Alberto Marinelli (638283)

Midterm 3 (Assignment 2)
CNN on CIFAR-10

Implement your own convolutional network, deciding how many layers, the type of layers and how they are interleaved, the type of pooling, the use of residual connections, etc. Discuss why you made each choice a provide performance results of your CNN on CIFAR-10.

Now that your network is trained, you might try an adversarial attack to it. Try the simple Fast Gradient Sign method, generating one (or more) adversarial examples starting from one (or more) CIFAR-10 test images. It is up to you to decide if you want to implement the attack on your own or use one of the available libraries (e.g. foolbox, CleverHans, ...). Display the original image, the adversarial noise and the final adversarial example.

CNN Model

The architecture of the **CNN model** is inspired by the architectural principles of the **VGG models** of the paper *"Very Deep Convolutional Networks for Large-Scale Image Recognition"*

In particular, the **architecture of the final model** is divided in **three block**.

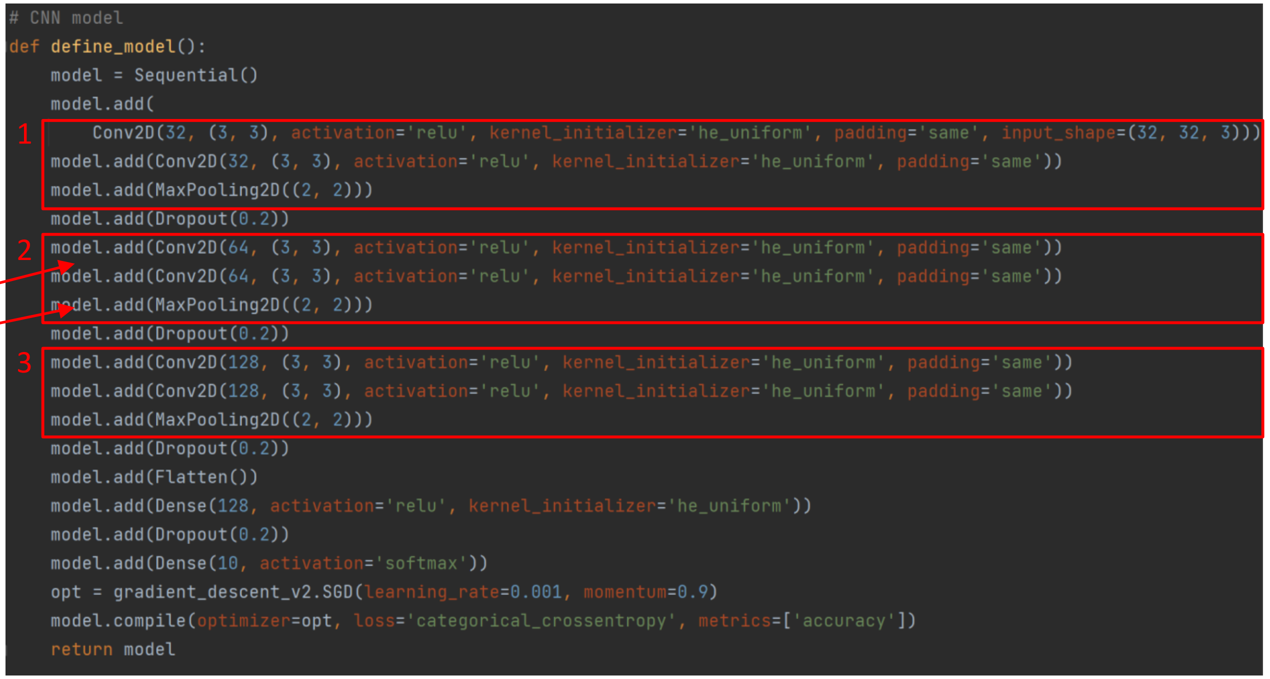
Each block is composed by:

1) **Convolutional layers with 3×3 filters**

2) **Max pooling layer**

The number of **filters in each block** is increased with the depth of the network.

[Code]



```
# CNN model
def define_model():
    model = Sequential()
    model.add(
        Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    opt = gradient_descent_v2.SGD(learning_rate=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

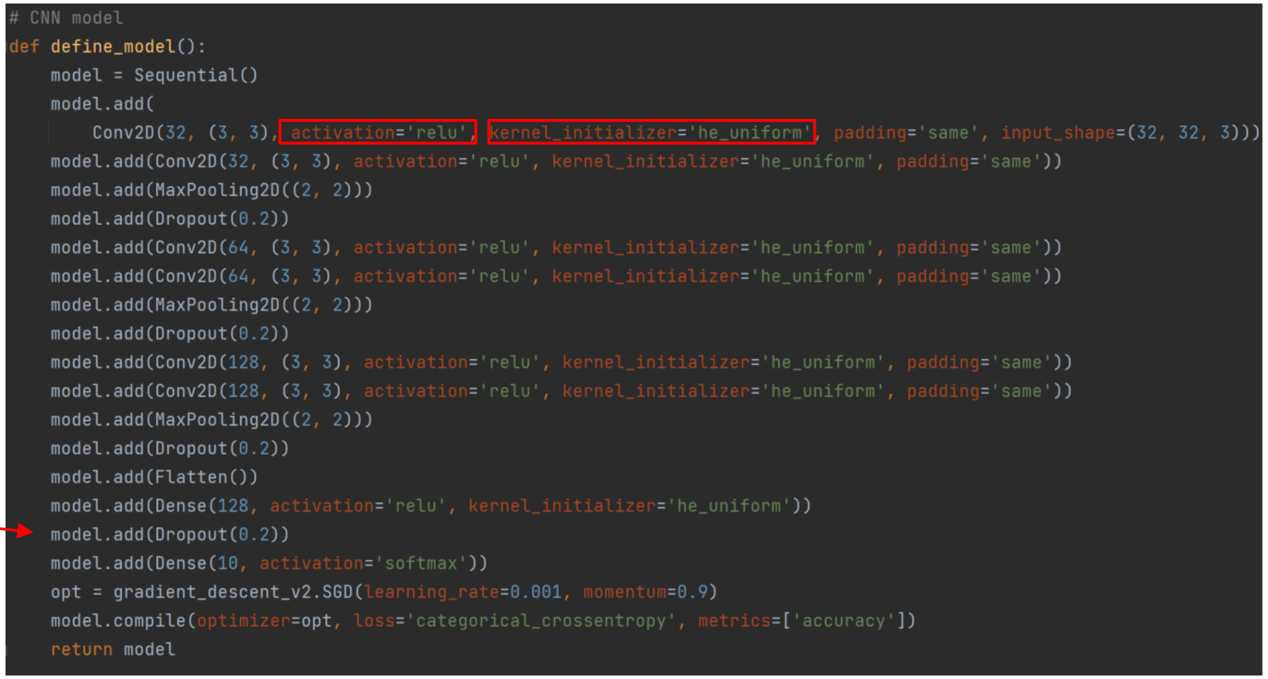
CNN Model

Each layer use the **ReLU activation function** and the **He weight initialization**.

To regularize the network and make the training process noisy, is used a dropout method that forces nodes within a layer to probabilistically take on more or less responsibility for the inputs.

After each max pooling layer and the fully connected layer there is a **dropout layer** with fixed **dropout rate of 20%**.

[Code]

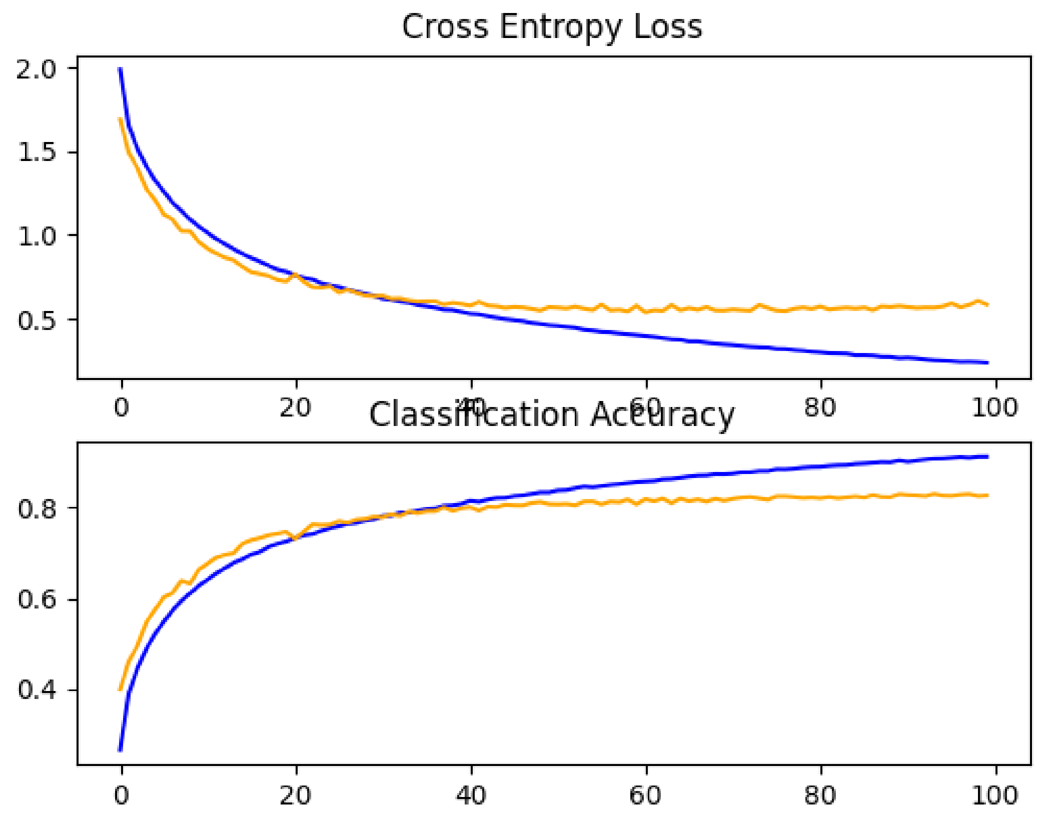


```
# CNN model
def define_model():
    model = Sequential()
    model.add(
        Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    opt = gradient_descent_v2.SGD(learning_rate=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

CNN Model Learning curves

Accuracy (training set): **98.79 %**

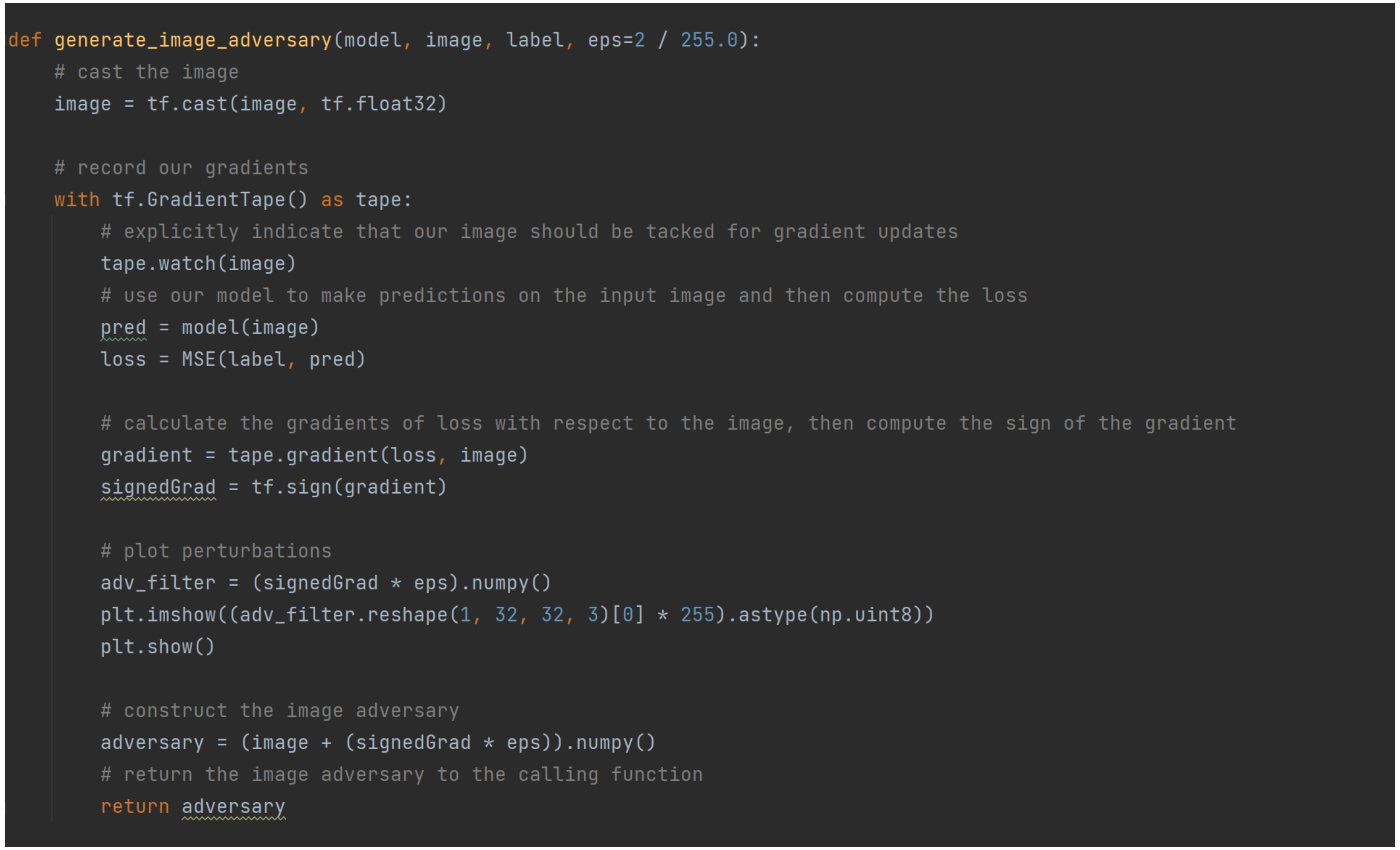
Accuracy (test set): **82.72 %**



[Blue = Training set | Orange = Test set]

Fast Gradient Sign Method

[Code]



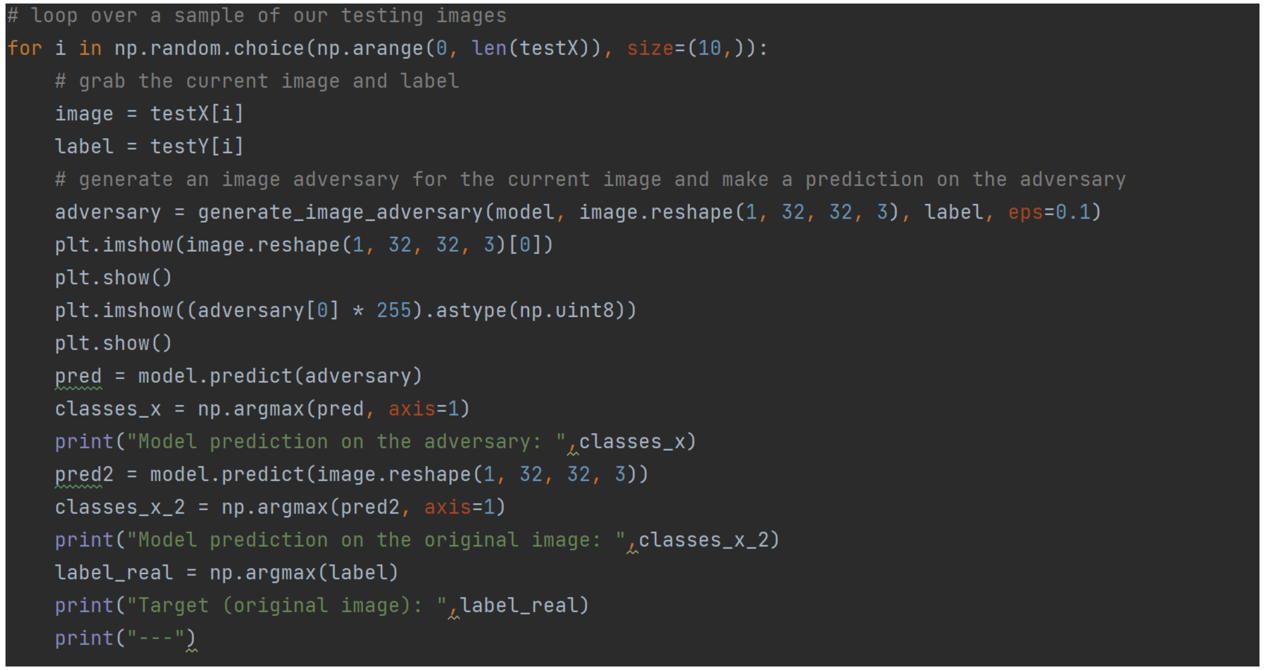
```python
def generate_image_adversary(model, image, label, eps=2 / 255.0):
    # cast the image
    image = tf.cast(image, tf.float32)

    # record our gradients
    with tf.GradientTape() as tape:
        # explicitly indicate that our image should be tacked for gradient updates
        tape.watch(image)
        # use our model to make predictions on the input image and then compute the loss
        pred = model(image)
        loss = MSE(label, pred)

        # calculate the gradients of loss with respect to the image, then compute the sign of the gradient
        gradient = tape.gradient(loss, image)
        signedGrad = tf.sign(gradient)

        # plot perturbations
        adv_filter = (signedGrad * eps).numpy()
        plt.imshow((adv_filter.reshape(1, 32, 32, 3)[0] * 255).astype(np.uint8))
        plt.show()

        # construct the image adversary
        adversary = (image + (signedGrad * eps)).numpy()
        # return the image adversary to the calling function
        return adversary
```

Adversarial attacks (using FGSM)

Each image used to perform the adversarial attacks is randomly selected from the test set.

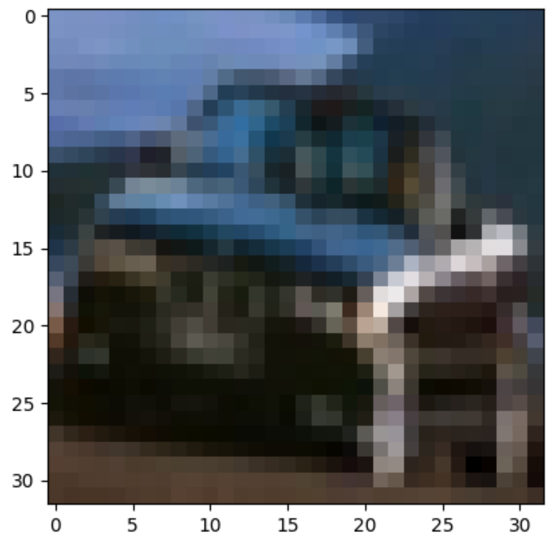
The method *"generate_image_adversary"*, that uses the FGSM, returns the adversarial image on which the perturbations are applied, then the model makes predictions both on the adversarial image and the original image.

[Code]

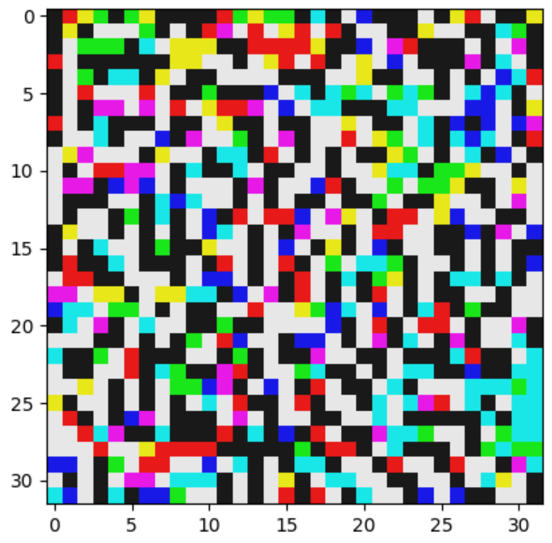


```python
# loop over a sample of our testing images
for i in np.random.choice(np.arange(0, len(testX)), size=(10,)):
    # grab the current image and label
    image = testX[i]
    label = testY[i]
    # generate an image adversary for the current image and make a prediction on the adversary
    adversary = generate_image_adversary(model, image.reshape(1, 32, 32, 3), label, eps=0.1)
    plt.imshow(image.reshape(1, 32, 32, 3)[0])
    plt.show()
    plt.imshow((adversary[0] * 255).astype(np.uint8))
    plt.show()
    pred = model.predict(adversary)
    classes_x = np.argmax(pred, axis=1)
    print("Model prediction on the adversary: ",classes_x)
    pred2 = model.predict(image.reshape(1, 32, 32, 3))
    classes_x_2 = np.argmax(pred2, axis=1)
    print("Model prediction on the original image: ",classes_x_2)
    label_real = np.argmax(label)
    print("Target (original image): ",label_real)
    print("---")
```

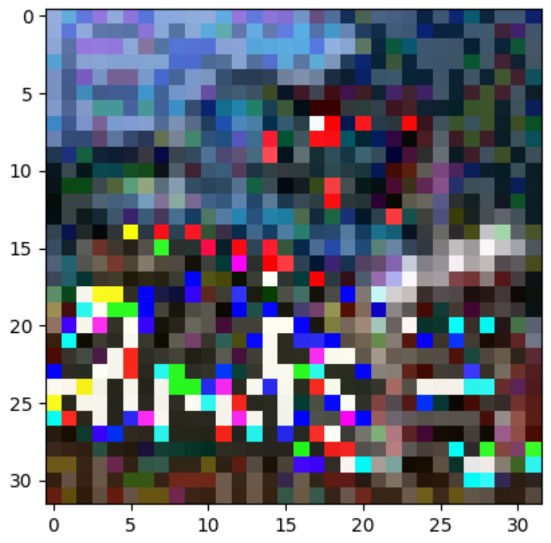
Adversarial attacks (using FGSM) (1/3)



[Original image]

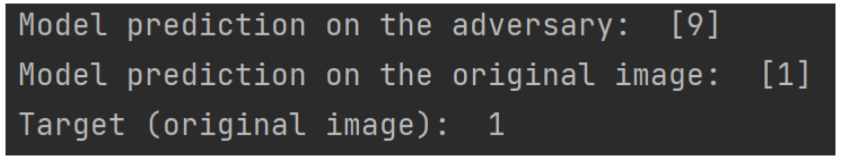


[Perturbations]



[Adversarial image]

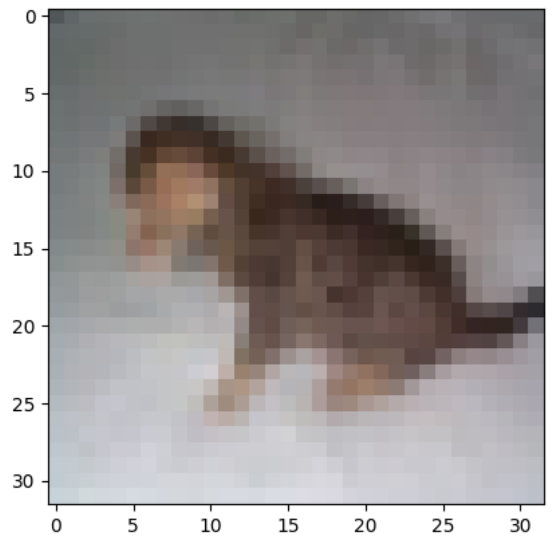
[Output]



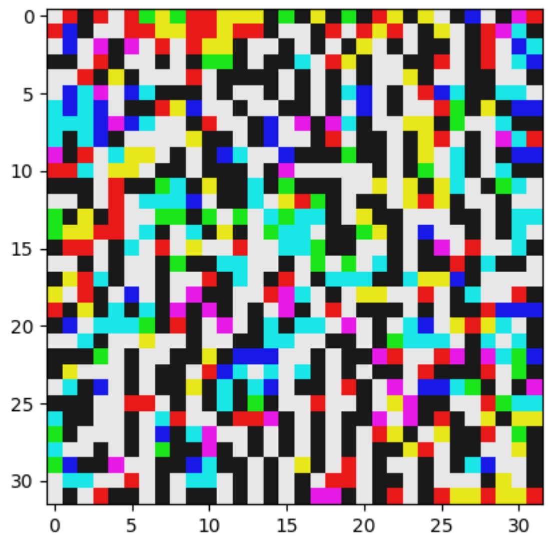
```
Model prediction on the adversary:  [9]
Model prediction on the original image:  [1]
Target (original image):  1
```

[Targets - 0: airplane, 1: car, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck]

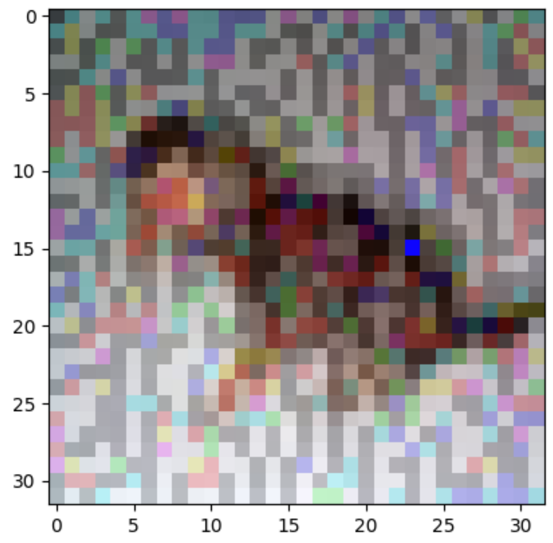
Adversarial attacks (using FGSM) (2/3)



[Original image]

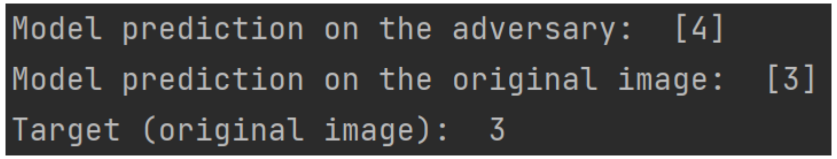


[Perturbations]



[Adversarial image]

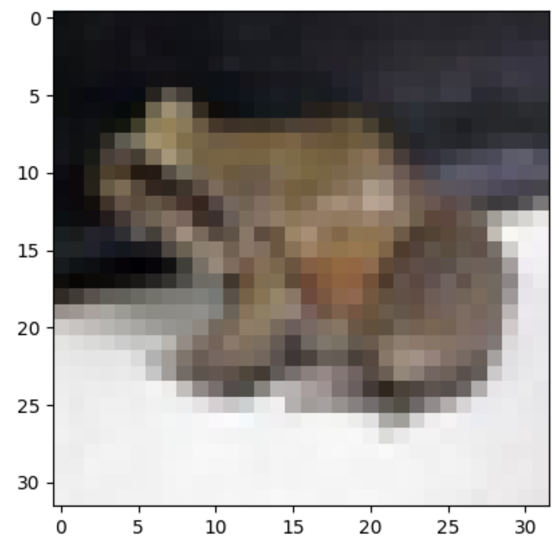
[Output]



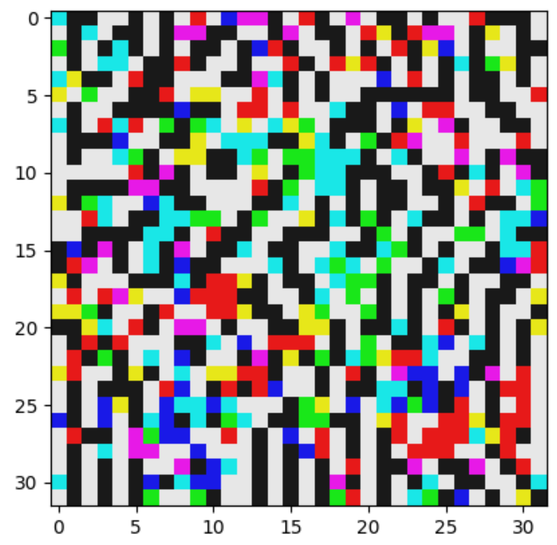
```
Model prediction on the adversary:  [4]
Model prediction on the original image:  [3]
Target (original image):  3
```

[Targets - 0: airplane, 1: car, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck]

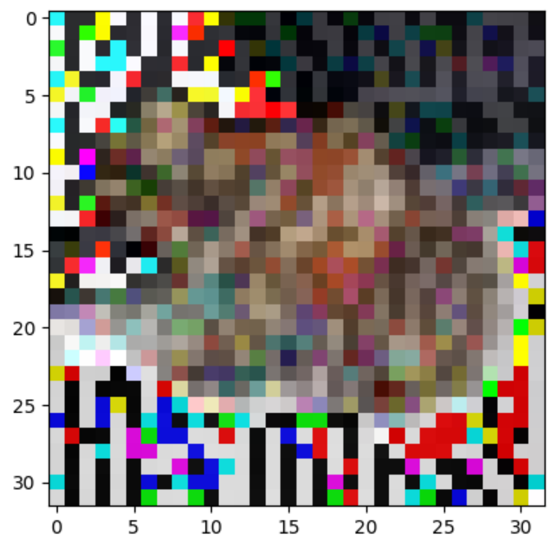
Adversarial attacks (using FGSM) (3/3)



[Orignal image]

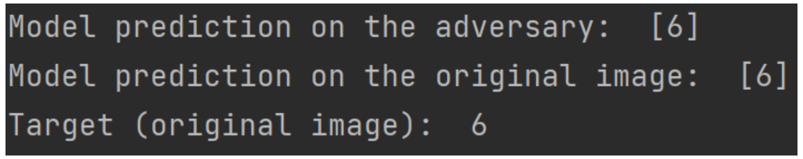


[Perturbations]



[Adversarial image]

[Output]



```
Model prediction on the adversary:   [6]
Model prediction on the original image:   [6]
Target (original image):   6
```

[Targets - 0: airplane, 1: car, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck]

Thanks for attention!

Alberto Marinelli