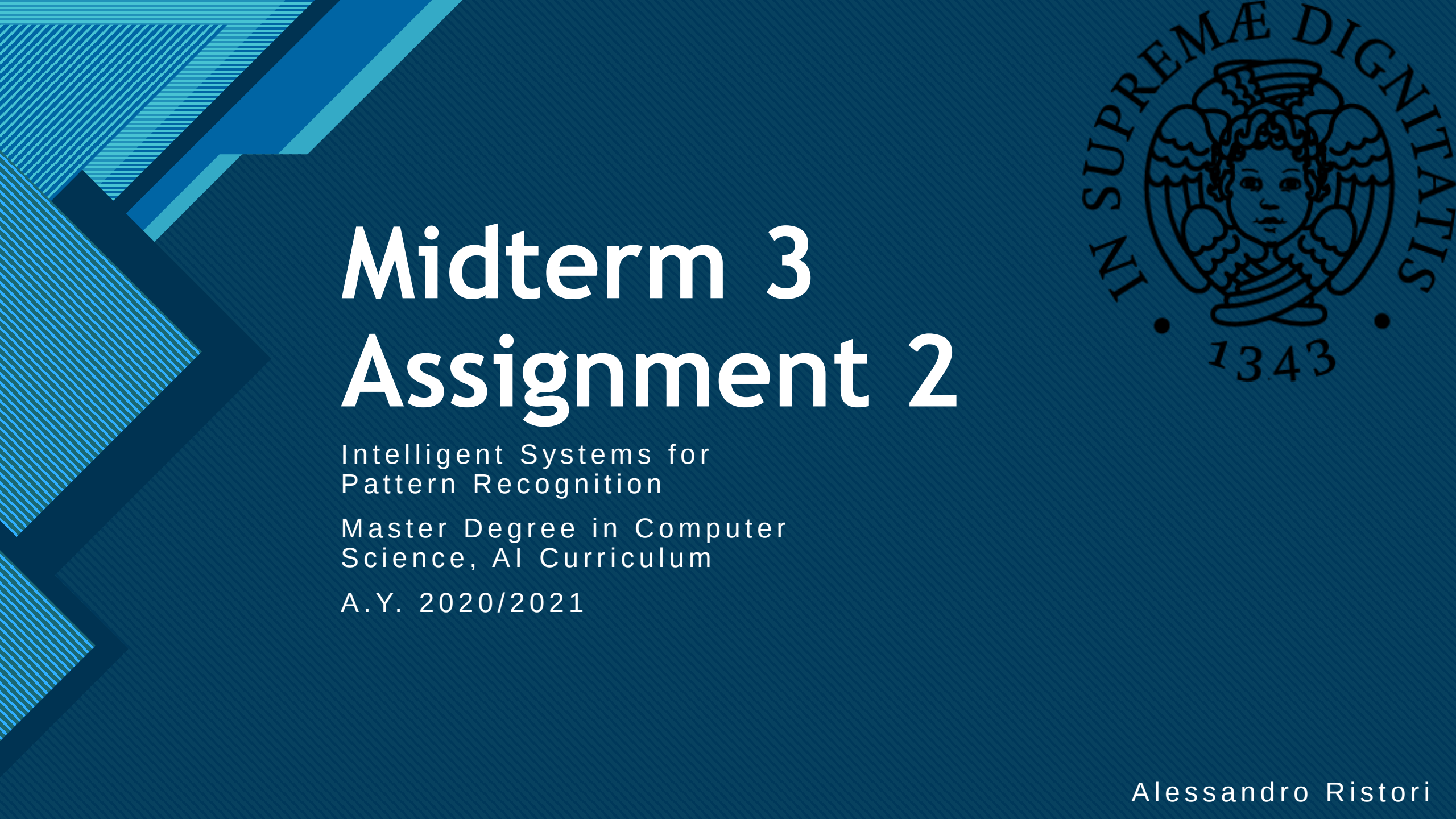


Midterm 3 Assignment 2

Intelligent Systems for Pattern Recognition

Master Degree in Computer Science, AI Curriculum

A.Y. 2020/2021

Alessandro Ristori

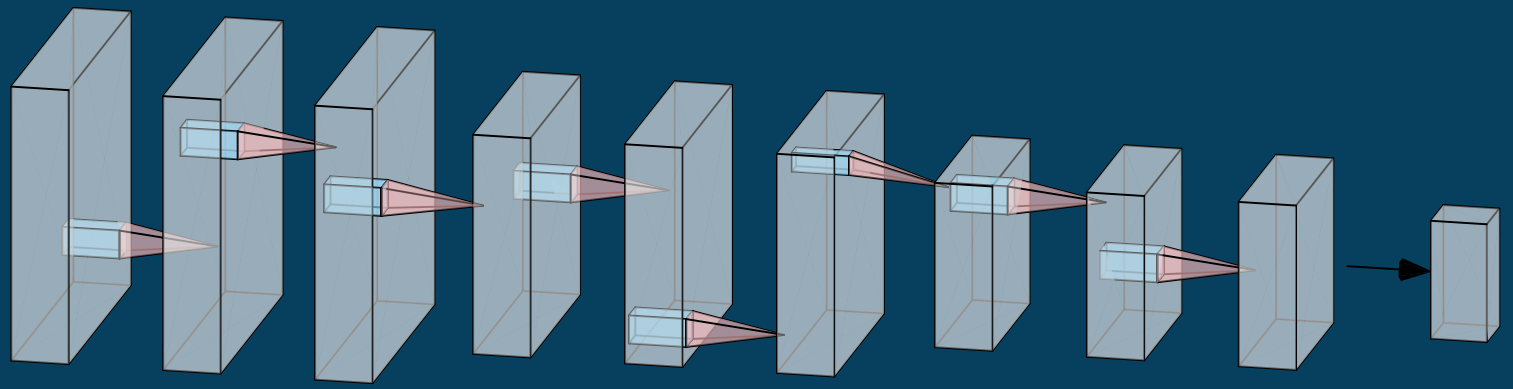
Code snippets

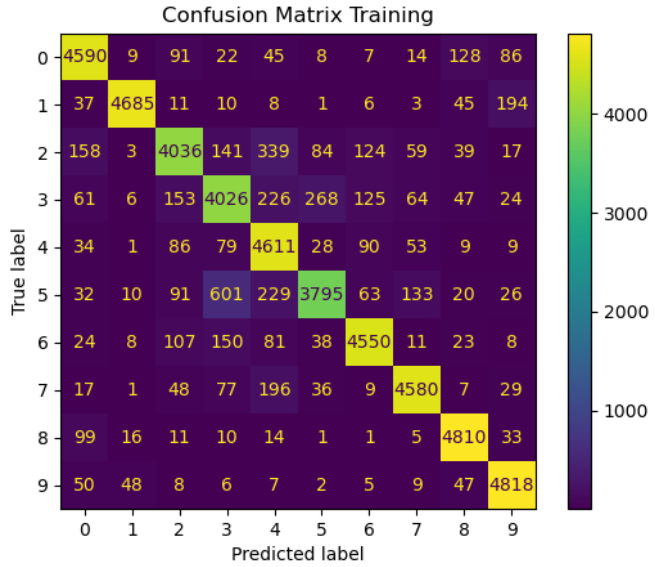
```python
def adversary_pattern(model, pattern, label, eps, show_noise=False):
    pattern = tf.cast(np.reshape(pattern, (1, 32, 32, 3)), tf.float32)
    with tf.GradientTape() as tape:
        tape.watch(pattern)
        pred = model(pattern)
        loss = MSE(label, pred)
        gradient = tape.gradient(loss, pattern)
        signed_grad = tf.sign(gradient)
        adversary = (pattern + (signed_grad * eps)).numpy()
        if show_noise:
            signed_grad = (signed_grad * eps).numpy()
            plt.imshow(np.clip(signed_grad.reshape(32, 32, 3) * 255, 0, 1))
            plt.show()

        return adversary
```

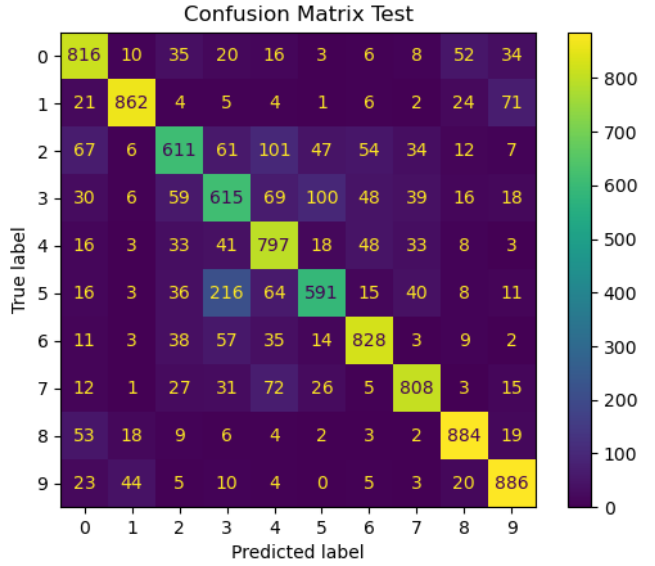
```python
def attack_pattern(model, pattern, label, eps, predict: bool, print_prediction: bool, show_noise=False):
    adversary_image = adversary_pattern(model, pattern, label, eps=eps, show_noise=show_noise)
    if predict:
        out_adversary = np.argmax(model.predict(adversary_image))
        out_pattern = np.argmax(model.predict(pattern.reshape(1, 32, 32, 3)))
        if print_prediction:
            print("True label: {0}, Predicted label: {1}, Predicted adversary label: {2}".
                  format(classes[np.argmax(label)], classes[out_pattern], classes[out_adversary]))

    adversary_image = adversary_image.reshape((32, 32, 3))
    adversary_image = np.clip(adversary_image * 255, 0, 255)
    image = np.copy(pattern) * 255
    plt.imshow(adversary_image.astype("uint8"))
    plt.show()
    plt.imshow(image.astype("uint8"))
    plt.show()
```

The network and the results

For the network's structure i took inspiration from VGG16 since a simple cnn didn't perform well.
Kernel size: (3, 3)
Pooling Stride: 2
Pooling: MaxPooling (2, 2)
Dropout: 0.3
Activation: ReLU
Batch nomalization after each convolution



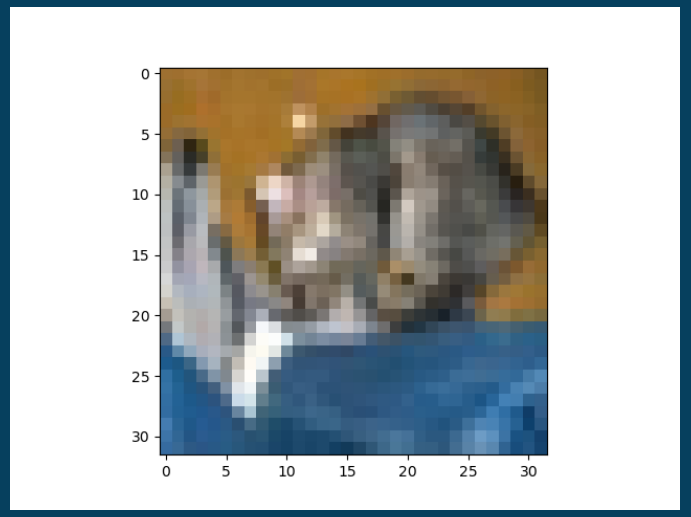




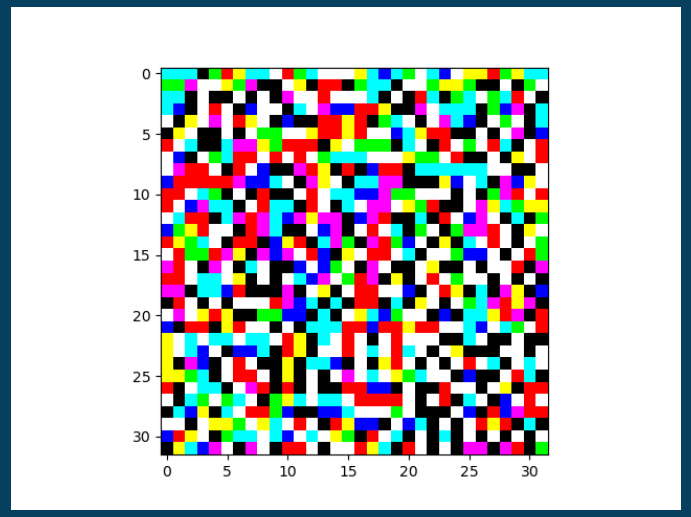

```
Training Error: 0.3862653970718384, Training accuracy: 0.9571400284767151
Test Error: 0.7658382058143616, Test accuracy: 0.8486999869346619
```

Adversarial attack, FGSM

Predicted: cat

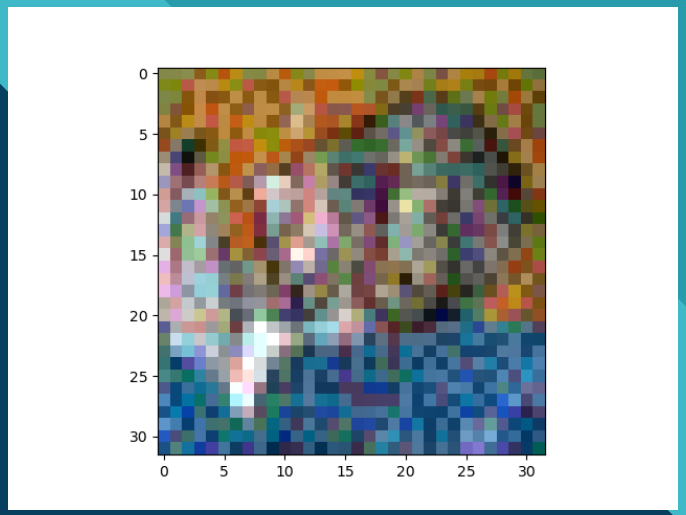


Original image



Noise

Predicted: dog



Attacked image

```
Test adversary eps=0.005 Error: 3.771026134490967, Test adversary eps=0.005 accuracy: 0.3700999915599823
Test adversary eps=0.01 Error: 6.874120235443115, Test adversary eps=0.01 accuracy: 0.15209999680519104
```

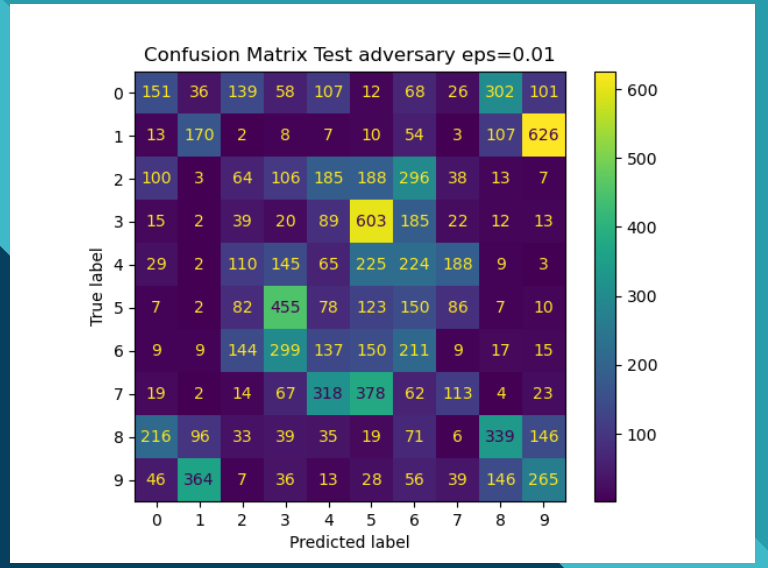
How can we defend from such attacks?

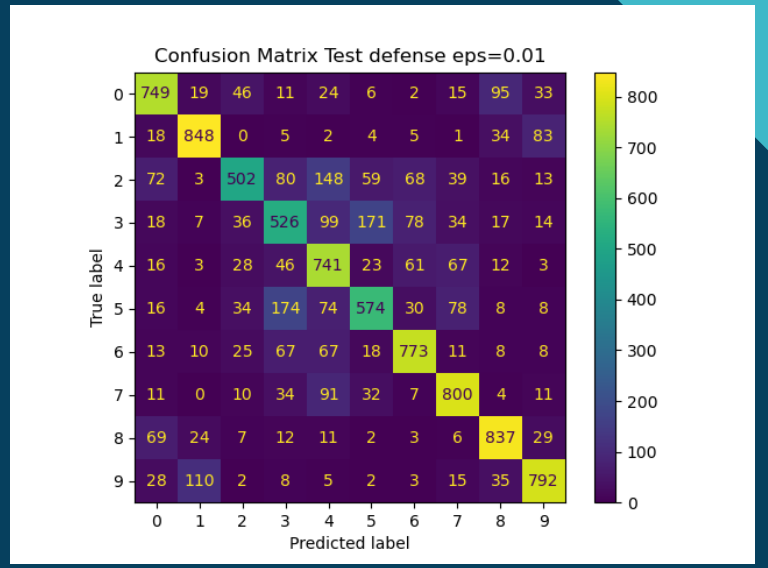
Adversarial training

- The most simple idea was augmenting the training set with adversarial patterns (each one with random noise);
- It makes the model more robust but makes the training more complex (and longer) and it doesn't always guarantee success.

Eps (noise)	No adv. Training	Adv. Training
0.001	0.7432	0.7886
0.005	0.37	0.7559
0.01	0.1520	0.7142
0.05	0.0535	0.4271
0.1	0.0891	0.2813

Test accuracy at different eps values, showing the difference beetween a network that wasn't trained on adversarial pattern and one that was.





Final considerations

- Using more filters as the we go deeper in the network gives better results in terms of accuracy;
- MaxPooling performed a little better than AvgPooling, it highlights the most important features (like edges), but the gap in terms of accuracy wasn’t that much;
- Simple data augmentation is not the best solution for adversarial training, more sophisticated approaches could perform better, such as distillation, and gaussian augmentation.

Thanks for the attention