dl-3-b

April 11, 2025

[ ]:
```
# import libraries
import pandas as pd # Import Pandas for data manipulation using dataframes
import numpy as np # Import Numpy for data statistical analysis
import matplotlib.pyplot as plt # Import matplotlib for data visualisation
import seaborn as sns
import random

%matplotlib inline
sns.set_style("whitegrid")
```

[12]:
```
# dataframes creation for both training and testing datasets
fashion_train_df = pd.read_csv('fashion-mnist_train.csv',sep=',')
fashion_test_df = pd.read_csv('fashion-mnist_test.csv', sep = ',')
```

[13]:
```
fashion_train_df.head()
```

[13]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      2       0       0       0       0       0       0       0       0
1      9       0       0       0       0       0       0       0       0
2      6       0       0       0       0       0       0       0       5
3      0       0       0       0       1       2       0       0       0
4      3       0       0       0       0       0       0       0       0

   pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       0  …       0.0       0.0       0.0       0.0       0.0       0.0
1       0  …       0.0       0.0       0.0       0.0       0.0       0.0
2       0  …       0.0       0.0       0.0      30.0      43.0       0.0
3       0  …       3.0       0.0       0.0       0.0       0.0       1.0
4       0  …       0.0       0.0       0.0       0.0       0.0       0.0

   pixel781  pixel782  pixel783  pixel784
0       0.0       0.0       0.0       0.0
1       0.0       0.0       0.0       0.0
2       0.0       0.0       0.0       0.0
3       0.0       0.0       0.0       0.0
4       0.0       0.0       0.0       0.0
```

```
[5 rows x 785 columns]
```

[14]:
```
fashion_train_df.tail()
```

[14]:
```
      label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
6151      5       0       0       0       0       0       0       0       0
6152      7       0       0       0       0       0       0       0       0
6153      7       0       0       0       0       0       0       0       0
6154      1       0       0       0       0       0       0       0       0
6155      8       0       0       0       0       0       0       0       0

      pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
6151       0  …       0.0       0.0       0.0       0.0       0.0       0.0
6152       0  …       0.0       0.0       0.0       0.0       0.0       0.0
6153       0  …       0.0       0.0       0.0       0.0       0.0       0.0
6154       0  …      22.0       0.0       0.0       0.0       0.0       0.0
6155       0  …       NaN       NaN       NaN       NaN       NaN       NaN

      pixel781  pixel782  pixel783  pixel784
6151       0.0       0.0       0.0       0.0
6152       0.0       0.0       0.0       0.0
6153       0.0       0.0       0.0       0.0
6154       0.0       0.0       0.0       0.0
6155       NaN       NaN       NaN       NaN

[5 rows x 785 columns]
```

[15]:
```
fashion_test_df.head()
```

[15]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      0       0       0       0       0       0       0       0       9
1      1       0       0       0       0       0       0       0       0
2      2       0       0       0       0       0       0      14      53
3      2       0       0       0       0       0       0       0       0
4      3       0       0       0       0       0       0       0       0

   pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       8  …     103.0      87.0      56.0       0.0       0.0       0.0
1       0  …      34.0       0.0       0.0       0.0       0.0       0.0
2      99  …       0.0       0.0       0.0       0.0      63.0      53.0
3       0  …     137.0     126.0     140.0       0.0     133.0     224.0
4       0  …       0.0       0.0       0.0       0.0       0.0       0.0

   pixel781  pixel782  pixel783  pixel784
0       0.0       0.0       0.0       0.0
1       0.0       0.0       0.0       0.0
2      31.0       0.0       0.0       0.0
```

```
3      222.0       56.0        0.0        0.0
4        0.0        0.0        0.0        0.0

[5 rows x 785 columns]
```

[16]:
```
fashion_test_df.tail()
```

[16]:
```
      label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
8034      4       0       0       0       0       0       3       0       0
8035      2       0       0       0       0       0       0       0       0
8036      1       0       0       0       0       0       0       0       0
8037      5       0       0       0       0       0       0       0       0
8038      9       0       0       0       0       0       0       0       0

      pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
8034       0  …     102.0       0.0       0.0       0.0      46.0     128.0
8035       0  …       0.0       2.0       0.0      25.0     201.0     169.0
8036       0  …       0.0       0.0       0.0       0.0       0.0       0.0
8037       0  …       0.0       0.0       1.0       0.0       0.0      91.0
8038       0  …       NaN       NaN       NaN       NaN       NaN       NaN

      pixel781  pixel782  pixel783  pixel784
8034     125.0      41.0       0.0       0.0
8035      61.0       0.0       0.0       0.0
8036       0.0       0.0       0.0       0.0
8037       0.0       0.0       0.0       0.0
8038       NaN       NaN       NaN       NaN

[5 rows x 785 columns]
```

[17]:
```
fashion_train_df.shape
```

[17]: (6156, 785)

[18]:
```
# Create training and testing arrays
train = np.array(fashion_train_df, dtype='float32')
test = np.array(fashion_test_df, dtype='float32')
```

[19]:
```
train.shape
```

[19]: (6156, 785)

[20]:
```
train
```

[20]:
```
array([[ 2.,  0.,  0., …,  0.,  0.,  0.],
       [ 9.,  0.,  0., …,  0.,  0.,  0.],
       [ 6.,  0.,  0., …,  0.,  0.,  0.],
```

```
       …,
       [ 7.,  0.,  0., …,  0.,  0.,  0.],
       [ 1.,  0.,  0., …,  0.,  0.,  0.],
       [ 8.,  0.,  0., …, nan, nan, nan]], dtype=float32)
```

[21]:
```
test
```

[21]:
```
array([[ 0.,  0.,  0., …,  0.,  0.,  0.],
       [ 1.,  0.,  0., …,  0.,  0.,  0.],
       [ 2.,  0.,  0., …,  0.,  0.,  0.],
       …,
       [ 1.,  0.,  0., …,  0.,  0.,  0.],
       [ 5.,  0.,  0., …,  0.,  0.,  0.],
       [ 9.,  0.,  0., …, nan, nan, nan]], dtype=float32)
```

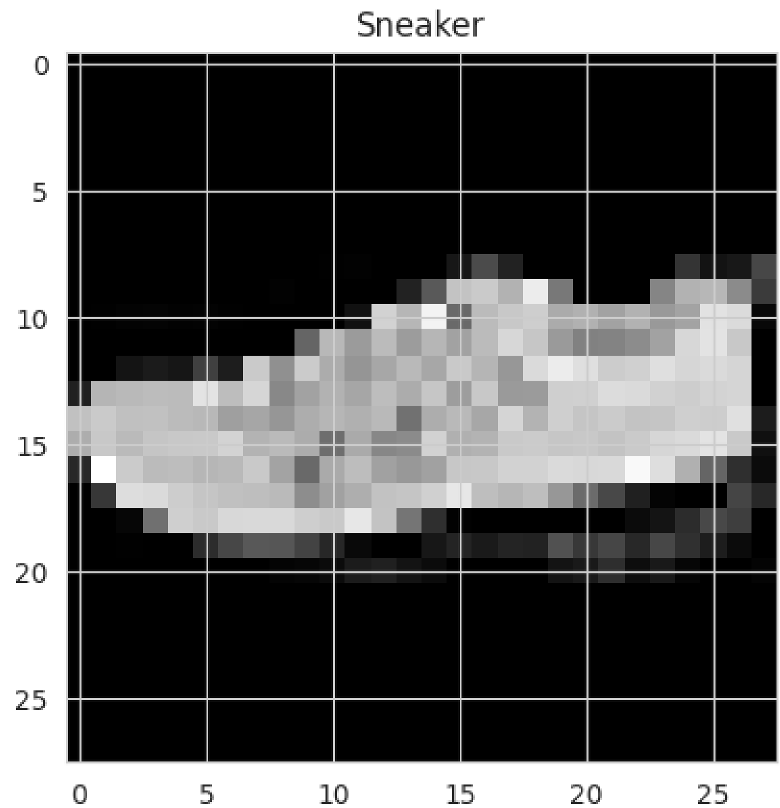
[23]:
```
class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Let's view some images!
i = random.randint(1,6000) # select any random index from 1 to 60,000
plt.imshow(train[i,1:].reshape((28,28))) # reshape and plot the image

plt.imshow(train[i,1:].reshape((28,28)) , cmap = 'gray') # reshape and plot the␣
 ↪image
label_index = fashion_train_df["label"][i]
plt.title(f"{class_names[label_index]}")


# Remember the 10 classes decoding is as follows:
# 0 => T-shirt/top
# 1 => Trouser
# 2 => Pullover
# 3 => Dress
# 4 => Coat
# 5 => Sandal
# 6 => Shirt
# 7 => Sneaker
# 8 => Bag
# 9 => Ankle boot
```

[23]:
```
Text(0.5, 1.0, 'Sneaker')
```



[24]:
```
label = train[i,0]
label
```

[24]: np.float32(7.0)

[25]:
```
W_grid = 15
L_grid = 15

fig, axes = plt.subplots(L_grid, W_grid, figsize=(17,17))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n_train = len(train) # get the length of the train dataset

# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables

    # Select a random number
    index = np.random.randint(0, n_train)
    # read and display an image with the selected index
    axes[i].imshow( train[index,1:].reshape((28,28)) )
```

```
    label_index = int(train[index,0])
    axes[i].set_title(class_names[label_index], fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```

[26]:
```
# Prepare the training and testing dataset
X_train = train[:, 1:] / 255
y_train = train[:, 0]

X_test = test[:, 1:] / 255
y_test = test[:,0]
```

[27]:
```python
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i].reshape((28,28)), cmap=plt.cm.binary)
    label_index = int(y_train[i])
    plt.title(class_names[label_index])
plt.show()
plt.tight_layout()
```

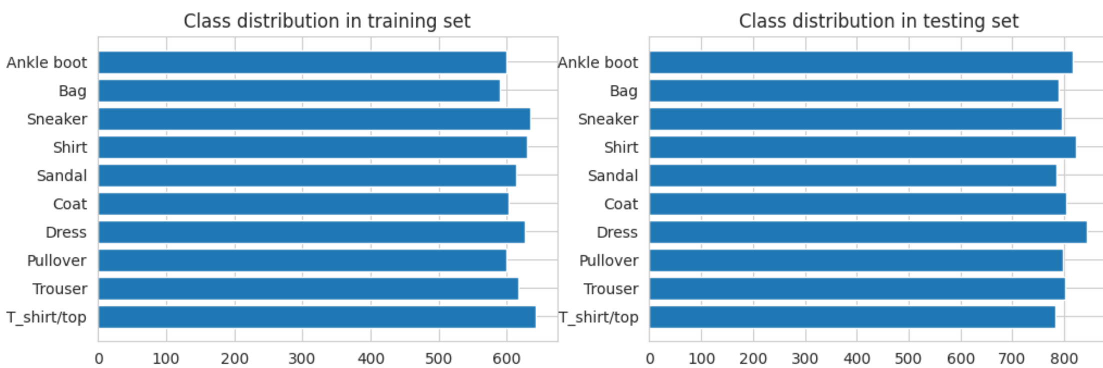

```
<Figure size 640x480 with 0 Axes>
```

[28]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
classes, counts = np.unique(y_train, return_counts=True)
plt.barh(class_names, counts)
plt.title('Class distribution in training set')

plt.subplot(2, 2, 2)
classes, counts = np.unique(y_test, return_counts=True)
plt.barh(class_names, counts)
plt.title('Class distribution in testing set')
```

[28]:
```
Text(0.5, 1.0, 'Class distribution in testing set')
```



[29]:
```
from sklearn.model_selection import train_test_split

X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train,␣
 ↪test_size=0.2, random_state=42)
```

[30]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(4924, 784)
(4924,)
```

[31]:
```
# * unpack the tuple
X_train = X_train.reshape(X_train.shape[0], * (28, 28, 1))
X_test = X_test.reshape(X_test.shape[0], * (28, 28, 1))
X_validate = X_validate.reshape(X_validate.shape[0], * (28, 28, 1))
```

[32]:
```
print(X_train.shape)
print(y_train.shape)
print(X_validate.shape)
print(y_validate.shape)
```

```
(4924, 28, 28, 1)
(4924,)
(1232, 28, 28, 1)
(1232,)
```

[33]:
```
import keras
import tensorflow as tf
```

[34]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
```

[37]:
```
cnn_model = Sequential()

# Try 32 fliters first then 64
cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(28,28,1),␣
 ↪activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(28,28,1),␣
 ↪activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(28,28,1),␣
 ↪activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(28,28,1),␣
 ↪activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.2))

cnn_model.add(Flatten())
cnn_model.add(Dense(units=128, activation='relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(Dense(units=10, activation='softmax'))
```

[38]:
```
METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
```

```
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer='adam'␣
↪,metrics=['accuracy'])
```

[39]:
```
epochs = 50
batch_size = 512

history = cnn_model.fit(
    X_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(X_validate, y_validate)
)
```

```
Epoch 1/50
10/10                39s 3s/step -
accuracy: 0.1515 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 2/50
10/10                37s 3s/step -
accuracy: 0.0986 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 3/50
10/10                43s 3s/step -
accuracy: 0.1025 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 4/50
10/10                41s 3s/step -
accuracy: 0.1017 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 5/50
10/10                29s 3s/step -
accuracy: 0.1013 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 6/50
10/10                41s 3s/step -
accuracy: 0.1072 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 7/50
10/10                42s 3s/step -
accuracy: 0.1026 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 8/50
10/10                40s 3s/step -
accuracy: 0.1049 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 9/50
10/10                40s 3s/step -
accuracy: 0.1010 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 10/50
10/10                28s 3s/step -
accuracy: 0.1112 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 11/50
10/10                29s 3s/step -
accuracy: 0.0991 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
```

```
Epoch 12/50
10/10            41s 3s/step -
accuracy: 0.1061 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 13/50
10/10            41s 3s/step -
accuracy: 0.1106 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 14/50
10/10            41s 3s/step -
accuracy: 0.1004 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 15/50
10/10            42s 3s/step -
accuracy: 0.0989 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 16/50
10/10            40s 3s/step -
accuracy: 0.1038 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 17/50
10/10            40s 3s/step -
accuracy: 0.1029 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 18/50
10/10            44s 3s/step -
accuracy: 0.1048 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 19/50
10/10            39s 3s/step -
accuracy: 0.1045 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 20/50
10/10            41s 3s/step -
accuracy: 0.1035 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 21/50
10/10            41s 3s/step -
accuracy: 0.1066 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 22/50
10/10            43s 3s/step -
accuracy: 0.0992 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 23/50
10/10            29s 3s/step -
accuracy: 0.1052 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 24/50
10/10            42s 3s/step -
accuracy: 0.1031 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 25/50
10/10            40s 3s/step -
accuracy: 0.1046 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 26/50
10/10            41s 3s/step -
accuracy: 0.1020 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 27/50
10/10            43s 3s/step -
accuracy: 0.1036 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
```

```
Epoch 28/50
10/10            43s 3s/step -
accuracy: 0.1070 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 29/50
10/10            38s 3s/step -
accuracy: 0.1002 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 30/50
10/10            41s 3s/step -
accuracy: 0.1062 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 31/50
10/10            40s 3s/step -
accuracy: 0.1046 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 32/50
10/10            40s 3s/step -
accuracy: 0.1066 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 33/50
10/10            41s 3s/step -
accuracy: 0.1010 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 34/50
10/10            41s 3s/step -
accuracy: 0.1096 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 35/50
10/10            41s 3s/step -
accuracy: 0.1016 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 36/50
10/10            41s 3s/step -
accuracy: 0.1035 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 37/50
10/10            42s 3s/step -
accuracy: 0.1026 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 38/50
10/10            28s 3s/step -
accuracy: 0.1029 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 39/50
10/10            41s 3s/step -
accuracy: 0.1063 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 40/50
10/10            41s 3s/step -
accuracy: 0.1020 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 41/50
10/10            28s 3s/step -
accuracy: 0.1044 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 42/50
10/10            42s 3s/step -
accuracy: 0.1068 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 43/50
10/10            39s 3s/step -
accuracy: 0.1004 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
```

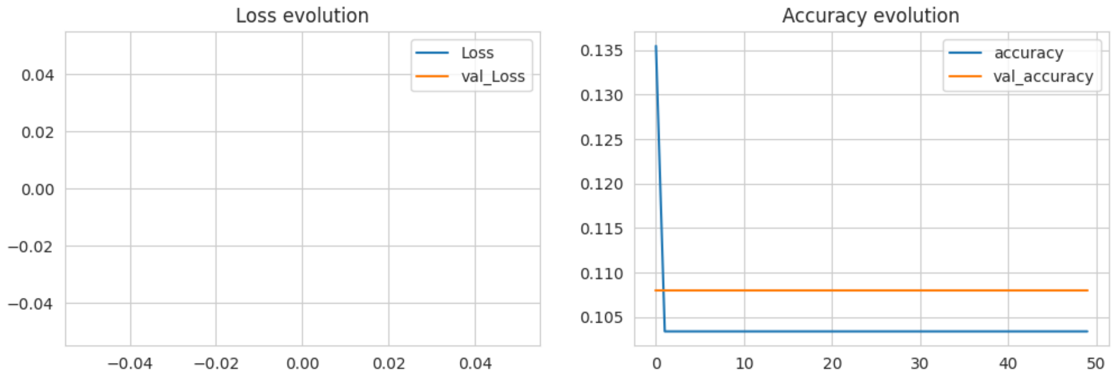
```
Epoch 44/50
10/10             41s 3s/step -
accuracy: 0.1013 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 45/50
10/10             41s 3s/step -
accuracy: 0.0951 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 46/50
10/10             28s 3s/step -
accuracy: 0.1000 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 47/50
10/10             42s 3s/step -
accuracy: 0.0977 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 48/50
10/10             40s 3s/step -
accuracy: 0.1042 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 49/50
10/10             29s 3s/step -
accuracy: 0.1060 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
Epoch 50/50
10/10             41s 3s/step -
accuracy: 0.1001 - loss: nan - val_accuracy: 0.1080 - val_loss: nan
```

[40]:
```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='val_Loss')
plt.legend()
plt.title('Loss evolution')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.legend()
plt.title('Accuracy evolution')
```

[40]:
```
Text(0.5, 1.0, 'Accuracy evolution')
```



[41]:
```python
evaluation = cnn_model.evaluate(X_test, y_test)
print(f'Test Accuracy : {evaluation[1]:.3f}')
```

```
252/252              12s 48ms/step -
accuracy: 0.0979 - loss: nan
Test Accuracy : 0.097
```

[ ]:
```

```