Adversarial attacks and Defences in Convolutional Neural Network
A Comprehensive Tutorial

- By: Sutharshan shanmugarajah (23092351)

Intended Learning Outcomes

By the end of this tutorial, learners will be able to:

- Explain the concept of convolutional neural network is and their real-world applications.
- Classify and describe the different types of adversarial attacks that can deceive CNN models.
- Identify and analyse the defensive mechanisms to enhance model robustness against the adversarial attacks.
- Implement and train a CNN model to classify handwritten digits using the MNIST dataset from Keras.
- Apply the Fast Gradient Sign Method (FGSM) to generate adversarial attacks on the trained model.
- Implement and evaluate defence strategies to mitigate adversarial attacks effectively.

What is CNN?

Convolutional Neural Network is a category of neural network in the areas like image recognition and audio video recognition. It's a deep learning technique that was created just like how the human brain thinks and performs as human brain is made up of millions of neurons.

During the process of convolutional neural network, the input image will be filtered through the number of layers like, kernel, stride, padding, pooling, and fattening.

The major advantage of the CNN could be, it reduces the image damageability as much as possible so we can get the outcome as expected.

Introduction to Adversarial attacks

These are the attacks that vulnerable to the CNN models where small, imperceptible perturbations to input images can cause a CNN to misclassify them.

These adversarial attacks poses the serious security risks in applications like autonomous vehicles, biometric security, and healthcare AI systems. For example, to the security thread, lets say in a company, the employees will be allowed inside the work station under the security face read scanner every time otherwise the access will be denied. That face scanning system was built by using the CNN. In that scenario, a hacker or a outside attacker can wear a mask of containing the similar facial feature of a employee and scan Infront of the face scanner and could be a attacker to the CNN. This example can be clearly explained on about the biometric security thread in a corporate industry.

Key concepts covered in this tutorial:

A. Adversarial examples: Inputs that have been intentionally modified to deceive a neural network.
B. Attack methods: Techniques used to generate adversarial examples
C. Defencive mechanisms: Strategies to make the CNN model robust against the attacks.

In this tutorial, we will explore and implement the common attack Fast Gradient Sign Method (FGSM) and used a defence mechanism – Adversarial Training to make the CNN model robust against the attack.

A. *Types of Adversarial attacks*

Adversarial attacks classified into Whitebox and Blackbox attacks based on the attacker's knowledge of the target model.

1. **Whitebox attack**
   The attacker has full knowledge of the CNN architecture, parameters and gradients.
   E.g.:
   - Fast Gradient Sign Methd (FGSM): Computes the gradient of the loss with respect to the input and adds a small perturbation in the direction that increases the loss.
   - Projected Gradient Descent (PGD): A more powerful iterative version of FGSM and expensive method as it needs multiple iterations to cover.
   - Carlini & Wagner (C&W) Attack: Optimizes perturbations to minimize detection.
2. **Blackbox attack**
   The attacker does not have access to model details and relies on alternative strategies.
   E.g.:
   - Transfer Based Attacks: Generates adversarial examples on one model and transfers them to another.
   - Query-Based Attacks: Uses limited queries to probe the model and construct adversarial inputs.

B. *Implementing an adversarial attack*

<u>Step 1:</u> Load and preprocess the MNIST dataset
MNIST dataset is a handwritten numerical dataset of images produced by Keras that should be load using the following command

```
tf.keras.datasets.mnist.load_data()
```

Once the dataset loaded need to perform few sorts of preprocessing like, normalise and reshape the image dataset to make the data clean to ready to use.
As we are using the numerical data, we need to convert the dataset labels to categorical format. This is necessary because the model is using categorical cross entropy as its loss function and a SoftMax activation function in the output layer.

```
tf.keras.utils.to_categorical(y_train,10)
```

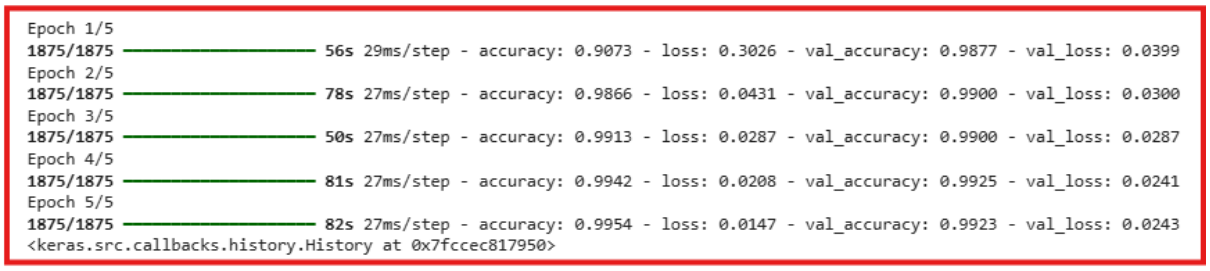
it converts each label into a binary vector of length 10, where the correct class is marked as 1 and all others as 0.

<u>Step 2:</u> Build and train a CNN model for digit classification
Once we finish the preprocessing steps, we need to train the model using the CNN technique as we need to let through number of layers. This is the main step of CNN technique that extracts important features from images, reduces unnecessary details using pooling, and finally classifies digits using fully connected layers.

```
def build_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3,3), activation='relu', in-
put_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    return model
```

the above function `build_model()` implemented to do the training process,

- ✓ First Convolution layer applies 32 filters of size 3 x 3 to extract low level features like edges and textures and uses ReLU activation to introduce non-linearity.
- ✓ First Pooling layer, down samples feature maps by taking the maximum value in 2 x 2 regions, reducing spatial dimensions and computational cost.
- ✓ Second Convolutional layer uses 64 filters of size 3 x 3, extracting more complex patterns from the previous layer.
- ✓ Second Pooling layer further reduces spatial dimensions while retaining important features.
- ✓ Flattening layer converts the 2D feature maps into a 1D vector for the fully connected layer.
- ✓ Fully connected (Dense) layer uses ReLU activation to introduce non-linearity with 128 neurons for learning complex patterns.



```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 56s 29ms/step - accuracy: 0.9073 - loss: 0.3026 - val_accuracy: 0.9877 - val_loss: 0.0399
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 78s 27ms/step - accuracy: 0.9866 - loss: 0.0431 - val_accuracy: 0.9900 - val_loss: 0.0300
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 50s 27ms/step - accuracy: 0.9913 - loss: 0.0287 - val_accuracy: 0.9900 - val_loss: 0.0287
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 27ms/step - accuracy: 0.9942 - loss: 0.0208 - val_accuracy: 0.9925 - val_loss: 0.0241
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 27ms/step - accuracy: 0.9954 - loss: 0.0147 - val_accuracy: 0.9923 - val_loss: 0.0243
<keras.src.callbacks.history.History at 0x7fccec817950>
```

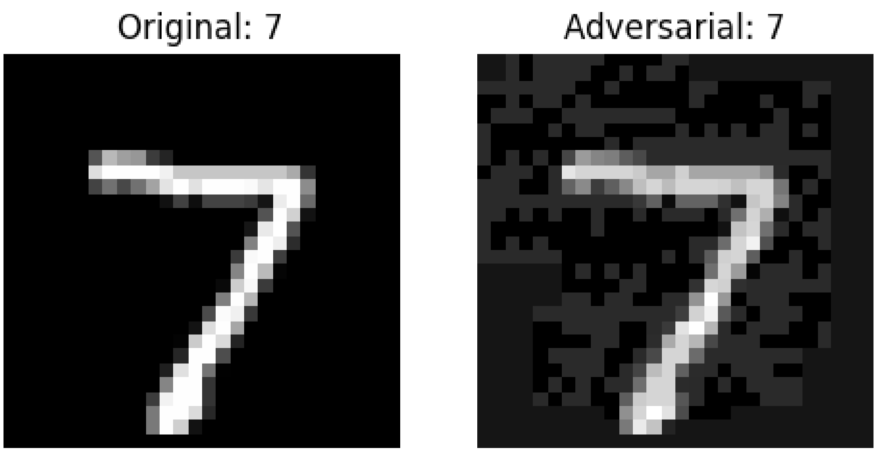
The above snapshot shows the training progress of the CNN model over 5 epochs while training on the MNIST dataset. Since MNIST has 60000 training images and the default batch size is 32, so the total steps per epoch are, 60000 / 32 = 1875 and each epoch takes between 50 – 82 seconds depending on system performance.

Step 3: Generate and analyse adversarial examples using FGSM

Next, we have to implement a function to generate an adversarial image that maximally increases the model's loss while remaining visually to the original and it should exploits the model's sensitivity to small changes leading to the misclassification.

```
def fgsm_attack(image, epsilon, gradient):
perturbation = epsilon * np.sign(gradient)
return image + perturbation
```

The function `fgsm_attack()` implements the Fast Gradient Sign Method an adversarial attack purposely used to make the model confuse in prediction of value. A small value controlling the attack's strength (perturbation magnitude) is the epsilon value used in the attack mechanism and the gradient of the loss of input image indicating how to modify the image to increase the loss.
`epsilon * np.sign(gradient)` Determines the direction of the gradient (whether positive or negative) and scales the direction by epsilon, creating a small but effective perturbation.
Finally, the function returns the added values of perturbation with the original image creating an adversarial example.



The above diagram shows the output of the adversarial attack with the noised image and the predicted values in the top. As this example says this attack doesn't affect the prediction.

C. *Implement the defensive mechanism – Adversarial Training*

The defence mechanism used in this code is Adversarial Training, where the model is retrained with both original and adversarially perturbed examples. This helps the model learn to recognise and correctly classify adversarial inputs, making it more robust against attacks like FGSM.
The code snippet below explains how the adversarial examples integrate into the training dataset to improve the model's robustness.

```
x_train_adv = np.array([fgsm_attack(x, 0.1, gradient) for x in
x_train])
x_train_adv = x_train_adv.reshape(x_train.shape)
x_train_combined = np.concatenate([x_train, x_train_adv])
y_train_combined = np.concatenate([y_train, y_train])

model.fit(x_train_combined, y_train_combined, epochs=5, valida-
tion_data=(x_test, y_test))

defended_prediction = model.predict(adversarial_image)
defended_label = np.argmax(defended_prediction)
```

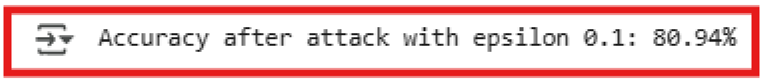
- ✓ The function `fgsm_attack()` is applied to each training image to create adversarially perturbed images.
- ✓ The adversarial examples are reshaped to match the original dataset shape. Both the original images and adversarial examples are combined into `x_train_combined` , effectively doubling the dataset.
- ✓ Then the model is retrained using the combined dataset (`x_train_combined`, `y_train_combined`) improving its ability to handle adversarial attacks.
- ✓ The retrained model is tested on a adversarial image to see if it correctly classifies it after training.
- ✓ The final predicted label is obtained to compare whether the model has successfully defended against the attack.

Finally, the full process of prediction need to be evaluated as follows,

```
def evaluate_model(model, x_test, y_test, epsilon):
    correct = 0
    for i in range(len(x_test)):
        # Convert image to a TensorFlow tensor before watching it
        image = tf.convert_to_tensor(x_test[i:i+1])
        label = y_test[i:i+1]

        with tf.GradientTape() as tape:
            tape.watch(image)
            prediction = model(image)
            loss = tf.keras.losses.categorical_crossentropy(label, pre-
diction)

        gradient = tape.gradient(loss, image)
        adversarial_image = fgsm_attack(image, epsilon, gradient)
        adversarial_prediction = model(adversarial_image)

        if np.argmax(adversarial_prediction) == np.argmax(label):
            correct += 1

    accuracy = correct / len(x_test)
    print(f"Accuracy after attack with epsilon {epsilon}: {accuracy *
100:.2f}%")
```

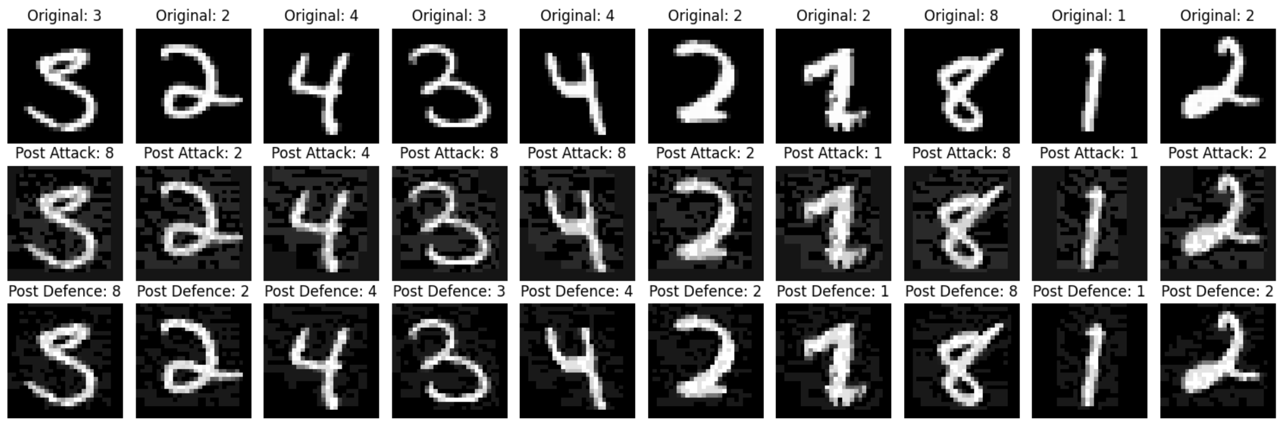
By executing the above function `evaluate_model ()` could be able to calculate the overall accuracy by setting the epsilon value

```
evaluate_model(model, x_test, y_test, epsilon=0.1)
```



```
Accuracy after attack with epsilon 0.1: 80.94%
```

This result shows that after applying an FGSM adversarial attack with the epsilon 0.1 , the model's accuracy dropped to 80.94%.



This means that the adversarial perturbations significantly affected the model's predictions, causing some misclassifications. Before the attack, the model likely had a much higher accuracy near 99% so the attack was successfully fooled the model to some extent, proving it's vulnerability. The higher epsilon value would likely decrease accuracy even further, making the attack stronger.

CONCLUSION

Adversarial attacks pose **serious challenges** to CNNs, especially in handwritten digit recognition applications. This tutorial covered:

- The concept of adversarial attacks and different attack types.
- Implementing an **FGSM attack** on the **MNIST dataset**.
- Defence mechanisms like **adversarial training**.
- Evaluating CNN robustness with before-and-after attack analysis.

Understanding adversarial attacks is essential for building **secure and reliable AI models** in the future.

REFERENCES

Git Repository: https://github.com/Sutharshan-art/Machine-Learning-Tutorial

1. Thangaraju and C. Merkel, "Exploring Adversarial Attacks and Defenses in Deep Learning," 2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 2022, pp. 1-6, doi: 10.1109/CONECCT55679.2022.9865841

2. Zhang, P. Benz, T. Imtiaz and I. S. Kweon, "Understanding Adversarial Examples From the Mutual Influence of Images and Perturbations," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 14509-14518, doi: 10.1109/CVPR42600.2020.01453.

3. K. Ren, T. Zheng, Z. Qin and X. Liu, "Adversarial attacks and defenses in Deep Learning", *Engineering*, vol. 6, no. 3, pp. 346-360, 2020, [online] Available: https://doi.org/10.1016/j.eng.2019.12.012.