# ResNets: Easy to train, easy to fool?

Di Wu, Samuel Huberman

## I. CONTRIBUTIONS

- Extended the ResNet architecture.
- Compared the performances of various network models.
- Proposed and implemented a parallelized approach to generating universal adversarial perturbations.
- Constructed adversarial examples for ResNets and observed shared characteristics with universal adversarial perturbations.

## II. INTRODUCTION

Convolutional neural networks (ConvNets) have become one of the most popular machine learning approaches for image recognition. The trend in the past decade has focused in one direction: make the network deeper. For instance, using deeper networks with relatively small receptive fields, the classification accuracy on the ImageNet competition increased from 80% to 95% [1]. Shallow layers in ConvNets represent low-level local features in the images such as edges and color contrasts, while deeper layers aim to capture more complex shape information and could be more specific. The motivation of designing deeper ConvNets is to achieve greater representative power from hierarchically composing shallower features into deeper ones. While deep neural networks demonstrate high performance on image classification tasks, these same networks are often difficult to train due to two reasons [1]:

- Vanishing / exploding gradients: sometimes a neuron dies during training process and depending on its activation function it might never come back. This problem can be addressed with initialization techniques that try to start the optimization process with an active set of neurons.
- Harder optimization: the model introduces more parameters as the number of layers increases, thus the complexity increases and as do the chances of getting stuck in a local optimum (in the non-convex scenario).

The recent success of Residual neural networks (introduced by Microsoft in 2015 [1]), or ResNets, is, in part, due to their ability to overcome these challenges. ResNets tweak the architecture of a deep neural network and allows deeper neural networks to be effectively trained. In the original work, the authors modify the layers functions to introduce identity mapping which serves as shortcut connections parallel to the normal convolutional layers. The identity connections in ResNets propagate the gradient throughout the model and therefore are always alive.

In this project, we aim to study the impact of various network architectures as well as universal adversarial perturbations (UAP) on the classification performance. Firstly, we replicated the ResNet and HighwayNet as described in literature (see [1],[2]), and compared the results with equivalent ConvNets on the benchmarking CIFAR10 dataset. Inspired intuitively by the idea of ResNet, we extended its hierarchical structure by introducing multi-level shortcut connections between layers and explored its impact on the classification behavior. Moreover, we investigated the fooling rates of these networks using the fast gradient sign approach.

In Section III, we review the basic methodology of designing different network architectures. In Section IV we will describe the experiments and implementation, and also discuss our results and show comparisons between various network models. In Section VI, we review adversarial examples and the concept of a universal adversarial perturbation. In Section VII, we present some characteristics of adversarial examples for our ResNet architectures. Finally in the conclusion section, we summarize the insights and propose future improvements.

## III. DESIGN OF NETWORK MODELS

### A. Residual Network

ResNet [1] is a neural network architecture which solves the problem of vanishing gradients and optimization difficulties in the simplest way possible. The main idea is to provide the network with a shortcut connection at each layer to send the gradient signal backwards. Consider a single image $x_0$ that is passed through a regular convolutional network. Typically a regular network consists of $L$ layers, each of which implements a non-linear transformation $H()$. $H()$ can be functions of operations such as convolution (Conv), batch normalization (BN), or rectified linear units (ReLU). The output of the $l_{th}$ layer is represented as $x_l$. Traditional ConvNets feed the output of the $l_{th}$ layer forward as input to the $(l+1)_{th}$ layer, which can be represented as :

$$x_{l+1} = H(x_l) \tag{1}$$

When the signal is sent backwards, the gradient must pass through $H(x_l)$, which can cause problems due to the nonlinearities involved. Instead, ResNets add a shortcut connection that bypasses the non-linear transformation

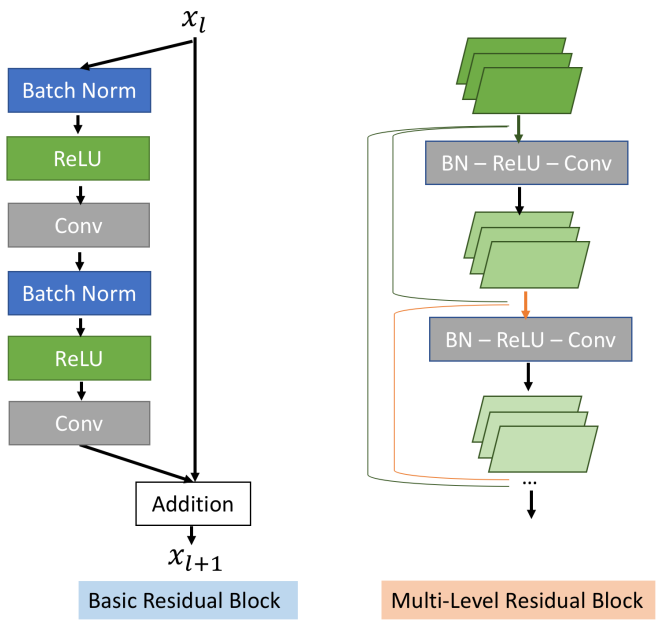



FIG. 1: Schematic Illustrations of ResNet and MultiResNet basic blocks

functions with an identity mapping:

$$x_{l+1} = H(x_l) + x_l \tag{2}$$

The $x_l$ at the end is the shortcut. It leads to the major advantage of ResNets that the gradient can pass backwards directly through the identity function from later layers to the earlier layers. Therefore, the issues of vanishing or exploding gradients and dead neurons can be resolved. For the actual implementation of ResNets, it takes the basic residual block as illustrated in Figure (1):

We could describe ResNets as multiple basic blocks that are serially connected to each other and there are also shortcut connections parallel to each basic block and it gets added to its output. However, the identity function and the output of $H(x_l)$ are combined by summation, which may impede the information flow in the network [3], [4].

### B. Highway Network

The second architecture we explored is the Highway Network (HighwayNet) as reported by Srivastava et al. [2]. The motivation of HighwayNet is also to address the gradient vanishing problem, especially when exacerbated the information flow in deeper layers. Intuitively, the information flow is blocked in the 'traffic problem'. And we set up 'special path' to restore the 'traffic' in the deep networks, just like a 'Highway' that we a familiar with from our physical world. The HighwayNet preserves the shortcuts introduced in the ResNet, but augments them with a learnable parameter to determine to what extent each layer should be a skip connection or a nonlinear connection. Layers in a HighwayNet are defined as follows:

$$x_{l+1} = H(x_l, W_H) \cdot T(x_l, W_T) + x_l \cdot C(x_l, W_C) \tag{3}$$

In Equation 3, we can find analogies with the previous two kinds of layers discussed: $H(x_l, W_H)$ mirrors the traditional transformation layer, and $H(x_l, W_H)+x_l$ mirrors the residual unit. The new functions are $T(x_l, W_T)$ and $C(x_l, W_C)$. They are referred as transform gate and carry gate respectively. For simplicity we set $C = 1 - T$. This serves at the switch to determine to what extent information should be sent through the primary pathway by nonlinear transformations or the skip pathway by identity mapping. Typically, the transformation gate is defined as a sigmoidal function: $T(x) = \sigma(W_T^T x + b_T)$, where $W_T$ is the weight matrix and $b_T$ is the bias.

### C. Multi-level Residual Network

Inspired by ResNets [3] and [4] and appropriating the insights of the skip connection, the basic architecture of ResNet could be further extended to explore the between its hierarchical structure and its performance. The idea here is that since connecting a skip connection from the previous layer improves performance, what would happen if we connect every layer to every other layer? In that case there is always a direct route for the information propagating backwards through the network. Initially, we proposed this multi-level connection architecture as inspired by Ke Zhange et al. [3] As our project progressed, we learnt that this idea was just very recently reported by Huang et al. on Dec. 6th. [4] It's no surprising to us since the deep learning is a super active research field. Instead of using an addition as Equation 4, however, the MultiResNet relies on stacking of layers. Mathematically it could be represented as

$$x_{l+1} = H_l([x_0, x_1, ..., x_{l-1}, x_l]) \tag{4}$$

Here $[x_0, x_1, ..., x_{l-1}, x_l]$ represents the concatenation of the feature maps of all preceding layers. This architecture makes intuitive sense in both the feed-forward and propagate-backward settings. From the feed-forward perspective, a deep learning task could benefit from being able to get shallow-level feature activations in addition to deep-level feature activations. For example, when classifying images, a lower layer of the network may determine edges in an image, whereas a higher layer would determine larger-scale features such as presence of objectives. There may be cases where the capability of utilizing information about edges can help with determining the correct object in a complex scene. From the backwards perspective, connecting all the layers allows to

quick transmission of gradients to their respective locations in the network. Figure 1 illustrates the architecture of Multi-ResNet schematically.

## IV. DATASET AND NETWORK IMPLEMENTATION

### A. Dataset: CIFAR10

We tested all the models on CIFAR10 dataset [5]. It consists of 60,000 colored natural scene images with $32 \times 32$ pixels each. There are 50k training images and 10k testing images in 10 classes. We preprocess the input data by subtracting the mean and dividing the standard deviation.

### B. Architecture

Our focus of this project is to compare the behaviors of various network models with equivalent depth, but not on achieving the state-of-the-art results, so we intentionally use relatively simple architectures as described below. Here we didn't specifically tune the hyperparameters for each model. We experimented network models with depth of 25-layer and 50-layer. The original ConvNet contains the regular convolutional layers mostly with $3 \times 3$ filters. The ResNets contains residual blocks with batch normalization (BN) before activation (ReLU) and then following by $3 \times 3$ convolution in residual mapping paths. If the input and output dimension for a basic block are equal, the shortcut connection is simply an identity matrix. Otherwise we use either average pooling (for reduction) or zero padding (for enlargement) to adjust the size. The network ends with a global average $2 \times 2$ pooling of stride 2, a 10-way fully-connected layer, and softmax. When implementing MultiResNets, similar with the ResNet, each block consists of three consecutive operations: batch normalization, followed by a rectified linear unit and a 3 convolution. But we can't just connect everything though. Only layers with the same height and width can be stacked. So we instead stack a set of convolutional layers, then apply a $2 \times 2$ average pooling layer, then stack another set of convolutional layers, etc. Taking the 25-layer MultiResNet as the example, each stack between the striding layers consists of 5 $3 \times 3$ convolutional layers with 64 hidden neurons each, then it's repeated for 5 times. The Figure 2 schematically illustrates the architectures.

### C. Training

In this project we use the Tensorflow library to implement and train different network models. Tensorflow has many predefined neural network layers and also enables us to run our training algorithms on GPUs. We train using Stochastic Gradient Descent with mini-batch size of 64. A learning rate of 0.001 is applied for the baseline trainings. One of the standard data augmentation techniques, random sampling is adopted in our experiments.

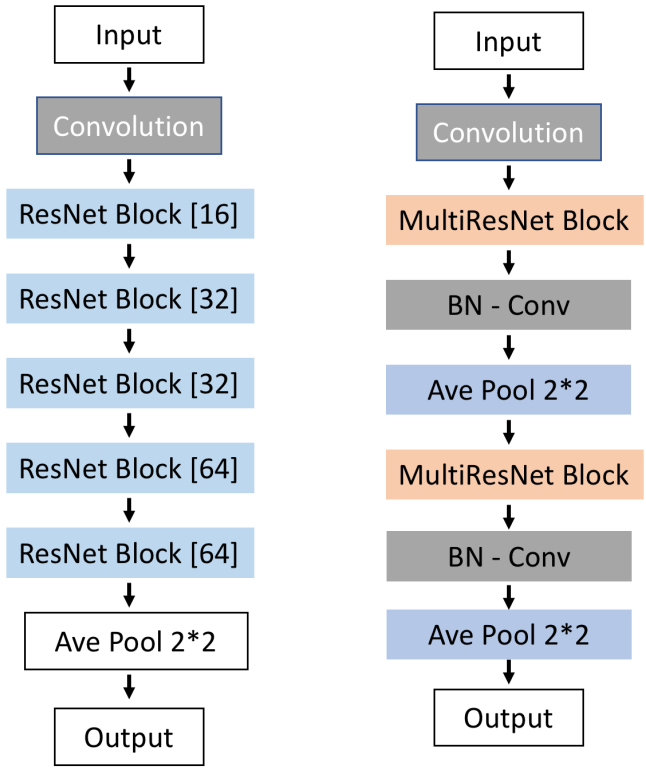


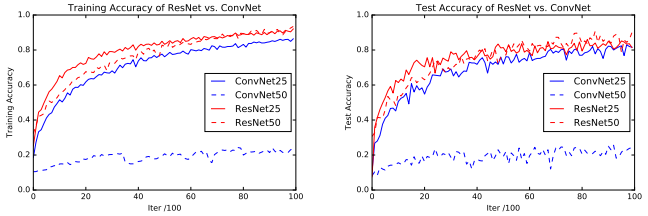

FIG. 2: Schematic Illustration of Network Architectures of ResNet(left) and MultiResNet(right).

| | 10K-step | Train Acc | Test Acc |
|---|---|---|---|
| ConvNet | 25-layer | 86.38% | 81.05% |
| | 50-layer | 23.48% | 22.85% |
| ResNet | 25-layer | 92.03% | 84.20% |
| | 50-layer | 93.47% | 84.45% |
| HighwayNet | 25-layer | 93.98% | 80.75% |
| | 50-layer | 91.22% | 78.59% |
| MultiResNet | 25-layer | 91.75% | 83.05% |
| | 50-layer | 92.74% | 85.84% |

TABLE I: 10k-step Training and Testing Accuracies on CIFAR10 dataset

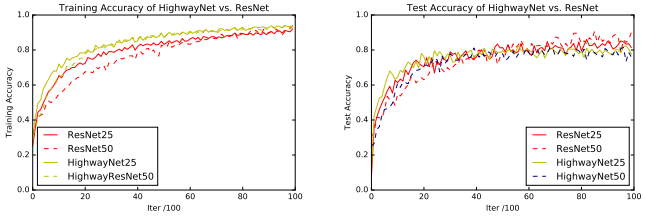
## V. CLASSIFICATION RESULTS AND DISCUSSION

The training and testing accuracies are summarized in the Table I.




(a) (b)

FIG. 3: ResNet vs ConvNet. Training accuracy(left) and test accuracy(right) on CIFAR10 with 25-layer and 50-layer networks. *Due to computational limitations, we demonstrated the baseline comparisons with 10K-step training procedure. Under this constraint, the networks have not yet converged to their optimal performance.**Average training accuracy of every 100-steps is reported.




(a) (b)

FIG. 4: HighwayNet vs ResNet. Training accuracy(left) and test accuracy(right) on CIFAR10 with 25-layer and 50-layer networks.

### A. ResNet vs. ConvNet

We observed the degradation problem of deeper ConvNet as described previously. The results in Figure 3 show that the deeper 50-layer ConvNet has much lower training accuracy (23%) than the shallower 25-layer one (86%) throughout the 10k-step training process, and thus the test accuracy. This suggests the existence of optimization difficulty in deep ConvNets. As discussed in [1], since batch normalization ensures the propagated information have non-zero variances, it ameliorates the vanishing gradients problem. The performance deterioration may indicate the fundamental convergence difficulty of deep ConvNets.

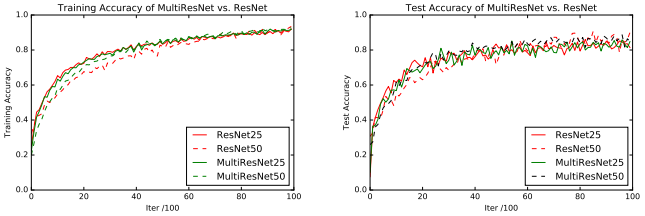
In contrast, ResNet exhibits higher training accuracy when the depth increases from 25 to 50 layers (93.47% vs. 92.03%). A testing accuracy of 84.5% is achieved with the 50-layer ResNet. Although the 50-layer ResNet seems to converge slower than 25-layer one in the early training stage, it grows faster later on and eventually achieves a higher accuracy level. We also find the testing accuracy at 10k-step is improved in a relatively small magnitude with increased depth. But we expect the discrepancy being larger when the algorithm convergences at a later stage with more training steps. Due to time and computation resource limitation, we leave it as one of the future tasks. It worth mentioning the fact that the equivalent ResNet require no extra parameters compared to the ConvNet counterparts. In addition, the ResNet converges faster at the early training stage than the ConvNet. Thus our results support the conclusion that the identity mappings in ResNet effectively help to overcome the optimization difficulty.

### B. HighwayNet vs. ResNet

As shown in Figure 4, HighwayNet also exhibits high training and testing accuracy when it goes deeper, which is contrary to the ConvNet. This demonstrates the effectiveness of highway connections to mitigate the optimization issue. Comparing HighwayNet and the ResNet counterparts, we find relatively lower testing accuracy of is achieved by the HighwayNet (81% vs 84%), although the training accuracy is higher (93.8% vs. 92%). ResNet creates the 'hard-wired' identity shortcuts consistently while the HighwayNet uses the adaptive gate mechanism. It seems might be an disadvantage since the ResNet is no longer adaptive, which is inflexible and more likely to be worse. But actually ResNet can be considered as a special HighwayNet where the gate is never closed, thus assuring the information flow during learning [6]. Additionally, although no longer suffering from the optimization issues, the 50-layer HighwayNet shows relatively lower testing (by 2.1%) and training accuracy (by 2.7%) than the 25-layer one. Combining with the previous result, it indicates the overfitting problem with our HighwayNet. One of the major differences between ResNet and HighwayNet is ResNet doesn't have the extra parameters but HighwayNet introduces the extra gate parameter. This difference may explain why the HighwayNet is more prone to overfit than the ResNet. In addition, as pointed out in [2], the initialization of the bias term in the gating function is critical for training the HighwayNet. It could also be a possible reasoning behind the worse performance of HighwayNets. More experiments on the appropriate initialization is necessary for HighwayNet to achieve better performance.

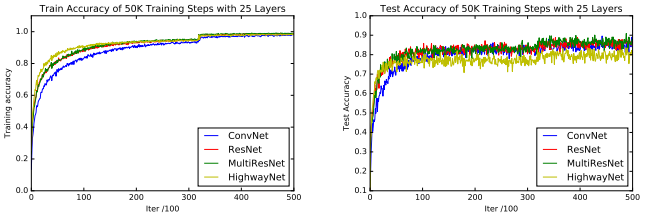
### C. Multi-ResNet vs. ResNet

As shown in Figure 5, MultiResNets also perform better as the depth increases. Moreover, our results show that the MultiResNet slightly outperforms the ResNet counterpart by 1% of testing accuracy in the 50-layer case. Intuitively, MultiResNets are quite similar to ResNets: Equation 4 differs from Equation 2 only in one




(a) (b)

FIG. 5: MultiResNet vs ResNet. Training accuracy(left) and test accuracy(right) on CIFAR10 with 25-layer and 50-layer networks.




(a) (b)

FIG. 6: 50K-step training experiments of 25-layer networks. *Learning rate is divided by 10 after 32k-steps.

aspect, the inputs to $H()$ are concatenated instead of summed. As a direct consequence of this small modification, the feature maps learned by any of the preceding layers can be accessed by their subsequent layers in a collective way [2]. This encourages feature reuse throughout the network, which further improves the information flow between layers. By comparing the testing/training accuracy ratios of MultiResNet and ResNet, which is 92.6% vs. 90.3%, our results indicate that MultiResNet is less prone to overfit compared with residual networks. The more efficient use of parameters of MultiResNet may lead to this advantage.

We also conducted 50k-step training experiments of the 25-layer networks to compare their converged performance, see Figure 6. The final training and testing accuracies are reported in Table II.

| 50K-step | Train Acc | Test Acc |
|---|---|---|
| ConvNet | 97.86% | 84.80% |
| ResNet | 98.71% | 85.96% |
| MultiResNet | 98.87% | 86.10% |
| HighwayNet | 98.27% | 80.38% |

TABLE II: 50k-step Training and Testing Accuracies on CIFAR10

## VI. ADVERSARIAL EXAMPLES

### A. Theory of Adversarial examples

First discovered by Szegedy et al. [7], adversarial examples are perturbations to an input vector such that the network provides high confidence misclassification, or in other words, is fooled. Goodfellow et al. [8] provide a simple way to construct adversarial examples, which is referred to as the "fast gradient sign method":

$$x_{adversarial} = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \tag{5}$$

Goodfellow et al. [8] conjecture that linear models have adversarial examples if the input dimension is large enough. The logic behind this idea is, while one expects that small changes (on the order of the precision of the features) to an input vector will not result in misclassification, the additive property of the dot product can trigger activation. Take, for example, dimension of size $n$, and weight vector $w$ with average element magnitude of $m$. Allowing for a noise floor of size $\epsilon$ (the error introduced in digitizing an analog signal for instance), we can have perturbations on the order of $\epsilon nm$, scaling linearly with input dimension.

Given the conjecture that linear models experience adversarial attacks, it is natural to ask, to which extent are linear-like models, such as ResNet vulnerable to these adversarial examples. We attempt to shed light on this question.

### B. Universal adversarial perturbations

Recently, Moosavi-Dezfooli et al. [9] coined the term "universal adversarial perturbation" (UAP) to describe a single vector that can be linearly added to any input and achieve a high fooling rate.

$$x_{adversarial} = x + v_{uap} \tag{6}$$

Note that unlike the fast gradient sign method, $v_{uap}$ does not depend on the input or the loss function. Imagine a scenario (or game) where a player can only send inputs to classifier and receive outputs (classification), can the party systematically construct adversarial examples? In this case, the player does not have access to the loss function, and thus cannot use the fast gradient sign method, can we do better than perturbing the input randomly? The authors proposed the following algorithm to determine $v_{uap}$.

**Algorithm 1** Original UAP algorithm

**Require:** Data $(x_i,\ y_i)$, Classifier $k$, Max norm of UAP $l_p$, Desired accuracy $\delta$

$v_{uap} \leftarrow 0$
**while** $Err(X_v) \leq 1 - \delta$ **do**
  **for** $x_i \in$ Data **do**
    **if** $k(x_i + v_{uap}) = k(x_i)$ **then**
      $\Delta v_i \leftarrow \arg\min_r ||r||_2$ s.t. $k(x_i + v_{uap} + r) \neq k(x_i)$
      $v_{uap} \leftarrow P(v_{uap} + \Delta v_i)$
**return** $v_{uap}$

Where

$$P_{p,\zeta}(v) = \arg\min_{v'} ||v - v'||_2 \text{ s.t. } ||v'||_p \leq \zeta \tag{7}$$

and $Err(X_v)$ is the desired fooling rate. We propose an alternative approach that enables one to take advantage of trivial parallelization. Since each data point is independent, we can split the optimization problem over data points.

**Algorithm 2** Modified UAP algorithm

**Require:** Data $(x_i,\ y_i)$, Classifier $k$, rank approximation $a$

$v \leftarrow 0$
**for** $x_i \in$ Data **do**
  $v[i] \leftarrow \arg\min_r ||r||_2$ s.t. $k(x_i + r) \neq k(x_i)$
$v_{uap} \leftarrow LRA(v, a)$
**return** $v_{uap}$

Here, $LRA$ represents a low-rank approximation to the vector $v_{uap}$, obtained from the singular value decomposition of matrix $v$:

$$v_{uap} = \sum_i^n f_i v_i \tag{8}$$

where $v_i$ are the right singular vectors and $f_i$ is a user-chosen coefficient. In the work presented here, $f_i$ is set to be at normally distributed random variable with mean set to 0 and variance set to 25. There is likely a clever approach to selecting $f_i$, but that is out the scope of this work and given Figure 7, seems to work surprisingly well.

Given the insight contained in both the fast gradient sign method and the UAP concept, it would seem natural to combine the two ideas (a common theme in ML). Keeping in mind that in doing so we ignore the rules of the game that originally motivated the UAP, we propose two algorithms for generating perturbation vectors. Algorithm 3 simply constructs a matrix $v$ using the perturbations obtained with the fast gradient sign method and then performs the LRA as done in algorithm 2. Algorithm 4 replaces the objective function of algorithm 2 with the gradient sign function, subject to the same constraints. Algorithm 4, while interesting, scales poorly (some class of NP presumably) and thus will not be implemented in this work.

**Algorithm 3** Combined Fast Adversarial algorithm

**Require:** Data $(x_i,\ y_i)$, Loss function $J$, rank approximation $a$

$v \leftarrow 0$
**for** $x_i \in$ Data **do**
  $v[i] \leftarrow \epsilon \text{sign}(\nabla_x J(\theta, x, y))$
$v_{uap} \leftarrow LRA(v, a)$
**return** $v_{uap}$

**Algorithm 4** Combined Fast-UAP algorithm

**Require:** Data $(x_i,\ y_i)$, Classifier $k$, Loss function $J$, rank approximation $a$

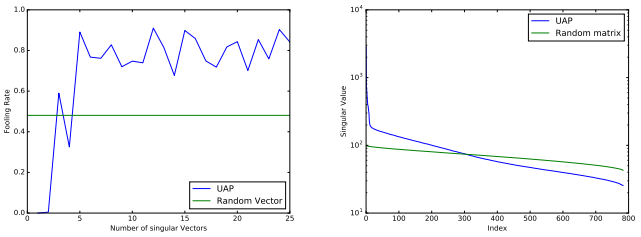
$v \leftarrow 0$
**for** $x_i \in$ Data **do**
  $v[i] \leftarrow \arg\min_\epsilon ||\epsilon \text{sign}(\nabla_x J(\theta, x, y))||_2$ s.t. $k(x_i + \epsilon \text{sign}(\nabla_{x_i} J(\theta, x_i, y))) \neq k(x_i)$
$v_{uap} \leftarrow LRA(v, a)$
**return** $v_{uap}$

### C. MNIST case study

To gain some intuition for the idea of UAPs, we use a five layer, hundred node, fully connected network with ReLu activation for the hidden nodes and a softmax function at the output layer is trained on the MNIST data set (500 samples of each class for training, 50 for validation, 50 for test). A training accuracy of 93% and a testing accuracy 91% is achieved.

Algorithm 1 was implemented, but the procedure was found to be impractically long, even on this relatively small input dimension ($32 \times 32$). Specifically, the projection procedure (see Equation 7) often resulted in the violation of the constraint from the first optimization step, requiring many iterations through the outer loop before achieving convergence. Instead, we propose and develop Algorithm 2, which run using the spicy.optimize library with COBYLA method used to perform the constrained optimization and trivially parallelized using the joblib library.

Our benchmark is the fooling rate achieved by perturbing the input by a random vector. Algorithm 2 outperforms the random vector approach if a sufficient number of singular vectors are used to represent the UAP vector. Moosavi-Dezfooli et al. [9] conjecture that the existence of a UAP vector for given a classifier is the result of the emergence of correlations between the decision boundaries, a claim which is supported by the trend of singular values as a function of index (in [9], the singular values decay more rapidly than those obtained from a random matrix). Here, we observe a similar trend, although the decay is not as rapid as that of [9]. It would be interesting to develop a theory that could predict the scaling of




(a) (b)

FIG. 7: Fooling rate and SVD of perturbation vectors for the 5-layer NN.

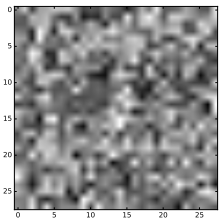
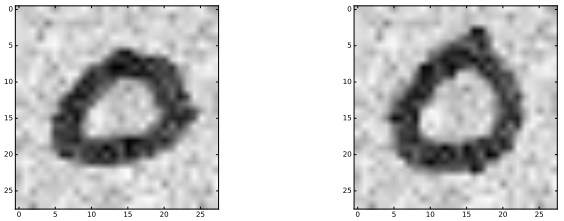
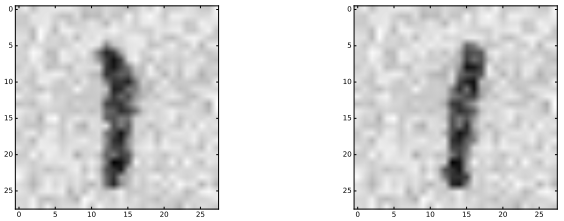


FIG. 8: $v_{uap}$ obtained using Algorithm 2. Scaled for viewing ease.

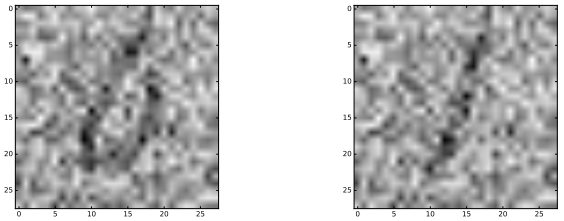


(a) (b)

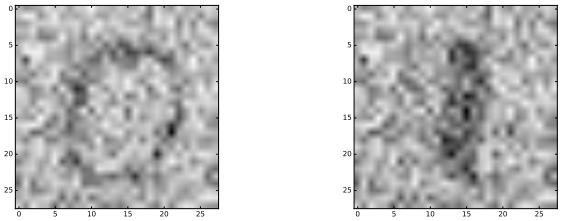


(c) (d)

FIG. 9: Example MNIST images perturbed with the same vector, Figure 8.



(a) (b)



(c) (d)

FIG. 10: Example MNIST images perturbed with a random vector.

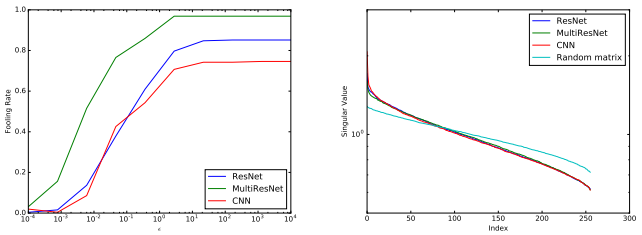
the singular values for a given classifier.

We present some adversarial examples obtained for the MNIST dataset. It is remarkable that with Algorithm 2, we can achieve a higher fooling rate than the random vector while, to the human eye, maintaining a closer likeness to the original image. This speculatively suggests that there is a fundamental difference between the process by which humans recognize object compared to the computation done via deep learning.

## VII. ADVERSARIAL EXAMPLES ON RESNETS

Our final step is the combine ResNets with the adversarial strategies. We attempted to use algorithm 2 to generate UAP for ResNets, but this proved to be too computationally costly. With this in mind, we employ the fast gradient sign method. We use the 10-layer networks that were trained for 10k steps (same structure as those presented in Section IV B with less layers to reduce the overall computational cost). Figure 11 shows the fooling rate as a function of $\epsilon$. The MultiResNet was found to have a greater sensitivity to perturbations than the CNN or the original ResNet, both of which show a similar fooling rate as function of $\epsilon$. This is surprising, since as Ref. [8] convincingly argues, the adversarial mechanism is a strong function of the input dimension, and less so the network architecture. This result warrants further work that is out the scope. Furthermore, applying algorithm 3, a similar scaling behavior to that seen in the MNIST case, was observed here for the singular values of the perturbation matrix.

Table III contains 4 adversarial examples generated using the fast sign gradient approach for different $\epsilon$ on the 10-layer ResNet. Like in the case of the UAP, we observe examples that fool the classifier but remain impercepti-




(a) (b)

FIG. 11: Fooling rate and SVD of perturbation vectors for 10-layer ResNet, MultiResNet, and CNN.

bly changed to the human eye. In the case of the first example, adding a small perturbation leads to a correct classification, while for the cat and bird, the perturbation leads to an incorrect classification. For completeness, a large perturbation is also shown, yielding an image that a human would consider to be noise.

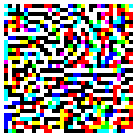
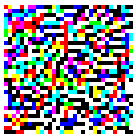
| | | | | |
|---|---|---|---|---|
|  Airplane with 37% confidence | + |  Perturbation $\epsilon = 0.0001$ | = |  Ship with 44% confidence |
|  Bird with 76% confidence | + |  Perturbation $\epsilon = 0.006$ | = |  Cat with 31% confidence |
|  Cat with 54% confidence | + |  Perturbation $\epsilon = 0.046$ | = |  Dog with 78% confidence |
|  Horse with 76% confidence | + |  Perturbation $\epsilon = 2.78$ | = |  Dog with 48% confidence |

TABLE III: Examples of adversarial examples obtained with ResNet with CIFAR-10. Similar examples were observed for the CNN and the MultiResNet.

One can harness these examples to reduce the fooling rates. In Ref.[8], the sign of the gradient is added to the overall loss function as regularizer. This was shown to reduce the error rate on adversarial examples for simple deep neural networks. We expect that the same trend to be recovered for the networks studied here.

## VIII. CONCLUSION

To summarize, we compared the performances of four different neural network architectures. The dependency on the network depth was discussed based on the results of 25-layer vs. 50-layer nets. When compared with the published results of ResNet, HighwayNet and MultiResNet of similar depth, our training accuracies are very close to the reported values, but the testing accuracies are relatively lower. In our experiments, we didn't obtain the average testing accuracy over 90%. In our opinion, this is mainly due to lack of hyper-parameter tuning, especially the regularizations. The results comparison indicates our models overfit. Adding different type and multiple level of shortcut connections can improve the accuracy of the traditional deep ConvNet models in the image classification task and make the training process faster. But the tradeoff is that some of the networks are more prone to overfitting which is undesirable. As for the future improvements on the network architecture, we'd like to 1) add regularizations (such as weight decay and dropout layers) as well as 2) explore the hyper-parameters in details (e.g. varying width and height, or building pyramidal architecture) to achieve state-of-art accuracy of the image classification.

We reviewed the original logic behind adversarial examples and presented the recently proposed concept of a universal adversarial perturbation. We implemented a modified algorithm to determine the UAP for a simple fully connected neural network, showing a fooling rate greater than 80%, comparable to what is found in [9]. We also construct adversarial examples for 10-layer ConvNet, ResNet and MultiResNet, finding sensitivity to the $\epsilon$ parameter in the fast gradient sign method. We propose a combined algorithm to unify UAP and that fast gradient sign method. In the future, it would be interesting to train with the inclusion of these adversarial examples.

Author 1 worked on Sections III and IV. Author 2 worked on Sections VI and VII.

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[2] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[3] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: Multi-level residual networks," *arXiv preprint arXiv:1608.02908*, 2016.

[4] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[5] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *arXiv preprint arXiv:1603.05027*, 2016.

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

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[9] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *arXiv preprint arXiv:1610.08401*, 2016.