Detecting Corruption in Real Video Game Graphics using Deep Convolutional Neural Networks

Matthieu Chan Chee[1][a], Vinay Pandit[2][b] and Max Kiehn[2][c]
[1] *University of Toronto, Ontario, Canada*
[2] *AMD, Inc., Thornhill, Ontario, Canada*

Keywords: Video Game Display Corruption, Image Corruption Detection, Deep Convolutional Neural Network, EfficientNet, Structural Similarity Index Measure, Grad-CAM.

Abstract: Early detection of video game display corruption is essential to maintain the highest quality standards and to reduce the time to market of new GPUs targeted for the gaming industry. This paper presents a Deep Learning approach to automate gameplay corruption detection, which otherwise requires labor-intensive manual inspection. Unlike prior efforts which are reliant on *synthetically generated* corrupted images, we collected *real-world* examples of corrupted images from over 50 game titles. We trained an EfficientNet to classify input game frames as corrupted or golden using a two-stage training strategy and extensive hyperparameter search. Our method was able to accurately detect a variety of geometric, texture, and color corruptions with a precision of 0.989 and recall of 0.888.

1 INTRODUCTION

With high-performance GPUs being widely used in the gaming community, it is crucial for graphics card companies to rigorously test their products on a variety of video games to ensure the highest standard of user experience. The codebase behind graphics drivers often comprises of several thousand lines of code, which makes driver updates susceptible to introducing bugs that lead to *corruption* in the graphics, commonly known as ‘glitches’.

Catching such corruption entails test plans that can easily span thousands of different configurations on a multitude of game titles and test machines, making manual gameplay testing time consuming and labor intensive. Thus, automatically detecting corruption in the video games being tested is critical to accelerating the time to market of graphics drivers.

Due to the scarcity of real corruption in rendered video game graphics, previous work has focused on generating synthetic corruption as a surrogate and developed methods to detect those artificially generated malfunctions (C. Ling and Gisslén, 2020), (P. Davarmanesh and Malaya, 2020). However, generating the synthetic data is a non-trivial task since it involves writing generators for each type of corruption by domain experts. Substantial effort would be needed to generate a synthetic dataset fully representative of the corruption types and the variations observed in the real world which makes this process both expensive and time consuming. Hence, in this paper, we lay out a workflow for collecting *real* graphical glitches from gameplay scenarios (shown in Fig. 1) and demonstrate that Deep Convolutional Neural Networks (DCNN) can effectively detect such corruption in actual video game images.

The main contributions of this paper are as follows:

- We propose a data collection workflow to collect *real* corrupted images encompassing a variety of corruption types from multiple game titles including different genres, viewpoints, and modes.
- We propose a two-stage training mechanism comprising of transfer learning and finetuning and highlight its efficacy over a single-stage training.
- We provide a detailed comparative study of several neural network architectures to detect corrupted images and show that EfficientNet (Tan and Le, 2019) is better suited for this task.
- We demonstrate the use of Grad-CAM to interpret the network’s predictions.

[a] https://orcid.org/0000-0002-5279-4979
[b] https://orcid.org/0000-0001-7684-4080
[c] https://orcid.org/0000-0001-5662-0602

Chan Chee, M., Pandit, V. and Kiehn, M.
Detecting Corruption in Real Video Game Graphics using Deep Convolutional Neural Networks.
DOI: 10.5220/0010788900003124
In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 4: VISAPP*, pages 901-908
ISBN: 978-989-758-555-5; ISSN: 2184-4321

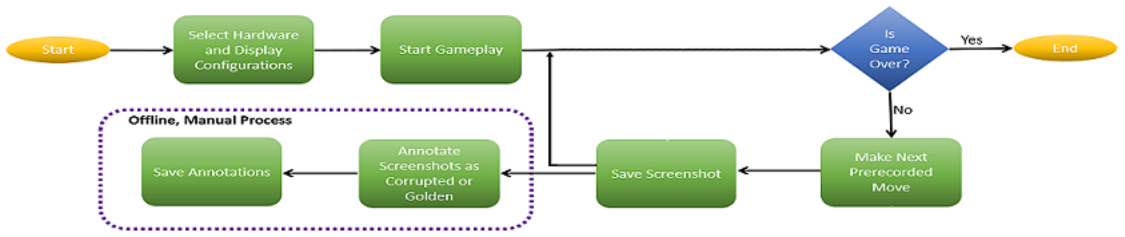

Start
Select Hardware and Display Configurations
Start Gameplay
Is Game Over?
Yes
End
No
Make Next Prerecorded Move
Save Screenshot
Offline, Manual Process
Annotate Screenshots as Corrupted or Golden
Save Annotations

Figure 1: Data collection workflow.

2 RELATED WORK

In recent years, there has been an increased interest in applying deep learning to visual defect detection tasks. This approach has the advantage of being able to find defects that cannot be detected by traditional computer vision algorithms (Y. Chen and Ho, 2019). In (E. McLaughlin and Narasimhan, 2019), a neural net is used for the pixel-wise segmentation of infra-red images of concrete delamination, while (Y. Chen and Ho, 2019) focused on the detection of surface scratches on plastic housings.

In the context of optical defect detection in video games, leveraging traditional methods would require specialized algorithms, such as corner detection (A. Nantes and Maire, 2008), to find different types of corruption. In contrast, (A. Nantes and Maire, 2013) proposed to use the game's internal state in addition to the rendered game graphics and leveraged multi-layer perceptrons (MLP) and self-organizing maps (SOM) to predict the presence of glitches.

Departing from previous approaches, (C. Ling and Gisslén, 2020) explored the use of DCNN to infer the presence of graphical malfunctions directly from the game's visual output, without the need for specialized pre-processing or extra information about the game state. This approach, being agnostic of what game is being tested and its internal mechanics, is highly versatile. While (C. Ling and Gisslén, 2020) demonstrated the effectiveness of DCNNs in detecting *artificially generated* corruption, our work focuses on collecting *real* graphical corruption from a large variety of games and demonstrating the performance of DCNNs in a real-world setting.

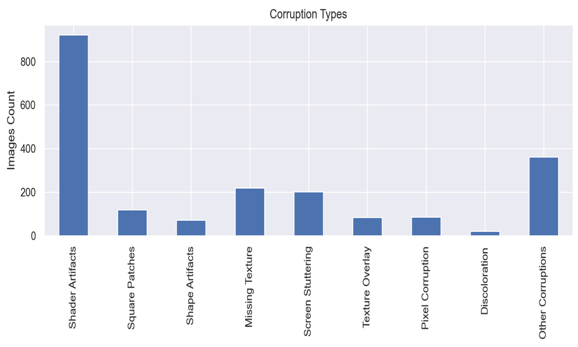
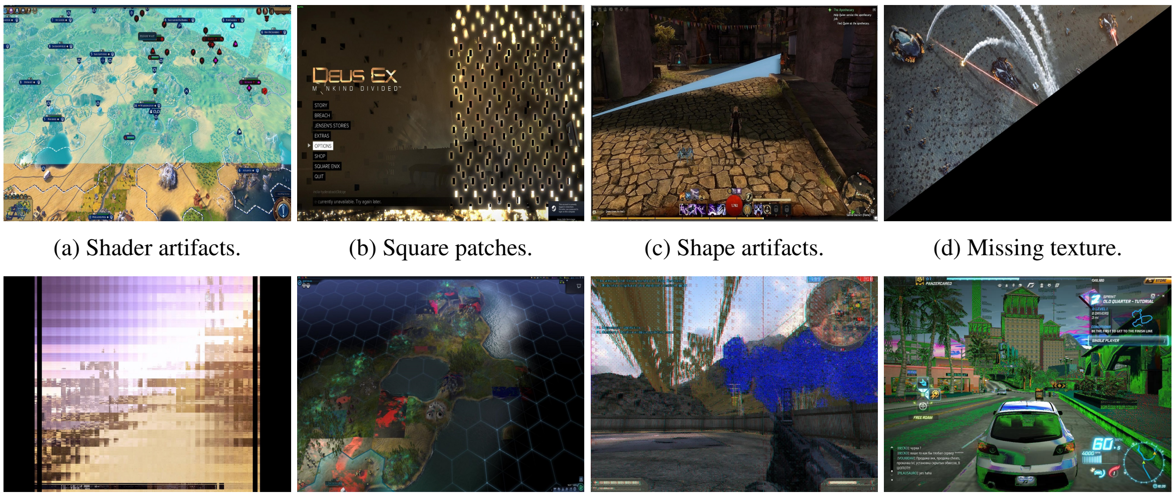


Figure 2: Distribution of corruption types.

3 DATA

3.1 Data Collection Workflow

Collecting a large, labeled dataset is an essential step towards training a DCNN. Though generating synthetic corrupted images is an option (C. Ling and Gisslén, 2020), (P. Davarmanesh and Malaya, 2020), since there are a variety of corruption types, a large effort is needed to generate corrupted images resembling real counterparts. Hence, in this work, we propose a procedure to collect *real* corrupted images. Since corruption rarely occurs in actual gameplay, collecting a large dataset of *real* glitches is challenging and involves playing games repeatedly. Hence, we designed a systematic data collection process in which repeating the gameplay and recording the rendered visual output is automated. The block diagram of our Data Collection Workflow is shown in Fig. 1. This process starts with selecting a test system with suitable hardware and display configurations. Next, the game client and the game under test are installed. Then an automated script starts the actual gameplay. To progress through the game, the script executes a series of pre-recorded commands consisting of timed keystrokes and mouse clicks. In parallel, the script also captures screenshots of the gameplay at regular intervals and stores them. At the end of the game, saved images are manually evaluated for the presence of display corruption and each image is labeled as *corrupted* or *golden* (not corrupted). Since the corruption occurrences are independent of the hardware or driver configurations, this process was repeated for different combinations of hardware configurations, driver versions, game titles, and game settings to produce the labeled dataset required for this work. In order to collect the data for multiple game titles, separate sets of pre-recorded commands were


(a) Shader artifacts. (b) Square patches. (c) Shape artifacts. (d) Missing texture.

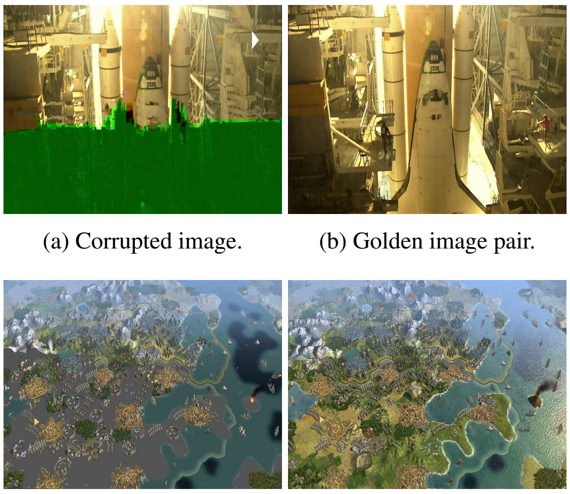
(e) Screen stuttering. (f) Texture overlay. (g) Pixel corruption. (h) Discoloration.

Figure 3: Examples of corrupted images.

prepared for each game.

3.2 Corruption Types

Our dataset is comprised of 2,087 *real* corrupted images and 6,000 golden images. While different characterizations of corruption exist, our categorization is similar to that in (P. Davarmanesh and Malaya, 2020). The distribution of the corruption types within our dataset is presented in Fig. 2. Broadly, glitches manifesting in distinct geometric patterns were subcategorized as Shader Artifacts, Square Patches, and Shape Artifacts. Similarly, texture malfunctions were subdivided as Missing Texture, Screen Stuttering, and Texture Overlay. Further, color rendering issues had two subtypes: Pixel Corruption and Discoloration.


(a) Corrupted image. (b) Golden image pair.

(c) Corrupted image. (d) Golden image pair.

Figure 4: Examples of corrupted and golden image pairs.

Some examples of corrupted images are shown in Fig. 3. While some images had more than one corruption types (Fig. 3g has both Pixel Corruption and Discoloration), others were random and were hard to categorize into any of the classes. Unclassifiable images were aggregated as *Other Corruptions* (see Fig. 2).

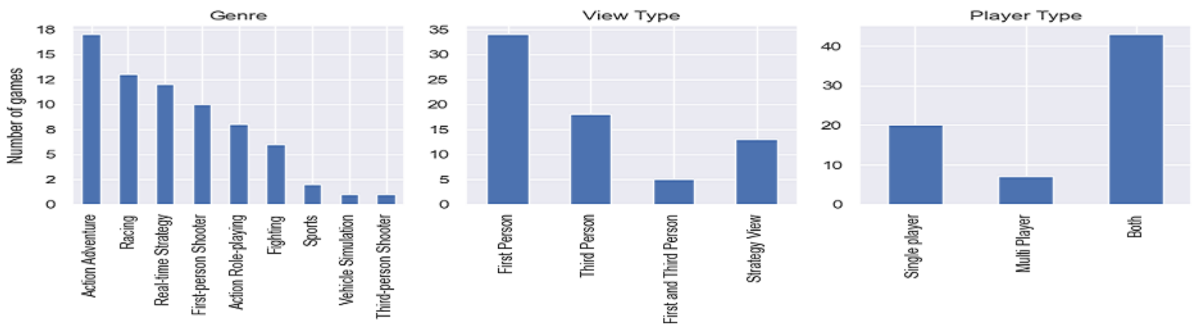
3.3 Golden Images

We refer to uncorrupted images as *golden*. Unlike corrupted images, we can generate an arbitrarily large number of new golden images by rerunning different game titles. However, these images are likely to be quite similar to the earlier runs. Further, since the screenshots are taken at a rate of one per second or faster, adjacent frames of a gameplay run are likely to be similar as well. To reduce this redundancy in the dataset, for every image we computed its Structural Similarity Index Measure (SSIM) (Z. Wang and Simoncelli, 2004) against the remaining images and filtered out the those with SSIM greater than a threshold. Of the filtered data points, we randomly selected 6,000 golden images.

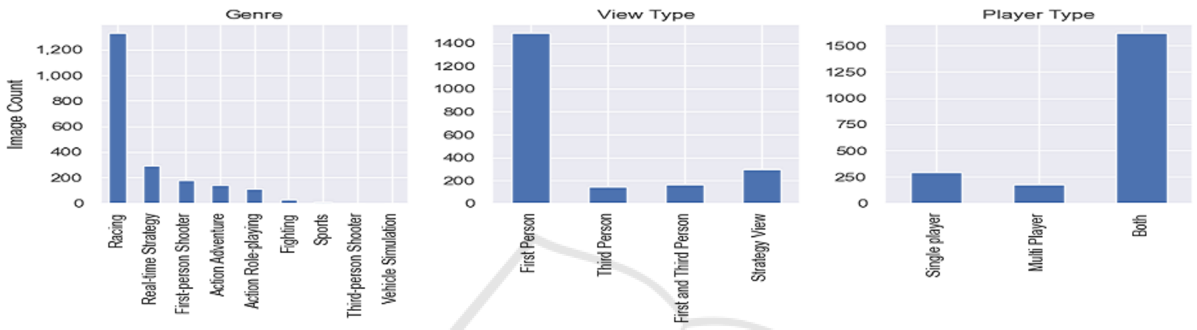
Since the images of both classes were derived from the same set of game titles, several of the corrupted images had a corresponding golden counterpart in the dataset. We believe this helped the network learn the difference between the classes and to produce more generalized model. Fig. 4 shows some examples of these pairs.

3.4 Game Titles

The dataset consisted of images from over 50 different game titles. Games included different genres (first



(a) Game titles distribution categorized by genre, view type, and player type.



(b) Corrupted images distribution categorized by genre, view type, and player type.

Figure 5: Distribution of game titles and corrupted images.

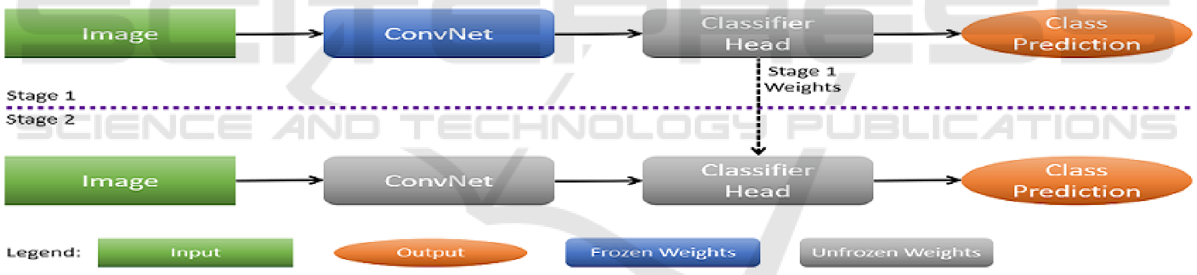


Figure 6: Two-stage training.

person shooter, racing etc.), multiple viewpoints (first person, third person, strategy, etc.), and modes (single player, multiplayer, or both). The distribution of the titles with respect to these properties is shown in Fig. 5a. Additionally, the distribution of corrupted images by game title categories is shown in Fig. 5b. Though several corrupted samples were from the *racing* genre, these images exhibited a variety of corruption types.

3.5 Training, Validation, and Test Split

Prior to training, 6,000 golden images of our dataset were randomly split into training, validation, and holdout test sets in a 0.75 : 0.1 : 0.15 ratio. The same ratio was applied to split the 2,087 corrupted images.

4 METHODOLOGY

Corruption in video games manifests in several forms, impacting the image quality in varying degrees of severity. While in some cases distortions are localized to a small image region, in others the entire image may be impacted. Hence, sophisticated scene-understanding capabilities are needed to detect corruption accurately. To meet these challenges, we propose a DCNN binary classifier which will analyze individual frames captured during gameplay for the presence of corruption.

We decided to treat this task as a binary classification rather than a multi-label classification problem for two reasons. Firstly, in the GPU testing workflow, corruption seldom occurs and even so, any im-

Table 1: High-performing models with hyperparameters and validation metrics.

Number of Parameters	Network	Input size	Class weights (golden, corrupted)	Validation			
				Precision	Recall	F1-score	Accuracy
3.5M	MobileNetV2	512×512	1,1	0.994	0.871	0.928	0.962
8.1M	DenseNet121	768×768	400,1	0.994	0.888	0.938	0.967
9.2M	**EfficientNetB2**	**768×768**	**1,1**	**0.994**	**0.899**	**0.944**	**0.970**
14.3M	DenseNet169	896×896	200,1	0.994	0.871	0.928	0.962
20.2M	DenseNet201	512×512	400,1	0.994	0.865	0.925	0.961
25.6M	ResNet50V2	1280×720	1,1	0.988	0.893	0.938	0.967
44.6M	ResNet101V2	896×896	1,1	0.975	0.871	0.920	0.958
60.4M	**ResNet152V2**	**768×768**	**400,1**	**0.994**	**0.904**	**0.946**	**0.972**
138.4M	VGG16	896×896	1,1	0.987	0.880	0.930	0.964
143.7M	VGG19	512×512	1,1	0.993	0.820	0.898	0.948

Table 2: Test set metrics for EfficientNetB2.

Network	Test			
	Prec.	Recall	F1-score	Acc.
EfficientNetB2	0.989	0.888	0.936	0.970

age flagged as corrupted will be manually inspected by a human capable of distinguishing between various corruption subtypes. Hence, the main objective of the ConvNet for this application is to detect the presence of corruption, regardless of which subtype it belongs to. Secondly, training a multi-label classifier requires a large number of training samples for each class. However, the distribution of the collected images indicates a significant imbalance, with some corruption subcategories comprising of much fewer samples than others (Fig. 2).

4.1 Two-stage Training

Our method for detecting corrupted video game frames relies on transfer learning and finetuning. We propose to train the DCNN in two stages. In the first stage, we initialize the network with ImageNet pre-trained weights and train it for few epochs with the entire network's weights frozen, except the classifier layer weights. In the second stage, we restart the training with the trained weights from the first stage, while bringing the following three changes. Firstly, we train the network end-to-end allowing the training process to alter the weights of all the layers. Secondly, we reduce the learning rate to a much lower value compared to the first stage. Finally, we significantly increase the number of training epochs. This process is depicted in the block diagram shown in Fig. 6.

4.2 Network Selection and Hyperparameter Tuning

Since the introduction of AlexNet (A. Krizhevsky and Hinton, 2012), DCNNs have been employed in innumerable computer vision applications. Further, many successful improvements of AlexNet also have been proposed in the literature. In this work, we explored several of these well-known architectures including VGGNet (Simonyan and Zisserman, 2014), ResNet (K. He and Sun, 2016), DenseNet (G. Huang and Weinberger, 2017), MobileNet V2 (M. Sandler and Chen, 2018), and EfficientNet (Tan and Le, 2019).

While the choice of the architecture was one of the important hyperparameters, we trained a wide variety of models by varying:

- number of layers in the classifier head;
- number of units in each layer;
- input image size;
- class weights;
- learning rate;
- batch size;
- *alpha* parameter controlling the width of MobileNet V2.

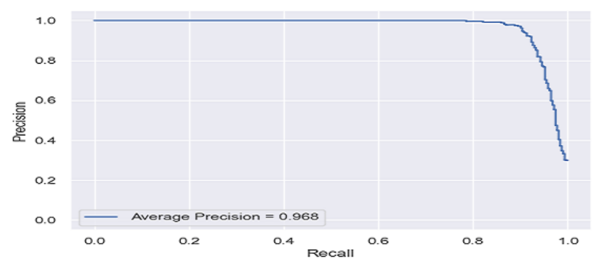
4.3 Model Selection

The process of testing GPUs involves running several game titles on different hardware and driver configurations repeatedly. This process generates a large number of images which are evaluated by the deep learning model and the images flagged as corrupted will be manually verified prior to further defect triaging. In order to reduce this manual effort in triaging, we prioritized reducing the number of false positives (i.e., golden images predicted as corrupted). Hence, we focused on improving the *precision* (1) of the model over the *recall* (2).

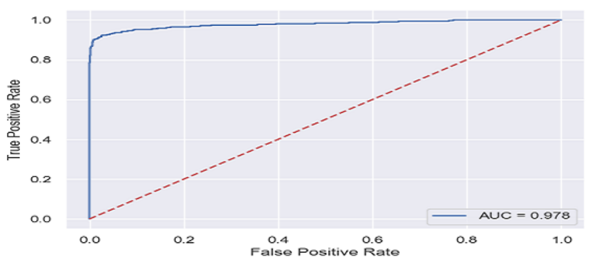
$$Precision = \frac{TP}{TP+FP} \quad (1)$$

$$Recall = \frac{TP}{TP+FN} \quad (2)$$

We used the Adam optimizer (Kingma and Ba, 2014) and minimized the sigmoid cross-entropy loss



(a) Precision-recall curve.



(b) Receiver Operating Characteristic (ROC) curve.

Figure 7: Test set metrics for EfficientNetB2.

to train the model. During the first phase of training, the model was optimized for the validation F1-score. In the second phase, the optimizer maximized the *precision* to a predefined threshold (0.99) and then picked a model with optimal *recall* such that the *precision* remained higher than the aforementioned threshold. The learning rate was lowered by a factor of 100 during this step. During both phases of the training, to achieve better generalization, data augmentation was carried out on the dataset. However, the augmentation was restricted to flipping images horizontally and vertically. Other augmentation techniques, such as image translation and shear resulted in images resembling corrupted images and hence were excluded.

5 RESULTS

We conducted a wide range of experiments and summarized our findings for high-performing models in Table 1. Having focused on maintaining precision above a threshold during training, we notice the precision on the validation set is significantly higher than the recall.

Two models, EfficientNetB2 and ResNet152V2, outperformed other architectures on the validation set. Having 6× fewer parameters than ResNet152V2 (see Table 1), EfficientNetB2 is lighter on memory and faster at inference time. Hence, we find that EfficientNetB2 is better suited to the task at hand. For the chosen EfficientNetB2 model, hyperparameter values were as follows: Image size of 768 × 768, equal class weights, learning rate of 0.001, batch size of 8, and the classifier head comprised a single sigmoid-activated unit.

We further evaluated EfficientNetB2 on the hold-out test set comprising of 300 corrupted images and 900 golden images. From Table 2, we observe that the model has successfully generalized to unseen data, achieving a precision of 0.989 and recall of 0.888. From Fig. 7a and Fig. 7b, the model also achieved an Average Precision of 0.968, with the area under its Receiver Operating Characteristic (ROC) curve attaining 0.978.

The main benefit of our two-stage training is that it provides significantly higher accuracy compared to single-stage training. In our experiments, with single-stage training we noticed that model accuracy plateaus beyond certain epochs. Training in two stages instead yielded more accurate models. The validation set F1-score progression for a typical training session for single-stage and two-stage training is shown in Fig. 8.

Fig. 9 and 10 show the breakdown of correct and incorrect predictions made by the model on corrupted images in the test set. The model exhibits excellent prediction capabilities on most of the corruption categories, except for Discoloration and 'Other Corruptions'. We attribute the poor performance on images with Discoloration to the relatively small num-

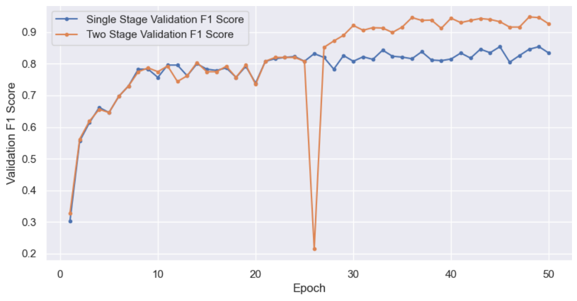


Figure 8: Validation F1-Score for single-stage and two-stage training.

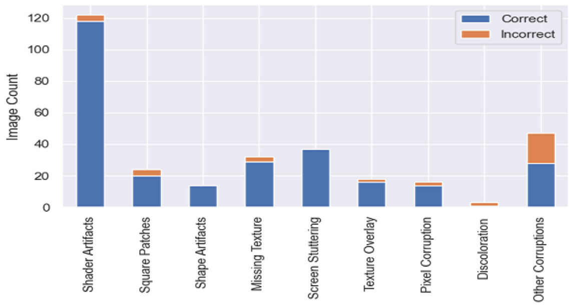


Figure 9: Distribution of correct and incorrect predictions across corruption types in the test set.

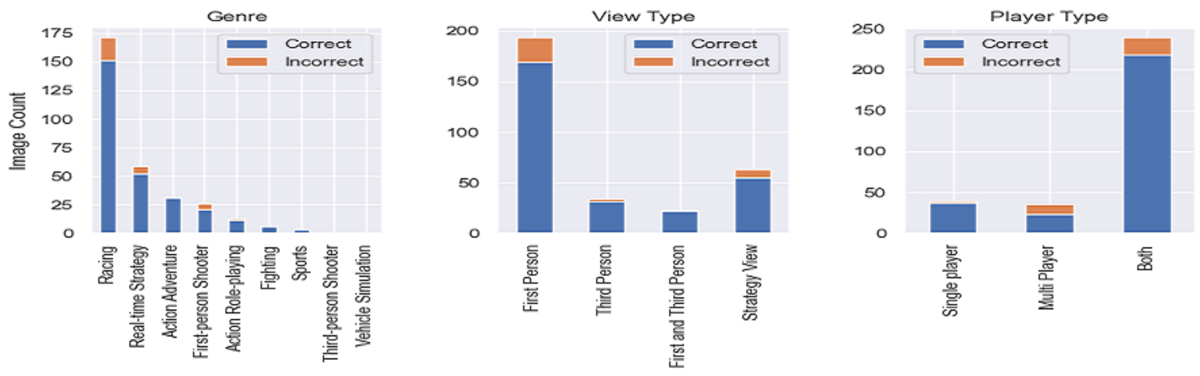
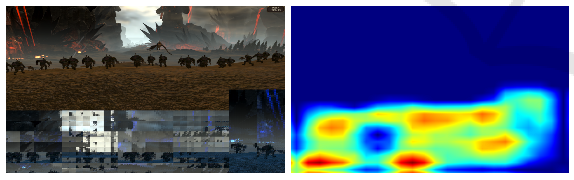


Figure 10: Distribution of correct and incorrect predictions across game title categories in the test set.

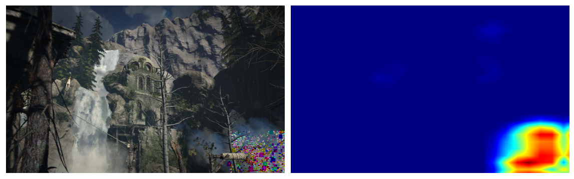
ber of images with discoloration in the training set (see Fig. 2). A similar argument can be applied to the 'Other Corruptions' category as it consists of images each displaying different subtypes of corruption, with each subtype appearing less often. On the other hand, game title category does not appear to have much impact on the model performance (see Fig. 10).

Having established that the model performed better for categories comprising of larger training samples, we conjecture that, given sufficient training data for each corruption type, a DCNN can achieve very high performance in detecting *real* display corruption.

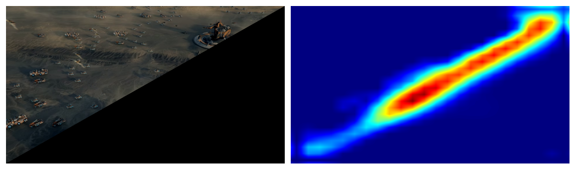
Note that, in (C. Ling and Gisslén, 2020), authors found that ShuffleNet V2 (N. Ma and Sun, 2018) provides optimal performance on their synthetic dataset. To the best of our knowledge, a reliable implementation of (C. Ling and Gisslén, 2020) is not publicly



(a) Screen stuttering. (b) Grad-CAM heatmap.



(c) Pixel corruption. (d) Grad-CAM heatmap.



(e) Screen tearing. (f) Grad-CAM heatmap.

Figure 11: Corrupted images with corresponding Grad-CAM heatmaps.

available. Hence, we have not included a comparison between ShuffleNet V2 and EfficientNetB2. Further, the authors also state "MobileNet V2 (Sandler et al. 2018) achieved similar performance as ShuffleNet V2, although with slower training times". Additionally, the results in Table 1 indicate that EfficientNetB2 outperforms MobileNet V2 in terms of both F1-score and accuracy. Hence, we conclude that EfficientNetB2 is better suited for corruption detection in images.

5.1 Interpretability

Interpretability plays a major role in the adoption of machine learning solutions to provide actionable insights in model predictions. In our work, we leverage Grad-CAM (Gradient-weighted Class Activation Mapping) proposed by (R. Selvaraju and D.Batra, 2017) to produce a heatmap-like visualization that highlights regions contributing to the prediction.

Fig. 11 shows heatmaps generated via Grad-CAM on EfficientNetB2. We observe that the model correctly gives high importance to regions presenting either (i) corruption (Fig. 11b and Fig. 11d) or (ii) a transition from a non-corrupted to a corrupted area (Fig. 11f).

A DCNN otherwise acting as a black-box, we argue that providing this level of interpretability increases the confidence of game testers in leveraging deep-learning approaches to automate the detection of display corruption. Further, Grad-CAM can be used as a helpful tool to determine the reason for model's wrong predictions and further improve the model.

6 CONCLUSION

We have presented a deep learning approach to automatically detect *real* video game display corruption and trained an EfficientNet achieving precision of 0.989 and recall of 0.888 on this task.

While previous approaches focused on *artificially generated* corruption, our study presents a workflow for collecting *real* display corruption of various types from a vast range of video games while using Structural Similarity Index Measure (SSIM) to ensure the collected images are visually diverse. We presented a two-stage training procedure and demonstrated its effectiveness through a variety of neural networks achieving high validation performance. We presented the distribution of correct and incorrect predictions on corrupted images for our top-performing model and argued that, with sufficient training data per corruption type, a DCNN can be successfully trained to detect a wide variety of graphical malfunctions. Finally, we showed that Grad-CAM can be leveraged to provide interpretability in our neural net's predictions.

7 FUTURE WORK

The main shortcoming of our pipeline is the gameplay being highly scripted throughout the game tests. In the future, we plan on extending our method to provide a gameplay testing experience close to human behavior. This allows for more in-game exploration, thus providing more visually varied gameplay scenarios and a potentially diverse source of corrupted images.

Moreover, our current method detects corruption in individual images without accounting for adjacent game frames. Incorporating video understanding with Long-term Recurrent Convolutional Networks (J. Donahue and Darrell, 2017) could provide valuable insights on the game context and potentially improve the overall performance.

In this study, we treated the task at hand as a binary classification problem. Given sufficient data per corruption category, a DCNN could be trained as a multi-label classifier to effectively detect each corruption subtype separately. In GPU testing workflow, this could further minimize the amount of manual triage required upon the detection of visual corruption.

REFERENCES

A. Krizhevsky, I. S. and Hinton, G. (2012). Imagenet classification with deep convolutional neural network. In *NIPS*.

A. Nantes, R. B. and Maire, F. (2008). A framework for the semi-automatic testing of video games. In *Proceedings of the Fourth AAAI Conference on AI and Interactive Digital Entertainment*, page 197–202.

A. Nantes, R. B. and Maire, F. (2013). Neural network based detection of virtual environment anomalies. In *Neural Computing and Applications*, page 1711–1728.

C. Ling, K. T. and Gisslén, L. (2020). Using deep convolutional neural networks to detect rendered glitches in video games. In *Proceedings of the AAAI Conference on AI and Interactive Digital Entertainment*, pages 66–73.

E. McLaughlin, N. C. and Narasimhan, S. (2019). Combining deep learning and robotics for automated concrete delamination assessment. In *Proceedings of the 36th ISARC*, pages 485–492.

G. Huang, Z. Liu, L. V. D. M. and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2261–2269.

J. Donahue, L. A. Hendricks, S. G. M. R. S. V. K. S. and Darrell, T. (2017). Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 677–691.

K. He, X. Zhang, S. R. and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *arXiv:1412.6980*.

M. Sandler, A. Howard, M. Z. A. Z. and Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 4510–4520.

N. Ma, X. Zhang, H. Z. and Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision*, page 116–131.

P. Davarmanesh, K. Jiang, T. O. A. V. S. I. M. K. S. J. and Malaya, N. (2020). Automating artifact detection in video games. arXiv:2011.15103.

R. Selvaraju, M. Cogswell, A. D. R. V. D. P. and D.Batra (2017). Grad-cam: Visual explanations from deep networks via gradientbased localization. In *IEEE International Conference on Computer Vision*, pages 618–626.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *arXiv:1409.1556*.

Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6105–6114.

Y. Chen, F. Yang, E. S. and Ho, C. (2019). Automatic defect detection system based on deep convolutional neural networks. In *International Conference on Engineering, Science, and Industrial Applications*, pages 1–4.

Z. Wang, A. Bovik, H. S. and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. In *IEEE Transactions on Image Processing*, pages 600–612.