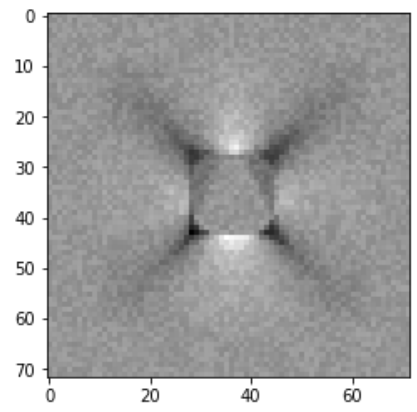
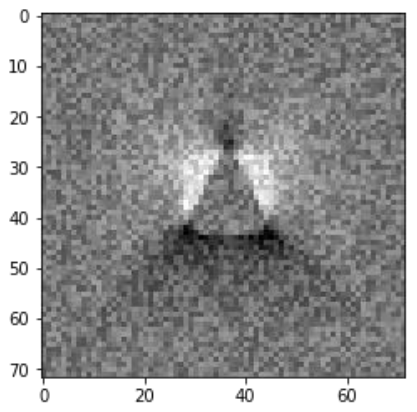
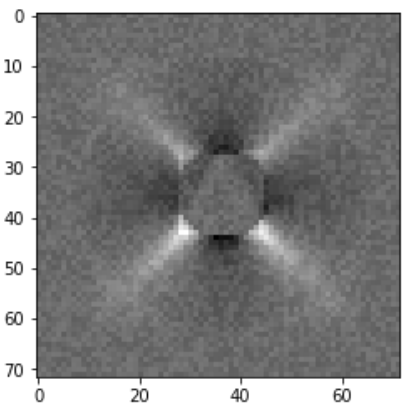
Nom: WANG Xingyi
xingyi.wang@student.ecp.fr

All the codes are available on my Github: https://github.com/StellaireXy/DeepLearning.git

2. Simple Classification

```
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical

#Generating data
[X_train, Y_train] = generate_dataset_classification(300, 20)

#Transforming to categorical data
y_train = to_categorical(Y_train)

#Building the neural network
n_cols = X_train.shape[1]
model = Sequential()
model.add(Dense(3, activation='softmax', input_shape = (n_cols,)))

#Compiling the model
#model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Training the model (the accuracy could reach 1.000000)
model.fit(X_train, y_train, nb_epoch=30, batch_size=32)

#Saving model
from keras.models import load_model
model.save('model_2.h5')
```

3. Visualization of the Solution

rectangle	triangle	disk
		

```
#Dimension of W is 5184*3
[W, c] = model.get_weights()
w0 = W[:,0]
w1 = W[:,1]
w2 = W[:,2]
plt.imshow(w0.reshape(72,72), cmap='gray')
plt.imshow(w1.reshape(72,72), cmap='gray')
plt.imshow(w2.reshape(72,72), cmap='gray')
```

Nom: WANG Xingyi
xingyi.wang@student.ecp.fr

4. A More Difficult Classification Problem

```
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten
from keras.models import Sequential
from keras.utils import to_categorical

#Generating data
[X_train, Y_train] = generate_dataset_classification(3000, 20, True)
[X_test, Y_test] = generate_test_set_classification()

#Transforming the data
X_train = X_train.reshape(len(X_train), 72, 72, 1)
X_test = X_test.reshape(len(X_test), 72, 72, 1)
y_train = to_categorical(Y_train)
y_test = to_categorical(Y_test)

#Building the neural network
model = Sequential()
model.add(Conv2D(kernel_size=(5, 5), padding='same', input_shape=(72, 72, 1), filters=16,
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(Conv2D(32, (5, 5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))

#Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Training the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

```
Train on 3000 samples, validate on 300 samples
Epoch 1/10
3000/3000 [==============================] - 37s 12ms/step - loss: 0.5176 - acc: 0.7850 - val_loss: 0.2112 - val_acc: 0.9267
Epoch 2/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.1159 - acc: 0.9597 - val_loss: 0.1241 - val_acc: 0.9467
Epoch 3/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0622 - acc: 0.9777 - val_loss: 0.0729 - val_acc: 0.9700
Epoch 4/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0596 - acc: 0.9800 - val_loss: 0.1106 - val_acc: 0.9700
Epoch 5/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0381 - acc: 0.9893 - val_loss: 0.1717 - val_acc: 0.9467
Epoch 6/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0313 - acc: 0.9890 - val_loss: 0.0637 - val_acc: 0.9700
Epoch 7/10
3000/3000 [==============================] - 37s 12ms/step - loss: 0.0266 - acc: 0.9923 - val_loss: 0.1772 - val_acc: 0.9433
Epoch 8/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0350 - acc: 0.9893 - val_loss: 0.0746 - val_acc: 0.9667
Epoch 9/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0204 - acc: 0.9923 - val_loss: 0.0804 - val_acc: 0.9767
Epoch 10/10
3000/3000 [==============================] - 36s 12ms/step - loss: 0.0077 - acc: 0.9977 - val_loss: 0.0564 - val_acc: 0.9767
```

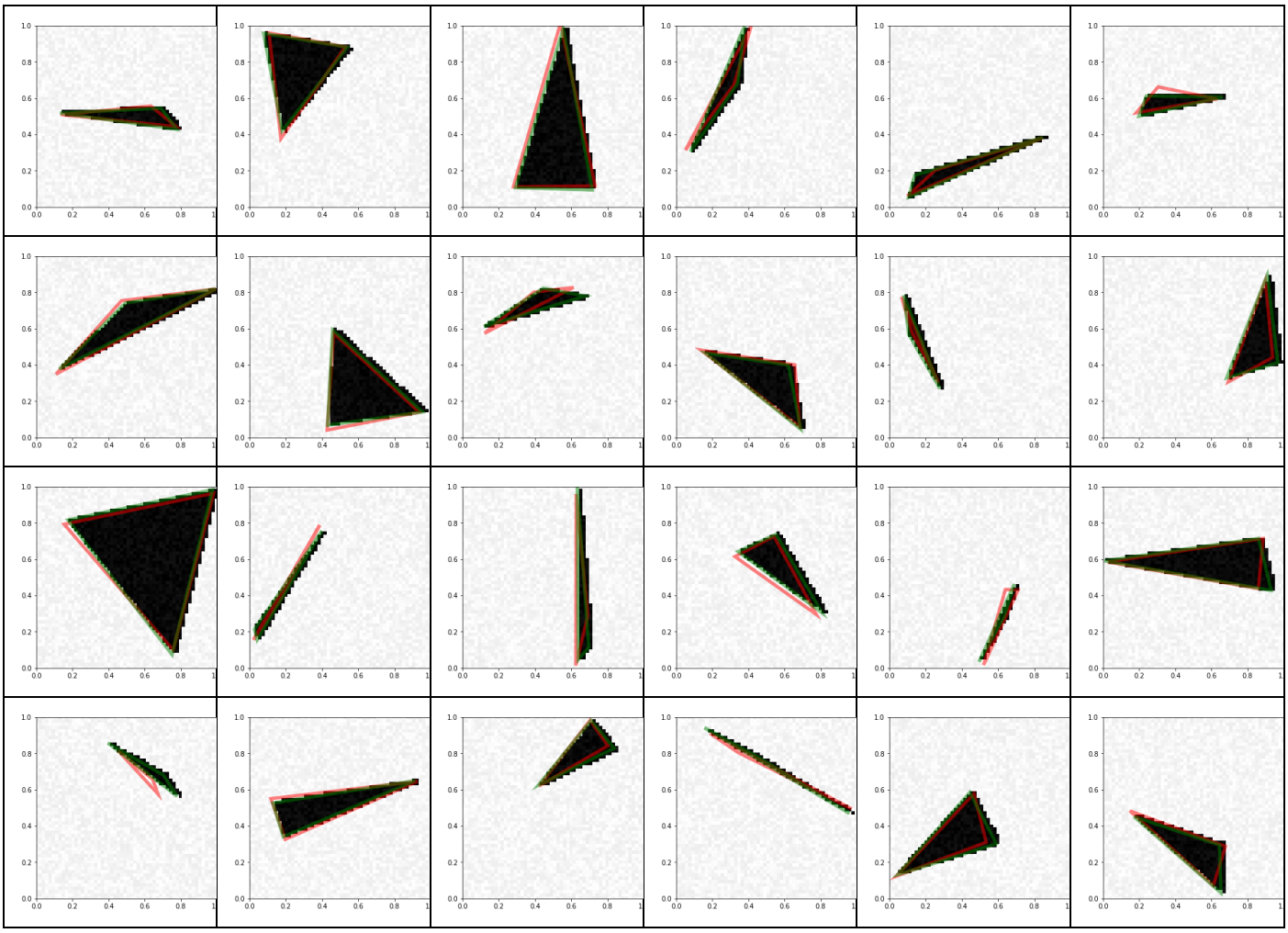
Nom: WANG Xingyi
xingyi.wang@student.ecp.fr

5. A Regression Problem

```
#Sorting the coordinates by distance from the original point
import numpy as np
def normal_triangle(Y):
    p =[[Y[0], Y[1], Y[0]**2 + Y[1]**2],
        [Y[2], Y[3], Y[2]**2 + Y[3]**2],
        [Y[4], Y[5], Y[4]**2 + Y[5]**2]]
    p.sort(key=lambda x:x[2])
    y = np.array([p[0][0], p[0][1], p[1][0], p[1][1], p[2][0], p[2][1]])
    return y

#Generating data
[X_train, Y_train] = generate_dataset_regression(6000, 20)
[X_test, Y_test] = generate_test_set_regression()

#Transforming the data
nb_train = len(X_train)
nb_test = len(X_test)
y_train = np.zeros([nb_train, 6])
y_test = np.zeros([nb_test, 6])
for i in range(nb_train):
    y_train[i] = normal_triangle(Y_train[i])
for i in range(nb_test):
    y_test[i] = normal_triangle(Y_test[i])
x_train = X_train.reshape(len(X_train), 72, 72, 1)
x_test = X_test.reshape(len(X_test), 72, 72, 1)

#Building the network (Conv – Conv – Dense – Dense - Dense)
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten
from keras.models import Sequential
model = Sequential()
model.add(Conv2D(filters=32, activation='relu', input_shape=(72, 72, 1), kernel_size=(3, 3),
padding="same"))
model.add(MaxPooling2D(pool_size=(2, 2), padding="same"))
model.add(Conv2D(32, (3, 3), activation="relu", padding="same"))
model.add(MaxPooling2D(pool_size=(2, 2), padding="same"))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(6))
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
Train on 6000 samples, validate on 300 samples
Epoch 1/10
6000/6000 [==============================] - 93s 15ms/step - loss: 0.0313 - acc: 0.6883 - val_loss: 0.0193 - val_acc: 0.8000
Epoch 2/10
6000/6000 [==============================] - 93s 16ms/step - loss: 0.0132 - acc: 0.7905 - val_loss: 0.0128 - val_acc: 0.8233
Epoch 3/10
6000/6000 [==============================] - 90s 15ms/step - loss: 0.0091 - acc: 0.8150 - val_loss: 0.0112 - val_acc: 0.8167
Epoch 4/10
6000/6000 [==============================] - 90s 15ms/step - loss: 0.0068 - acc: 0.8377 - val_loss: 0.0102 - val_acc: 0.7967
Epoch 5/10
6000/6000 [==============================] - 93s 16ms/step - loss: 0.0052 - acc: 0.8548 - val_loss: 0.0097 - val_acc: 0.8067
Epoch 6/10
6000/6000 [==============================] - 93s 16ms/step - loss: 0.0040 - acc: 0.8712 - val_loss: 0.0096 - val_acc: 0.8267
Epoch 7/10
6000/6000 [==============================] - 97s 16ms/step - loss: 0.0032 - acc: 0.8803 - val_loss: 0.0086 - val_acc: 0.8567
Epoch 8/10
6000/6000 [==============================] - 97s 16ms/step - loss: 0.0023 - acc: 0.8908 - val_loss: 0.0083 - val_acc: 0.8633
Epoch 9/10
6000/6000 [==============================] - 102s 17ms/step - loss: 0.0017 - acc: 0.9098 - val_loss: 0.0085 - val_acc: 0.8467
Epoch 10/10
6000/6000 [==============================] - 97s 16ms/step - loss: 0.0014 - acc: 0.9060 - val_loss: 0.0079 - val_acc: 0.8567
```

```
#Visualizing the prediction
import random
for i in range(24):    #pick 24 images randomly, Green is the correct answer and Red is the prediction
    j = random.randint(0, 300)
    visualize_prediction(X_test[j], Y_pred[j], Y_test[j])
```



```
#24 images are picked randomly to present the result visually. Generally speaking, they are not perfect but acceptable
```

6. Bonus Question

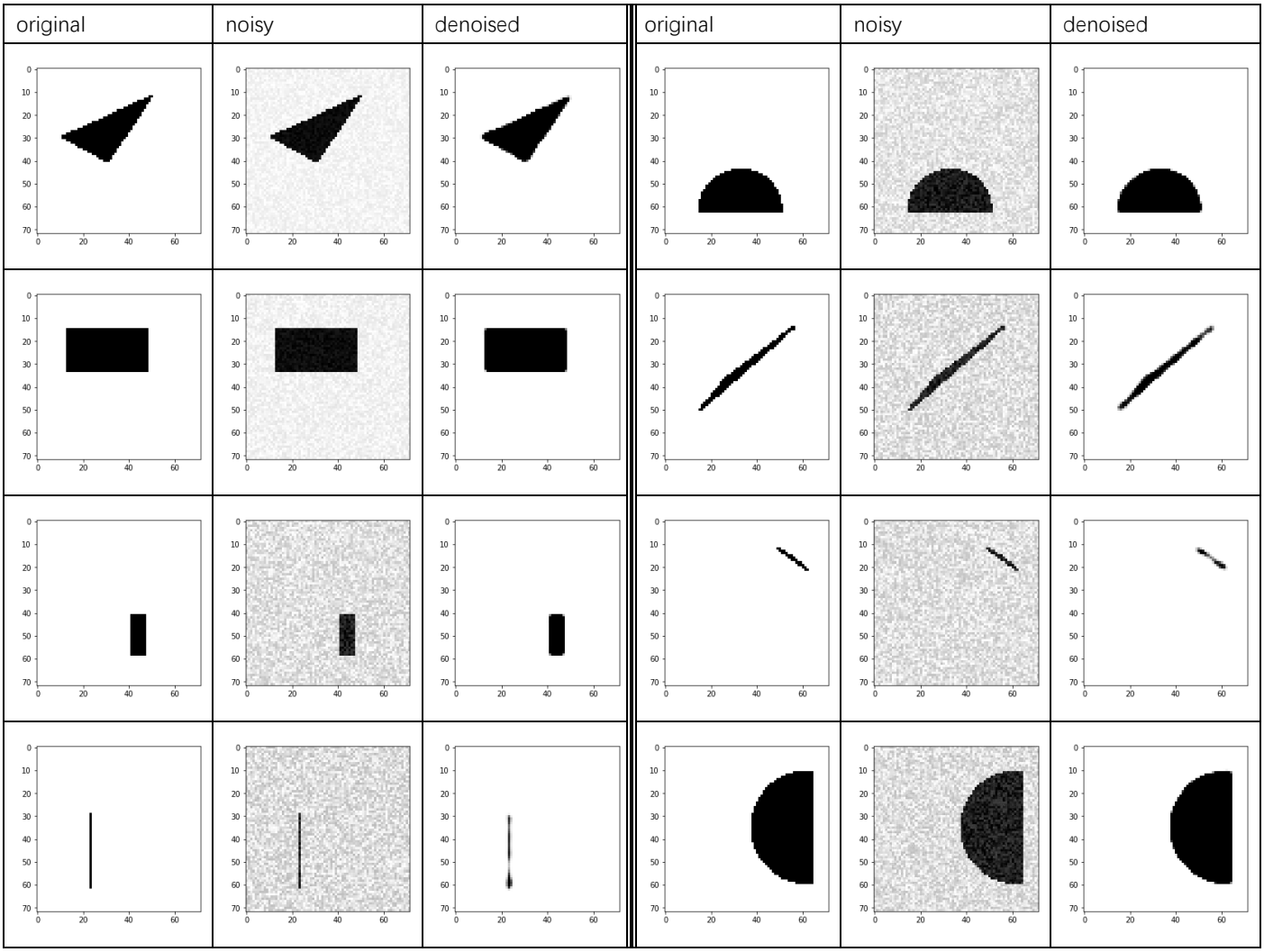
```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D, Activation
from keras.models import Model

#Generating data
[X_train_noise, X_train_clean] = generate_dataset_denoising(3000, True)
[X_test_noise, X_test_clean] = generate_test_set_denoising()

#Transforming the data
x_train_clean = X_train_clean.reshape(len(X_train_clean), 72, 72, 1)
x_train_noise = X_train_noise.reshape(len(X_train_noise), 72, 72, 1)
x_test_clean = X_test_clean.reshape(len(X_test_clean), 72, 72, 1)
x_test_noise = X_test_noise.reshape(len(X_test_noise), 72, 72, 1)

#Building an autoencoder (using Model)
input_img = Input(shape=(72, 72, 1))
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
```

Nom: WANG Xingyi
xingyi.wang@student.ecp.fr

```
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train_noise, x_train_clean, epochs=10, batch_size=32,
                shuffle=True, validation_data=(x_test_noise, x_test_clean))
```

```
Train on 3000 samples, validate on 300 samples
Epoch 1/10
3000/3000 [==============================] - 82s 27ms/step - loss: 0.0872 - val_loss: 0.0166
Epoch 2/10
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0110 - val_loss: 0.0075
Epoch 3/10
3000/3000 [==============================] - 82s 27ms/step - loss: 0.0064 - val_loss: 0.0051
Epoch 4/10
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0048 - val_loss: 0.0046
Epoch 5/10
3000/3000 [==============================] - 86s 29ms/step - loss: 0.0039 - val_loss: 0.0035
Epoch 6/10
3000/3000 [==============================] - 82s 27ms/step - loss: 0.0034 - val_loss: 0.0034
Epoch 7/10
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0029 - val_loss: 0.0027
Epoch 8/10
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0026 - val_loss: 0.0025
Epoch 9/10
3000/3000 [==============================] - 84s 28ms/step - loss: 0.0026 - val_loss: 0.0026
Epoch 10/10
3000/3000 [==============================] - 81s 27ms/step - loss: 0.0022 - val_loss: 0.0023
```

```
x_pred_clean = autoencoder.predict(x_test_noise, batch_size=32)
```



#Here I listed some images to present the result. The denoised image is a little bit slur, which is especially visible when the form is thin.