Machine Learning Project Report*

Hao Sun
516030910362

Bohong Wu
516030910365

1. Main Ideas

In this project, we are to implement deep networks on the facial expression dataset *fer2013*. We have implemented two basic network backbones. One is simple VGG-19, the other one is from our innovation, which a modified VGG, with privileged information embedded.

Apart from the basic requirements, we have also implemented attacks and defenses algorithms to validate the robustness our network. We first generate an adversarial dataset based on the attack on Public Test dataset. We then compared the accuracy before adding defenses and accuracy after adding defenses on the adversarial dataset.

In terms of the performance of our model, we achieved 72.56% accuracy in Private test set, and 70.97% in Public test set. In attacks and defenses, we successfully improved and enhanced the robustness of our model by DAE pre-processing or Gaussian augmenting.

2. Methods

2.1. Baseline-VGG19

In recent years, deep convolutional neural network has achieved great success in computer vision area. Networks including VGG, ResNet, Googlenet and Densenet have shown competitive results in the image classification area.

In our prject, we used "VGG19" [6] as the backbone of network. The overall structure of VGG is shown in fig. 1.

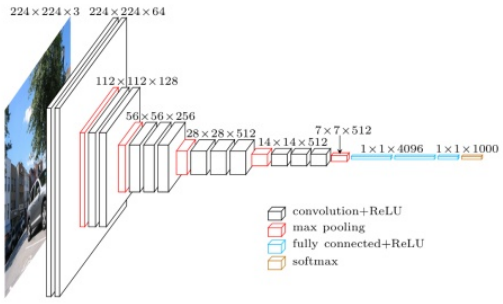


Figure 1. The overall architecture of VGG19

*Our codes can be downloaded in https://github.com/TissueC/ML-Project-2019

2.2. Privileged Information Embedded Network

As deep convolutional neural network is often considered as a black box and lack of interpretability, more and more researches tend to feed the deep network with privileged information to promote the performance. We also incorporate privileged information in our network, and in this section, we will briefly introduce the privileged information we used in our updated network. The overall architecture of the PI-embedded VGG is shown in fig. 2.

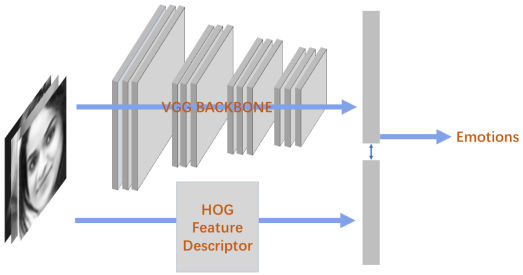


Figure 2. The overall architecture of PI-embedded VGG.

2.2.1 Histogram of Oriented Gradients.

Histogram of Oriented Gradients(HOG)[1] was brought up in 2005 as a image descriptor for human detection. This technique counts occurrences of gradient orientation in localized portions of an image. It is computed on a dense grid of uniformly spaced cells, and uses overlapping local contrast normalization for improved accuracy. The basic procedures of HOG is listed as follows.

Image normalization. Image normalization is a common process for image prepossessing, to improve the robustness of the model against the illumination differences of images, as these images may be taken from different environments. Although in this project, the "fer2013" dataset is already processed into gray scale.

Image Gridding. The gradients calculated are local features of an big image, so the performance might suffer when performing HOG on the raw image. In practice, big images

are usually segmented into many small patches by gridding, and hog features are calculated separately in each patch.

Image gradients calculation. Before calculating the image gradients, the images are usually smoothed by Gaussian filters to remove the noises. But in HOG, image smoothing is not encouraged because this might reduce the contrast in the edges of images, which might be of vital importance in classification. To calculate the gradients, a gradient calculation operator like "sobel" or "laplacian" should be chosen first to perform convolution on the segmented patches. HOG then calculates the amplitudes for gradients on different orientations, which could be written as follows in eq. (1) and eq. (2).

$$M(x,y) = \sqrt{I_x^2 + I_y^2} \tag{1}$$

$$\theta(x,y) = \tan^{-1}\frac{I_y}{l_x} \in [0, 360^\circ) \text{ or } \in [0, 180^\circ) \tag{2}$$

where $M(x,y)$ represents the amplitude of gradient on the certain orientation. HOG then samples these amplitudes on each patch, in order to build the histogram of amplitudes. Finally, the features of each patches are concatenated together as a long $1-d$ vector.

3. Algorithm: Stochastic Gradient Descent

Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It is called stochastic because the method uses randomly selected (or shuffled) samples to evaluate the gradients, hence SGD can be regarded as a stochastic approximation of gradient descent optimization.

Algorithm 1: Stochastic Gradient Descent

Input: An object function $Q(\mathbf{w})$, cross entropy loss in this experiment
Output: An approximate local solution $\mathbf{w}$
Initialization: Choose an initial vector of parameters $\mathbf{w}$ and learning rate η;
while *an approximate minimum is not obtained* **do**
  Randomly shuffle examples in the training set;
  for $i = 1, 2, \cdots, n$ **do**
    $w := w - \eta \nabla Q_i(w)$
  end
end

Generally and in this experiment, for getting a higher accuracy, we set a max_epoch and run the program above many times to seek a checkpoint which has the highest accuracy.

4. Experiments

The codes can be downloaded in https://github.com/TissueC/ML-Project-2019. You can have an overview of our experiments there.

4.1. Dataset

We use the provided dataset **fer2013** [2] to do our experiments. In this section we will briefly introduce the fer2013 dataset. The dataset is divided into 3 parts *training set, private test set, and public test set*. The composition and categories are shown as table 1.

Table 1. The information of dataset fer2013

	Sample size	Seven categories
Training set	28709	0=Angry 1=Disgust
Private test set	3589	2=Fear 3=Happy 4=Sad
Public test set	3589	5=Surprise 6=Neutral

Data prepossessing For data in "fer2013.csv", we first split them in to "training.csv", "public.csv", "private.csv" for training, validating and testing respectively. Afterwards, we resize each image into (48,48), and duplicate these images 3 times on the channel dimension, in order to make the data into "RGB" format, making our network more adaptative to other datasets. We then perform **Image Cropping** to randomly crop the image into a shape of $(44, 44, 3)$, and **Random Flipping** to make our model more robust to the spatial transformations of datasets. Finally, we perform normalization on the datasets.

4.2. Experimental Settings

We use Stochastic Gradient Descent(SGD) in section 3 as the network optimizer with a momentum of 0.9. As for learning rate, we adopt the **learning rate decay** strategy. In the first 60 epochs, the learning rate is set to 0.01 to accelerate the converge speed. Afterwards, the learning rate decays with a ratio of 0.9 every 5 epochs. Also, to avoid the "Gradient Explosion" problem caused by depth of network, we adopt **Clip Gradient** strategy, by setting the maximum gradient to 0.1. The learning batch size is set to 128 and the number of epochs is set to 200. For simplicity, the parameter settings is shown in table 2.

4.3. Results and Analysis

The results of both simple VGG19 and HOG-embedded VGG19 is shown in table 3.

From the tables above, we could see that the HOG-embedded VGG19 performs slightly better than the simple VGG network on the Public Test dataset, with a 0.5 percentage rise. Also, they share similar performance on the private test dataset.

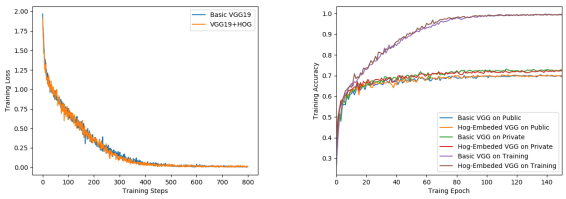
To better present the results, the loss curves and accuracy curves of both models are shown in fig. 3.

Table 2. Parameter Settings for both VGG and HOG-embedded VGG.

Parameters	Values
learning rate	0.01
decay start epoch	60
decay every epoch	5
decay rate	0.9
batch size	128
Momentum	0.9
clip-gradient	0.1
total epochs	200

Table 3. The results of our models on fer2013 dataset.

Type	Basic	HOG-embedded
Private Acc	72.67%	72.56%
Public Acc	70.44%	70.97%



(a) Loss Curves (b) Accuracy Curves

Figure 3. Loss and Acc Curves for basic VGG19 and hog-embedded VGG19.

Both networks share similar curves. We could see that the accuracy on the training set reaches over 90%, but the accuracy on the test set stays around 70%. It is because *fer2013* is relatively a hard dataset, with many samples mislabeled, and many samples that are even hard for humans to classify. Also, we have calculated the number of samples for each class on public dataset, which is listed in table 4. We could see that the numbers of samples for each class are **heavily unbalanced**, which is also a cause that why the classification result is not that good.

Table 4. Number of samples for each class on Public Test.

Class	Angry	Disgust	Fear	Happy
Sample size	448	33	438	1043
Class	Sad	Surprise	Neutral	
Sample size	775	311	541	

To illustrate the advantages of PI(HOG)-embedded networks, we zoomed in the accuracy curves to compare the differences, which is shown in fig. 4. From the curves above, we could find that the starting accuracy of HOG-embedded model is over 40%, which is much higher than simple vgg model. We think that it's because the privileged

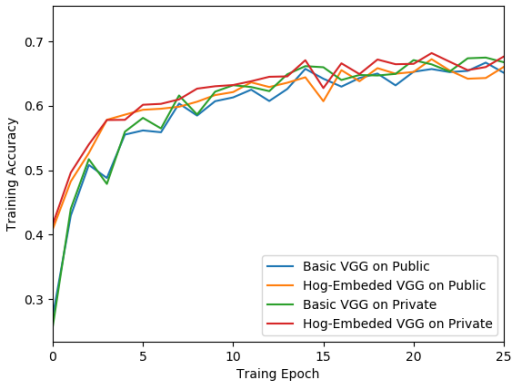


Figure 4. The zoomed in accuracy curved on both networks for both training set and test set respectively.

information guided the process of training a network, thus making the network converge faster.

Also, to better visualize when the network tends to make mistakes, we have also done error analysis on the Public Test dataset, which is shown in fig. 5. From the figure, we could easily see that the network suffers on classifying "Disgust" images, and is tend to label them "Angry". It is partially because that the number of "Disgust" images is relatively too small, which only contains 33 examples on Public Test, and "Disgust" has a close relationship with "Angry".

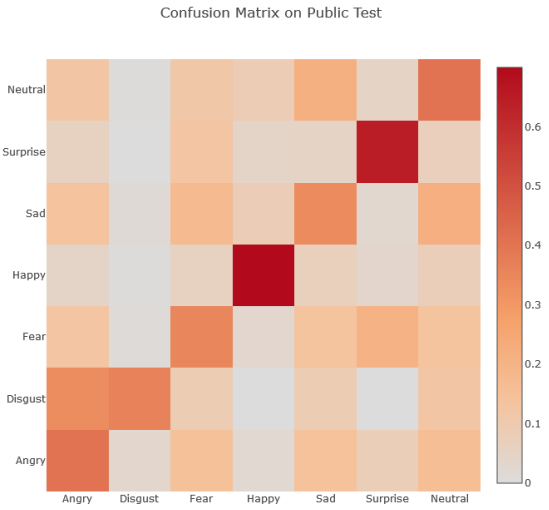


Figure 5. The confusion matrix on Public Test. The y-value represents the true labels, while the x-value represents the predicted labels.

5. Additional Experiments on Attack and Defenses of Neural Networks

Deep neural networks are found to be vulnerable to adversarial samples. These samples are imperceptible to human, but can easily fool deep neural networks. Recently there have been a number of methods proposed to attack the deep neural networks includes Deep Fool[5] and Fast Gradient Sign Method[3], and in consequences, many defense algorithms have also been introduced. In this project, we have also done auxiliary experiments on the additional assignment.

5.1. Attacks

In this section, we will introduce multiple attack methods used in our experiments, and explain the consequent results.

First of all, attacks consist of targeted attack and untargeted attack. For untargeted attack, the image will be processed to confuse classifier to make it into another category, which is not specific. For targeted attack, the attack on a targeted target makes it easier for the image to be recognized by the classifier as a specific category. We do both of targeted and untargetd attack in the following two attack methods, ℓ_∞ PGD Attack and FGSM attack

5.2. Linf PGD Attack

5.2.1 Attack Theory

[4] proposes Projected Gradient Descent (PGD) and shows that PGD is a universal adversary among first-order approaches, which is multi-step variant k-FGSM (k represents the number of iterations). The general idea is that FGSM only makes one iteration and takes a big step, while PGD makes several iterations and takes a small step each time. The loss function of PGD is

$$x^{t+1} = \Pi_{x+S}\left(x^t + \alpha \operatorname{sgn}\left(\nabla_x L(\theta, x, y)\right)\right) \qquad (3)$$

Linf PGD Attack is short for the projected gradient descent attack with linf order, where "linf" order means L-infinite norm. For a vector $\mathbf{x}$, the L-infinite norm is $\max\{|x_i|\}$.

An artificially constructed perturbation, that is imperceptible to human, can cause a significant drop in the prediction accuracy of an otherwise accurate network. The accuracy on the perturbed data depends on the magnitude of the perturbations, L-inifinite ℓ_∞ is an example.

5.2.2 Attack result

We experimented with our previous trained model. And the settings are that the number of iterations are 40, the maximum distortion is 0.15, and the attack step size is 0.01. We selected five figures from the test set to visualize the result as fig. 6

Figure 6. Linf PGD attack. The first row consists of the clean five figures, which is the original figure from test set without processing. The second row consists of the five faces after untargeted linf PGD attack. And the last row consists of the five faces after targeted linf PGD attack. We can see that by attacking, there is no difference in the expression of faces with some noise that doesn't affect our artificial judgment. And for untargeted attack, the misjudged category is fluid, while for targeted attack, we choose the target is "Happy class", we can see that all these five attacked faces are classified as happy face.

5.3. Fast Gradient Sign Method

5.3.1 Attack Theory

In the deep neural network phase, small difference in the high-dimensional training data might result in huge bias in classification tasks. [3] illustrated that with simple **linear model**, it could still have adversarial samples when the input features have enough dimensionality.

Usually, the digital images use only 8 bits per pixel, so information below $1/255$ of the dynamic range is discarded, resulting in the limited precision of the image features. Therefore, it makes no sense for classifier to respond differently to an input x than an adversarial input $\tilde{x} = x + \eta$ if every element of the perturbation $\eta < precision\, of\, features$. However, consider the dot product between a weight vector w and an adversarial sample $\tilde{x}$:

$$\boldsymbol{w}^\top \tilde{\boldsymbol{x}} = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{\eta}$$

We could see that the adversarial perturbation causes the activation to grow by $\boldsymbol{w}^\top \boldsymbol{\eta}$. We could then maximize this increase error by setting constraints on the adversarial perturbation $\boldsymbol{\eta}$.

Therefore, Goodfellow *et al* assumes that the deep networks are too linear(LSTMs, ReLUs, etc.) to resist linear perturbation, and proposed a new linear-based attack method Fast Gradient Sign Method(FGSM) to generate lin-

ear adversarial samples. This method could also be generalized in non-linear models(Sigmoids etc.) with careful fine-tuning.

Denote θ to be the parameters of a model, x to be the input to the model, y to be the labels associate with x, and η to be the adversarial perturbation. We also denote $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ to be the cost function of the network. FGSM constraints the perturbation η to have the same direction with the gradient, which will increase the loss function, result in the maximum change of the classification result, shwon in fig. 7. Therefore, the formula of generating adversarial samples by

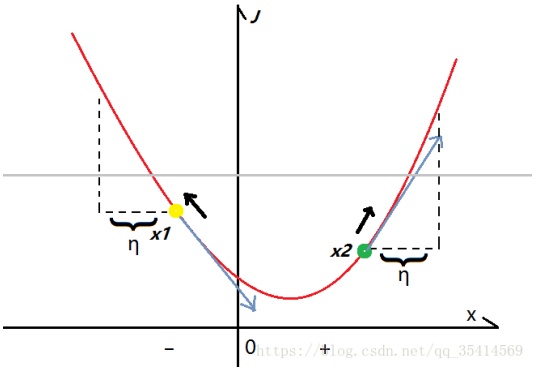


Figure 7. The process of FGSM. we could see that if sample x_1 and x_2 follows the gradient of loss function, they will generate the adversarial samples with maximal error.

FGSM is shown as follows in eq. (4).

$$\boldsymbol{\eta} = \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right) \tag{4}$$

5.3.2 Attack Result

Similar as PGD attack, we experimented with our previous trained model. And the settings are that the maximum distortion is 0.02, and the attack step size is 0.01. It's worth mentioning that the FGSM does not need iterations, which is the main reason that FGSM runs much faster than PGD. The result of visualization is shown as fig. 8, the accuracy can be seen in section 5.5.

5.4. Sensitivity on the noise of attack

For exploring the impact of noise on the classification result. We did experiments about sensitivity on the noise of attack, including two attack methods: linf PGD attack and FGSM. We select a cute face and the following experiments results are form that. Almost all faces in the dataset have the similar sensitivity on the noise. The sensitivity on the noise of attack linf PGD attack is shown as fig. 9.

From the curves and the cute baby face result we can see that the classification is pretty sensitive on the noise of attack. And there always exists a mutation point which is the key to change the result of classification of our network.

Figure 8. Fast Gradient Sign Method result. The first row consists of the clean five figures, which is the original figure from test set without processing. The second row consists of the five faces after untargeted FGSM attack. And the last row consists of the five faces after targeted FGSM attack. We can see that by attacking, there is no difference in the expression of faces with some noise that doesn't affect our artificial judgment. And compared with PGD, the noise is much fewer. The attack runs pretty fast, the cost of which is that the effect of attack is not as good as linf PGD attack. We can see that in the second row, two figures are still correctly classified. And in the third row, the targeted attack specify the classification "Fear class", but there exists two faces are not labeled as specified by attack.

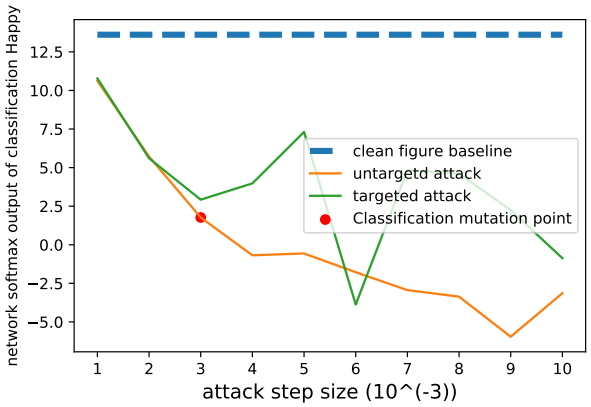


Figure 9. The sensitivity result of attack linf PGD attack. The untargeted attack is more regular than targeted attack. The red point means that from the red point, the classification of this baby face has been changed because network cannot stand the attack.

The sensitivity on the noise of attack FGSM is shown as fig. 12.

Figure 10. The classification result of a baby face by linf PGD attack with eps=0.001. From left to right, the faces are original face, face under untargeted attack, and face under targeted attack.

Figure 11. The classification result of a baby face by linf PGD attack with eps=0.01. From left to right, the faces are original face, face under untargeted attack, and face under targeted attack.

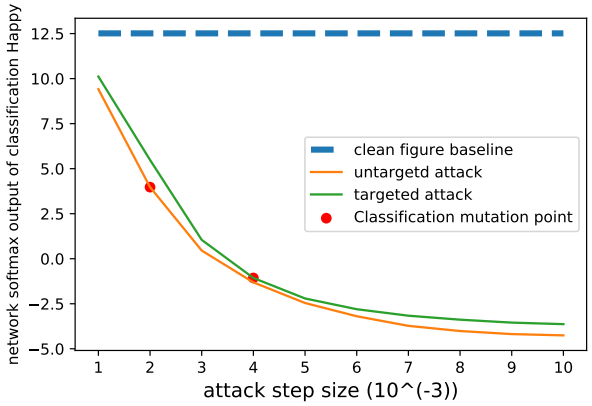


Figure 12. The sensitivity result of attack FGSM attack. Compared with PGD, both of the untargeted attack and targeted attack more regular. We can find that the mutation point is not the same while the curves of them are exactly similar. We guess this is because our network can stand more noise when faced with targeted attack.

Figure 13. The classification result of a baby face by FGSM attack with eps=0.001. From left to right, the faces are original face, face under untargeted attack, and face under targeted attack.

5.5. Defenses

For simplicity, we only used the "Public Test" dataset for testing in this part. **We also didn't use the ten-crop accuracy used in the previous experiments for simplicity,**

Figure 14. The classification result of a baby face by FGSM attack with eps=0.01. From left to right, the faces are original face, face under untargeted attack, and face under targeted attack.

which results in a little lower accuracy. In this subsection, we have done 3 experiments with different ideas on defensing against the attacks introduced in the previous section.

5.5.1 Train Samples with Gaussian Noises.

One simple way to defense against Black-box attack is to augment the training data. Here we chose to augment the training data with random noise, which turns out to have a relatively good performance on the adversarial datasets generated by the previous attack methods. We have kept the basic structure of VGG. However, before feeding the raw training data into the network, we add Gaussian noises ϵ, with *mean* set to 0 and *std* set to 0.05. The relative results is then shown in table 5.

Table 5. The accuracy for raw VGG and Gaussian enhanced VGG on both the raw public test datasets and the adversarial datasets generated.

Dataset	Public Test	Adversarial Test
Gaussian Augmented	68.32%	54.86%
Simple VGG	65.28%	49.04%

From the results, we could see that augmenting the original datasets with Gaussian noise is not only helpful in classifying the adversarial samples (**with a 5% lead**), but also helpful in classifying the original datasets(**with a 3% lead**). Thus, augmenting the original dataset with gaussian noise is effective in defensing the Black-box attack methods.

5.5.2 Preprocess the Images with Denoising Auto Encoder.

There exists defensing methods which aims at preprocessing the adversarial images, in order to make the adversarial images more "clean". Methods include mean filtering, feature squeezing shows competitive results in this area. In our experiments, we have tried to denoise the adversarial images with a simple denoising autoencoder(DAE)[7].

We have designed a simple 14-layer autoencoder, 7 for encoding and 7 for decoding. To train the autoencoder, we set the raw image augmented with gaussian noise as input,

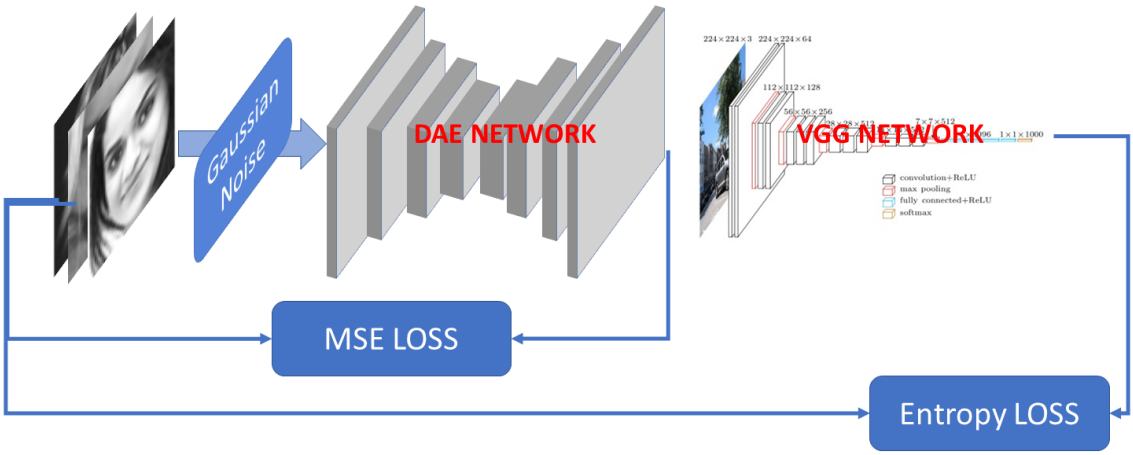


Figure 15. The network structure of the DAE-combined VGG. Samples are first augmented with gaussian noises. After processed by DAE, the network will generate a middle output as the denoised clean samples. Then VGG will perform classification tasks on the denoised samples. Loss functions of this network is coupled with reconstruction error(defined by MSE loss) and the classification error(defined by cross entropy).

and the original raw image as the output. Then, for a testing image, this image is first fed into DAE to reconstruct a clean image, and then processed by the classification network.

The relative results is shown in table 6.

Table 6. The accuracy for raw VGG and DAE preprocessed VGG on both the raw public test datasets and the adversarial datasets generated.

Dataset	Public Test	Adversarial Test
DAE preprocessed	49.04%	46.64%
Simple VGG	65.28%	49.04%

From the dataset, we could see that although this method performs bad on both the raw image dataset and the adversarial datasets, but the accuracy difference Δ between the raw images and the adversarial difference is very small(**less than 3%**). It means that this method could partially solve the attack problem, as the attack's influence only causes a 3% decrease on this dataset.

The reason why the base accuracy on the raw image dataset is low could be summarized in two aspects.

- **The DAE network is relatively simple.** Because of the limit of time, we only implement a very shallow and simple Denoising Auto Encoder, which might be the cause why the base accuracy on the raw image dataset is relatively low.
- **Errors are accumulated during the reconstruction phase and the classification phase.** In this experiments, there are two neural networks, which might lead the error of these two neural network accumulates, resulting in a bad performance. This problem will be fixed in the next section.

5.5.3 Modifying Network Structure by combining DAE and VGG.

From the previous section, we could see that the effectiveness of DAE is tightly limited, as the errors are accumulated during both DAE's denoising task and VGG's classification task. Thus it is a natural idea that these two networks could be coupled in a single network, which denoises the input images and classify the types at the same time.

The network structure is then shown in fig. 15. Still, we add random gaussian noise to the training samples first, and process these samples by DAE, which returns a denoised middle output. Afterwards, we use VGG to process this middle output, and returns a classification result. The loss function are combined with both **Mean Squared Error** produced by DAE and **CrossEntropy** produced by the classification task.

The results of this modified network is shown in table 7, compared with the basic network structure.

Table 7. The accuracy for raw VGG and DAE+VGG on both the raw public test datasets and the adversarial datasets generated.

Dataset	Public Test	Adversarial Test
Modified VGG	65.12%	57.15%
Simple VGG	65.28%	49.04%

From table 7, we could clearly see that the modified VGG structure combined with DAE has a much better performance than simple VGG on the adversarial datasets. On classification result on the raw dataset, it only shows a 0.16% decrease. However, **on the classification result on the adversarial dataset, it shows a 8% increase.**

Compared with the previous section, we could also conclude that coupling the reconstruction task and classification task with result in a better performance: Not only the classification result on the raw dataset gets better than the decoupled networks(**with a 16% increase**), the classification result on the adversarial dataset has also shown giant growth(**with a 11% lead**).

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005.

[2] I. Goodfellow, D. Erhan, P.-L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio. Challenges in representation learning: A report on three machine learning contests, 2013.

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

[4] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2018.

[5] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.