The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Testing the Channels of Convolutional Neural Networks

Kang Choi, Donghyun Son, Younghoon Kim, Jiwon Seo*

Hanyang University
{kangchoi21, sondh, nogaussian, seojiwon}@hanyang.ac.kr

Abstract

Neural networks have complex structures, and thus it is hard to understand their inner workings and ensure correctness. To understand and debug convolutional neural networks (CNNs) we propose techniques for testing the channels of CNNs. We design FtGAN, an extension to GAN, that can generate test data with varying the intensity (i.e., sum of the neurons) of a channel of a target CNN. We also proposed a channel selection algorithm to find representative channels for testing. To efficiently inspect the target CNN's inference computations, we define *unexpectedness* score, which estimates how similar the inference computation of the test data is to that of the training data. We evaluated FtGAN with five public datasets and showed that our techniques successfully identify defective channels in five different CNN models.

1 Introduction

Deep neural networks (DNNs) are used in many application domains. As more DNN models are deployed, it becomes more important to ensure that they function correctly and reliably. However, due to their complexity, it is difficult to analytically verify the correctness of DNNs (Katz et al. 2017).

To practically test neural networks, test input generation has been studied. Most well-known is adversarial example generation that finds minimal perturbation to input to deceive a target neural network (Goodfellow, Shlens, and Szegedy 2014). The perturbed input, i.e. adversarial examples, may be used to assess the robustness of DNNs for adversarial attack. While the techniques are effective in finding adversarial examples, they focus on low-level neuron operations and do not examine, for example, the interactions of the feature maps in convolutional neural networks (CNNs).

Testing is a widely studied topic in software engineering. Many techniques have been developed to test the correctness of software systems. For example, test input generation explores the ranges of certain variables such as array indices and find the inputs to induce buffer overflow (Haller et al. 2013; Xie, Chou, and Engler 2003). Also, a common technique in software testing is to check for inconsistencies in parts of larger systems (Engler et al. 2001); Researchers discovered that implicit invariants exist for certain functions or modules, and their violations often result in invalid system states (Ernst et al. 2007; Martin, Livshits, and Lam 2005).

In this paper, we employ these testing strategies in software engineering for neural networks. In particular, we propose *channel-wise* testing of CNNs, which is a test generation technique for convolutional neural networks. The channels in CNNs are, to some extent, similar to the functions and modules in software; they both are logical units of larger systems. As with unit testing in software engineering, testing individual channels in a modular manner helps to debug and understand neural networks. We generate test data to separately examine the behavior of individual channels and check for their (in)consistencies. With this consistency information, we rank the test data and report (potentially) defect-inducing inputs and the corresponding channels.

With channel-wise testing, we aim to find defects in CNNs (i.e., unintended inference outcome) that are caused by channels having deviant behavior in high or low activation levels. To this end, we designed FtGAN, an extension to GAN that tests the channels of target CNNs. FtGAN is trained, in an unsupervised manner, to find the latent variables that are correlated with the CNNs' channels. Using FtGAN, we gradually vary the latent variables in the generated test data, which then affects the correlated channels in the tested CNNs. To identify inconsistent behavior of the channels with generated test data (compared to that with training data), we define *unexpectedness score* that compares the inference computations and estimates their inconsistency.

This paper proposes channel-wise testing of CNNs. Our contributions are threefold: 1) we designed FtGAN to test selected channels of CNNs (Section 3), 2) we developed channel selection algorithm to find representative channels for testing (Section 4), and 3) we identify defect-inducing test data with unexpectedness score, which uses channel correlations to estimate inconsistencies in the inference computation (Section 5). Our evaluation shows that FtGAN helps to find real and synthetic defects in neural networks.

2 Related Work and Motivation

We describe existing studies that are closely relevant to our work, that is, adversarial attacks, semantic image transformations, and coverage-guided testings. Then we discuss our preliminary experiments that motivated this study.

*Corresponding author and principal investigator

Adversarial Attacks. Neural networks are known to be susceptible to imperceptible perturbations. Deceiving neural networks by exploiting this property is referred to as adversarial attack (Szegedy et al. 2013). Techniques for adversarial attack have been extensively studied,(Carlini and Wagner 2017; Madry et al. 2017; Moosavi-Dezfooli, Fawzi, and Frossard 2016). However, these techniques search only in raw pixel space to find adversarial examples, and they cannot handle certain realistic variations of attributes, such as light conditions (Qiu et al. 2019). Recently, adversarial attack techniques with distance metrics other than L_p norm are studied (Kang et al. 2019; Xiao et al. 2018).

Semantic Image Transformation. To further explore diverse attacks on neural networks, techniques based on semantic image transformations are studied (Bhattad et al. 2019; He et al. 2019; Joshi et al. 2019; Qiu et al. 2019). Particularly, studies based on deep generative models are extensively conducted. For example, Bhattad et al. make use of texture transfer models to attack neural networks. For more general semantic adversarial attack, attribute-conditioned image editing models are exploited (Dorta et al. 2020; He et al. 2019; Wu et al. 2019; Xu et al. 2020). Joshi et al. leveraged an attribute-editing GAN to search over the range of attributes to generate adversarial examples; Wu et al. proposed RelGAN that progressively modifies attributes with its *relative attributes*. While these techniques effectively generate semantic adversarial examples, they require manual annotation of attributes, which is costly.

Coverage-Guided Neural Network Testing. In software engineering, test coverage metrics, such as path coverage, measure the fraction of code that is exercised by a test suite; it also assesses the quality of test suites. Similar metrics are recently proposed for neural networks (Gerasimou et al. 2020; Ma et al. 2018; Odena et al. 2019; Pei et al. 2017; Riccio and Tonella 2020). DeepXplore introduced the notion of *neuron coverage* that represents the fraction of neurons activated by a set of test inputs. The metric is then used to simulate domain specific perturbations (Pei et al. 2017). Other metrics, such as neuron boundary coverage, are proposed as test coverage metrics for neural networks (Ma et al. 2018; Sun et al. 2018; Xie et al. 2019). TensorFuzz adopted coverage-guided fuzzing to efficiently find test input that violates certain properties in application domains (Odena et al. 2019).

Motivating Channel-Wise Testing. Odena et al. applied coverage-guided testing for neural networks (2019). Their technique, i.e. TensorFuzz, generates a corpus of test inputs to find the inputs that violate certain domain properties. For an efficient search over the input space, TensorFuzz records the internal states (i.e., activation vectors) of the neural network and creates the corpus consisting of *dissimilar* test inputs. They show that TensorFuzz can find error-inducing test inputs for fault-injected models and real-world ones.

TensorFuzz shows that coverage-guided test generation is helpful for debugging and understanding neural networks. While it employs internal neuron activations for the coverage metric, we considered higher-level metrics may be also useful. Specifically we conjectured that CNNs' trained features (i.e. channels of hidden layers) are well-suited for the coverage metric as they are activated in various degrees of *intensities* for different inputs, where intensity denotes the sum of the channel's neuron values. Moreover, trained features are higher-level measure than the neuron activations and thus testing with feature intensities may give different perspective in understanding neural networks.

Our preliminary experiments showed that TensorFuzz cannot test the diverse channels of CNNs. The details are discussed in Section 7.4 but TensorFuzz covers less than 35% of all features in tested CNNs; majority of the features are not well tested. When we systemically test them, it uncovers hidden issues in tested CNNs. Consider the data corruption problem (Jagielski et al. 2018) that may cause defects vulnerable to input distribution shift. Specifically, in a face identification task, assume that the training data is corrupted such that faces with a certain attribute (e.g. green hair) are all identified as a same person. Again the details are in Section 7.2 but FtGAN successfully revealed the problem by identifying the channel that is correlated to the attribute and generating defect-inducing test data. In contrast, TensorFuzz is based on randomized fuzzing, and its noise-augmented test data does not help to find the problem.

3 Taming GAN for Testing CNN's Channels

3.1 Introducing FtGAN

GAN-based techniques have been studied to generate realistic test input for neural networks (Bhattad et al. 2019; Joshi et al. 2019). We also make use of GAN and propose FtGAN for testing CNN's channels. In particular, we use GAN's capability to learn latent variables in the training data and generate images with varying the latent variables (Chen et al. 2016). However, instead of learning *any* latent variables, we train FtGAN to learn those that are highly related to the selected channels of a tested CNN (Huang and Belongie 2017; Xu et al. 2020). Specifically, FtGAN is conditioned on an auxiliary input I that indicates the *intensity* of the CNN's selected channel, i.e., the summation of the channel's neuron values. By varying I, we can control the generated images so that they activate the selected channel with varying intensity levels. We use the channel intensity as the control input because CNN's channel-wise mean and variance are known to capture the styles and attributes of images (Xu et al. 2020; Huang and Belongie 2017; Gatys, Ecker, and Bethge 2016); also, the channel mean and variance are previously used to control style features such as textures in generator networks (Huang and Belongie 2017; Xu et al. 2020).

FtGAN consists of generator G and discriminator D as shown in Fig. 1. FtGAN makes use of the target CNN T and its selected channel c for its training. The encoder G_{enc} is optional, and if used, the generated output is reconstructed from the input with the decoder's transformation. The input I is a scalar value that controls channel c's intensity (T_c), that is, the sum of c's neuron values. The discriminator has two parts; one that distinguishes real data from the generated ones and another that infers the value of I. The target CNN T, which is being tested, is used by FtGAN as a reference point. Its selected channel's intensity is directed to G, giving the guidance to find the latent variable related to the channel.

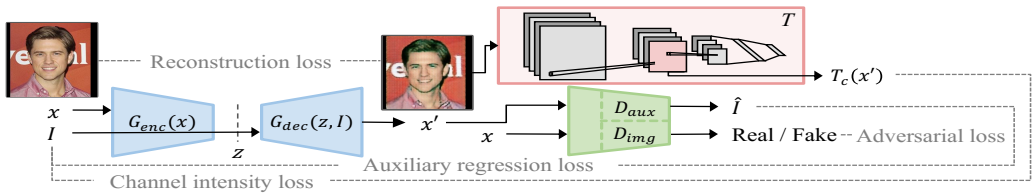

Figure 1: FtGAN Architecture (G: generator, D: discriminator, T: the target CNN, and T_c: its tested channel's intensity).

3.2 Architecture of FtGAN

Our goal is to train a generator G that learns to generate images with varying the intensity of target channel c. To achieve this, we train $G : (x, I) \rightarrow x'$ to transform input image x into output image x' that make the model T to have $T_c(x') \sim I$ on target channel c. T_c is the function returning the intensity of channel c and I is the target intensity calculated as $I{=}T_c(x) \cdot (1 + r)$ with a distortion rate r. We set r such that $1 + r$ ranges from 0.33 to 3 in our experiments. We also have an auxiliary classifier in the discriminator to predict the channel intensity I. For training FtGAN we define four loss terms, namely, adversarial loss, channel intensity loss, auxiliary regression loss, and reconstruction loss. We describe these four loss terms in the following.

Adversarial loss: For stable training, we adopt WGAN's adversarial loss (Arjovsky, Chintala, and Bottou 2017) that minimizes the Wasserstein-1 distance between the real and generated distributions. Let D_{img} be the discriminator which outputs the probability that its input is a real data. Then the adversarial loss is

$$L_{adv} = E_x[D_{img}(x)] - E_{x,I}[D_{img}(G(x,I))] \quad (1)$$

We also apply the gradient penalty for the Lipschitz constraint (Gulrajani et al. 2017).

Channel intensity loss: We control the intensity of the channel c separately from the remaining channels in the same layer L. Therefore, we employ the channel intensity loss L_{int} for the generator to constrain the generated image x' to produce the desired channel intensity I, formulated as follows (L is the set of all channels in layer L),

$$L_{int} = |T_c(x') - I| + \frac{1}{|L|-1} \sum_{c' \in L \wedge c \neq c'} |T_{c'}(x') - T_{c'}(x)| \quad (2)$$

Auxiliary regression loss: Our goal is to transform an input image x into a realistic image x', which yields the intended intensity on the target channel. Therefore we add an auxiliary regressor D_{aux} that shares the convolution layers with D_{img} and define the auxiliary regression loss for both D and G. The auxiliary regression loss for real images is defined as

$$L_{aux}^{real} = E_x[-\log Q_{aux}(x, T_c(x))] \quad (3)$$

where $Q_{aux}(x, I)$ is modeled with a normal distribution whose mean $\mu_{aux}(x)$ and variance $\sigma^2_{aux}(x)$ are estimated by the auxiliary regressor D_{aux} for input x. The log-normal $\log[Q_{aux}(x, I)]$ is calculated as

$$-\frac{1}{2}\log\left(2\pi\sigma^2_{aux}(x) + \epsilon\right) - \frac{1}{2}\left(\frac{I - \mu_{aux}(x)}{\sigma_{aux}(x) + \epsilon}\right)^2 \quad (4)$$

where ϵ is a small positive value. By minimizing the objective, D_{aux} learns to estimate the channel intensity $T_c(x)$ of a real image x. The pairs of real images x and their target channel intensities $(x, T_c(x))$ are used to train D_{aux} using the above loss L_{aux}^{real}.

Furthermore, the loss function for the auxiliary regression with fake image x', which is generated by G with the intensity $I = T_c(x) \cdot (1 + r)$, is formulated as following:

$$L_{aux}^{fake} = E_{x,I}[-\log Q_{aux}(G(x,I), I)] \quad (5)$$

The above loss function is then used to train G to generate images with the intended intensity I on the channel c. For each input x in the training dataset and randomly selected distortion rate r, we generate $x'{=}G(x, T_c(x) \cdot (1 + r))$ and the pair $(x', T_c(x) \cdot (1 + r))$ is then used for the training.

Reconstruction Loss: We use the reconstruction (He et al. 2019) loss to train the generator to produce images that are similar to the original images, given the same channel intensity. This is achieved by the following objective:

$$L_{rec} = E_{x,T_c(x)}[\|x - G(x, T_c(x))\|_1] \quad (6)$$

By minimizing this loss, the generator can produce images that closely approximate the original images when the same channel intensity is given as input.

Overall Objective: The final objective for the generator is

$$L_{enc,dec} = \lambda_{adv} L_{adv} + \lambda_{rec} L_{rec} + \lambda_{int} L_{int} + \lambda_{aux,f} L_{aux}^{fake} \quad (7)$$

and the objective for the discriminator/auxiliary classifier is

$$L_{dis,aux} = -\lambda_{dis} L_{adv} + \lambda_{aux,r} L_{aux}^{real} + \lambda_{gp} GP \quad (8)$$

We set the coefficients in Eqns. 7 and 8 to make the loss terms be in the same order of magnitude (He et al. 2019; Wu et al. 2019; Dorta et al. 2020). We set $\lambda_{adv} = \lambda_{dis}{=}1$, $\lambda_{rec}{=}100$, $\lambda_{aux,*}{=}5$, $\lambda_{gp}{=}1$ (for the gradient penalty GP); the value of λ_{int} varies for the datasets (e.g. 0.5 for CelebA).

3.3 Reducing the Cost of Training FtGAN

To test a CNN's channel, we need to train an FtGAN instance. Although the cost of training FtGAN is not trivial, it can be reduced with pre-training. That is, we pre-train FtGAN to generate realistic images without the target CNN; then we fine-tune FtGAN with the intensity loss from the target CNN. This way, we pre-train FtGAN once and fine-tune it multiple times to test multiple channels. The cost of the fine-tuning is less than a quarter of that of the pre-training.

4 Coverage-Guided Channel Selection

FtGAN generates realistic images for testing a CNN's channels. However, recent CNN models have many layers and channels, and thus it is not feasible to test all those channels with FtGAN. Thus, we propose to test a subset of the channels that is *representative* of all or most of the channels. The key idea is to exploit the correlations between CNN channel intensities (Bengio and Bergstra 2009; Rodríguez et al. 2016); i.e. we select a subset S of the channels having high correlations (either positive or negative) with the channels that are not in S. Then, testing the channels in S would have

the effect of indirectly testing the other channels. We now formally define the channel selection problem as follows.

Let $corr(c_i, c_j)$ be the Pearson correlation between channel c_i and c_j. For given test inputs, $corr(c_i, c_j)$ is computed with the pairs of the intensities of c_i and c_j. We compute the correlations for all pairs of the convolutional channels.

Let V be the set of all channels in the convolutional layers of the target network. With subset $S \subseteq V$, we presume that a channel $c_i \notin S$ can be *indirectly* tested by one of the channels in S if its correlation with c_i is larger than a given threshold. We denote by $corr(c_i, S)$ the maximum correlation between a channel in S and c_i, i.e., $\max_{\gamma \in S} corr(\gamma, c_i)$. Let θ be the minimum correlation threshold to indirectly test the channels not in S. Then, the problem of finding the minimum subset S for testing all channels in V is formulated as

$$\arg\min_{S \subseteq V} |S| \text{ s.t. } \min_{c_i \in V} corr(c_i, S) \geq \theta. \quad (9)$$

In other words, our channel selection problem finds the smallest set S such that for all channels c_i in V, there exists at least a channel c_j in S with $corr(c_i, c_j) \geq \theta$.

Proposition 4.1 *Minimal channel selection (Eqn. 9) is equivalent to the minimal hitting set problem.*

Let $\delta(c_i)$ for $c_i \in V$ denote the set of channels $c_j \in V$ with $corr(c_i, c_j) \geq \theta$. That is, $\delta(c_i)$ is a set of channels that can be tested instead of c_i as their correlations are higher than θ. A feasible solution to the channel selection problem is the set that has at least one channel in $\delta(c_i)$ for all $c_i \in V$. Formulated in this way, this problem is equivalent to the minimal hitting set problem (Ausiello, D'Atri, and Protasi 1980).

The minimal hitting set problem is NP-complete and equivalent to the minimal set cover problem. Due to Proposition 4.1, we can obtain a greedy approximate algorithm as shown in Algorithm 1 whose approximation ratio is proven to be $\ln(n) + 1$ where n is the number of channels.

Algorithm 1: Greedy channel selection algorithm.

Input: $\forall c_i \in V, \delta(c_i) = \{c_j \in V | corr(c_i, c_j) \geq \theta\}$
Output: Channel set S

```
Initialize C ← ∅ and S ← ∅;
while C ≠ V do
    c* ← arg min_{c_i ∈ V−C} |δ(c_i)/C|;
    C ← C ∪ δ(c*);   S ← S ∪ {c*};
end while
```

5 Testing Channels with Unexpectedness

Canonical correlation analysis (CCA) has been used to analyze and understand the latent representations, i.e., features in channels, of neural networks (Hardoon, Szedmak, and Shawe-Taylor 2004; Li et al. 2015; Morcos, Raghu, and Bengio 2018; Raghu et al. 2017). The method finds linear transformations that maximize the correlations between multidimensional variables (Hotelling 1992). Recently, Raghu et al. applied CCA to ResNet models and demonstrated that the correlation of the hidden neurons to different labeling classes are distinctly different.

We use CCA to analyze the target CNN's inference computation for generated test data and find inconsistent channel behavior. Since FtGAN varies the intensity of tested channels, we apply CCA at a channel level and compute the correlation of channel intensities. Also, because CCA is very expensive to compute (Raghu et al. 2017), we approximate it by computing *pair-wise* channel correlations and use them as the reference points of the inference computation. That is, we calculate the pair-wise channel correlations using the training data labeled as a same class, and find the top-*k* channel pairs of maximum correlation coefficients. Then we use them as the reference point for the class and compare them to the correlations computed with generated test data of the same class. Our experiments (similar to the one by Raghu et al. and not described due to space limit) show that the comparison reflects the similarity (or unexpectedness) of their inference computations. We do not claim that unexpectedness score accurately measures the similarity (or inconsistency) of inference computations; we argue with experiments that the score helps to identify deviant channel behavior. This is similar to many testing techniques in software engineering that rank the test results (potential bugs) by certain evaluation scores and report the higher-ranked ones.

More formally, for testing a selected channel in layer L, we define *unexpectedness* score as the L1 distance of L's top-*k* channel correlations with the training data to those with the generated test data of a same class; i.e., the unexpectedness score is $\sum_{(c_i,c_j) \in \text{TopK}} |corr_{X(T)}(c_i, c_j) - corr_{X'(T)}(c_i, c_j)|$, where $X(T)$ and $X'(T)$ are the training data and generated test data in class T, $corr_{X(T)}$ is the correlation computed with $X(T)$, and TopK is the set of channel pairs in L with top-*k* correlations for the training data; we set k=5% in our evaluation. After we generate test data for selected channels, we measure and rank the unexpectedness of the generated data in each class. Then we report the tested channels, the classes, and the test data of the classes ranked by their unexpectedness with highlighting the test data that changed the inference outcome.

6 Discussion

Limitation. Testing with FtGAN is limited by the capability of GAN. That is, FtGAN learns the attributes that are present in the training dataset. Thus we can only test with those attributes in the dataset that are correlated to the selected channels of target CNNs. This limits the types of bugs that our technique can detect. However, testing is generally considered to be opportunistic, and similar limitations exist in many testing tools, especially in those that check deviant runtime behaviors (Engler et al. 2001; Ernst et al. 2007; Haller et al. 2013). It is more important for a testing tool to find real bugs in practice, which we show in our evaluation.

Another limitation is that our testing requires human examination of the test results. We minimize this by computing unexpectedness scores and ranking the results by the scores. Thus only a small subset of the test results needs to be examined. This is similar to many software testing tools that rank test results (potential bugs) by certain scores (Engler et al. 2001; Ernst et al. 2007; Haller et al. 2013). In all our experiments, buggy channels are in top-5 by unexpectedness score, which made human intervention reasonably small.

Multi-Channel Testing. Although FtGAN tests a single

Dataset	Model	Task
MNIST/SVHN	LeNet & AlexNet	Digit recognition
VGG Face	VGG-16 & ResNet-50	Face identification
CelebA	AlexNet	Face attribute detection
CARLA	CARLA-CNN	Autonomous driving

Table 1: Evaluated Datasets and Models.

channel at a time, we observed that its generated test data incorporates the changes of other correlated (or anti-correlated) channels. Hence those correlated channels are collectively tested in effect.

Moreover, testing multiple (non-correlated) channels is supported with FtGAN by chaining multiple FtGAN instances. That is, if we want to test two channels c_1 and c_2, we can direct the output of the FtGAN trained for c_1 as the input of another FtGAN trained for c_2. We have tested multiple channels in this way for a subset of our experiments, which we describe in Section 7.2. Chaining multiple GANs in a similar manner was previous studied for generating high resolution images (Zhang et al. 2017) or transforming poses and expressions of facial images (Zheng et al. 2017).

7 Evaluation

We evaluated whether FtGAN can effectively test CNNs' channels with three sets of experiments: 1) one for making realistic (and error-inducing) attribute variations (Section 7.1), 2) another for finding the channels that are correlated to error-inducing attributes in bug-injected and real-world CNN models (7.2 and 7.3), and 3) the last for ensuring the test coverage of our greedy channel selection (7.4). Complete analysis, experimental results, and supplementary material are in the paper's extended version (Choi et al. 2023).

We evaluated our technique with five datasets in four domains; they are MNIST, SVHN, VGG Face, CelebA, and CARLA (Codevilla et al. 2018). MNIST and SVHN are for digit recognition, VGG Face is for face identification, CelebA is for face attribute recognition, and CARLA is for autonomous driving. Table 1 shows the datasets. Moreover, we tested with five models – LeNet, AlexNet, VGG-16, ResNet-50, and CARLA-CNN (a custom CNN model for autonomous driving). LeNet/AlexNet are trained to classify the categories in the dataset (MNIST/SVHN) or to detect a subset of the attributes in the dataset (CelebA); VGG/ResNet are trained to detect the identities in the VGG Face dataset. We used NVIDIA Titan XP for all the experiments. On a single Titan XP, the fine-tuning of FtGAN for one channel takes ten minutes for SVHN and an hour for CelebA.

We mainly evaluated our testing techniques qualitatively. This is similar to the evaluation of software testing techniques that analyze program behavior to infer implicit invariants and report their violations (Elbaum et al. 2006; Engler and Ashcraft 2003; Flanagan et al. 2002; Gligoric et al. 2010; Hangal and Lam 2002, 2009; Saxena et al. 2009).

7.1 Efficacy of FtGAN's Test Data Generation

We first evaluate whether FtGAN generates realistic images that induce the tested channels with varying intensities. Then we examine the unexpectedness scores of the generated data and discuss the validity of the scores. For this evaluation, we mostly used the face datasets and the corresponding CNN models. We run greedy channel selection to select twenty channels for each CNN instance. Then we trained FtGAN for the selected channels and generated test data using the images in the test sets as the seeds. We set the intensity input I to be between 0.33 and 3 times that of the seeds for the test data generation. We show some of the generated test data in Fig. 2. As we vary the intensity input I, the correlated latent attributes are gradually changing in all the generated test images. We also observed that some of the latent attributes for the channels are human recognizable (a–d in Fig. 2) and others are not (e–h). The attributes that are human recognizable are, namely, hair color, face mask, face color, and age, respectively for a–d; we call these channels by their correlated attribute names, e.g. age channel. We can see that the test data for these attributes show realistic variations.

Validity of Unexpectedness Score. We analyzed the unexpectedness scores of the channels for Fig 2. Due to the space limit, we do not discuss the details here, but our analysis shows that the higher scores generally indicate inconsistent inference computations. The channels a–c in Fig 2 have high unexpectedness scores and we observed several inconsistent behaviors for them. For example, testing the face color channel with the *pale-skin* class data generated pink-ish face data, which confused the classification of the age attribute; i.e., the generated images are classified as not *young* even though the seed images are labeled as such. Fig. 2(c, right) shows an example; with the pink-ish skin she is incorrectly classified as not *young*. Also, the hair color channel shows similar behavior and confused *pale-skin* classifier.

7.2 Finding Data Corruption Bugs with FtGAN

We evaluate if FtGAN helps to find bugs in bug-injected CNN instances. We trained two CNN models to have defects (of being vulnerable to input attribute distribution shift) on purpose by corrupting the training dataset. For the defect injection, we use Morpho-MNIST, which extends MNIST with morphometric transformations such as "thickening" or "swelling" (Castro et al. 2019). We trained two AlexNet models with the combined dataset of MNIST and Morpho-MNIST, one with the thickening and another with the swelling transformation. The images in Morpho-MNIST are given a same incorrect label (i.e., 0) so that the two AlexNets have the defects of incorrectly classifying the

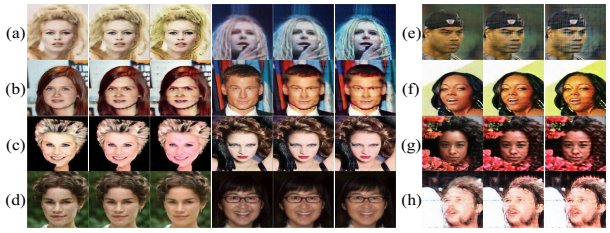

Figure 2: Test data made by FtGAN. The latent attributes for a–d are human-recognizable and those for e–h are not.

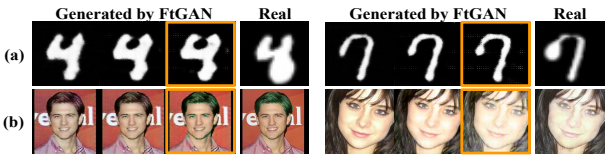

Figure 3: Testing defective CNN instances that misclassify (a) swollen digits or (b) faces with green hair or pale skin.

Figure 4: Test images by FtGAN, TensorFuzz, and semantic adv attack for VGG-hair (left) & ResNet-skin (right).

thickened or swollen digits. The accuracy of the models is 99% for the normal digits. For the thickened or swollen digits, the models are only 2% and 1% accurate, respectively.

For the two AlexNet instances, namely AlexNet-TH(ick) and AlexNet-SW(ell), 1) we apply the greedy channel selection and train FtGAN with the twenty selected channels, 2) generate test images with the channel intensity from 0.33 to 3 times of the original, and 3) measure the unexpectedness scores for the channels; we rank by the score the test data for the channels and examined top-5 channels' test data in detail. For both AlexNet-TH and AlexNet-SW, we noticed that for one particular channel, namely thick channel and swollen channel, its test data result in incorrect inference outcome at a high rate. Table 2 shows the rate of the generated test data for thick and swollen channels resulting in mis-classification for varying intensity input I; the table also shows the ranking of those channels by the score in parentheses in the header.

Fig. 3 (a) shows the images that are generated for AlexNet -SW for the swollen channel. The generated images with three different intensity are shown for each digit; images in Morpho-MNIST are also shown (marked as "Real"). The generated images have swollen strokes like those in Morpho-MNIST. The swelling is subtle in our test data but it is sufficient for the classifier to output incorrect labels; the orange boxes in Fig. 3 indicate that the samples are incorrectly classified by AlexNet-SW. The prototype of FtGAN is publicly available at our repository[1].

We also tested the interactions of two channels of AlexNet-SW by feeding the output of FtGAN that is trained for one channel as the input of another FtGAN that is trained for a different channel. We tested the pairs of top-5 channels by their unexpectedness scores. Fig. 5 shows the results with comparing the test data generated for both channels (denoted by *Both*) with the data generated for one channel (denoted by *ch.* followed by the channel id). The data is generated with 1.5× channel intensity; data generated with 2.25× channel intensity is also shown for comparison. The orange box denotes that the sample is mis-classified by AlexNet-SW.

[1]https://github.com/mlsys-seo/FtGAN

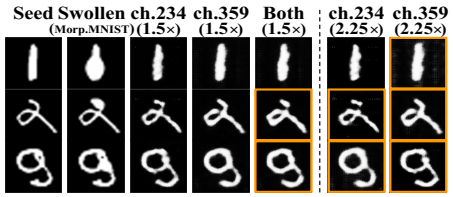

Figure 5: Testing of two channels in AlexNet-SW.

From the results, we observed that for some seed inputs the swelling is more noticeable (hence more mis-classifications) when the intensities of both channels are adjusted. Although our focus in this paper is modular testing of individual CNN channels, this experiment shows that FtGAN can be used to jointly test multiple CNN channels.

Moreover, we simulated the data corruption bugs with the VGG Face dataset. We trained two VGG-16 networks to identify the faces in the dataset but we injected certain bugs. For one instance (namely VGG-hair) we trained it to misclassify any faces with green hair as a certain target person; for another one (namely VGG-skin) we made it to infer any faces with pale skin as another target identity. We have repeated the process in a similar manner to train two faulty ResNet instances (ResNet-{hair,skin}). For ResNet though, we only considered 3x3 convolutional layers for the testing as 1x1 convolutions are mainly designed to control the computational complexity by reducing or expanding the channel dimensions (Szegedy et al. 2015; He et al. 2016).

To test the faulty CNN instances, we selected forty channels (or sixty for ResNet) with the greedy selection in Algorithm 1 and tested them with FtGAN. We examined the test data of top-5 unexpected channels in detail and identified one most defective channel for each of the instances. Let us describe the details of the manual examination for the VGG models. We describe the case for VGG-skin, and the case for VGG-hair is not described as it is similar and also due to the space limit. Among the five channels with high unexpectedness scores, four channels are related to human-recognizable attributes (nose-color, liver-spots, pale-skin-a, and pale-skin-b); we refer these channels as *semantic* channels. The fifth channel adds grid patterns to the images. For the five channels, we examined how the inference outcome changes as the intensity changes. The inference outcomes for all five channels changed in a biased manner towards the target identity that VGG-skin is trained for. For the three semantic channels (except the pale-skin-b channel), 20–30%

Range of Intensity I	**Model (score rank / tested channels)**		
	AlexNet-TH (2/20)	**AlexNet-SW (4/20)**	**VGG-hair (3/40)**
0.9 - 2.0	31.8%	16.8%	56.7%
0.7 - 3.0	67.8%	56.2%	80.4%
0.5 - 4.0	84.8%	85.2%	87.4%
0.4 - 4.5	89.0%	90.2%	89.4%

Table 2: The rate of mis-classified test data generated for the defective channels. The rankings of the channels by their unexpectedness scores are shown in parentheses in the header.

Figure 6: Test images generated by FtGAN for CARLA-CNN. We observe (a) the center line wear-out and road texture changes, and (b) the color tone changes.

of the incorrect outcomes are inferred as the target identity. For the pale-skin-b channel, 60% of the incorrect outcomes are inferred as the target identity. Fig. 3 shows an example of generated test data. Again with our testing techniques we successfully identified the defect in VGG-skin. For other channels having low unexpectedness scores among the forty selected ones, we did not observe such biased inference changes.

If we test the faulty models with TensorFuzz or semantic adversarial attack (Joshi et al. 2019), the defects are not detected. Fig. 4 shows the generated images of FtGAN and the two techniques. TensorFuzz generates noise-augmented images; the semantic attack inconsistently changes a few attributes. Most importantly, the images made by the two techniques are not identified as the target person, thus they did not find the defect but simply generated adversarial examples. We also applied other adversarial example generation techniques (Carlini and Wagner 2017; Moosavi-Dezfooli, Fawzi, and Frossard 2016), but they generated minimally perturbed images (that are not identified as the target person) and thus we do not show them here. Furthermore, we used Grad-CAM (Selvaraju et al. 2017), the state of the art XAI technique, to FtGAN's test images and confirmed that the changes made by FtGAN caused the inference changes. For additional details on the results, please refer to the extended version of the paper (Choi et al. 2023).

7.3 Finding Bugs in a Public CNN Model

We further evaluated FtGAN with a pre-trained, publicly-available CNN instance for autonomous driving that is developed and trained by Codevilla et al. (Codevilla et al. 2018). This CNN instance, which we call CARLA-CNN, has eight convolution layers and two dense layers. We tested the channels of the convolution layers in the same way as the previous experiments. The images generated for a subset of the channels have certain semantic variations such as the center line wear-out, road texture changes, or color tone changes. Fig. 6 shows the generated images; these channels generally have high unexpectedness scores. The white arrows on the images are the steering decision made by CARLA-CNN. We can see that the variations caused by the intensity changes make CARLA-CNN to make wrong steering decision (marked by orange boxes).

7.4 Coverage Gain by Greedy Channel Selection

Metric for channel coverage. With a test coverage metric, we aim to quantify the fraction of the channels in a target CNN that satisfy the test condition for a test suite made

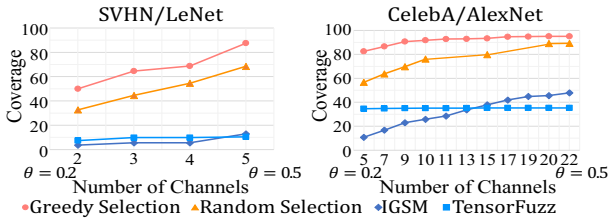

Figure 7: Channel boundary coverage of FtGAN (greedy & random selection), IGSM, and TensorFuzz. θ below the x-axis is the correlation threshold for the channel selection.

by a test image generator. Borrowing the concept of neuron boundary coverage (Ma et al. 2018), we define *channel boundary coverage*. Let $T_c(x)$ be the intensity of channel c when image x is given as input to target network T. Given training dataset $\mathbb{X}$ for T, let I_c^{upper} be the maximum intensity of the channel c for $\mathbb{X}$; i.e., $I_c^{upper}=\max_{x\in\mathbb{X}} T_c(x)$. For the channel c, if $T_c(x)$ for a test image x from the test suite $\mathbb{T}$ is larger than I_c^{upper}, the boundary of c is said to be *covered* by $\mathbb{T}$. Let V denote the set of all channels to be tested in T. The *boundary coverage* is then defined as $\frac{|\{c\in V|\exists x\in\mathbb{T}, T_c(x)>I_c^{upper}\}|}{|V|}$, that is, the ratio of channels whose boundaries are covered by at least an image in $\mathbb{T}$.

Coverage evaluation. With varying the minimum correlation θ from 0.2 to 0.5 in Algorithm 1, we selected test channels of the target networks trained for SVHN (LeNet) and CelebA (AlexNet). For the baselines, we selected the same number of test channels with random selection. For the two networks, we then generated $2,000$ test images per channel based on the seed dataset using FtGAN and plotted the boundary coverage of each test in Fig. 7. We also applied two other techniques, IGSM (adversarial example generation based on iterative gradient sign method) (Kurakin, Goodfellow, and Bengio 2016) and TensorFuzz, to generate the same number of test images for each experiment; we cannot apply semantic adversarial attack as its execution time is too long, taking more than a few seconds to generate a single image. The graphs show that the test suite obtained by our greedy algorithm achieves the highest coverage in both networks and verify that the channels selected by our greedy algorithm efficiently cover for the untested channels.

8 Conclusion

This paper proposes techniques for testing the channels of CNNs. We designed FtGAN, an extension to GAN that generates realistic test data for a target CNN with varying its selected channels intensities. We developed a channel selection algorithm that finds a subset of a CNN's representative channels by using the correlations between the channels. To investigate inconsistency in the target CNN's inference with FtGAN's test data, we defined unexpectedness score and rank the test data by this score. In our evaluation, we investigated five CNN models that are trained with five datasets. By applying our testing techniques, we successfully found defects in both synthetic and real-world CNN instances.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2018R1D1A1A02086132 and No.2020R1G1A1011471) and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2020-0-01373, Artificial Intelligence Graduate School Program (Hanyang University)). We thank Nahun Kim for his help with the experiments. Jiwon Seo is the corresponding author.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Ausiello, G.; D'Atri, A.; and Protasi, M. 1980. Structure Preserving Reductions among Convex Optimization Problems. *J. Comput. Syst. Sci.*, 21(1): 136–153.

Bengio, Y.; and Bergstra, J. S. 2009. Slow, decorrelated features for pretraining complex cell-like networks. In *NIPS*, 99–107.

Bhattad, A.; Chong, M. J.; Liang, K.; Li, B.; and Forsyth, D. A. 2019. Unrestricted adversarial examples via semantic manipulation. *arXiv preprint arXiv:1904.06347*.

Carlini, N.; and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *SP*, 39–57. IEEE.

Castro, D. C.; Tan, J.; Kainz, B.; Konukoglu, E.; and Glocker, B. 2019. Morpho-MNIST: Quantitative Assessment and Diagnostics for Representation Learning. *JMLR*, 20(178).

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2172–2180.

Choi, K.; Son, D.; Kim, Y.; and Seo, J. 2023. Testing the Channels of Convolutional Neural Networks. arXiv:2303.03400.

Codevilla, F.; Müller, M.; López, A.; Koltun, V.; and Dosovitskiy, A. 2018. End-to-end Driving via Conditional Imitation Learning. In *ICRA*, 4693–4700.

Dorta, G.; Vicente, S.; Campbell, N. D. F.; and Simpson, I. J. A. 2020. The GAN That Warped: Semantic Attribute Editing With Unpaired Data. In *CVPR*, 5356–5365.

Elbaum, S.; Chin, H. N.; Dwyer, M. B.; and Dokulil, J. 2006. Carving differential unit test cases from system test cases. In *SIGSOFT'06/FSE-14*, 253–264.

Engler, D.; and Ashcraft, K. 2003. RacerX: Effective, static detection of race conditions and deadlocks. *ACM SIGOPS Operating Systems Review*, 37(5): 237–252.

Engler, D.; Chen, D. Y.; Hallem, S.; Chou, A.; and Chelf, B. 2001. Bugs as deviant behavior: A general approach to inferring errors in systems code. *ACM SIGOPS Operating Systems Review*, 35(5): 57–72.

Ernst, M. D.; Perkins, J. H.; Guo, P. J.; McCamant, S.; Pacheco, C.; Tschantz, M. S.; and Xiao, C. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3): 35–45.

Flanagan, C.; Leino, K. R. M.; Lillibridge, M.; Nelson, G.; Saxe, J. B.; and Stata, R. 2002. Extended static checking for Java. In *PLDI*, 234–245.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2016. Image Style Transfer Using Convolutional Neural Networks. In *CVPR*, 2414–2423.

Gerasimou, S.; Eniser, H. F.; Sen, A.; and Cakan, A. 2020. Importance-driven deep learning system testing. In *ICSE*, 702–713. IEEE.

Gligoric, M.; Gvero, T.; Jagannath, V.; Khurshid, S.; Kuncak, V.; and Marinov, D. 2010. Test generation through programming in UDITA. In *ICSE*, 225–234. IEEE.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. C. 2017. Improved training of wasserstein gans. In *NIPS*, 5767–5777.

Haller, I.; Slowinska, A.; Neugschwandtner, M.; and Bos, H. 2013. Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations. In *USENIX Security*, 49–64.

Hangal, S.; and Lam, M. S. 2002. Tracking down software bugs using automatic anomaly detection. In *ICSE*, 291–301. IEEE.

Hangal, S.; and Lam, M. S. 2009. Automatic dimension inference and checking for object-oriented programs. In *ICSE*, 155–165. IEEE.

Hardoon, D. R.; Szedmak, S.; and Shawe-Taylor, J. 2004. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12): 2639–2664.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

He, Z.; Zuo, W.; Kan, M.; Shan, S.; and Chen, X. 2019. Attgan: Facial attribute editing by only changing what you want. *IEEE TIP*, 28(11): 5464–5478.

Hotelling, H. 1992. Relations between two sets of variates. In *Breakthroughs in statistics*, 162–190. Springer.

Huang, X.; and Belongie, S. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 1501–1510.

Jagielski, M.; Oprea, A.; Biggio, B.; Liu, C.; Nita-Rotaru, C.; and Li, B. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *SP*, 19–35. IEEE.

Joshi, A.; Mukherjee, A.; Sarkar, S.; and Hegde, C. 2019. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. In *ICCV*, 4773–4783.

Kang, D.; Sun, Y.; Hendrycks, D.; Brown, T.; and Steinhardt, J. 2019. Testing robustness against unforeseen adversaries. *arXiv preprint arXiv:1908.08016*.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, 97–117. Springer.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

Li, Y.; Yosinski, J.; Clune, J.; Lipson, H.; Hopcroft, J. E.; et al. 2015. Convergent learning: Do different neural networks learn the same representations? In *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, 196–212.

Ma, L.; Juefei-Xu, F.; Zhang, F.; Sun, J.; Xue, M.; Li, B.; Chen, C.; Su, T.; Li, L.; Liu, Y.; et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *ASE*, 120–131.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Martin, M.; Livshits, B.; and Lam, M. S. 2005. Finding application errors and security flaws using PQL: a program query language. *Acm Sigplan Notices*, 40(10): 365–383.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2574–2582.

Morcos, A.; Raghu, M.; and Bengio, S. 2018. Insights on representational similarity in neural networks with canonical correlation. In *NIPS*, 5732–5741.

Odena, A.; Olsson, C.; Andersen, D.; and Goodfellow, I. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *ICML*, 4901–4911.

Pei, K.; Cao, Y.; Yang, J.; and Jana, S. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *SOSP*, 1–18.

Qiu, H.; Xiao, C.; Yang, L.; Yan, X.; Lee, H.; and Li, B. 2019. SemanticAdv: Generating adversarial examples via attribute-conditional image editing. *arXiv preprint arXiv:1906.07927*.

Raghu, M.; Gilmer, J.; Yosinski, J.; and Sohl-Dickstein, J. 2017. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NIPS*, 6078–6087.

Riccio, V.; and Tonella, P. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *ESEC/FSE*, 876–888.

Rodríguez, P.; Gonzalez, J.; Cucurull, G.; Gonfaus, J. M.; and Roca, X. 2016. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*.

Saxena, P.; Poosankam, P.; McCamant, S.; and Song, D. 2009. Loop-extended symbolic execution on binary programs. In *ISSTA*, 225–236.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 618–626.

Sun, Y.; Huang, X.; Kroening, D.; Sharp, J.; Hill, M.; and Ashmore, R. 2018. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Wu, P.-W.; Lin, Y.-J.; Chang, C.-H.; Chang, E. Y.; and Liao, S.-W. 2019. RelGAN: Multi-Domain Image-to-Image Translation via Relative Attributes. In *ICCV*, 5914–5922.

Xiao, C.; Zhu, J.-Y.; Li, B.; He, W.; Liu, M.; and Song, D. 2018. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*.

Xie, X.; Ma, L.; Juefei-Xu, F.; Xue, M.; Chen, H.; Liu, Y.; Zhao, J.; Li, B.; Yin, J.; and See, S. 2019. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *ISSTA*, 146–157.

Xie, Y.; Chou, A.; and Engler, D. 2003. Archer: using symbolic, path-sensitive analysis to detect memory access errors. In *ESEC/FSE*, 327–336.

Xu, Q.; Tao, G.; Cheng, S.; Tan, L.; and Zhang, X. 2020. Towards feature space adversarial attack. *arXiv preprint arXiv:2004.12385*.

Zhang, H.; Xu, T.; Li, H.; Zhang, S.; Wang, X.; Huang, X.; and Metaxas, D. N. 2017. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 5907–5915.

Zheng, Z.; Yu, Z.; Zheng, H.; Wang, C.; and Wang, N. 2017. Pipeline generative adversarial networks for facial images generation with multiple attributes. *arXiv preprint arXiv:1711.10742*.