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration1_analysis'));
      end
      save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration1_analysis/',...
         now,'_',string(CNN),'_coloration1_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_COLORATION2_TEST.m

```
clearvars;
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing COLORATION2');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/');
test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/coloration2_data');
train_class_type_par = train_class_type;
fixed_threshold_par = fixed_threshold;
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   trained_CNN_dir = strcat(string(CNN_dir),'/',CNNtype(aa),'_',CNN_training_date);
   for bb = 1:num_trials   % loop over statistical trial
      test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/coloration2_data/', ...
         num2str(bb),'/');
      parfor cc=1:21  % loop over percent
         percent = (cc-1)*5;
         local_test_dir = strcat(test_dir,num2str(percent));
         disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)))
         [TPR_local,FPR_local,pos_local] = ...
            test_tuned_cnn(trained_CNN_dir,local_test_dir, ...
            train_class_type_par,fixed_threshold_par);
         TPR(bb,cc,:,:) = TPR_local;
         FPR(bb,cc,:,:) = FPR_local;
         pos(bb,cc,:,:) = pos_local;
      end
   end
   % write out results
   if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration2_analysis'),'dir')
      mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration2_analysis'));
   end
   save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_coloration2_analysis/',...
```

```
        now,'_',string(CNN),'_coloration2_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_DEFILADE_TEST.m

```
clearvars;
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing DEFILADE');
CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/');
test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/defilade_data/');
train_class_type_par = train_class_type;
fixed_threshold_par = fixed_threshold;
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   trained_CNN_dir = strcat(string(CNN_dir),'/',CNNtype(aa),'_',CNN_training_date);
   parfor cc=1:21    % cycle through stress increments
      percent = (cc-1)*5;
      local_test_dir = strcat(test_dir,num2str(percent));
      disp(strcat(string(CNN),':',num2str(cc)))
      [TPR_local,FPR_local,pos_local] = ...
         test_tuned_cnn(trained_CNN_dir,local_test_dir, ...
         train_class_type_par,fixed_threshold_par);
      TPR(cc,:,:) = TPR_local;
      FPR(cc,:,:) = FPR_local;
      pos(cc,:,:) = pos_local;
   end
   % write out results
   if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_defilade_analysis'),'dir')
      mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_defilade_analysis'));
   end
   save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_defilade_analysis/',...
      now,'_',string(CNN),'_defilade_ROC_data'),'TPR','FPR');
end
```

PARALLEL_DISTORTION_TEST.m

```
clearvars;
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing DISTORTION');
%%% user input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_trials = 25;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/');
test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/distortion_data/');
train_class_type_par = train_class_type;
fixed_threshold_par = fixed_threshold;
for aa = 1:length(CNNtype)
   CNN = CNNtype(aa);
   trained_CNN_dir = strcat(string(CNN_dir),'/',CNNtype(aa),'_',CNN_training_date);
   for bb = 1:num_trials   % loop over statistical trial
      test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/distortion_data/', ...
         num2str(bb),'/');
      parfor cc=1:21
         percent = (cc-1)*5;
```

```
        local_test_dir = strcat(test_dir,num2str(percent));
        disp(strcat(string(CNN),':',num2str(bb),':',num2str(cc)))
        [TPR_local,FPR_local,pos_local] = ...
          test_tuned_cnn(trained_CNN_dir,local_test_dir, ...
          train_class_type_par,fixed_threshold_par);
        TPR(bb,cc,:,:) = TPR_local;
        FPR(bb,cc,:,:) = FPR_local;
        pos(bb,cc,:,:) = pos_local;
      end
    end
  if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_distortion_analysis'),'dir')
    mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_distortion_analysis'));
  end
  save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_distortion_analysis/',...
    now,'_',string(CNN),'_distortion_ROC_data'),'TPR','FPR','pos');
end
```

PARALLEL_JAM_TEST.m

```
clearvars;
global now CNN_training_date CNNtype train_class_type fixed_threshold;
disp('testing JAMMING');
 CNN_dir = strcat(pwd,'/../../../STAGING-TRAIN_MILITARY_VEHICLES/TRAINED_CNN/');
test_dir = strcat(pwd,'/../../../TEST_MILITARY_VEHICLES/jam_data/');
train_class_type_par = train_class_type;
fixed_threshold_par = fixed_threshold;
for aa = 1:length(CNNtype)
  CNN = CNNtype(aa);
  trained_CNN_dir = strcat(string(CNN_dir),'/',CNNtype(aa),'_',CNN_training_date);
  parfor cc=1:21    % cycle through stress increments
    percent = (cc-1)*5;
    local_test_dir = strcat(test_dir,num2str(percent));
    disp(strcat(string(CNN),':',num2str(cc)))
    [TPR_local,FPR_local,pos_local] = ...
      test_tuned_cnn(trained_CNN_dir,local_test_dir,...
      train_class_type_par,fixed_threshold_par);
    TPR(cc,:,:) = TPR_local;
    FPR(cc,:,:) = FPR_local;
    pos(cc,:,:) = pos_local;
  end
  if ~exist(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_jam_analysis'),'dir')
    mkdir(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_jam_analysis'));
  end
  save(strcat(pwd,'/../TEST_RESULTS/complete_',now,'_jam_analysis/',...
    now,'_',string(CNN),'_jam_ROC_data'),'TPR','FPR','pos');
end
```

test_tuned_cnn.m

```
function[TPR_local,FPR_local, pos_local] = ...
  test_tuned_cnn(CNN_dir,local_test_dir,train_class_type)
load(strcat(CNN_dir,'/1_testnet'),'net');  % load weights and biases
TPR_local = zeros(101,length(train_class_type));   % initialize
FPR_local = zeros(101,length(train_class_type));   % initialize
```

```
inputSize = net.Layers(1).InputSize;       % input image size
% read in test images into datastore
imdsTest = imageDatastore(local_test_dir, ...
   'IncludeSubfolders',true, 'LabelSource','foldernames');
% augment test images by resizing to needed size and insuring rgb
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest,'ColorPreprocessing','gray2rgb');
 [labels,probs] = classify(net,augimdsTest,'MiniBatchSize',15);
%accuracy = mean(YPred == labels);
probs2 = zeros(20,length(train_class_type),length(train_class_type));   % initialize
% reshape probability matrix dim = (20,6,6)
for aa = 1:length(train_class_type)
   M = ((aa-1) * 20) + (1:20);
   probs2(:,:,aa) = probs(M,:);
end
% reshape label matrix to dim = (20,6)
label2 = reshape(labels,20,length(train_class_type));
pos_local = zeros(101,length(train_class_type));  % initialize (#classes)
for  aa = 1:101     % cycle through thresholds
   threshold = (aa-1) / 100;
   if threshold == 0; threshold = .0005; end
   for bb = 1:length(train_class_type) % cycle through class
      table_1 = zeros(20,length(train_class_type));   % initialize (#test images, #classes)
      table_0 = zeros(20,length(train_class_type));   % initialize (#test images, #classes)
      for cc = 1:20   % cycle through test images
         % count success if probability is greater then threshold
         if label2(cc,bb) == train_class_type(bb) && probs2(cc,bb,bb) >= threshold
            pos_local(aa,bb) = pos_local(aa,bb) + 1;
         end
         for dd = 1:length(train_class_type) % cycle through class
            if probs2(cc,dd,bb) >= threshold
               table_1(cc,dd) = 1;
            else
               table_0(cc,dd) = 1;
            end
         end
      end
      sum_table_1 = squeeze(sum(table_1,1)); % dim = class(7)
      sum_table_0 = squeeze(sum(table_0,1)); % dim = class(7)
      TP = sum_table_1(bb);
      FP = sum(sum_table_1) - TP;
      FN = sum_table_0(bb);
      TN = sum(sum_table_0) - FN;
      TPR_local(aa,bb) = TP / (TP + FN);
      FPR_local(aa,bb) = FP / (FP + TN);
      if (FP == 0) && (TN == 0); FPR_local(aa,bb) = 0; end
   end
end
% sort entries according to ascending abscissa
[FPR_local,ind] = sort(FPR_local);
TPR_local = TPR_local(ind);
% last point at 1
FPR_local(end+1,:) = 1;    % force ROC curves to go up to (1,1)
TPR_local(end+1,:) = 1;    % force ROC curves to go up to (1,1)
```

```
end
```

DATA_REDUCTION.m

```
clearvars all; close all; clc;
global now CNNtype;
TPR_mat = zeros(11,length(CNNtype),25,21,102,6);
FPR_mat = zeros(11,length(CNNtype),25,21,102,6);
pos_mat = zeros(11,length(CNNtype),25,21,101,6);
%%% boil cammo data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for bb = 1:6    % cycle through number of patches
   for aa =1:length(CNNtype)
      load(strcat(pwd,'/CAMOUFLAGE_ANALYSIS/TEST_RESULTS/complete_',...
         now,'_cammo_analysis/',num2str(bb*5),'_',string(CNNtype(aa)),...
         '_cammo_ROC_data'));
      TPR_mat(bb,aa,:,:,:,:) = TPR;
      FPR_mat(bb,aa,:,:,:,:) = FPR;
      pos_mat(bb,aa,:,:,:,:) = pos;
   end
end
%%% coloration1 data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clearvars pos;
for aa = 1:length(CNNtype)
   load(strcat(pwd,'/COLORATION1_ANALYSIS/TEST_RESULTS/complete_',...
      now,'_coloration1_analysis/',now,'_',string(CNNtype(aa)),...
      '_coloration1_ROC_data'));
   TPR_mat(7,aa,:,:,:,:) = TPR;
   FPR_mat(7,aa,:,:,:,:) = FPR;
   pos_mat(7,aa,:,:,:,:) = pos;
end
%%% coloration2 data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clearvars pos;
for aa = 1:length(CNNtype)
   load(strcat(pwd,'/COLORATION2_ANALYSIS/TEST_RESULTS/complete_',...
      now,'_coloration2_analysis/',now,'_',string(CNNtype(aa)),...
      '_coloration2_ROC_data'));
   TPR_mat(8,aa,:,:,:,:) = TPR;
   FPR_mat(8,aa,:,:,:,:) = FPR;
   pos_mat(8,aa,:,:,:,:) = pos;
end
%%% defilade data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clearvars pos;
for aa =1:length(CNNtype)
   load(strcat(pwd,'/DEFILADE_ANALYSIS/TEST_RESULTS/complete_',...
      now,'_defilade_analysis/',now,'_',string(CNNtype(aa)),...
      '_defilade_ROC_data'));
   % replicate 25 trials
   TPR_mat(9,aa,:,:,:,:) = permute(repmat(TPR,[1,1,1,25]),[4,1,2,3]);
   FPR_mat(9,aa,:,:,:,:) = permute(repmat(FPR,[1,1,1,25]),[4,1,2,3]);
   pos_mat(9,aa,:,:,:,:) = permute(repmat(pos,[1,1,1,25]),[4,1,2,3]);
end
%%% distortion data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clearvars pos;
```

```
for aa = 1:length(CNNtype)
   load(strcat(pwd,'/DISTORTION_ANALYSIS/TEST_RESULTS/complete_',...
            now,'_distortion_analysis/',now,'_',string(CNNtype(aa)),...
      '_distortion_ROC_data'));

   TPR_mat(10,aa,:,:,:,:) = TPR;
   FPR_mat(10,aa,:,:,:,:) = FPR;
   pos_mat(10,aa,:,:,:,:) = pos;
end
%%% jam data %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clearvars pos;
for aa =1:length(CNNtype)
   load(strcat(pwd,'/JAM_ANALYSIS/TEST_RESULTS/complete_',...
      now,'_jam_analysis/',now,'_',string(CNNtype(aa)),...
      '_jam_ROC_data'));
   % replicate 25 trials
   TPR_mat(11,aa,:,:,:,:) = permute(repmat(TPR,[1,1,1,25]),[4,1,2,3]);
   FPR_mat(11,aa,:,:,:,:) = permute(repmat(FPR,[1,1,1,25]),[4,1,2,3]);
   pos_mat(11,aa,:,:,:,:) = permute(repmat(pos,[1,1,1,25]),[4,1,2,3]);
end
% save large matrices to file
save(strcat(pwd,'/ROC_TRUTH_TABLES/',now,'_ROC_data'),'TPR_mat','FPR_mat');
save(strcat(pwd,'/ROC_TRUTH_TABLES/',now,'_pos_data'),'pos_mat');
```

Appendix E – Metrics Visualization Scripts

The classification failure (CF), receiver operating characteristic (ROC), ROC-area under the curve (AUC), and CF-AUC curves are generated from a single script that prompts the user to choose which parameter to compare (convolutional neural networks [CNNs], stresses, or classes) and then proceeds to calculate and plot these values.

TOTAL_ANALYSIS.m

```
clearvars;clc;close all;
% plot out ROC curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
now = '20220427';
weights_used = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% incorporate full training into the file name
if weights_used ~= 1
   now = strcat(now,'_from_neutral');
end
stress = ["Cammo-5","Cammo-10","Cammo-15","Cammo-20","Cammo-25",...
   "Cammo-30","Coloration-1","Coloration-2","Defilade","Distortion",...
   "Obscuration"];
class = ["abrams","bradley","hmmwv","mrap","stryker","T72"];
CNN_type = ["squeezenet","googlenet","resnet18","mobilenetv2",...
   "resnet50","resnet101","inceptionv3", "inceptionresnetv2", "nasnetlarge"];
sym1 = ["-o","-x","-v","-+","-<","-*","-s","-d","-^"];
sym2 = ["-o","-x","-v","-+","-<","-*","-s","-d","-^","-p","-o"];
sym3 = ["-o","-x","-v","-+","-<","-*"];
load(strcat(pwd,'/../STAGING-TEST_MILITARY_VEHICLES/ROC_TRUTH_TABLES/',...
   now,'_ROC_data'));
load(strcat(pwd,'/../STAGING-TEST_MILITARY_VEHICLES/ROC_TRUTH_TABLES/',...
   now,'_pos_data'));
% % dim    = stress, CNN, trial, %stress, threshold, class
% % TPR_mat =   11    9   25     21     102      6
% % pos_mat =   11    9   25     21     101      6
%
% average over 25 tests
TPR = squeeze(mean(TPR_mat,3));    % dim = 11,5,21,102,6
FPR = squeeze(mean(FPR_mat,3));    % dim = 11,5,21,102,6
pos = squeeze(mean(pos_mat,3));    % dim = 11,5,21,101,6
% develop ROC_auc matrix
[sa,sb,sc,~,se] = size(TPR);
for a = 1:sa
   for b = 1:sb
      for c = 1:sc
         for e = 1:se
            % form ROC_auc (dim = 11,5,21,6)
            ROC_auc(a,b,c,e) = trapz(squeeze(FPR(a,b,c,:,e)),...
               squeeze(TPR(a,b,c,:,e)));
         end
      end
   end
end
% develop pos_auc matrix
```

```
[sa,sb,~,sd,se] = size(pos);
for a = 1:sa
   for b = 1:sb
      for d = 1:sd
         for e = 1:se
            % form pos_auc (dim = 11,5,101,6)
            pos_auc(a,b,d,e) = trapz(squeeze((0:20)*5),pos(a,b,:,d,e));
         end
      end
   end
end
% determine what comparisons are desired
prompt = 'Compare CNNs(1), Stresses(2), Classes(3)?';
type_comp = input(prompt);
switch type_comp
   case 1     % compare CNN
      mTPR = squeeze(mean(TPR,[1,5]));    % dim = 5,21,102
      mFPR = squeeze(mean(FPR,[1,5]));    % dim = 5,21,102
      % plot results
      figure
      sgtitle('ROC comparisons of CNNs','fontWeight','bold');
      for a = 1:21
         subplot(3,7,a)
         title(strcat(num2str((a-1)*5),'% stress'))
         hold on
         for b = 1:length(CNN_type)
            plot(squeeze(mFPR(b,a,:)),squeeze(mTPR(b,a,:)))
         end
         grid on
         legend('squeezenet','googlenet','resnet18','mobilenetv2',...
            'resnet50','resnet101','inceptionv3', 'inceptionresnetv2',...
            'nasnetlarge','location','se')
         hold off
      end
      m_ROC_auc = squeeze(mean(ROC_auc,[1,4]));    % dim = 5,21
      figure
      hold on
      for a = 1:length(CNN_type)
         plot(((1:21)-1)*5,m_ROC_auc(a,:),sym1(a))
      end
      hold off
      grid on
      xlabel('percent stress')
      ylabel('normalized ROC-AUC')
      axis([0 100 .5 1])
      title('ROC-AUC comparison of CNNs')
      legend('squeezenet','googlenet','resnet18','mobilenetv2',...
         'resnet50','resnet101','inceptionv3', 'inceptionresnetv2',...
         'nasnetlarge','location','sw')
      m_pos = squeeze(mean(pos,[1,5]));   % dim = 5,21,101
      figure
      hold on
      for a = 1:length(CNN_type)
```

```
        plot((0:20)*5,m_pos(a,:,81)/20,sym1(a))
    end
    hold off
    grid on
    xlabel('percent stress')
    ylabel('classification success')
    title('CF comparisons of CNNs at .8 threshold');
    legend(CNN_type,'location','sw')
    m_pos_auc = squeeze(mean(pos_auc,[1,4]));   % dim = 5,101
    figure
    hold on
    for a = 1:length(CNN_type)
        plot((0:100)*.01,m_pos_auc(a,:)/max(m_pos_auc(a,:)),sym1(a))
    end
    hold off
    grid on
    xlabel('threshold')
    ylabel('CF-AUC')
    axis([0 1 0 1])
    title('CF-AUC comparison of CNNs')
    legend('squeezenet','googlenet','resnet18','mobilenetv2',...
        'resnet50','resnet101','inceptionv3', 'inceptionresnetv2',...
        'nasnetlarge','location','sw')
    clearvars mFPR mTPR;
case 2     % compare stress
    mTPR = squeeze(mean(TPR,[2,5]));    % dim = 11,21,102
    mFPR = squeeze(mean(FPR,[2,5]));    % dim = 11,21,102
    % plot results
    figure
    sgtitle('ROC comparisons of Stresses','fontWeight','bold');
    for a = 1:21
        subplot(3,7,a)
        title(strcat(num2str((a-1)*5),'% stress'))
        hold on
        for b = 1:length(stress)
            plot(squeeze(mFPR(b,a,:)),squeeze(mTPR(b,a,:)))
        end
        grid on
        legend(stress,'location','se')
        hold off
    end
    m_ROC_auc = squeeze(mean(ROC_auc,[2,4]));    % dim = 11,21
    figure
    hold on
    for a = 1:length(stress)
        plot(((0:20))*5,m_ROC_auc(a,:),sym2(a))
    end
    hold off
    grid on
    xlabel('percent stress');
    ylabel('ROC-AUC');
    axis([0 100 .6 1]);
    title('ROC-AUC comparison of Stresses')
```

```
152         legend(stress,'location','sw')
153         m_pos = squeeze(mean(pos,[2,5]));   % dim = 11,21,101
154         figure
155         hold on
156         for a = 1:length(stress)
157            plot((0:20)*5,m_pos(a,:,81)/20,sym2(a))
158         end
159         hold off
160         grid on
161         xlabel('percent stress')
162         ylabel('classification success')
163         title('CF comparisons of Stresses at .8 threshold');
164         legend(stress,'location','sw')
165         m_pos_AUC = squeeze(mean(pos_auc,[2,4]));   % dim = 11,101
166         figure
167         hold on
168         for a = 1:length(stress)
169            plot((0:100)*.01,m_pos_AUC(a,:)/max(m_pos_AUC(a,:)),sym2(a))
170         end
171         hold off
172         grid on
173         xlabel('threshold')
174         ylabel('CF-auc')
175         axis([0 1 0 1])
176         title('CF-AUC comparison of Stresses')
177         legend(stress,'location','sw')
178         clearvars mFPR mTPR;
179      case 3     % compare class
180         mTPR = squeeze(mean(TPR,[1,2]));    % dim = 21,102,6
181         mFPR = squeeze(mean(FPR,[1,2]));    % dim = 21,102,6
182         % plot results
183         figure
184         sgtitle('ROC comparison of Classes','fontWeight','bold');
185         for a = 1:21
186            subplot(3,7,a)
187            title(strcat(num2str((a-1)*5),'% stress'))
188            hold on
189            for b = 1:length(class)
190               plot(squeeze(mFPR(a,:,b)),squeeze(mTPR(a,:,b)))
191            end
192            grid on
193            legend(class,'location','se')
194            hold off
195         end
196         m_ROC_auc = squeeze(mean(ROC_auc,[1,2]));    % dim = 21,6
197         figure
198         hold on
199         for a = 1:length(class)
200            plot(((1:21)-1)*5,m_ROC_auc(:,a),sym3(a))
201         end
202         hold off
203         grid on
204         xlabel('percent stress');
```

```
ylabel('ROC-AUC');
axis([0 100 .7 1]);
title('ROC-AUC comparison of Classes')
legend(class,'location','sw')
m_pos = squeeze(mean(pos,[1,2]));   % dim = 21,101,6
figure
hold on
for a = 1:length(class)
   plot((0:20)*5,m_pos(:,81,a)/20,sym3(a))
end
hold off
grid on
xlabel('percent stress')
ylabel('classification success')
title('CF comparisons of Classes at .8 threshold');
legend(class,'location','sw')
m_pos_AUC = squeeze(mean(pos_auc,[1,2]));   % dim = 101,6
figure
hold on
for a = 1:length(class)
   plot((0:100)*.01,m_pos_AUC(:,a)/max(m_pos_AUC(:,a)),sym3(a))
end
hold off
grid on
xlabel('threshold')
ylabel('CF-auc')
axis([0 1 0 1])
title('CF-AUC comparison of Classes')
legend(class,'location','sw')
clearvars mFPR mTPR
```

LIST OF ACRONYMS

2D	two-dimensional
3D	three-dimensional
4D	four-dimensional
AI	artificial intelligence
AUC	area under the curve
CF	classification failure
CNN	convolutional neural network
DAC	DEVCOM Analysis Center
DEVCOM	U.S. Army Combat Capabilities Development Command
FN	false negative
FP	false positive
FPR	false positive rate
GPU	graphics processing unit
HMMWV	high-mobility multipurpose wheeled vehicle
MRAP	mine-resistant ambush-protected
ONNX	open neural network exchange
ROC	receiver operating characteristic
TN	true negative
TP	true positive
TPR	true positive rate
VIS	visible spectrum

ORGANIZATION

DEVCOM Analysis Center
FCDD-DAE-E/P. Debroux
1624 Headquarters Ave.
White Sands Missile Range, NM 88002

DEVCOM Army Research Laboratory
FCDD-RLD-DCI/Tech Library
2800 Powder Mill Rd.
Adelphi, MD 20783

Defense Technical Information Center
ATTN: DTIC-O
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6218