Course Project: Deep Learning and Reinforcement Learning

Objective:

The objective of this report is to analyze the MNIST dataset, which contains 70,000 handwritten black-and-white images, which are traditionally split into 60k training images and 10k validation images. The study will be carried out through 3 Convolutional Neural Networks models. The objective of this project will be to compare the results obtained by the 3 models to validate the model of greater convenience.

Description of the Data Set and its attributes

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

Exploratory Data Analisis

Initial plan for data exploration and actions for data cleaning and feature engineering.

The data is already correct concerning Data Cleaning. However, we will do the corresponding scaling transformations to carry out the different models of the adequate tide. Then we will preliminarily reduce the dimensionality of the features, in order to compare the execution and training times of the algorithms by analyzing the accuracy obtained for the study cases.

Data Cleaning and Feature Engineering

In [76]:
```
import keras
from keras.datasets import mnist
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import random
```

Analyzing the data set.

In [77]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data();
```

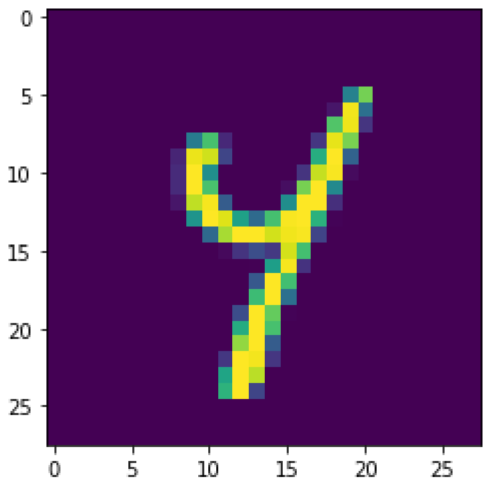
In [78]:
```
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

In [79]:
```
num = random.randint(1, len(x_train))
print(y_train[num])
plt.imshow(x_train[num])
```

```
4
```

Out[79]:
```
<matplotlib.image.AxesImage at 0x1688ff2b850>
```



PCA

We will perform a dimensionality reduction of the image using the Principal Component Analisis.

First, lets flatten the pixels to a 1d-array.

In [80]:
```
x_train_flat = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test_flat = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train_flat.shape, x_test_flat.shape)
```

```
(60000, 784) (10000, 784)
```

We will scale the pixels of the images so that the values are between 0 and 1. This standardization is necessary to carry out the PCA method.

In [81]:
```
from sklearn.preprocessing import MinMaxScaler
MMS = MinMaxScaler().fit(x_train_flat)
x_train_flat = MMS.transform(x_train_flat)
MMS = MinMaxScaler().fit(x_test_flat)
x_test_flat = MMS.transform(x_test_flat)
```

In [82]:
```
def data_pca(x_data, n_components):
    pca = PCA(n_components=n_components)
    fit_pca = pca.fit(x_data)
    print("Variance explained with {0} components:".format(n_components), round(sum(fi
```

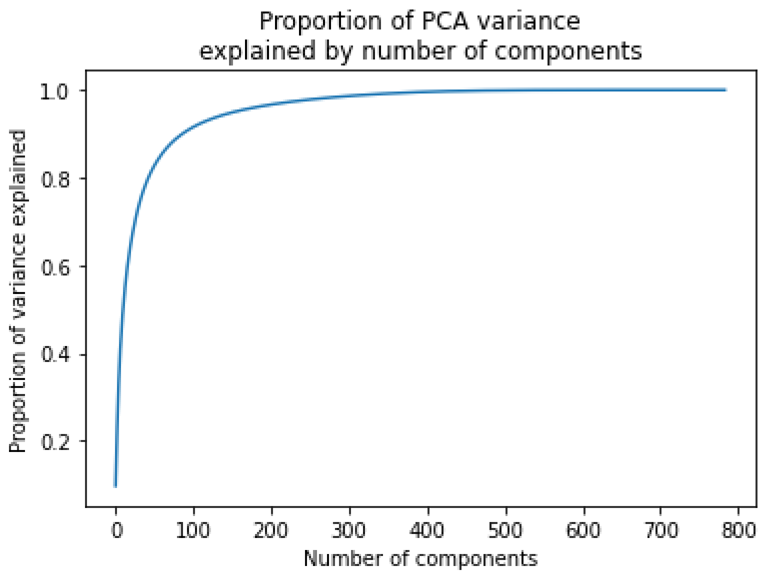
```
    return fit_pca, fit_pca.transform(x_data)
```

The variance explained for all the component must be equal to 1.

In [83]:
```
pca_full, mnist_data_full = data_pca(x_train_flat, 784)
```

```
Variance explained with 784 components: 1.0
```

In [84]:
```
plt.plot(np.cumsum(pca_full.explained_variance_ratio_))
plt.title("Proportion of PCA variance\nexplained by number of components")
plt.xlabel("Number of components")
plt.ylabel("Proportion of variance explained");
```



In [85]:
```
for i in range (100, 400, 50):
    data_pca(x_train_flat, i)
```

```
Variance explained with 100 components: 0.9142
Variance explained with 150 components: 0.948
Variance explained with 200 components: 0.966
Variance explained with 250 components: 0.9777
Variance explained with 300 components: 0.986
Variance explained with 350 components: 0.9918
```

We are going to reduce the dimensionality to 18x18 pixels = 324. Given that, for a number of components of 350, 99% of the variance is preserved. At the same time, the dimensionality is reduced by 41%, considering that the original dimension is 28x28 pixels = 784.

In [86]:
```
# 18*18 = 324pixels
dim = 324
pca_324, mnist_data_324 = data_pca(x_train_flat, dim)
```

```
Variance explained with 324 components: 0.989
```

In [87]:
```
x_test_flat_324 = pca_324.transform(x_test_flat)
x_test_predict_324 = pca_324.inverse_transform(x_test_flat_324)
print(x_test_flat_324.shape, x_test_predict_324.shape)
```

```
(10000, 324) (10000, 784)
```

A low mean square error is obtained.

In [88]:
```
def mse_reconstruction(true, reconstructed):
    return round(np.sum(np.power(true - reconstructed, 2) / true.shape[1]), 2)
mse_reconstruction(x_test_flat, x_test_predict_324)
```

Out[88]:
```
7.33
```

Let's reshape x_test and x_train to the square format and graphically see the result of the dimensionality reduction.

In [89]:
```
x_test_324 = [[] for x in range(len(x_test))]
for i in range(len(x_test)):
    x_test_324[i] = x_test_flat_324[i].reshape((int(np.sqrt(dim)), int(np.sqrt(dim)),1

x_test_324 = np.array(x_test_324)
x_test_324.shape
```

Out[89]:
```
(10000, 18, 18, 1)
```

In [91]:
```
x_train_324 = [[] for x in range(len(x_train))]
for i in range(len(x_train)):
    x_train_324[i] = mnist_data_324[i].reshape((int(np.sqrt(dim)), int(np.sqrt(dim)),1

x_train_324 = np.array(x_train_324)
x_train_324.shape
```

Out[91]:
```
(60000, 18, 18, 1)
```

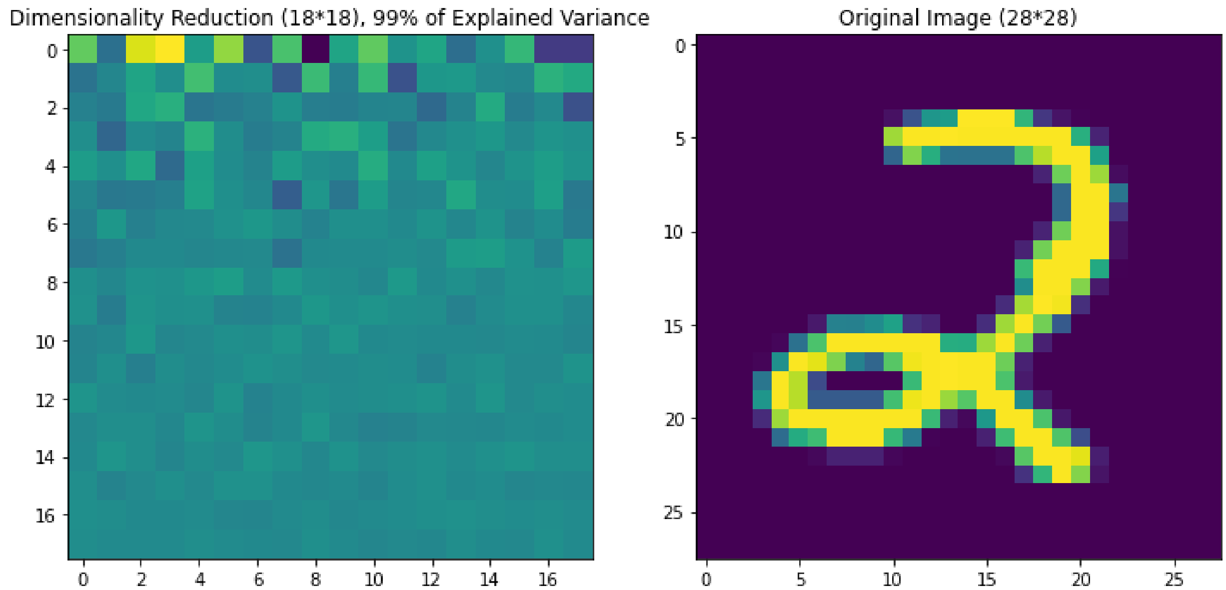
In [92]:
```
num = random.randint(1, len(x_train))
print('Number:', y_train[num])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.set_title('Dimensionality Reduction (18*18), 99% of Explained Variance')
ax.imshow(x_train_324[num])

ax = fig.add_subplot(1, 2, 2)
ax.set_title('Original Image (28*28)')
ax.imshow(x_train[num])
```

```
Number: 2
```

Out[92]:
```
<matplotlib.image.AxesImage at 0x1688ff63400>
```



Convolutional Neural Network Models (CNN)

Now we are going to analyze the following CNN models.

- Model 1: CNN with dimensonality reduction
- Model 2: CNN with dimensonality reduction and a deeper network
- Model 3: CNN without dimensonality reduction

Model 1

Lets one hot encode the categorcal variables.

In [93]:
```
num_classes = 10

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
y_train[num]
```

Out[93]:
```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Let's build a CNN using Keras Sequential capabilities.

In [94]:
```
model_1 = Sequential()

## 4*4 convolution with 2*2 stride and 32 filters
model_1.add(Conv2D(32, (4, 4), strides = (2,2), padding='same', input_shape=x_train_32
model_1.add(Activation('relu'))

## 4*4 convolution with 2*2 stride and 32 filters
model_1.add(Conv2D(32, (4, 4), strides = (2,2)))
model_1.add(Activation('relu'))
```

```
model_1.add(Flatten())

## 2 fully conected layers 288 to 342 and 342 to 10
model_1.add(Dense(288))
model_1.add(Dense(324))
model_1.add(Activation('relu'))
model_1.add(Dropout(0.5))
model_1.add(Dense(num_classes))
model_1.add(Activation('softmax'))

model_1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 9, 9, 32)          544       

 activation_12 (Activation)  (None, 9, 9, 32)          0         

 conv2d_7 (Conv2D)           (None, 3, 3, 32)          16416     

 activation_13 (Activation)  (None, 3, 3, 32)          0         

 flatten_3 (Flatten)         (None, 288)               0         

 dense_10 (Dense)            (None, 288)               83232     

 dense_11 (Dense)            (None, 324)               93636     

 activation_14 (Activation)  (None, 324)               0         

 dropout_3 (Dropout)         (None, 324)               0         

 dense_12 (Dense)            (None, 10)                3250      

 activation_15 (Activation)  (None, 10)                0         

=================================================================
Total params: 197,078
Trainable params: 197,078
Non-trainable params: 0
_________________________________________________________________
```

In [95]:

```
batch_size = 32

model_1.compile(loss='binary_crossentropy',
                optimizer=Adam(lr=1e-5),
                metrics=['accuracy'])

run_hist_1 = model_1.fit(x_train_324,
                         y_train,
                         batch_size=batch_size,
                         epochs=10,
                         validation_data=(x_test_324, y_test),
                         shuffle=True)
```

```
Epoch 1/10
```

```
c:\Users\enzof\Desktop\ML_IBM\5_Deep_Learning_and_Reinforcement_Learning\env\lib\site
-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument i
s deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
1875/1875 [==============================] - 12s 6ms/step - loss: 0.3904 - accuracy:
0.1143 - val_loss: 0.3182 - val_accuracy: 0.4340
Epoch 2/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.3072 - accuracy:
0.2772 - val_loss: 0.2635 - val_accuracy: 0.6854
Epoch 3/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.2304 - accuracy:
0.5883 - val_loss: 0.1741 - val_accuracy: 0.7867
Epoch 4/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1669 - accuracy:
0.7234 - val_loss: 0.1272 - val_accuracy: 0.8301
Epoch 5/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.1352 - accuracy:
0.7783 - val_loss: 0.1055 - val_accuracy: 0.8505
Epoch 6/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.1181 - accuracy:
0.8097 - val_loss: 0.0928 - val_accuracy: 0.8675
Epoch 7/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1067 - accuracy:
0.8313 - val_loss: 0.0842 - val_accuracy: 0.8781
Epoch 8/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0979 - accuracy:
0.8468 - val_loss: 0.0778 - val_accuracy: 0.8847
Epoch 9/10
1875/1875 [==============================] - 10s 6ms/step - loss: 0.0906 - accuracy:
0.8602 - val_loss: 0.0721 - val_accuracy: 0.8935
Epoch 10/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0849 - accuracy:
0.8685 - val_loss: 0.0677 - val_accuracy: 0.8996
```

In [96]:

```python
run_hist_1.history.keys()
```

Out[96]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [97]:

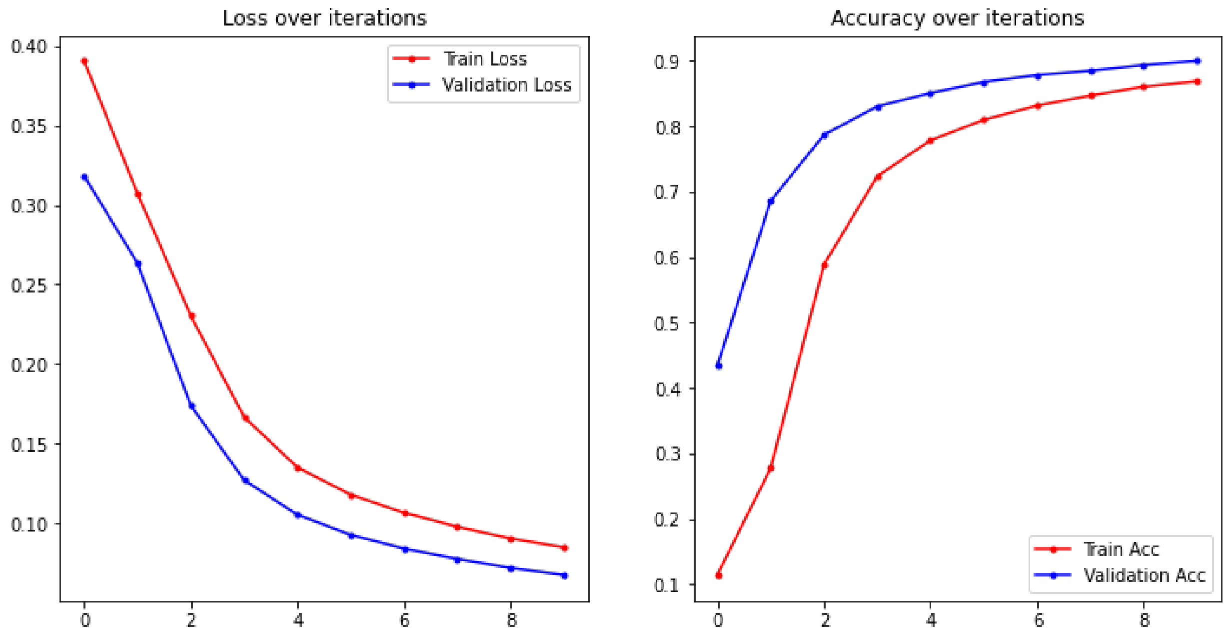
```python
n = len(run_hist_1.history["loss"])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(range(n), (run_hist_1.history["loss"]),'r', marker='.', label="Train Loss")
ax.plot(range(n), (run_hist_1.history["val_loss"]),'b', marker='.', label="Validation
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1, 2, 2)
ax.plot(range(n), (run_hist_1.history["accuracy"]),'r', marker='.', label="Train Acc")
ax.plot(range(n), (run_hist_1.history["val_accuracy"]),'b', marker='.', label="Validat
ax.legend(loc='lower right')
ax.set_title('Accuracy over iterations')
```

Out[97]:

```
Text(0.5, 1.0, 'Accuracy over iterations')
```



In [98]:
```
predict_y1 = model_1.predict(x_test_324)
classes_y1 = np.argmax(predict_y1,axis=1)
print(classes_y1[:30])
print(np.argmax(y_test[:30], axis=1))
```

```
313/313 [==============================] - 1s 2ms/step
[7 2 1 0 4 1 4 9 2 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
```

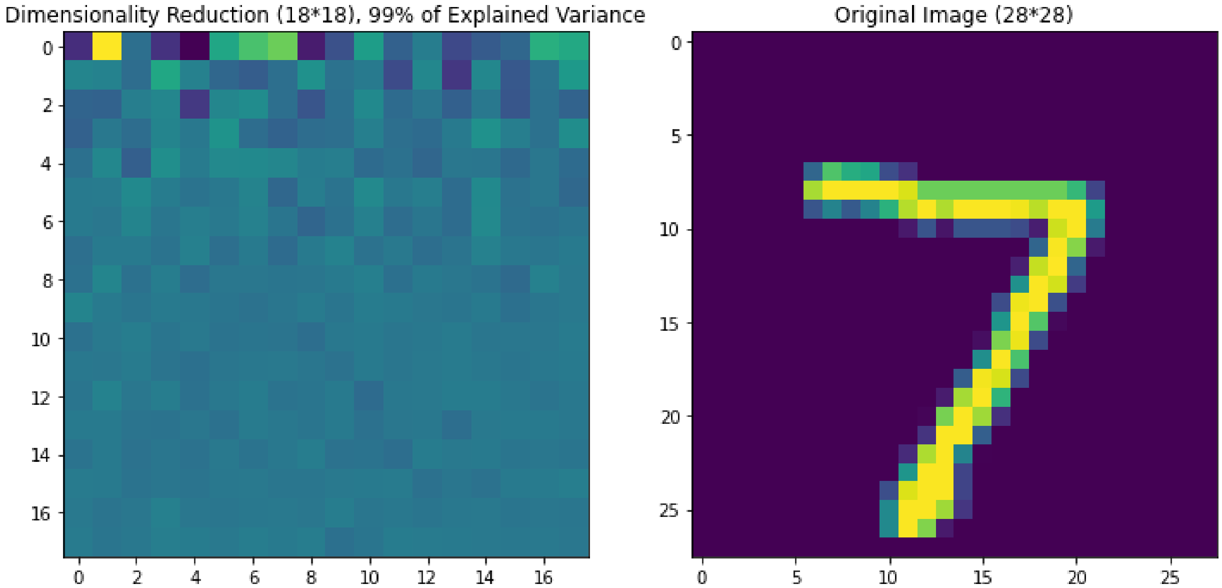
In [99]:
```
print('Number:', np.argmax(y_test[0]))

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.set_title('Dimensionality Reduction (18*18), 99% of Explained Variance')
ax.imshow(x_test_324[0])

ax = fig.add_subplot(1, 2, 2)
ax.set_title('Original Image (28*28)')
ax.imshow(x_test[0])
```

```
Number: 7
```

Out[99]:
```
<matplotlib.image.AxesImage at 0x16892000820>
```



In [100…
```
dic = {}
dic['Accuracy Model 1:'] = accuracy_score(np.argmax(y_test, axis=1), classes_y1)
dic
```

Out[100]:
```
{'Accuracy Model 1:': 0.8996}
```

Model 2

Let's increase the depth of the Neural Network by adding 1 more fully conected layer.

In [101…
```
model_2 = Sequential()

## 4*4 convolution with 2*2 stride and 32 filters
model_2.add(Conv2D(64, (4, 4), strides = (2,2), padding='same', input_shape=x_train_32
model_2.add(Activation('relu'))

## 4*4 convolution with 2*2 stride and 32 filters
model_2.add(Conv2D(64, (4, 4), strides = (2,2)))
model_2.add(Activation('relu'))

## 3 fully conected layers 288 to 342, 342 to 324 and 342 to 10
model_2.add(Flatten())
model_2.add(Dense(288))
model_2.add(Dense(324))
model_2.add(Dense(324))
model_2.add(Activation('relu'))
model_2.add(Dropout(0.5))
model_2.add(Dense(num_classes))
model_2.add(Activation('softmax'))

model_2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 9, 9, 64)          1088      
                                                                 
 activation_16 (Activation)  (None, 9, 9, 64)          0         
                                                                 
 conv2d_9 (Conv2D)           (None, 3, 3, 64)          65600     
                                                                 
 activation_17 (Activation)  (None, 3, 3, 64)          0         
                                                                 
 flatten_4 (Flatten)         (None, 576)               0         
                                                                 
 dense_13 (Dense)            (None, 288)               166176    
                                                                 
 dense_14 (Dense)            (None, 324)               93636     
                                                                 
 dense_15 (Dense)            (None, 324)               105300    
                                                                 
 activation_18 (Activation)  (None, 324)               0         
                                                                 
 dropout_4 (Dropout)         (None, 324)               0         
                                                                 
 dense_16 (Dense)            (None, 10)                3250      
                                                                 
 activation_19 (Activation)  (None, 10)                0         
                                                                 
=================================================================
Total params: 435,050
Trainable params: 435,050
Non-trainable params: 0
_________________________________________________________________
```

In [102…

```
batch_size = 32

model_2.compile(loss='binary_crossentropy',
                optimizer=Adam(lr=1e-5),
                metrics=['accuracy'])

run_hist_2 = model_2.fit(x_train_324, y_train,
                        batch_size=batch_size,
                        epochs=10,
                        validation_data=(x_test_324, y_test),
                        shuffle=True)
```

```
Epoch 1/10
c:\Users\enzof\Desktop\ML_IBM\5_Deep_Learning_and_Reinforcement_Learning\env\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

```
1875/1875 [==============================] - 16s 8ms/step - loss: 0.3648 - accuracy:
0.1648 - val_loss: 0.2719 - val_accuracy: 0.6826
Epoch 2/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.2043 - accuracy:
0.6288 - val_loss: 0.1243 - val_accuracy: 0.8421
Epoch 3/10
1875/1875 [==============================] - 16s 8ms/step - loss: 0.1291 - accuracy:
0.7896 - val_loss: 0.0893 - val_accuracy: 0.8800
Epoch 4/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.1030 - accuracy:
0.8407 - val_loss: 0.0735 - val_accuracy: 0.8971
Epoch 5/10
1875/1875 [==============================] - 16s 8ms/step - loss: 0.0876 - accuracy:
0.8688 - val_loss: 0.0633 - val_accuracy: 0.9105
Epoch 6/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0769 - accuracy:
0.8869 - val_loss: 0.0565 - val_accuracy: 0.9157
Epoch 7/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0690 - accuracy:
0.8971 - val_loss: 0.0516 - val_accuracy: 0.9209
Epoch 8/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0636 - accuracy:
0.9054 - val_loss: 0.0480 - val_accuracy: 0.9249
Epoch 9/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0585 - accuracy:
0.9130 - val_loss: 0.0449 - val_accuracy: 0.9281
Epoch 10/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0549 - accuracy:
0.9182 - val_loss: 0.0424 - val_accuracy: 0.9323
```

In [103…

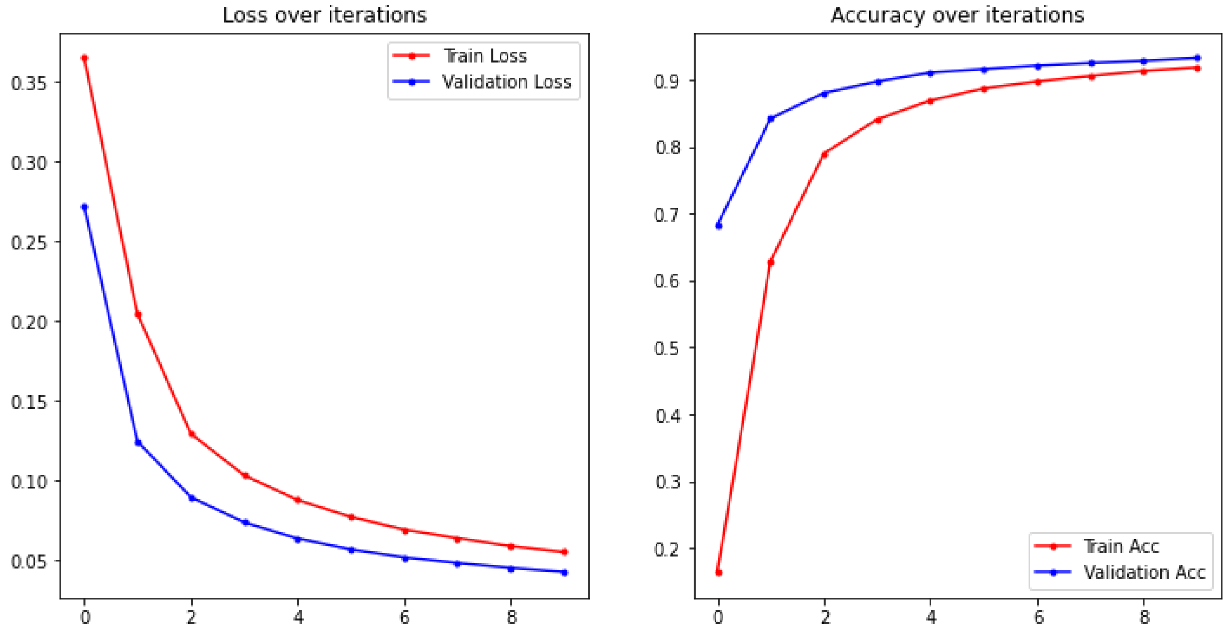
```
n = len(run_hist_2.history["loss"])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(range(n), (run_hist_2.history["loss"]),'r', marker='.', label="Train Loss")
ax.plot(range(n), (run_hist_2.history["val_loss"]),'b', marker='.', label="Validation
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1, 2, 2)
ax.plot(range(n), (run_hist_2.history["accuracy"]),'r', marker='.', label="Train Acc")
ax.plot(range(n), (run_hist_2.history["val_accuracy"]),'b', marker='.', label="Validat
ax.legend(loc='lower right')
ax.set_title('Accuracy over iterations')
```

Out[103]:

```
Text(0.5, 1.0, 'Accuracy over iterations')
```



In [104…
```
predict_y2 = model_2.predict(x_test_324)
classes_y2 = np.argmax(predict_y2,axis=1)
print(classes_y2[:30])
print(np.argmax(y_test[:30], axis=1))
```

```
313/313 [==============================] - 1s 3ms/step
[7 2 1 0 4 1 4 9 2 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
```

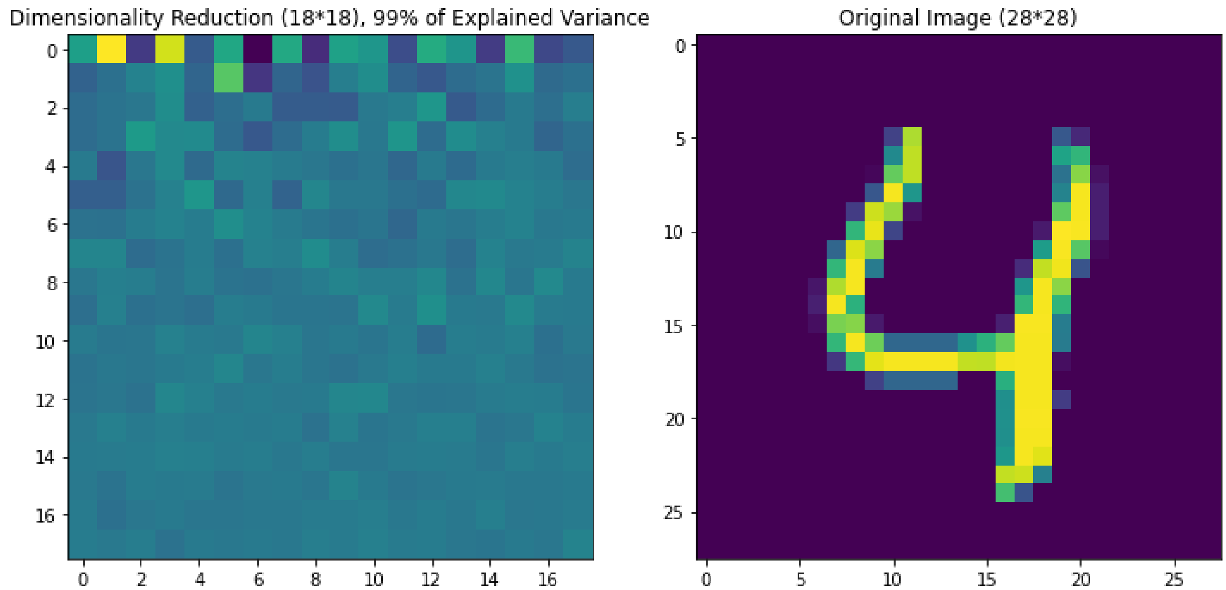
In [105…
```
print('Number:', np.argmax(y_test[4]))

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.set_title('Dimensionality Reduction (18*18), 99% of Explained Variance')
ax.imshow(x_test_324[4])

ax = fig.add_subplot(1, 2, 2)
ax.set_title('Original Image (28*28)')
ax.imshow(x_test[4])
```

```
Number: 4
```

Out[105]:
```
<matplotlib.image.AxesImage at 0x1689704ecb0>
```



In [106…
```
dic['Accuracy Model 2:'] = accuracy_score(np.argmax(y_test, axis=1), classes_y2)
dic
```
Out[106]:
```
{'Accuracy Model 1:': 0.8996, 'Accuracy Model 2:': 0.9323}
```

Model 3

Let's reshape x_test and x_train to the original format (28x28) for the data without th e dimensionality reduction.

In [107…
```
x_train_784 = [[] for x in range(len(x_train))]
for i in range(len(x_train)):
    x_train_784[i] = x_train[i].reshape((28,28,1))

x_train_784 = np.array(x_train_784)
x_train_784.shape
```
Out[107]:
```
(60000, 28, 28, 1)
```

In [109…
```
x_test_784 = [[] for x in range(len(x_test))]
for i in range(len(x_test)):
    x_test_784[i] = x_test[i].reshape((28,28,1))

x_test_784 = np.array(x_test_784)
x_test_784.shape
```
Out[109]:
```
(10000, 28, 28, 1)
```

Let's build the same CNN of the Model 1 and train it with the original data.

In [110…
```
model_3 = Sequential()

## 4*4 convolution with 2*2 stride and 32 filters
model_3.add(Conv2D(32, (4, 4), strides = (2,2), padding='same', input_shape=x_train_78
model_3.add(Activation('relu'))
```

```
## 4*4 convolution with 2*2 stride and 32 filters
model_3.add(Conv2D(32, (4, 4), strides = (2,2)))
model_3.add(Activation('relu'))

## 2 fully conected layers 288 to 342 and 342 to 10
model_3.add(Flatten())
model_3.add(Dense(288))
model_3.add(Dense(324))
model_3.add(Activation('relu'))
model_3.add(Dropout(0.5))
model_3.add(Dense(num_classes))
model_3.add(Activation('softmax'))

model_3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 14, 14, 32)        544

 activation_20 (Activation)  (None, 14, 14, 32)        0

 conv2d_11 (Conv2D)          (None, 6, 6, 32)          16416

 activation_21 (Activation)  (None, 6, 6, 32)          0

 flatten_5 (Flatten)         (None, 1152)              0

 dense_17 (Dense)            (None, 288)               332064

 dense_18 (Dense)            (None, 324)               93636

 activation_22 (Activation)  (None, 324)               0

 dropout_5 (Dropout)         (None, 324)               0

 dense_19 (Dense)            (None, 10)                3250

 activation_23 (Activation)  (None, 10)                0

=================================================================
Total params: 445,910
Trainable params: 445,910
Non-trainable params: 0
_________________________________________________________________
```

In [111…

```
batch_size = 32

model_3.compile(loss='binary_crossentropy',
                optimizer=Adam(lr=1e-5),
                metrics=['accuracy'])

run_hist_3 = model_3.fit(x_train_784, y_train,
                         batch_size=batch_size,
                         epochs=10,
                         validation_data=(x_test_784, y_test),
                         shuffle=True)
```

```
Epoch 1/10
```

```
c:\Users\enzof\Desktop\ML_IBM\5_Deep_Learning_and_Reinforcement_Learning\env\lib\site
-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument i
s deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
1875/1875 [==============================] - 20s 10ms/step - loss: 0.7767 - accuracy:
0.4460 - val_loss: 0.1297 - val_accuracy: 0.8245
Epoch 2/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2157 - accuracy:
0.7086 - val_loss: 0.0769 - val_accuracy: 0.8925
Epoch 3/10
1875/1875 [==============================] - 20s 11ms/step - loss: 0.1376 - accuracy:
0.8072 - val_loss: 0.0551 - val_accuracy: 0.9221
Epoch 4/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0990 - accuracy:
0.8657 - val_loss: 0.0429 - val_accuracy: 0.9385
Epoch 5/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0780 - accuracy:
0.8952 - val_loss: 0.0354 - val_accuracy: 0.9488
Epoch 6/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0631 - accuracy:
0.9167 - val_loss: 0.0299 - val_accuracy: 0.9546
Epoch 7/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0533 - accuracy:
0.9296 - val_loss: 0.0262 - val_accuracy: 0.9599
Epoch 8/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0460 - accuracy:
0.9400 - val_loss: 0.0229 - val_accuracy: 0.9642
Epoch 9/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0400 - accuracy:
0.9484 - val_loss: 0.0211 - val_accuracy: 0.9666
Epoch 10/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0351 - accuracy:
0.9551 - val_loss: 0.0189 - val_accuracy: 0.9698
```

In [112…

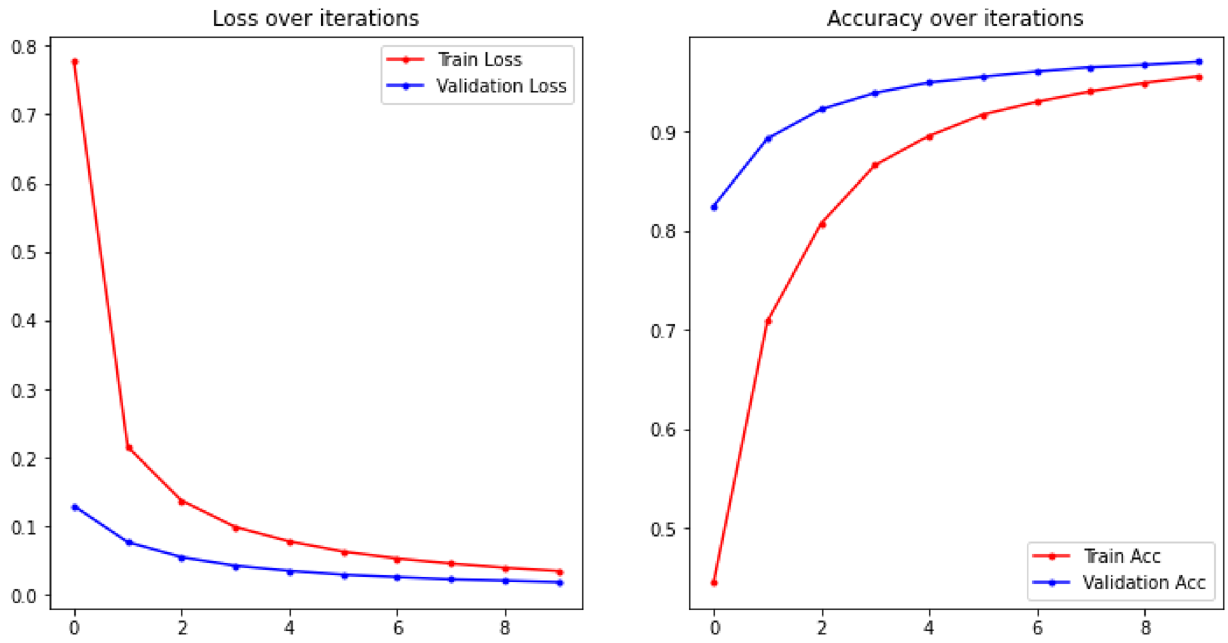
```python
n = len(run_hist_3.history["loss"])

fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(range(n), (run_hist_3.history["loss"]),'r', marker='.', label="Train Loss")
ax.plot(range(n), (run_hist_3.history["val_loss"]),'b', marker='.', label="Validation
ax.legend()
ax.set_title('Loss over iterations')

ax = fig.add_subplot(1, 2, 2)
ax.plot(range(n), (run_hist_3.history["accuracy"]),'r', marker='.', label="Train Acc")
ax.plot(range(n), (run_hist_3.history["val_accuracy"]),'b', marker='.', label="Validat
ax.legend(loc='lower right')
ax.set_title('Accuracy over iterations')
```

Out[112]:

```
Text(0.5, 1.0, 'Accuracy over iterations')
```



In [113…
```
predict_y3 = model_3.predict(x_test_784)
classes_y3 = np.argmax(predict_y3, axis=1)
print(classes_y3[:30])
print(np.argmax(y_test[:30], axis=1))
```

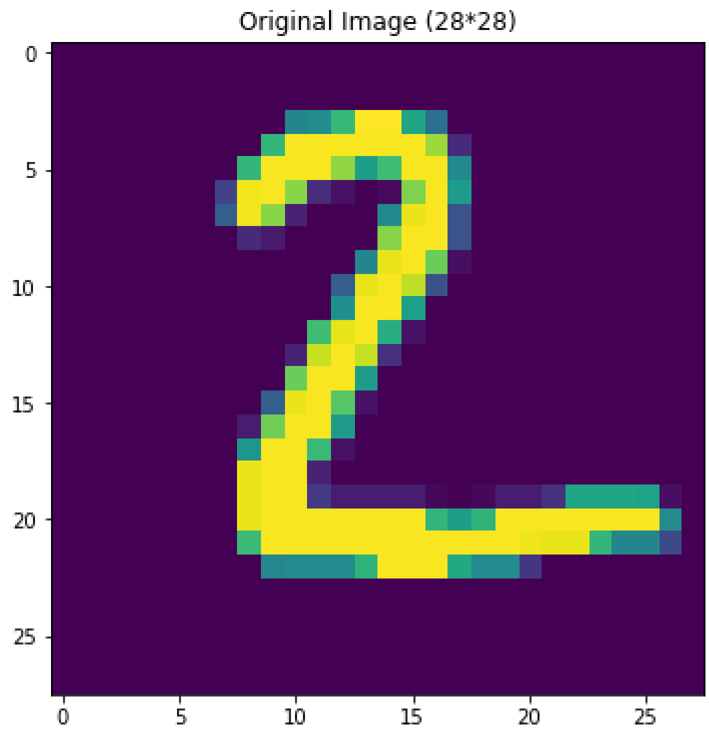
```
313/313 [==============================] - 1s 3ms/step
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1]
```

In [116…
```
print('Number:', np.argmax(y_test[1]))

fig = plt.figure(figsize=(12, 6))
plt.title('Original Image (28*28)')
plt.imshow(x_test[1])
```

```
Number: 2
```

Out[116]:
```
<matplotlib.image.AxesImage at 0x168979718a0>
```



In [115…

```
dic['Accuracy Model 3:'] = accuracy_score(np.argmax(y_test, axis=1), classes_y3)
dic
```

Out[115]:

```
{'Accuracy Model 1:': 0.8996,
 'Accuracy Model 2:': 0.9323,
 'Accuracy Model 3:': 0.9698}
```

Summary

In summary, the following can be highlighted.

All the classifications were made yielded excelent results in terms of Accuracy. Broadly speaking, an increase in accuracy is appreciable as new models are proposed.

Results:

Model	#N Parameters	Execution Time	Accuracy
Model 1	197,078	1m 46,4s	0.90
Model 2	435,050	2m 34,2s	0.93
Model 3	445,910	3m 10,9s	0.97

As we can see all the models present strong results. So any of them can be recommended to optimize the accuracy, in particular the model with highest accuracy is the Model 3. Despite the fact that this model has the best accuracy, since it has been trained with the original data (without dimensionality reduction), it has the largest number of parameters and therefore, the longest execution time. On the other hand, in the models trained with the dimensionality reduction applied to the dataset, the accuracy does not increase significantly as the depth of the neural network increases. Therefore the increase of parameters and execution time are not justified for this case.