ABSTRACT
Systematic error, which is not determined by chance, often refers to the inaccuracy (invoking either the observation or measurement process) inherent to a system. In this paper, we exhibit some long-neglected but frequent-happening adversarial examples caused by systematic error. More specifically, we find the trained neural network classifier can be fooled by inconsistent implementations of image decoding and resize. This tiny difference between these implementations often causes an accuracy drop from training to deployment. To benchmark these real-world adversarial examples, we propose ImageNet-S dataset, which enables researchers to measure a classifier’s robustness to systematic error. For example, we find a normal ResNet-50 trained on ImageNet can have 1%–5% accuracy difference due to the systematic error. Together our evaluation and dataset may aid future work toward real-world robustness and practical generalization.

1 INTRODUCTION
Recently deep learning has shown remarkable success in image recognition [14, 17, 25], speech recognition [13, 19] and natural language processing [4, 8]. However, their security and robustness are greatly challenged by the delicately designed adversarial attacks [6, 12, 20]. The adversarial examples usually fool the neural network without significant visual differences, demonstrating the vulnerability of deep neural networks. This phenomenon attracts broad research interests for different kinds of attacks and defenses.

Currently, the mainstream attack noises can be categorized into adversarial noises and natural noises. The adversarial noises need to be generated according to the model’s specific information such as gradients calculated in backpropagation. Thus, they often achieve satisfactory attack success rates but are rare in real-world scenarios due to the model dependency. In the contrast, there exist natural noises such as blur or corruption [15, 16] that are agnostic to model and frequently encountered in the natural scenario. Although they have less damage on model robustness compared with adversarial noises, it is impossible to overlook them because of the border range of influence in nature. Therefore, evaluating the robustness of both types of noises is necessary. Some existing defense work [18] surprisingly finds that deep neural networks optimized by adversarial training might be sensitive to natural noises. This indicates that we need diverse noises for a comprehensive understanding of the model’s robustness.

However, the commonly adopted noises are not diverse enough. Besides the man-made noise and natural noise, we point out a new kind of noise-induced by the inconsistent resize or decoder implementation widely existing in the inference system of deep learning model. To be specific, we call it systematic noise since they are not corresponded to the model or the input image, but is only related to the inference software and hardware system. Once the software or hardware-software changes, the noise will occur. For example, we always utilize the Pillow package for image decoding and resizing in the PyTorch [21] training system. But when it is deployed on the edge device, it may not support Pillow and we have to seek an alternative for the practical deployment. Sometimes due to the hardware and software’s restriction, the alternative has no way to keep alignment with the origin Pillow implementation, leading to an inevitable systematic noise. The maximum perturbation caused by this systematic noise is only a 1-pixel value but may introduce an obvious accuracy drop (about 1% ~ 5%) in our experiment. Different from the extensively studied adversarial noises and natural noises, this kind of systematic noise is long-term neglected and needs to be included in the robustness evaluation.

In order to attract attention to systematic noise, we summarize the major systematic error sources in the deep learning system and propose a corresponding ImageNet-S dataset, providing provides 3 kinds of decoder noises and 6 kinds of resize noises. It also supports the flexible custom extension by users. We conduct a pioneering benchmark for some prevalent neural architectures on the systematic robustness and find some useful insights: (1) The effect of using different decoders is small while it may have a big impact when using different resize methods; (2) Usually, models in the same architecture will have better robustness of resize with FLOPs (Floating Point Operations) of it get larger, but it doesn’t work for the robustness of decoder; (3) Adversarial methods can increase the robustness of resize method, but we need to find a balance between robustness and clean accuracy. Along with the
ImageNet-S benchmark dataset, we also propose a simple yet effective mixed training strategy to improve systematic robustness. Our contributions can be summarized as below:

- We firstly indicate a kind of long-neglected but frequent-happening systematic noise caused by the inconsistent deploy software and hardware environment. It may harm the model’s performance in practical application.
- To comprehensively evaluate the effect of systematic noise, we construct an ImageNet-S dataset, which consists of 6 different kinds of resize functions and 3 different kinds of decoder functions. We benchmark the robustness of the prevalent ResNet, RegNetX, MobileNet-V2 networks, and “robust” networks with adversarial training. Even though the systematic noises are slight, they may introduce an up to 2% accuracy drop in accuracy.
- To improve the stability when facing systematic noise, we propose a simple yet effective mix training strategy. It can be easily applied in the training routine without much effort but enable the network to enjoy a consistent performance under different deployment systems.

2 RELATED WORK

Adversarial Examples. The adversarial examples were first introduced by [26]. From then on, various adversarial attacks were proposed, such as FGSM [12], PGD [20], C&W [5] and some black-box attack methods [2]. Driven by the emergence of adversarial noises, corresponding defense techniques also arose, including adversarial training [10, 24, 28], data augmentation [11] and regularization [9]. The adversarial examples are always dependent on the model to attack, especially for the block-box attack. Thus they suffer a low transferability and rarely occur in the practical scenario.

Natural Noises. Besides the adversarial examples, the community also realizes the importance of natural noises that are widely existing in the real world. Some representative datasets are constructed to simulate the natural noise, such as ImageNet-P, ImageNet-C [15], and ImageNet-A, ImageNet-O [16]. These noises are model-agnostic and may cause perceptible perturbation. Natural noises such as Snow noise and Frost noise can measure the robustness of a model in the wild.

3 ADVERSARIAL CORRUPTIONS FROM SYSTEMATIC ERROR

In this section, we outline the different system implementations for image decoding and resize. Then, we introduce our ImageNet-S dataset to benchmark the robustness from the different sources of decoding and resizing.

3.1 Notations

Let us denote a RGB image tensor as \( X \in [0, 1]^{w \times h \times 3} \) where \( w \) and \( h \) are the image width and height, respectively. RGB images have 3 channels standing for red, green, and blue colors. However, in computer systems, the images are not stored in this format. Typically there is a raw file (e.g. JPEG) which encodes the tensor \( X \) to \( V \). In the case of JPEG encoding, \( X \) will be processed by color space conversion, discrete cosine transform (DCT), quantization, and Huffman encoding to \( V \). For image decoding, we should reconstruct the image tensor \( X \) from \( V \). As for image resize, the operation can be represented by \( \text{Resize}(X) : [0, 1]^{w' \times h' \times 3} \rightarrow [0, 1]^{w'' \times h'' \times 3} \), where \( w' \) and \( h' \) are the resized width and height, respectively.

Generally, in neural network inference, the image is processed by decoding and resize \((V \rightarrow X \rightarrow X')\). With the different implementations of these two operations, output \( X' \) can have different values, which will be discussed in the next two sections.

3.2 Image Decoding

Image decoding refers to the process of translating the raw file (e.g. JPEG) back to an RGB or YUV 3 channel image tensor \((V \rightarrow X)\). Image decoding can be divided into several steps: (1) read the raw file from disk as bytes, (2) variable-length decoding (3) zigzag scan, (3) dequantization, (4) inverse discrete cosine transform (iDCT), (5) color conversion and reorder, shown as Figure 1. The fourth step iDCT occupies the majority of the computational cost in the decoding process, which is given by:

\[
f[m, n] = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} \alpha(k)\alpha(l) F(k, l) \cos \left( \frac{(2m+1)\pi k}{2N} \right) \cos \left( \frac{(2n+1)\pi l}{2N} \right)
\]

where,

\[
\alpha(k) = \begin{cases} \sqrt{\frac{2}{N}} & \text{if } k = 0 \\ \frac{1}{\sqrt{2}} & \text{otherwise} \end{cases}
\]

In theory, the principle of this process is fixed, but we find decoding one image file in different third-party libraries (e.g. OpenCV [3], Pillow [29], FFMpeg [27]) will output different RGB tensors. This is because they have a unique self-implemented tool to decode the images, especially the iDCT step. We find some libraries prefer to use fast inverse discrete cosine transform (Fast iDCT) instead of the vanilla one, which sacrifices the image quality for the decoding speed. Furthermore, there would be some minor errors in the implementation, such as (ADD EXAMPLES.) These minor errors can cause a shift in the pixel values of the final RGB tensor. Eventually, a single difference in pixel value could fool the neural network to change its prediction. Most of the time, when changing the decoding tools used in training to another one in inference, we observe a drop in accuracy.

\[\text{Sometimes the center crop operation is also utilized. It has a similar impact with the resize operation, therefore we omit its discussion in this paper.}\]
Image resize is adopted to scale the image resolution to a different size, either up (increase resolution) or down (decrease resolution). In resize operation, one needs to predict the pixel value at an unseen position. This is often performed by different interpolation algorithms. We summarize several common algorithms, like nearest neighbor, bilinear, bicubic, etc.

### 3.3.1 Nearest-neighbor Interpolation
This algorithm, also known as Proximal Interpolation, directly selects the value of its nearest known pixel. This mode does not consider the relative magnitude of its neighbor pixels. Therefore, it can preserve sharp details of known pixel. This mode does not consider the relative magnitude of its neighbor pixels. Therefore, it can preserve sharp details.

### 3.3.2 Bilinear Interpolation
Bilinear interpolation is a simple extension from linear interpolation by operating on the 2D variables. Unlike Nearest-neighbor algorithm, this mode consider the value of its neighbor and has a rather continuous interpolation effect. Mathematically, for 2D images where spatial coordinate is represented by $(x, y)$, the nearest neighbor interpolation will find the existing pixel with lowest Euclidean distance i.e., $X[\arg\min_{x,y}((x-x')^2 + (y-y')^2)]$.

### 3.3.3 Bicubic Interpolation
Bicubic interpolation is a simple extension from linear interpolation by operating on the 2D variables. Unlike Nearest-neighbor algorithm, this mode consider the value of its neighbor and has a rather continuous interpolation effect. Mathematically, for 2D images where spatial coordinate is represented by $(x, y)$, the nearest neighbor interpolation will find the existing pixel with lowest Euclidean distance i.e., $X[\arg\min_{x,y}((x-x')^2 + (y-y')^2)]$.

To evaluate the influence of different image decoders, in Section 4, we will compare with three different decoder tools including OpenCV, Pillow and FFmpeg which are widely used in the field of computer vision.

### 3.4 Image Resize Library
We summarize two most commonly used image resize tools in the field of computer vision. Table 1 shows the image resize tools and their corresponding resize method.

| Library | Nearest | Bilinear | Cubic | Lanczos | Area | Box | Hamming |
|---------|---------|----------|-------|---------|------|-----|---------|
| OpenCV  | ✓       | ✓        | ✓     | ✓       | ✓    | ✗   | ✗       |
| Pillow  | ✓       | ✓        | ✗     | ✗       | ✗    | ✗   | ✓       |

where,

\[
f(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij}x^iy^j
\]  

(3)

To find the total 16 coefficients $a_{ij}, i, j \in \{0, 1, 2, 3\}$, we need to solve a system of linear equations $Ax = b$. Due to the complexity of this algorithm, we refer the readers to this link\(^2\) for more details. Bicubic interpolation yields better performance than the previous two algorithms, however, it also needs huge time to solve the linear equations to find optimal interpolated values.

### 3.5 Visualization of Resize Algorithms
Apart from our mentioned algorithms, many more exists. Here we do not introduce them one by one. To provide an intuitive understanding of the difference, we visualize the results of various resize methods. From this figure, we can see that the gap between different tools using the same method is not small. In the real-world images, different resize tools applying on the same image may not lead to an obvious distance between results, but the gap between them can result in a destabilization of model robustness.

\(^2\)https://www.ece.mcmaster.ca/~xwu/interp_1.pdf
which—as expected—reduces half information. Another major difference comes from the third-party resize libraries. As shown in Fig. 2b and 2g, Pillow and OpenCV also share opposed directions of gradients in color. In practice, the effect of resize operation on real-world images is not as obvious as our example here, however, it is enough to cause wrong predictions for a trained neural network.

4 THE IMAGENET-S ROBUSTNESS BENCHMARK

4.1 ImageNet-S Design

To benchmark the robustness of this unique systematic noise, we aim to build an ImageNet-S dataset. This dataset consists of 3 commonly used decoder types and 6 commonly used resize types. For decoder, we include the implementation from Pillow [29], OpenCV [3] and FFmpeg [27]. For resize operation, we include nearest, cubic and bilinear interpolation modes from both OpenCV and Pillow tools.

To verify the effect of decoding and resize separately, we must fix the decoding method when evaluating the resize and vice versa. In this paper, we set Pillow bilinear mode as the default resize method when testing the decoding robustness. This setting is also the default training setting in PyTorch [21] official code example. Similarly, we set Pillow as the default image decoding tool (same as PyTorch).

We provide not only the validation set of ImageNet with different decoding and resize methods, but also the training set to expose how different decoding and resize methods influence the process of training on a specific model. To enable reproducibility, we save each image file after decoding and resize it as a $3 \times width \times height$ matrix in a.npy file instead of JPEG. According to the commonly used transform on ImageNet [7] train set and test set, we provided two kinds of preprocessing for training and testing respectively. For training, we provide a matrix of an image after random resize crop to $3 \times 224 \times 224$. While for testing, we provide a matrix of an image after the process of resizing to $3 \times 256 \times 256$ then applied a center crop to $3 \times 224 \times 224$.

4.2 Dataset Generator

We provide the generator of ImageNet-S on GitHub¹, so that the community can generate system noise dataset and test the robustness of their own models. The basic usage of it is as follows:

```python
from imagenet_s_gen import ImageTransfer
ImageTransfer(root_dir='~/images/val',
meta_file='~/images/meta/val.txt',
save_dir='~/dataset-decoder-resize',
decoder_type='pill',
transform_type=val,
resize_type='pill-bilinear')```

5 EXPERIMENTS

Our experiments consist of four parts. First, we evaluate the robustness of existing pre-trained ImageNet models. These models all use the standard [Pillow decoder and Pillow bilinear resize] to train. Second, we study whether the commonly adopted $L_2$ or $L_{\infty}$ [5] robust models can defend our ImageNet-S case. Last, we study the training choice of decoding and resize to compare the results on ImageNet-S.

5.1 Network Architecture Experiments

In this section, we test pre-trained neural network models from different architecture to find out the relationship between architecture and systematic noise robustness. Additionally, we aim to find whether the complexity (e.g. FLOPs of the model) of a model can improve the robustness or not. We select architecture from ResNet [14] (including 18, 34, 50, 101 layers models), MobileNet-V2 [22] (including width multiplier 0.5, 0.75, 1 and 1.4 $\times$ models) and RegNetX [22] (including FLOPs 200, 400, 600, 800, 1600M models). We summarize the results in Table 2 and Figure 3. As can be seen, we find several results: (1) Decoding corruptions are much less obvious than resize corruptions. In the decoding test, we find OpenCV, Pillow and FFmpeg have similar accuracy, with only < 0.01% accuracy difference. NVIDIA DALI, however, has a moderately different decoding process than these 3 libraries, which generally causes 1%

---

¹https://github.com/TheGreatCold/Imagenet-S
where robustness to our system noise. We download the pre-trained robust
the ground truth label
and robustness. Given a model
robustness of different resize
operation comes from the nearest
erence. For resize method, we observe a relatively large
https://github.com/microsoft/robust-models-transfer

![Figure 3: Architecture robustness of systematic noise.](image)

accuracy difference. For resize method, we observe a relatively large
gap. The biggest gap for resize operation comes from the nearest
neighbor and bilinear interpolation, which generally costs 1 ~ 2%
accuracy gap. (2) Different architecture has different robustness,
even with similar FLOPs. (3) For models in the same architecture,
bigger FLOPs usually result in better robustness in different resize
methods. (4) There is no clear relationship between FLOPs and the
robustness of different decoders.

### 5.2 Adversarial Experiments

Adversarial training is a defensive technique that gives model de-
ceptive images during training, thus improving its generalization
and robustness. Given a model \( f_0 \) and an input image batch \( X \) with
the ground truth label \( Y \), an adversarial example of \( X_{\text{adv}} \) satisfies
\[
\begin{align*}
\hat{f}_a(X_{\text{adv}}) &\neq Y \\
\|X - X_{\text{adv}}\|_p &\leq \epsilon,
\end{align*}
\]
where \( \| \cdot \|_p \) is a distance metric. Commonly, \( \| \cdot \| \) is measured by
the \( l_p \)-norm (\( p \in [1,2,\infty] \)).

Adversarial training is one of the most effective approaches to
help deep learning models defend against adversarial examples.
Thus, it is reasonable to use some adversarial models to test
their robustness to our system noise. We download the pre-trained robust
trained ResNet-50 [14] from this GitHub repository. 4 The ResNet-50

![Figure 4: Architecture robustness of systematic noise.](image)

Table 3: Top-1 accuracy of various robust ResNet-50 on ImageNet-S.

| Robust \( \epsilon \) | Decoder | Resize |
|-------------------|---------|--------|
| Type              | DALI    | OpenC-V | Pillow | Ffmpeg | Mean ± Std | Pillow-N | Pillow-L | Pillow-C | OpenC-N | OpenC-L | OpenC-C | Mean ± Std |
| None 0            | 75.058  | 75.386  | 75.397  | 75.397  | 75.397 ± 1.36E-02 | 70.626  | 75.386  | 75.200  | 71.578  | 74.442  | 73.312  | 63.067 ± 1.74E+00 |
| \( \epsilon \) = 0.1 | 75.058  | 75.386  | 75.397  | 75.397  | 75.397 ± 1.36E-02 | 71.490  | 75.386  | 75.200  | 71.578  | 74.442  | 73.312  | 63.067 ± 1.74E+00 |
| \( \epsilon \) = 0.5 | 75.058  | 75.386  | 75.397  | 75.397  | 75.397 ± 1.36E-02 | 71.490  | 75.386  | 75.200  | 71.578  | 74.442  | 73.312  | 63.067 ± 1.74E+00 |
| \( \epsilon \) = 1.0 | 75.058  | 75.386  | 75.397  | 75.397  | 75.397 ± 1.36E-02 | 71.490  | 75.386  | 75.200  | 71.578  | 74.442  | 73.312  | 63.067 ± 1.74E+00 |

![Figure 3: Architecture robustness of systematic noise.](image)

![Figure 4: Architecture robustness of systematic noise.](image)

is trained with either \( \epsilon_2 \) or \( \epsilon_\infty \) training method. The result is shown
in Table 3. We find several interesting insights: (1) Neither of these
two adversarial training methods improves the image decoder’s
robustness. (2) Both adversarial training methods can improve
the robustness of different resize methods. The robustness increase as
\( \epsilon \) gets larger, but a large \( \epsilon \) would harm the accuracy.

### 5.3 Robustness Enhancements

As aforementioned, we have verified that adversarial training can
enhance resize method robustness, but it cannot help with the ro-
busness on different decoders. This indicates that decoder noise
does not belong to the \( L_p \) norm perturbation. To overcome this
problem, we need other techniques to enhance the systematic
robustness.

#### 5.3.1 Mix Training

To solve this problem, a natural way is to make
the model ‘see’ all kinds of decoders and resize methods during the
training process. Based on this principle, we introduce mix training
method to enhance the model’s robustness on system noise.

The main process of mix training is to select decoder or resize
method randomly instead of just using one kind of method during
the whole process of training. The pseudocode of our algorithm
is shown in algorithm 1.
Table 4: Mix training on resize method.

| Train       | Test          | Pillow-bilinear | Pillow-nearest | Pillow-cubic | OpenCV-nearest | OpenCV-bilinear | OpenCV-cubic | Mean  | Std.  |
|-------------|---------------|-----------------|----------------|--------------|----------------|-----------------|--------------|-------|-------|
| Pillow-bilinear | Pillow-bilinear | 76.572          | 72.168         | 76.512       | 72.090         | 75.346          | 74.072       | 74.460 | 2.02E+00 |
| Pillow-nearest       | Pillow-nearest       | 74.872          | 75.988         | 75.548       | 75.970         | 76.002          | 76.056       | 75.739 | 4.63E-01 |
| Pillow-cubic               | Pillow-cubic               | 76.312          | 72.828         | 76.596       | 72.876         | 75.810          | 74.666       | 74.848 | 1.68E+00 |
| OpenCV-nearest            | OpenCV-nearest            | 74.818          | 76.298         | 75.474       | 76.092         | 76.082          | 76.192       | 75.826 | 5.71E-01 |
| OpenCV-bilinear          | OpenCV-bilinear          | 75.840          | 75.268         | 76.446       | 75.248         | 76.682          | 76.436       | 75.987 | 6.29E-01 |
| OpenCV-cubic              | OpenCV-cubic              | 76.194          | 72.812         | 76.510       | 72.940         | 75.736          | 74.818       | 74.835 | 1.62E+00 |
| mix                    | mix                      | 76.154          | 75.876         | 76.344       | 75.736         | 76.444          | 76.350       | 76.156 | 2.70E-01 |

Algorithm 1: Mixed training for improving robustness on systematic noise.

**Input:** Resize set RS; Decoder set D; Model to train
Set Pillow-bilinear as default Resize;
Set Pillow as default Decoder;
for all \( j = 1, 2, \ldots, T \)-iteration in training do
  if use mix-decoder strategy then
    Randomly sample a Decoder from D;
  if use mix-resize strategy then
    Randomly sample a Resize function from RS;
    Calling API to load the images from ImageNet-S according to the Decoder type and Resize type;
Model Optimization.
return An optimized robust model for systematic noise.

Table 5: Mix training on decoder.

| Train | Test | Pillow | OpenCV | FFmpeg | Mean  | Std.  |
|-------|------|--------|--------|--------|-------|-------|
| Pillow| Pillow-bilinear | 76.430 | 76.426 | 75.310 | 76.055 | 6.45E-01 |
| OpenCV | OpenCV-bilinear | 76.510 | 76.510 | 75.368 | 76.126 | 6.56E-01 |
| FFmpeg| FFmpeg-bilinear  | 75.730 | 75.664 | 76.318 | 75.904 | 3.60E-01 |
| mix   | mix             | 76.53  | 76.524 | 76.414 | 76.489 | 6.53E-02 |

To test the effect of mix training, we set up the following experiment. We use ResNet50 as the base model of this experiment. To comprehensively demonstrate the training effect, we train single decoding and resize as well as our mix training models. We set the default decoder as Pillow and the default resize method as Pillow bilinear when conducting ablation studies on resize method or decoder, respectively. Then we use top-1 accuracy as well as their mean and standard deviation as assessments.

The results of this experiment are shown in Table 5 and Table 4. From these tables, we can know that: (1) The model has a better performance (usually the best) when we train and test using the same decoder and resize method. (2) There is a big gap when we train and test the same model using different methods. For example, the accuracy of ResNet50 trained by Pillow bilinear method is about 2.4% lower than that trained by OpenCV bilinear on the test set which is resized by OpenCV cubic. While, on the whole, the gap between different decoders is smaller than that between different resize methods. (3) Mix training can improve the robustness of a model on system noise greatly without hurting the clean accuracy.

The Std. using mix training drop from 0.36 to 0.0653 on decoder experiment, and drop from 0.463 to 0.270 on resize experiment. Meanwhile, it can maintain the model’s accuracy at about 76%. As a contrast, the same ResNet50 model using L∞ – Robust adversarial training drop the Std. from 1.07 to 0.420 by paying 19.2% drop of clean accuracy.

6 CONCLUSION

In this paper, we introduced what is to our knowledge the first dataset, ImageNet-S, for real-world robustness from systematic noise. We focused on a problem that exists in the process of practical application, but it is easy to be ignored in practice. We finished lots of experiments that do show that the difference between different decoders and resize methods can influence on model’s accuracy. Therefore, we used mean and standard deviation which are used to measure the robustness of systematic noise. With this matrix, we did experiments and found the relation between architecture, adversarial, and the robustness of a model. What’s more, we found a way called mix training to enhance the system noise robustness without hurting the clean accuracy.
REFERENCES

[1] 2017. Nearest-neighbor interpolation. https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation

[2] Arjun Nitin Bhaogoi, Warren He, Bo Li, and Dawei Song. 2018. Practical Black-box Attacks on Deep Neural Networks using Efficient Query Mechanisms. In Proceedings of the European Conference on Computer Vision (ECCV).

[3] G. Bradski. 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000).

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Sussbاه, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranay Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse. Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp). IEEE: 39–57.

[6] Francesco Croce and Matthias Hein. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In IJCAI.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2009).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805 (2018).

[9] Chaohao Fu, Hongbin Chen, Na Ruan, and Weijia Jia. 2020. Label Smoothing and Adversarial Robustness. arXiv preprint arXiv:2009.08523 (2020).

[10] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Franцois Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. The journal of machine learning research 17, 1 (2016), 2096–2030.

[11] Chengyang Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. 2021. MaxUp: Light-weight Adversarial Training With Data Augmentation Improves Neural Network Training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2474–2483.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014).

[13] Wei Han, Zhendong Zhang, Yu Zhang, Jiuhui Yu, Chung-Cheng Chiu, James Qin, Anmol Gulati, Ruoming Pang, and Yonghui Wu. 2020. Contextnet: Improving Adversarial Robustness. arXiv preprint arXiv:2009.08523 (2020).

[14] Alexander V. Engel, Ilya Sutskever, and Quoc V. Le. 2020. The journal of machine learning research 21, 1 (2020), 3846–3894.

[15] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp). IEEE: 39–57.

[16] Francesco Croce and Matthias Hein. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In IJCAI.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2009).

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805 (2018).

[19] Chaohao Fu, Hongbin Chen, Na Ruan, and Weijia Jia. 2020. Label Smoothing and Adversarial Robustness. arXiv preprint arXiv:2009.08523 (2020).

[20] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Francois Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. The journal of machine learning research 17, 1 (2016), 2096–2030.

[21] Chengyang Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. 2021. MaxUp: Light-weight Adversarial Training With Data Augmentation Improves Neural Network Training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2474–2483.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014).

[23] Wei Han, Zhendong Zhang, Yu Zhang, Jiuhui Yu, Chung-Cheng Chiu, James Qin, Anmol Gulati, Ruoming Pang, and Yonghui Wu. 2020. Contextnet: Improving adversarial robustness for automatic speech recognition with global context. arXiv preprint arXiv:2005.03191 (2020).

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[25] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. Proceedings of the International Conference on Learning Representations (2019).

[26] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. 2021. Natural Adversarial Examples. CVPR (2021).

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classiﬁcation with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[28] Alfred Lauwros, Alice Caplier, and Matthieu Ospici. 2019. Are adversarial robustness and common perturbation robustness independent attributes?. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops. 0–0.

[29] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gaddde. 2019. Jasper: An end-to-end convolutional neural acoustic model. arXiv preprint arXiv:1904.05288 (2019).

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. Proceedings of the International Conference on Learning Representations (2018).

[31] Adam Pazie. Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[32] Ilya Radozvovanov, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. 2020. Designing Network Design Spaces. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[33] Mark Sandier, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chi Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[34] Ali Shahri, Mahyar Naihi, Amin Ghiy, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free!. arXiv preprint arXiv:1904.12843 (2019).

[35] Karen Simonson and Andrew Zisserman. 2014. Very deep-convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[36] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013).

[37] Suramya Tomar. 2006. Converting video formats with FFmpeg. Linux Journal 2006, 146 (2006), 10.

[38] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1707.07204 (2017).

[39] P. Umesh. 2012. Image Processing in Python. CSI Communications 25 (2012).