Taller 3 - Deep Learning

Curso: Machine Learning y Deep Learning

Diplomado en Data Science

Nombre: Dr. Ing. Rodrigo Salas (rodrigo.salas@uv.cl)

- Nombre integrante 1:Juan Pablo González Collao
- Nombre integrante 2:Pablo Omar Walters Barraza
- Nombre integrante 3:___

Fecha de entrega: Lunes 25 de Julio 2022

A continuación coloque todos los toolbox que utilizó para el desarrollo de esta actividad.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import max_error
from collections import defaultdict
from scipy.stats import pearsonr
from scipy.stats import spearmanr
import statsmodels as sm
import scipy as sp
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.compat import lzip

import tensorflow as tf
import keras as k
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras import optimizers
from keras.layers import Dropout
from keras.layers import LSTM


import os

import random

from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D, Conv2D, MaxPooling2D
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import train_test_split



#from google.colab import drive
#drive.mount('/content/drive')
```

In [2]:
```
def performance_results(y_real, y_predict, text):

    val={'Mean Absolute Error': mean_absolute_error(y_real, y_predict), 'Root Mean Squared Error': mean_squared_
    ds=pd.DataFrame(val,index=[text])
```

```
    #print('Mean Absolute Error', mean_absolute_error(y_real, y_predict) )
    #print('Root Mean Squared Error', mean_squared_error(y_real, y_predict, squared=False) )
    #print('MAPE', mean_absolute_percentage_error(y_real, y_predict))
    #print('R2 score:',r2_score(y_real, y_predict))
    #print('Spearman r:', spearmanr(y_real, y_predict)[0])
    #print('Pearson r:', pearsonr(y_real, y_predict)[0])

    return ds

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

1. Deep Learning aplicado a la predicción en Series DataFrameiempo (30 puntos)

En esta pregunta se realizará un estudio para predecir el nivel de contaminación diario en la Región Metropoltana de Santiago. Para ello se utilizarán los registros de PM10 que provee el Sistema de Información Nacional de la Calidad del Aire SINCA

- SINCA
- Datos

La Región Metropolitana posee varias estaciones de monitoreo. Para ello su grupo deberá seleccionar una de las estaciones y descargar los registros diarios a partir del 1ero de enero del 2018 (buscar alguna estación que tenga estos registros)

1.1. Realizar la lectura del archivo y guardar la información en un DataFrame de pandas. Convertir la variable de las fechas a un tipo de dato Tiempo. Graficar la Serie de Tiempo.

In [3]:
```
df = pd.read_csv('C:\\Users\\pablo.walters\\Desktop\\clase\\taller 3\\datos_180101_220713.csv',sep=";").fillna(r
df.dtypes
```

Out[3]:
```
FECHA (YYMMDD)              int64
HORA (HHMM)                 int64
Registros validados       float64
Registros preliminares    float64
Registros no validados    float64
Unnamed: 5                float64
dtype: object
```

In [4]:
```
df['Registros validados'][1642:1654]=df['Registros preliminares'].tail(12)
df['n_fecha']=pd.date_range('2018-01-01', periods=1654, freq='D')
df=df.drop(['FECHA (YYMMDD)', 'HORA (HHMM)','Registros preliminares', 'Registros no validados', 'Unnamed: 5'], a
```

```
C:\Users\pablo.walters\AppData\Local\Temp\ipykernel_15036\3200011525.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df['Registros validados'][1642:1654]=df['Registros preliminares'].tail(12)
```

In [5]:
```
print(df.dtypes)
df=df.set_index('n_fecha')
df
```

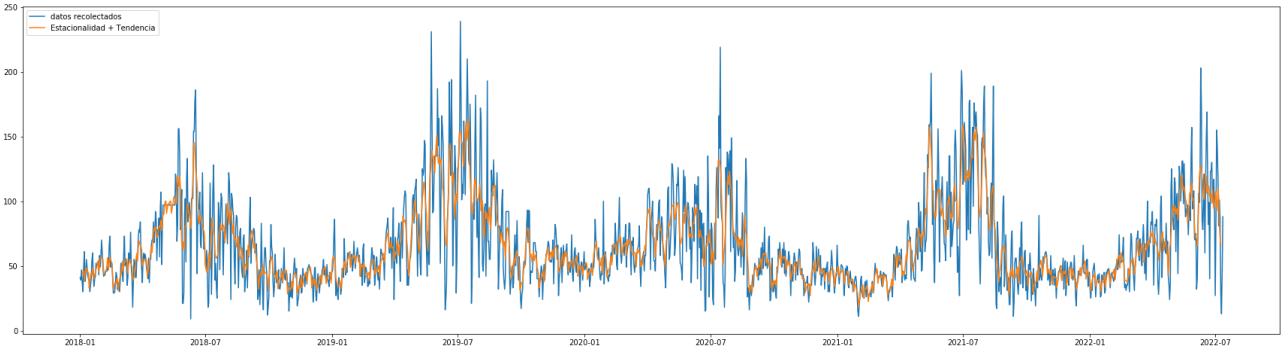
```
Registros validados           float64
n_fecha                 datetime64[ns]
dtype: object
```

Out[5]:

n_fecha	Registros validados
2018-01-01	41.0
2018-01-02	39.0
2018-01-03	47.0
2018-01-04	37.0
2018-01-05	30.0
...	...
2022-07-08	86.0
2022-07-09	29.0
2022-07-10	13.0
2022-07-11	46.0
2022-07-12	88.0

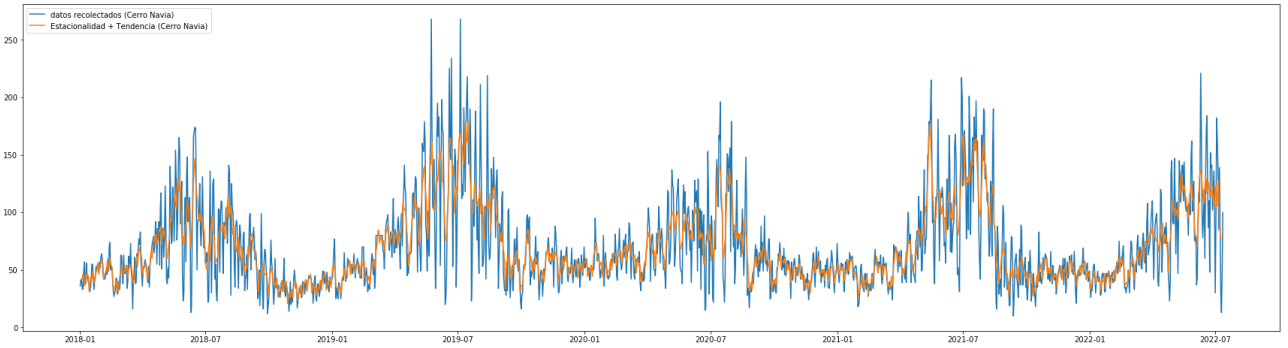
1654 rows × 1 columns

In [6]:
```
plt.figure(figsize = (30,8))
result=seasonal_decompose(df, model='additive', period=7)
trend_estimate = result.trend
seasonal_estimate = result.seasonal
residual_estimate = result.resid


plt.plot(df, label='datos recolectados')
plt.plot(trend_estimate+seasonal_estimate, label='Estacionalidad + Tendencia')
plt.legend(loc='upper left');
```



In [7]:
```
from IPython.display import Image
Image(filename='CERRONAVIA.PNG', width = 3000, height = 800)
```

Out[7]:



A modo de comparación, se grafica e importa imagen de la serie de tiempo de otra estación de monitoreo (Cerro Navia). A simple vista se puede observar, una muy similar tendencia y estacionalidad, con un nivel de ruido de magnitudes similares.

In [8]:
```
#Datos sin residuos
```

```
dfs=trend_estimate+seasonal_estimate
dfs=dfs.dropna()
dfs
```

Out[8]:
```
n_fecha
2018-01-04     44.388088
2018-01-05     45.808185
2018-01-06     46.227071
2018-01-07     37.980638
2018-01-08     43.217720
                  ...
2022-07-05    109.728359
2022-07-06    102.935654
2022-07-07     82.388088
2022-07-08     70.665327
2022-07-09     66.369928
Length: 1648, dtype: float64
```

1.2. Explicar brevemente cuál es el objetivo de este conjunto de datos. Indicar cuánto es la cantidad de datos. Explicar brevemente el significado de la variable.

Respuesta:

El dataset consiste en las concentraciones ambientales promedio diario de material particulado MP10, basado en el monitoreo continuo de este contaminante, en la estación de Monitoreo de Pudahuel, para el periodo 1 de enero de 2018 al 13 de julio del 2022, disponible en el Sistema de Información Nacional de la Calidad del Aire SINCA.

El objetivo del conjunto de datos es realizar una predicción de los niveles de contaminación por MP10 en la región metropolitana, utilizando herramientas de machine learning.

El conjunto de datos cuenta con 5 variables, 2 variables de enteros que muestran la fecha y hora y 3 variable de registros con distinto niveles de validación (Registro validado, preliminar y no validado).

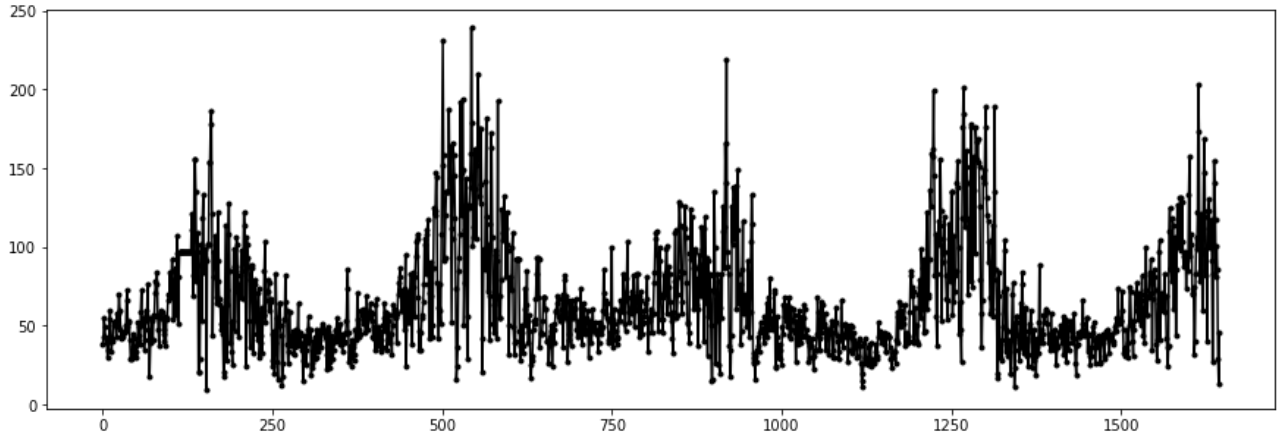
Se tienen 1654 observaciones, donde 1630 cuentan con registros, al menos validados o preliminares, y de las 24 obseraciones restantes, 6 son registros no validados y 18 faltantes.

1.3. Construir la matriz de entrada X considerando un lag de 7 (rezagos de una semana). Separar el conjunto de datos en una matriz X con las variables de entrada y en el target y.

In [9]:
```
def create_dataset(dataset, look_back = 1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        #print(a)
        #print(dataset[i + look_back, 0])
        #print("_____________")
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

In [10]:
```
lags = 7
ts= df['Registros validados'].values
ts1= dfs.values




X, y = create_dataset(ts.reshape(-1,1), look_back=lags)
X1, y1 = create_dataset(ts1.reshape(-1,1), look_back=lags)




plt.figure(figsize=(15,5))
plt.plot(y,'.-k')
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x1bae6c50370>]
```



1.4. Separar el conjunto de datos en conjunto de entrenamiento y en conjunto de test. Considerar el período comprendido entre el 2018-2021 para los datos para entrenamiento, y el año 2022 para test.

In [11]:
```
Train_X = X[0:1461]
Train_y = y[0:1461]
Test_X = X[1462:1654]
Test_y = y[1462:1654]

Train_X1 = X1[0:1461]
Train_y1 = y1[0:1461]
Test_X1 = X1[1462:1654]
Test_y1 = y1[1462:1654]


print('Train_X:',Train_X.shape)
print('Train_y:',Train_y.shape)
print('Test_X:',Test_X.shape)
print('Test_y:',Test_y.shape)

print('Train_X1:',Train_X1.shape)
print('Train_y1:',Train_y1.shape)
print('Test_X1:',Test_X1.shape)
print('Test_y1:',Test_y1.shape)

#Train: (1461, 1)
#Test: (193, 1)
```

```
Train_X: (1461, 7)
Train_y: (1461,)
Test_X: (184, 7)
Test_y: (184,)
Train_X1: (1461, 7)
Train_y1: (1461,)
Test_X1: (178, 7)
Test_y1: (178,)
```

In [12]:
```
Train_y
```

Out[12]:
```
array([38., 43., 55., ..., 46., 49., 52.])
```

1.5. Estandarizar los datos de entrada del conjunto de entrenamiento y test utilizando el "standard scaler". (El standar scaler se ajusta al conjunto de entranamiento y luego se transforman los datos del conjunto de entrenamiento y el de test)

In [13]:
```
ss = StandardScaler().fit(Train_X)
train_data = ss.transform(Train_X)
test_data = ss.transform(Test_X)

ss1 = StandardScaler().fit(Train_X1)
train_data1 = ss.transform(Train_X1)
test_data1 = ss.transform(Test_X1)
train_data
```

Out[13]:
```
array([[-0.68921935, -0.7465431 , -0.51559497, ..., -1.00477007,
        -0.51557162, -0.11228185],
       [-0.74686308, -0.51600608, -0.80369083, ..., -0.51513022,
        -0.11224025, -0.77491038],
       [-0.51628817, -0.80417736, -1.00535793, ..., -0.1118974 ,
        -0.77485607, -0.6308607 ],
       ...,
       [-0.8909724 , -0.60245746, -0.57321414, ..., -1.1487818 ,
        -0.83247483, -0.97657994],
       [-0.60275376, -0.57364034, -0.97654835, ..., -0.83195601,
        -0.97652175, -0.54443089],
       [-0.5739319 , -0.97708013, -1.14940586, ..., -0.97596773,
        -0.544381  , -0.45800108]])
```

NOTA: Es importante destacar que para evitar fugas en el modelo, se realizó la estandarización de los datos de entrada, ajustando el escalamiento previamente al conjunto de entrenamiento.

1.6. Ajustar un modelo de MLP, definir un número adecuado de neuronas escondidas.

In [14]:
```
def create_MLP_model(neurons = 7):
    model = Sequential()
    model.add(Dense(int(neurons*2), input_dim=neurons, activation='sigmoid'))
    model.add(Dropout(0.25))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

mlp_model = create_MLP_model(7)
mlp_model1 = create_MLP_model(7)
print(mlp_model.summary())
print(mlp_model1.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 14)                112
_________________________________________________________________
dropout (Dropout)            (None, 14)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 15
=================================================================
Total params: 127
Trainable params: 127
Non-trainable params: 0
_________________________________________________________________
None
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 14)                112
_________________________________________________________________
dropout_1 (Dropout)          (None, 14)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 15
=================================================================
Total params: 127
Trainable params: 127
Non-trainable params: 0
_________________________________________________________________
None
```

In [15]:
```
mlp_model.fit(train_data, Train_y, epochs=100, batch_size=10,  shuffle=True, verbose=1)
```

```
Epoch 1/100
147/147 [==============================] - 0s 504us/step - loss: 5267.9688
Epoch 2/100
147/147 [==============================] - 0s 463us/step - loss: 5086.0317
Epoch 3/100
147/147 [==============================] - 0s 460us/step - loss: 4896.7295
Epoch 4/100
147/147 [==============================] - 0s 468us/step - loss: 4706.0015
Epoch 5/100
147/147 [==============================] - 0s 455us/step - loss: 4493.0864
Epoch 6/100
147/147 [==============================] - 0s 462us/step - loss: 4272.8081
Epoch 7/100
147/147 [==============================] - 0s 457us/step - loss: 4043.9368
Epoch 8/100
147/147 [==============================] - 0s 495us/step - loss: 3826.8047
Epoch 9/100
147/147 [==============================] - 0s 502us/step - loss: 3602.5867
Epoch 10/100
147/147 [==============================] - 0s 543us/step - loss: 3398.3833
Epoch 11/100
147/147 [==============================] - 0s 563us/step - loss: 3188.2034
Epoch 12/100
147/147 [==============================] - 0s 543us/step - loss: 2982.9829
Epoch 13/100
147/147 [==============================] - 0s 590us/step - loss: 2819.9810
Epoch 14/100
147/147 [==============================] - 0s 909us/step - loss: 2625.9001
Epoch 15/100
147/147 [==============================] - 0s 838us/step - loss: 2500.18550s - loss: 2425.
Epoch 16/100
147/147 [==============================] - 0s 868us/step - loss: 2361.6758
Epoch 17/100
147/147 [==============================] - 0s 910us/step - loss: 2236.4673
Epoch 18/100
147/147 [==============================] - 0s 740us/step - loss: 2110.8318
Epoch 19/100
147/147 [==============================] - 0s 787us/step - loss: 2034.5269
Epoch 20/100
147/147 [==============================] - 0s 644us/step - loss: 1926.2849
Epoch 21/100
147/147 [==============================] - 0s 636us/step - loss: 1824.1981
Epoch 22/100
147/147 [==============================] - 0s 581us/step - loss: 1761.1306
Epoch 23/100
147/147 [==============================] - 0s 570us/step - loss: 1666.1664
Epoch 24/100
147/147 [==============================] - 0s 577us/step - loss: 1620.0724
Epoch 25/100
147/147 [==============================] - 0s 488us/step - loss: 1550.2305
Epoch 26/100
147/147 [==============================] - 0s 482us/step - loss: 1503.4835
Epoch 27/100
147/147 [==============================] - 0s 475us/step - loss: 1449.5017
Epoch 28/100
147/147 [==============================] - 0s 482us/step - loss: 1427.5497
Epoch 29/100
147/147 [==============================] - 0s 475us/step - loss: 1370.4902
Epoch 30/100
147/147 [==============================] - 0s 495us/step - loss: 1347.8783
Epoch 31/100
147/147 [==============================] - 0s 476us/step - loss: 1279.8843
Epoch 32/100
147/147 [==============================] - 0s 470us/step - loss: 1261.0166
Epoch 33/100
147/147 [==============================] - 0s 364us/step - loss: 1197.7375
Epoch 34/100
147/147 [==============================] - 0s 550us/step - loss: 1163.4564
Epoch 35/100
147/147 [==============================] - 0s 462us/step - loss: 1166.2180
Epoch 36/100
147/147 [==============================] - 0s 468us/step - loss: 1131.4897
Epoch 37/100
147/147 [==============================] - 0s 485us/step - loss: 1131.8378
Epoch 38/100
```

```
147/147 [==============================] - 0s 455us/step - loss: 1098.2805
Epoch 39/100
147/147 [==============================] - 0s 360us/step - loss: 1043.5222
Epoch 40/100
147/147 [==============================] - 0s 590us/step - loss: 1044.1664
Epoch 41/100
147/147 [==============================] - 0s 475us/step - loss: 1050.7512
Epoch 42/100
147/147 [==============================] - 0s 591us/step - loss: 1005.7706
Epoch 43/100
147/147 [==============================] - 0s 597us/step - loss: 987.6974
Epoch 44/100
147/147 [==============================] - 0s 577us/step - loss: 986.4111
Epoch 45/100
147/147 [==============================] - 0s 583us/step - loss: 966.6644
Epoch 46/100
147/147 [==============================] - 0s 577us/step - loss: 971.1749
Epoch 47/100
147/147 [==============================] - 0s 646us/step - loss: 938.1542
Epoch 48/100
147/147 [==============================] - 0s 545us/step - loss: 933.8284
Epoch 49/100
147/147 [==============================] - 0s 495us/step - loss: 917.1176
Epoch 50/100
147/147 [==============================] - 0s 486us/step - loss: 908.9355
Epoch 51/100
147/147 [==============================] - 0s 489us/step - loss: 895.8358
Epoch 52/100
147/147 [==============================] - 0s 463us/step - loss: 907.2165
Epoch 53/100
147/147 [==============================] - 0s 467us/step - loss: 910.1230
Epoch 54/100
147/147 [==============================] - 0s 482us/step - loss: 871.2175
Epoch 55/100
147/147 [==============================] - 0s 495us/step - loss: 868.9306
Epoch 56/100
147/147 [==============================] - 0s 556us/step - loss: 873.0338
Epoch 57/100
147/147 [==============================] - 0s 608us/step - loss: 857.3921
Epoch 58/100
147/147 [==============================] - 0s 593us/step - loss: 867.7589
Epoch 59/100
147/147 [==============================] - 0s 556us/step - loss: 836.1980
Epoch 60/100
147/147 [==============================] - 0s 473us/step - loss: 818.7571
Epoch 61/100
147/147 [==============================] - 0s 483us/step - loss: 828.9366
Epoch 62/100
147/147 [==============================] - 0s 571us/step - loss: 796.9357
Epoch 63/100
147/147 [==============================] - 0s 585us/step - loss: 819.8849
Epoch 64/100
147/147 [==============================] - 0s 609us/step - loss: 790.4391
Epoch 65/100
147/147 [==============================] - 0s 499us/step - loss: 792.0330
Epoch 66/100
147/147 [==============================] - 0s 495us/step - loss: 793.3122
Epoch 67/100
147/147 [==============================] - 0s 515us/step - loss: 801.9669
Epoch 68/100
147/147 [==============================] - 0s 621us/step - loss: 799.6877
Epoch 69/100
147/147 [==============================] - 0s 645us/step - loss: 754.5707
Epoch 70/100
147/147 [==============================] - 0s 594us/step - loss: 771.0053
Epoch 71/100
147/147 [==============================] - 0s 597us/step - loss: 785.3549
Epoch 72/100
147/147 [==============================] - 0s 557us/step - loss: 764.2362
Epoch 73/100
147/147 [==============================] - 0s 617us/step - loss: 758.3680
Epoch 74/100
147/147 [==============================] - 0s 619us/step - loss: 749.3642
Epoch 75/100
147/147 [==============================] - 0s 482us/step - loss: 752.3863
```

```
Epoch 76/100
147/147 [==============================] - 0s 490us/step - loss: 734.0854
Epoch 77/100
147/147 [==============================] - 0s 598us/step - loss: 779.1061
Epoch 78/100
147/147 [==============================] - 0s 562us/step - loss: 738.8687
Epoch 79/100
147/147 [==============================] - 0s 592us/step - loss: 714.5424
Epoch 80/100
147/147 [==============================] - 0s 675us/step - loss: 732.5166
Epoch 81/100
147/147 [==============================] - 0s 482us/step - loss: 727.9882
Epoch 82/100
147/147 [==============================] - 0s 489us/step - loss: 726.9222
Epoch 83/100
147/147 [==============================] - 0s 740us/step - loss: 754.9154
Epoch 84/100
147/147 [==============================] - 0s 645us/step - loss: 750.7722
Epoch 85/100
147/147 [==============================] - 0s 610us/step - loss: 702.5828
Epoch 86/100
147/147 [==============================] - 0s 632us/step - loss: 724.8196
Epoch 87/100
147/147 [==============================] - 0s 587us/step - loss: 738.0126
Epoch 88/100
147/147 [==============================] - 0s 527us/step - loss: 712.9473
Epoch 89/100
147/147 [==============================] - 0s 482us/step - loss: 696.3907
Epoch 90/100
147/147 [==============================] - 0s 516us/step - loss: 722.8041
Epoch 91/100
147/147 [==============================] - 0s 624us/step - loss: 701.5349
Epoch 92/100
147/147 [==============================] - 0s 631us/step - loss: 706.0829
Epoch 93/100
147/147 [==============================] - 0s 627us/step - loss: 674.1100
Epoch 94/100
147/147 [==============================] - 0s 550us/step - loss: 678.1767
Epoch 95/100
147/147 [==============================] - 0s 543us/step - loss: 692.7820
Epoch 96/100
147/147 [==============================] - 0s 597us/step - loss: 666.3368
Epoch 97/100
147/147 [==============================] - 0s 563us/step - loss: 690.4524
Epoch 98/100
147/147 [==============================] - 0s 583us/step - loss: 684.8415
Epoch 99/100
147/147 [==============================] - 0s 509us/step - loss: 691.0079
Epoch 100/100
147/147 [==============================] - 0s 468us/step - loss: 682.8928
```

Out[15]:
```
<tensorflow.python.keras.callbacks.History at 0x1bae6fed790>
```

In [16]:
```
y_predict = mlp_model.predict(test_data)

mlp_r=performance_results(Test_y,y_predict.reshape(-1),"MPL con ruido")
mlp_r
```

Out[16]:

	Mean Absolute Error	Root Mean Squared Error	MAPE	R2 score	Spearman r	Pearson r
MPL con ruido	16.687112	24.034368	27.319943	0.503052	0.754615	0.7166

In [17]:
```
mlp_model1.fit(train_data1, Train_y1, epochs=100, batch_size=10,  shuffle=True, verbose=1)
```

```
Epoch 1/100
147/147 [==============================] - 0s 489us/step - loss: 4914.5439
Epoch 2/100
147/147 [==============================] - 0s 475us/step - loss: 4722.6782
Epoch 3/100
147/147 [==============================] - 0s 475us/step - loss: 4524.3936
Epoch 4/100
147/147 [==============================] - 0s 466us/step - loss: 4332.8135
Epoch 5/100
147/147 [==============================] - 0s 474us/step - loss: 4140.7007
Epoch 6/100
147/147 [==============================] - 0s 492us/step - loss: 3931.9551
Epoch 7/100
147/147 [==============================] - 0s 485us/step - loss: 3732.6777
Epoch 8/100
147/147 [==============================] - 0s 488us/step - loss: 3528.6560
Epoch 9/100
147/147 [==============================] - 0s 488us/step - loss: 3299.6707
Epoch 10/100
147/147 [==============================] - 0s 485us/step - loss: 3078.3320
Epoch 11/100
147/147 [==============================] - 0s 516us/step - loss: 2870.4893
Epoch 12/100
147/147 [==============================] - 0s 509us/step - loss: 2681.9263
Epoch 13/100
147/147 [==============================] - 0s 491us/step - loss: 2484.2107
Epoch 14/100
147/147 [==============================] - 0s 495us/step - loss: 2342.0623
Epoch 15/100
147/147 [==============================] - 0s 488us/step - loss: 2156.0002
Epoch 16/100
147/147 [==============================] - 0s 495us/step - loss: 2024.4155
Epoch 17/100
147/147 [==============================] - 0s 495us/step - loss: 1861.8663
Epoch 18/100
147/147 [==============================] - 0s 485us/step - loss: 1767.3027
Epoch 19/100
147/147 [==============================] - 0s 491us/step - loss: 1628.7310
Epoch 20/100
147/147 [==============================] - 0s 492us/step - loss: 1515.8927
Epoch 21/100
147/147 [==============================] - 0s 466us/step - loss: 1457.7133
Epoch 22/100
147/147 [==============================] - 0s 475us/step - loss: 1382.5698
Epoch 23/100
147/147 [==============================] - 0s 498us/step - loss: 1297.4760
Epoch 24/100
147/147 [==============================] - 0s 475us/step - loss: 1195.6846
Epoch 25/100
147/147 [==============================] - 0s 482us/step - loss: 1137.7874
Epoch 26/100
147/147 [==============================] - 0s 495us/step - loss: 1067.1353
Epoch 27/100
147/147 [==============================] - 0s 477us/step - loss: 1021.4697
Epoch 28/100
147/147 [==============================] - 0s 485us/step - loss: 979.6259
Epoch 29/100
147/147 [==============================] - 0s 475us/step - loss: 941.9092
Epoch 30/100
147/147 [==============================] - 0s 495us/step - loss: 899.7845
Epoch 31/100
147/147 [==============================] - 0s 475us/step - loss: 856.4018
Epoch 32/100
147/147 [==============================] - 0s 494us/step - loss: 837.6951
Epoch 33/100
147/147 [==============================] - 0s 502us/step - loss: 773.2847
Epoch 34/100
147/147 [==============================] - 0s 497us/step - loss: 733.6254
Epoch 35/100
147/147 [==============================] - 0s 482us/step - loss: 714.2640
Epoch 36/100
147/147 [==============================] - 0s 495us/step - loss: 649.2344
Epoch 37/100
147/147 [==============================] - 0s 495us/step - loss: 643.1937
Epoch 38/100
```

```
147/147 [==============================] - 0s 475us/step - loss: 619.4158
Epoch 39/100
147/147 [==============================] - 0s 477us/step - loss: 621.8521
Epoch 40/100
147/147 [==============================] - 0s 488us/step - loss: 610.2939
Epoch 41/100
147/147 [==============================] - 0s 490us/step - loss: 564.9497
Epoch 42/100
147/147 [==============================] - 0s 509us/step - loss: 544.5977
Epoch 43/100
147/147 [==============================] - 0s 496us/step - loss: 526.3455
Epoch 44/100
147/147 [==============================] - 0s 482us/step - loss: 513.5757
Epoch 45/100
147/147 [==============================] - 0s 495us/step - loss: 524.5382
Epoch 46/100
147/147 [==============================] - 0s 502us/step - loss: 486.5485
Epoch 47/100
147/147 [==============================] - 0s 488us/step - loss: 457.8842
Epoch 48/100
147/147 [==============================] - 0s 482us/step - loss: 461.7959
Epoch 49/100
147/147 [==============================] - 0s 474us/step - loss: 444.5551
Epoch 50/100
147/147 [==============================] - 0s 465us/step - loss: 406.6651
Epoch 51/100
147/147 [==============================] - 0s 482us/step - loss: 427.0295
Epoch 52/100
147/147 [==============================] - 0s 482us/step - loss: 424.4584
Epoch 53/100
147/147 [==============================] - 0s 475us/step - loss: 396.3512
Epoch 54/100
147/147 [==============================] - 0s 483us/step - loss: 406.0442
Epoch 55/100
147/147 [==============================] - 0s 475us/step - loss: 401.1647
Epoch 56/100
147/147 [==============================] - 0s 475us/step - loss: 368.5135
Epoch 57/100
147/147 [==============================] - 0s 502us/step - loss: 397.1411
Epoch 58/100
147/147 [==============================] - 0s 482us/step - loss: 359.7706
Epoch 59/100
147/147 [==============================] - 0s 461us/step - loss: 348.0246
Epoch 60/100
147/147 [==============================] - 0s 475us/step - loss: 346.9980
Epoch 61/100
147/147 [==============================] - 0s 488us/step - loss: 338.7008
Epoch 62/100
147/147 [==============================] - 0s 482us/step - loss: 329.8430
Epoch 63/100
147/147 [==============================] - 0s 503us/step - loss: 326.3825
Epoch 64/100
147/147 [==============================] - 0s 482us/step - loss: 338.8360
Epoch 65/100
147/147 [==============================] - 0s 509us/step - loss: 303.7723
Epoch 66/100
147/147 [==============================] - 0s 495us/step - loss: 324.5133
Epoch 67/100
147/147 [==============================] - 0s 495us/step - loss: 288.5981
Epoch 68/100
147/147 [==============================] - 0s 495us/step - loss: 306.0306
Epoch 69/100
147/147 [==============================] - 0s 509us/step - loss: 280.0136
Epoch 70/100
147/147 [==============================] - 0s 488us/step - loss: 301.9595
Epoch 71/100
147/147 [==============================] - 0s 502us/step - loss: 295.3869
Epoch 72/100
147/147 [==============================] - 0s 502us/step - loss: 288.4769
Epoch 73/100
147/147 [==============================] - 0s 495us/step - loss: 263.4995
Epoch 74/100
147/147 [==============================] - 0s 495us/step - loss: 269.3831
Epoch 75/100
147/147 [==============================] - 0s 482us/step - loss: 248.0098
```

```
Epoch 76/100
147/147 [==============================] - 0s 495us/step - loss: 253.6552
Epoch 77/100
147/147 [==============================] - 0s 468us/step - loss: 249.8549
Epoch 78/100
147/147 [==============================] - 0s 502us/step - loss: 239.9296
Epoch 79/100
147/147 [==============================] - 0s 475us/step - loss: 236.9337
Epoch 80/100
147/147 [==============================] - 0s 475us/step - loss: 238.9472
Epoch 81/100
147/147 [==============================] - 0s 491us/step - loss: 233.1591
Epoch 82/100
147/147 [==============================] - 0s 471us/step - loss: 228.4537
Epoch 83/100
147/147 [==============================] - 0s 475us/step - loss: 222.2745
Epoch 84/100
147/147 [==============================] - 0s 488us/step - loss: 211.2859
Epoch 85/100
147/147 [==============================] - 0s 516us/step - loss: 220.0665
Epoch 86/100
147/147 [==============================] - 0s 495us/step - loss: 213.7090
Epoch 87/100
147/147 [==============================] - 0s 482us/step - loss: 207.4207
Epoch 88/100
147/147 [==============================] - 0s 475us/step - loss: 214.2911
Epoch 89/100
147/147 [==============================] - 0s 495us/step - loss: 204.9503
Epoch 90/100
147/147 [==============================] - 0s 509us/step - loss: 200.4987
Epoch 91/100
147/147 [==============================] - 0s 502us/step - loss: 195.8930
Epoch 92/100
147/147 [==============================] - 0s 502us/step - loss: 207.4517
Epoch 93/100
147/147 [==============================] - 0s 502us/step - loss: 189.7364
Epoch 94/100
147/147 [==============================] - 0s 481us/step - loss: 193.8488
Epoch 95/100
147/147 [==============================] - 0s 509us/step - loss: 183.6449
Epoch 96/100
147/147 [==============================] - 0s 488us/step - loss: 190.3744
Epoch 97/100
147/147 [==============================] - 0s 472us/step - loss: 171.3988
Epoch 98/100
147/147 [==============================] - 0s 504us/step - loss: 181.4353
Epoch 99/100
147/147 [==============================] - 0s 484us/step - loss: 176.0667
Epoch 100/100
147/147 [==============================] - 0s 495us/step - loss: 189.1415
```

Out[17]:
```
<tensorflow.python.keras.callbacks.History at 0x1bae969e0d0>
```

In [18]:
```
y_predict1 = mlp_model1.predict(test_data1)
mlp_sr=performance_results(Test_y1,y_predict1.reshape(-1),"MLP sin ruido")
mlp_sr
```

Out[18]:

	Mean Absolute Error	Root Mean Squared Error	MAPE	R2 score	Spearman r	Pearson r
MLP sin ruido	5.671728	7.303217	8.012593	0.920306	0.965481	0.965316

1.7. Ajustar un modelo de LSTM, definir un número adecuado de bloques.

In [19]:
```
def create_LSTM_model(neurons = 7, look_back = 1):
    model = Sequential()
    model.add(LSTM(int(neurons)*10, input_shape = (1, look_back)))
    model.add(Dropout(0.25))
    model.add(Dense(1,activation='linear'))
    model.compile(loss = 'mean_squared_error', optimizer = 'adam')
    return model
```

In [20]:
```
trainX = np.reshape(train_data, (train_data.shape[0], 1, train_data.shape[1]))
testX = np.reshape(test_data, (test_data.shape[0], 1, test_data.shape[1]))
```

```
trainX1 = np.reshape(train_data1, (train_data1.shape[0], 1, train_data1.shape[1]))
testX1 = np.reshape(test_data1, (test_data1.shape[0], 1, test_data1.shape[1]))
```

In [21]:
```
lstm_model = create_LSTM_model(20,lags)
print(lstm_model.summary())
lstm_model.fit(trainX, Train_y, epochs = 100, batch_size = 4, shuffle=False, verbose = 1)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 200)               166400
_________________________________________________________________
dropout_2 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 201
=================================================================
Total params: 166,601
Trainable params: 166,601
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
366/366 [==============================] - 1s 2ms/step - loss: 4929.0483
Epoch 2/100
366/366 [==============================] - 1s 2ms/step - loss: 2743.0010
Epoch 3/100
366/366 [==============================] - 1s 2ms/step - loss: 1406.2841
Epoch 4/100
366/366 [==============================] - 1s 2ms/step - loss: 802.2679
Epoch 5/100
366/366 [==============================] - 1s 2ms/step - loss: 628.9745
Epoch 6/100
366/366 [==============================] - 1s 2ms/step - loss: 588.4514
Epoch 7/100
366/366 [==============================] - 1s 2ms/step - loss: 577.9658
Epoch 8/100
366/366 [==============================] - 1s 1ms/step - loss: 558.1147
Epoch 9/100
366/366 [==============================] - 1s 2ms/step - loss: 546.7028
Epoch 10/100
366/366 [==============================] - 1s 2ms/step - loss: 537.2849
Epoch 11/100
366/366 [==============================] - 1s 2ms/step - loss: 532.3879
Epoch 12/100
366/366 [==============================] - 1s 2ms/step - loss: 526.3365
Epoch 13/100
366/366 [==============================] - 1s 2ms/step - loss: 517.9775
Epoch 14/100
366/366 [==============================] - 1s 2ms/step - loss: 522.9538
Epoch 15/100
366/366 [==============================] - 1s 2ms/step - loss: 509.7554
Epoch 16/100
366/366 [==============================] - 1s 2ms/step - loss: 519.4034
Epoch 17/100
366/366 [==============================] - 1s 2ms/step - loss: 510.6876
Epoch 18/100
366/366 [==============================] - 1s 2ms/step - loss: 511.8047
Epoch 19/100
366/366 [==============================] - 1s 2ms/step - loss: 513.1556
Epoch 20/100
366/366 [==============================] - 1s 2ms/step - loss: 508.2737
Epoch 21/100
366/366 [==============================] - 1s 2ms/step - loss: 495.4721
Epoch 22/100
366/366 [==============================] - 1s 2ms/step - loss: 502.4818
Epoch 23/100
366/366 [==============================] - 1s 2ms/step - loss: 505.0544
Epoch 24/100
366/366 [==============================] - 1s 2ms/step - loss: 502.6264
Epoch 25/100
366/366 [==============================] - 1s 2ms/step - loss: 499.6124
Epoch 26/100
366/366 [==============================] - 1s 2ms/step - loss: 502.6111
Epoch 27/100
366/366 [==============================] - 1s 2ms/step - loss: 498.2654
Epoch 28/100
366/366 [==============================] - 1s 2ms/step - loss: 493.4319
Epoch 29/100
366/366 [==============================] - 1s 2ms/step - loss: 501.5533
Epoch 30/100
366/366 [==============================] - 1s 2ms/step - loss: 497.3630
```

```
Epoch 31/100
366/366 [==============================] - 1s 2ms/step - loss: 496.1007
Epoch 32/100
366/366 [==============================] - 1s 2ms/step - loss: 483.4846
Epoch 33/100
366/366 [==============================] - 1s 2ms/step - loss: 491.3396
Epoch 34/100
366/366 [==============================] - 1s 2ms/step - loss: 486.3485
Epoch 35/100
366/366 [==============================] - 1s 2ms/step - loss: 499.3552
Epoch 36/100
366/366 [==============================] - 1s 2ms/step - loss: 495.4048
Epoch 37/100
366/366 [==============================] - 1s 2ms/step - loss: 477.0920
Epoch 38/100
366/366 [==============================] - 1s 2ms/step - loss: 490.2419
Epoch 39/100
366/366 [==============================] - 1s 2ms/step - loss: 497.2462
Epoch 40/100
366/366 [==============================] - 1s 2ms/step - loss: 486.3542
Epoch 41/100
366/366 [==============================] - 1s 2ms/step - loss: 486.4242
Epoch 42/100
366/366 [==============================] - 1s 2ms/step - loss: 483.7357
Epoch 43/100
366/366 [==============================] - 1s 2ms/step - loss: 490.7452
Epoch 44/100
366/366 [==============================] - 1s 2ms/step - loss: 489.8795
Epoch 45/100
366/366 [==============================] - 1s 2ms/step - loss: 481.4828
Epoch 46/100
366/366 [==============================] - 1s 2ms/step - loss: 496.1435
Epoch 47/100
366/366 [==============================] - 1s 2ms/step - loss: 475.2236
Epoch 48/100
366/366 [==============================] - 1s 2ms/step - loss: 481.3419
Epoch 49/100
366/366 [==============================] - 1s 2ms/step - loss: 481.8268
Epoch 50/100
366/366 [==============================] - 1s 2ms/step - loss: 480.7569
Epoch 51/100
366/366 [==============================] - 1s 2ms/step - loss: 477.8250
Epoch 52/100
366/366 [==============================] - 1s 2ms/step - loss: 477.4001
Epoch 53/100
366/366 [==============================] - 1s 2ms/step - loss: 475.3209
Epoch 54/100
366/366 [==============================] - 1s 2ms/step - loss: 488.9156
Epoch 55/100
366/366 [==============================] - 1s 2ms/step - loss: 482.5419
Epoch 56/100
366/366 [==============================] - 1s 2ms/step - loss: 473.8824
Epoch 57/100
366/366 [==============================] - 1s 2ms/step - loss: 475.7103
Epoch 58/100
366/366 [==============================] - 1s 2ms/step - loss: 483.5769
Epoch 59/100
366/366 [==============================] - 1s 2ms/step - loss: 473.7039
Epoch 60/100
366/366 [==============================] - 1s 2ms/step - loss: 478.1961
Epoch 61/100
366/366 [==============================] - 1s 2ms/step - loss: 475.9818
Epoch 62/100
366/366 [==============================] - 1s 2ms/step - loss: 475.3704
Epoch 63/100
366/366 [==============================] - 1s 2ms/step - loss: 465.5941
Epoch 64/100
366/366 [==============================] - 1s 2ms/step - loss: 469.3299
Epoch 65/100
366/366 [==============================] - 1s 2ms/step - loss: 471.8680
Epoch 66/100
366/366 [==============================] - 1s 2ms/step - loss: 472.4709
Epoch 67/100
366/366 [==============================] - 1s 2ms/step - loss: 479.1653
Epoch 68/100
```

```
366/366 [==============================] - 1s 2ms/step - loss: 474.2932
Epoch 69/100
366/366 [==============================] - 1s 2ms/step - loss: 472.8345
Epoch 70/100
366/366 [==============================] - 1s 2ms/step - loss: 464.2877
Epoch 71/100
366/366 [==============================] - 1s 2ms/step - loss: 471.5324
Epoch 72/100
366/366 [==============================] - 1s 2ms/step - loss: 470.6287
Epoch 73/100
366/366 [==============================] - 1s 2ms/step - loss: 470.5486
Epoch 74/100
366/366 [==============================] - 1s 2ms/step - loss: 468.4950
Epoch 75/100
366/366 [==============================] - 1s 2ms/step - loss: 467.8003
Epoch 76/100
366/366 [==============================] - 1s 2ms/step - loss: 469.1608
Epoch 77/100
366/366 [==============================] - 1s 2ms/step - loss: 467.0385
Epoch 78/100
366/366 [==============================] - 1s 2ms/step - loss: 475.6812
Epoch 79/100
366/366 [==============================] - 1s 2ms/step - loss: 469.6342
Epoch 80/100
366/366 [==============================] - 1s 3ms/step - loss: 460.1815
Epoch 81/100
366/366 [==============================] - 1s 3ms/step - loss: 465.3875
Epoch 82/100
366/366 [==============================] - 1s 3ms/step - loss: 457.2628
Epoch 83/100
366/366 [==============================] - 1s 2ms/step - loss: 455.3539
Epoch 84/100
366/366 [==============================] - 1s 2ms/step - loss: 469.1591
Epoch 85/100
366/366 [==============================] - 1s 2ms/step - loss: 464.5729
Epoch 86/100
366/366 [==============================] - 1s 2ms/step - loss: 455.2595
Epoch 87/100
366/366 [==============================] - 1s 2ms/step - loss: 463.6300
Epoch 88/100
366/366 [==============================] - 1s 2ms/step - loss: 460.4900
Epoch 89/100
366/366 [==============================] - 1s 2ms/step - loss: 459.0284
Epoch 90/100
366/366 [==============================] - 1s 2ms/step - loss: 461.2994
Epoch 91/100
366/366 [==============================] - 1s 2ms/step - loss: 460.8718
Epoch 92/100
366/366 [==============================] - 1s 2ms/step - loss: 443.1979
Epoch 93/100
366/366 [==============================] - 1s 2ms/step - loss: 457.4888
Epoch 94/100
366/366 [==============================] - 1s 2ms/step - loss: 453.3166
Epoch 95/100
366/366 [==============================] - 1s 2ms/step - loss: 456.2263
Epoch 96/100
366/366 [==============================] - 1s 3ms/step - loss: 455.1035
Epoch 97/100
366/366 [==============================] - 1s 2ms/step - loss: 458.2699
Epoch 98/100
366/366 [==============================] - 1s 2ms/step - loss: 455.1377
Epoch 99/100
366/366 [==============================] - 1s 2ms/step - loss: 447.7731
Epoch 100/100
366/366 [==============================] - 1s 2ms/step - loss: 462.5804
```

Out[21]:
```
<tensorflow.python.keras.callbacks.History at 0x1bae9d0e490>
```

In [22]:
```
y_predict2 = lstm_model.predict(testX)
lstm_r=performance_results(Test_y,y_predict2.reshape(-1),"LSTM con ruido")
lstm_r
```

Out[22]:

	Mean Absolute Error	Root Mean Squared Error	MAPE	R2 score	Spearman r	Pearson r
LSTM con ruido	16.028496	22.309703	27.219646	0.571813	0.756128	0.75897

In [23]:
```
lstm_model1 = create_LSTM_model(20,lags)
print(lstm_model1.summary())
lstm_model1.fit(trainX1, Train_y1, epochs = 100, batch_size = 4, shuffle=False, verbose = 1)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 200)               166400
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 201
=================================================================
Total params: 166,601
Trainable params: 166,601
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
366/366 [==============================] - 1s 2ms/step - loss: 4460.1201
Epoch 2/100
366/366 [==============================] - 1s 2ms/step - loss: 2297.5081
Epoch 3/100
366/366 [==============================] - 1s 2ms/step - loss: 944.9974
Epoch 4/100
366/366 [==============================] - 1s 2ms/step - loss: 320.4531
Epoch 5/100
366/366 [==============================] - 1s 2ms/step - loss: 169.4916
Epoch 6/100
366/366 [==============================] - 1s 2ms/step - loss: 147.2931
Epoch 7/100
366/366 [==============================] - 1s 2ms/step - loss: 123.1693
Epoch 8/100
366/366 [==============================] - 1s 2ms/step - loss: 105.4118
Epoch 9/100
366/366 [==============================] - 1s 2ms/step - loss: 96.1884
Epoch 10/100
366/366 [==============================] - 1s 2ms/step - loss: 85.4019
Epoch 11/100
366/366 [==============================] - 1s 2ms/step - loss: 78.7268
Epoch 12/100
366/366 [==============================] - 1s 2ms/step - loss: 74.5032
Epoch 13/100
366/366 [==============================] - 1s 2ms/step - loss: 69.8641
Epoch 14/100
366/366 [==============================] - 1s 2ms/step - loss: 65.4711
Epoch 15/100
366/366 [==============================] - 1s 2ms/step - loss: 68.0123
Epoch 16/100
366/366 [==============================] - 1s 2ms/step - loss: 64.3062
Epoch 17/100
366/366 [==============================] - 1s 2ms/step - loss: 58.6522
Epoch 18/100
366/366 [==============================] - 1s 2ms/step - loss: 61.1773
Epoch 19/100
366/366 [==============================] - 1s 2ms/step - loss: 59.7393
Epoch 20/100
366/366 [==============================] - 1s 2ms/step - loss: 57.8554
Epoch 21/100
366/366 [==============================] - 1s 2ms/step - loss: 56.8119
Epoch 22/100
366/366 [==============================] - 1s 2ms/step - loss: 53.9605
Epoch 23/100
366/366 [==============================] - 1s 2ms/step - loss: 55.4284
Epoch 24/100
366/366 [==============================] - 1s 2ms/step - loss: 53.1064
Epoch 25/100
366/366 [==============================] - 1s 2ms/step - loss: 48.4745
Epoch 26/100
366/366 [==============================] - 1s 2ms/step - loss: 50.9590
Epoch 27/100
366/366 [==============================] - 1s 2ms/step - loss: 51.4371
Epoch 28/100
366/366 [==============================] - 1s 2ms/step - loss: 50.5071
Epoch 29/100
366/366 [==============================] - 1s 2ms/step - loss: 48.9117
Epoch 30/100
366/366 [==============================] - 1s 2ms/step - loss: 51.4359
```

```
Epoch 31/100
366/366 [==============================] - 1s 2ms/step - loss: 50.1102
Epoch 32/100
366/366 [==============================] - 1s 2ms/step - loss: 48.0948
Epoch 33/100
366/366 [==============================] - 1s 2ms/step - loss: 48.2612
Epoch 34/100
366/366 [==============================] - 1s 2ms/step - loss: 50.2335
Epoch 35/100
366/366 [==============================] - 1s 2ms/step - loss: 48.6762
Epoch 36/100
366/366 [==============================] - 1s 2ms/step - loss: 47.5642
Epoch 37/100
366/366 [==============================] - 1s 2ms/step - loss: 51.3102
Epoch 38/100
366/366 [==============================] - 1s 2ms/step - loss: 48.7838
Epoch 39/100
366/366 [==============================] - 1s 2ms/step - loss: 48.5221
Epoch 40/100
366/366 [==============================] - 1s 2ms/step - loss: 50.2045
Epoch 41/100
366/366 [==============================] - 1s 2ms/step - loss: 49.0926
Epoch 42/100
366/366 [==============================] - 1s 1ms/step - loss: 49.1978
Epoch 43/100
366/366 [==============================] - 1s 2ms/step - loss: 47.2160
Epoch 44/100
366/366 [==============================] - 1s 2ms/step - loss: 47.3896
Epoch 45/100
366/366 [==============================] - 1s 2ms/step - loss: 45.2021
Epoch 46/100
366/366 [==============================] - 1s 2ms/step - loss: 45.6178
Epoch 47/100
366/366 [==============================] - 1s 2ms/step - loss: 47.1028
Epoch 48/100
366/366 [==============================] - 1s 2ms/step - loss: 47.3453
Epoch 49/100
366/366 [==============================] - 1s 2ms/step - loss: 47.2734
Epoch 50/100
366/366 [==============================] - 1s 2ms/step - loss: 46.8310
Epoch 51/100
366/366 [==============================] - 1s 2ms/step - loss: 46.6980
Epoch 52/100
366/366 [==============================] - 1s 2ms/step - loss: 47.2124
Epoch 53/100
366/366 [==============================] - 1s 2ms/step - loss: 45.7002
Epoch 54/100
366/366 [==============================] - 1s 2ms/step - loss: 47.4084
Epoch 55/100
366/366 [==============================] - 1s 2ms/step - loss: 47.0147
Epoch 56/100
366/366 [==============================] - 1s 2ms/step - loss: 45.0181
Epoch 57/100
366/366 [==============================] - 1s 2ms/step - loss: 44.7146
Epoch 58/100
366/366 [==============================] - 1s 2ms/step - loss: 44.5901
Epoch 59/100
366/366 [==============================] - 1s 2ms/step - loss: 47.0523
Epoch 60/100
366/366 [==============================] - 1s 2ms/step - loss: 47.9815
Epoch 61/100
366/366 [==============================] - 1s 2ms/step - loss: 46.3653
Epoch 62/100
366/366 [==============================] - 1s 2ms/step - loss: 47.1460
Epoch 63/100
366/366 [==============================] - 1s 2ms/step - loss: 47.4269
Epoch 64/100
366/366 [==============================] - 1s 2ms/step - loss: 44.5486
Epoch 65/100
366/366 [==============================] - 1s 2ms/step - loss: 48.9914
Epoch 66/100
366/366 [==============================] - 1s 2ms/step - loss: 44.8356
Epoch 67/100
366/366 [==============================] - 1s 2ms/step - loss: 45.2982
Epoch 68/100
```

```
366/366 [==============================] - 1s 2ms/step - loss: 44.4051
Epoch 69/100
366/366 [==============================] - 1s 2ms/step - loss: 47.2791
Epoch 70/100
366/366 [==============================] - 1s 2ms/step - loss: 44.3279
Epoch 71/100
366/366 [==============================] - 1s 2ms/step - loss: 43.5524
Epoch 72/100
366/366 [==============================] - 1s 2ms/step - loss: 45.6268
Epoch 73/100
366/366 [==============================] - 1s 2ms/step - loss: 44.2594
Epoch 74/100
366/366 [==============================] - 1s 2ms/step - loss: 44.7758
Epoch 75/100
366/366 [==============================] - 1s 2ms/step - loss: 44.6075
Epoch 76/100
366/366 [==============================] - 1s 2ms/step - loss: 45.9572
Epoch 77/100
366/366 [==============================] - 1s 2ms/step - loss: 43.6693
Epoch 78/100
366/366 [==============================] - 1s 2ms/step - loss: 44.9120
Epoch 79/100
366/366 [==============================] - 1s 2ms/step - loss: 47.8812
Epoch 80/100
366/366 [==============================] - 1s 2ms/step - loss: 45.1761
Epoch 81/100
366/366 [==============================] - 1s 2ms/step - loss: 46.6581
Epoch 82/100
366/366 [==============================] - 1s 2ms/step - loss: 45.6381
Epoch 83/100
366/366 [==============================] - 1s 3ms/step - loss: 43.0268
Epoch 84/100
366/366 [==============================] - 1s 2ms/step - loss: 44.3188
Epoch 85/100
366/366 [==============================] - 1s 2ms/step - loss: 43.2963
Epoch 86/100
366/366 [==============================] - 1s 2ms/step - loss: 46.4588
Epoch 87/100
366/366 [==============================] - 1s 2ms/step - loss: 43.9100
Epoch 88/100
366/366 [==============================] - 1s 2ms/step - loss: 44.5885
Epoch 89/100
366/366 [==============================] - 1s 2ms/step - loss: 42.4340
Epoch 90/100
366/366 [==============================] - 1s 2ms/step - loss: 43.9510
Epoch 91/100
366/366 [==============================] - 1s 2ms/step - loss: 44.2094
Epoch 92/100
366/366 [==============================] - 1s 2ms/step - loss: 45.1435
Epoch 93/100
366/366 [==============================] - 1s 2ms/step - loss: 44.0984: 0s - loss: 44.25
Epoch 94/100
366/366 [==============================] - 1s 2ms/step - loss: 43.6625
Epoch 95/100
366/366 [==============================] - 1s 2ms/step - loss: 41.4797
Epoch 96/100
366/366 [==============================] - 1s 2ms/step - loss: 46.2172
Epoch 97/100
366/366 [==============================] - 1s 2ms/step - loss: 42.7811
Epoch 98/100
366/366 [==============================] - 1s 2ms/step - loss: 45.0474
Epoch 99/100
366/366 [==============================] - 1s 2ms/step - loss: 46.4472
Epoch 100/100
366/366 [==============================] - 1s 2ms/step - loss: 44.2555
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x1baedf46730>
```

In [24]:
```
y_predict4 = lstm_model1.predict(testX1)
lstm_sr=performance_results(Test_y1,y_predict4.reshape(-1),"LSTM sin ruido")
lstm_sr
```

Out[24]:

	Mean Absolute Error	Root Mean Squared Error	MAPE	R2 score	Spearman r	Pearson r
LSTM sin ruido	4.581897	5.949508	6.703573	0.947111	0.969842	0.97419

1.8. Obtener las siguientes métricas de desempeño para el conjunto de Test:

- Mean Absolute Error
- Root Mean Squared Error
- MAPE
- R2 score
- Spearman r
- Person r

Realizar una tabla comparativa y concluir.

In [25]:
```
pd.concat([mlp_r,mlp_sr,lstm_r,lstm_sr])
```

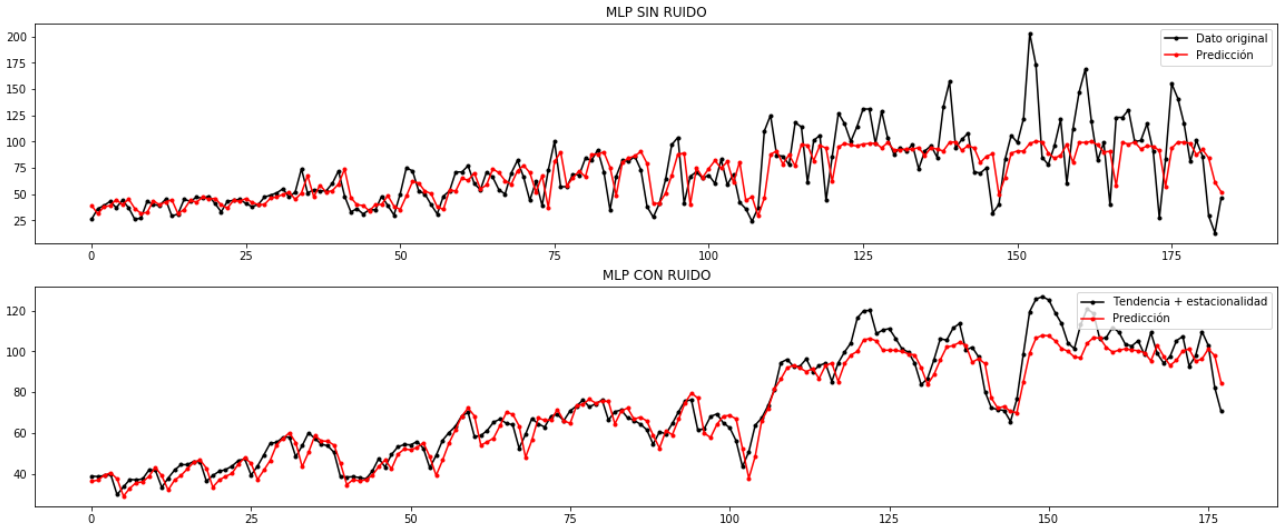
Out[25]:

	Mean Absolute Error	Root Mean Squared Error	MAPE	R2 score	Spearman r	Pearson r
MPL con ruido	16.687112	24.034368	27.319943	0.503052	0.754615	0.716600
MLP sin ruido	5.671728	7.303217	8.012593	0.920306	0.965481	0.965316
LSTM con ruido	16.028496	22.309703	27.219646	0.571813	0.756128	0.758970
LSTM sin ruido	4.581897	5.949508	6.703573	0.947111	0.969842	0.974190

1.9. Graficar las predicciones para el período 2022 en conjunto con los datos reales.

In [26]:
```
fig, axs = plt.subplots(2, 1,figsize=(20,8))

axs[0].plot(Test_y,'.-k', label='Dato original')
axs[0].plot(y_predict,'.-r', label='Predicción')
axs[0].title.set_text('MLP SIN RUIDO')

axs[1].title.set_text('MLP CON RUIDO')
axs[1].plot(Test_y1,'.-k', label='Tendencia + estacionalidad')
axs[1].plot(y_predict1,'.-r', label='Predicción')
axs[0].legend(loc="upper right")
axs[1].legend(loc="upper right")
```

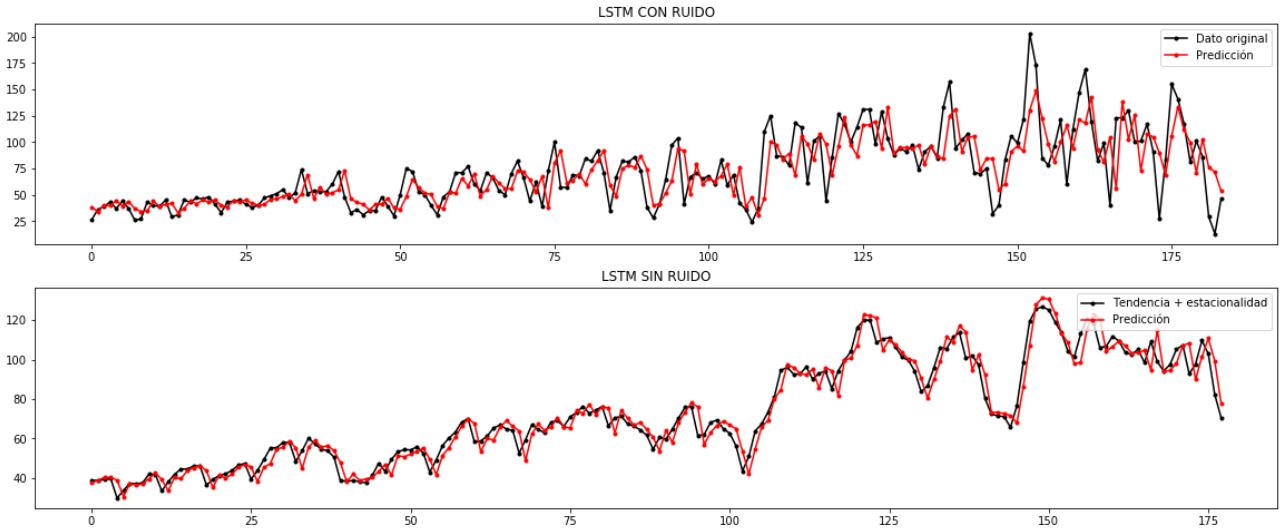
Out[26]:
```
<matplotlib.legend.Legend at 0x1baf20aef70>
```



In [27]:
```
fig, axs = plt.subplots(2, 1,figsize=(20,8))

axs[0].plot(Test_y,'.-k', label='Dato original')
axs[0].plot(y_predict2,'.-r', label='Predicción')
axs[0].title.set_text('LSTM CON RUIDO')
axs[1].title.set_text('LSTM SIN RUIDO')
```

```
axs[1].plot(Test_y1,'.-k', label='Tendencia + estacionalidad')
axs[1].plot(y_predict4,'.-r', label='Predicción')
axs[0].legend(loc="upper right")
axs[1].legend(loc="upper right")
```

Out[27]: <matplotlib.legend.Legend at 0x1baf0d6c0d0>



Concluir

Para la creación de los modelos, se realizó pruebas con distintas configuraciones, las cuales no mostraban grandes diferencias al variar la cantidad de neuronas y capas. Se obtuvo el mejor desempeño con las siguientes configuraciones:

- MLP de 4 capas secuenciales, con 7 lags y 14 neuronas de entrada con activación sigmoidea, con un 25% de Dropout para reducir el ajuste y una capa de salida con 1 neurona con activación lineal. Lo que resulta en 127 parámetros a entrenar (112 en la capa densa y 15 a la salida).
- LSTM de 4 capas secuenciales, con 7 lags, 40 bloques, 14 neuronas de entrada con activación sigmoidea, con un 25% de Dropout para reducir el ajuste y una capa de salida con 1 neurona con activación lineal. Lo que resulta en 166601 parámetros a entrenar (166400 en la capa densa y 201 a la salida).

Teniendo en cuenta el alto nivel de ruido la serie de tiempo, se consideraron datos con y sin ruido, para evaluar los modelos, obteniendo desempeños considerablemente mejores para la serie de tiempo sin ruido, mejorando las métricas de correlación en más de un 20%, alcanzando valores sobre 0.96 en el Spearman y Pearson y sobre 0.92 para el R2 score. Por otro lado, el MAPE mejoró en un 20%, llegando a valores menores de 8%, para ambos modelos.

Tanto las métricas de error como las de correlación, muestran un mejor desempeño para el modelo LSTM, lo que era esperado, ya que se trata de un modelo de mayor complejidad que tiene la capacidad de memorizar secuencias. Además, existe una gran diferencia en la cantidad de parámetros entrenados. Sin embargo, la diferencia en el desempeño logrado por el LSTM se podría decir que es marginal, considerando las métricas MAPE, Spearman y Pearson, que lograron mejoras menores a un punto porcentual.

Se concluye que, para el set de datos seleccionado, se podría usar indistintamente los modelos MLP y LSTM, ya que presentan desempeños similares, sin embargo, dependiendo de la disponibilidad de recursos, podría ser recomendable utilizar el modelo más simple.

2. Clasificación de Imágenes usando redes neuronales convolucionales (30 puntos)

En esta pregunta se estudiará un modelo de red neuronal convolucional para la clasificación de imágenes. Para ello se utilizará el conjunto de datos de "piedra", "papel" y "tijeras" de Kaggle.

- Rock, paper, scissors

Para la implementación de esta pregunta se pueden apoyar de los siguientes recursos:

Ejemplo de CNN

Ejemplo con Keras

In [28]:
```
def input_target_split(train_dir,labels):
    dataset = []
    count = 0
    for label in labels:
        folder = os.path.join(train_dir,label)
        for image in os.listdir(folder):
            img=load_img(os.path.join(folder,image), target_size=(150,150))
            img=img_to_array(img)
            img=img/255.0

            dataset.append((img,count))
        print(f'\rCompleted: {label}',end='')
        count+=1
    random.shuffle(dataset)
    X, y = zip(*dataset)

    return np.array(X),np.array(y)
```

2.1 Realizar la descarga del conjunto de datos

In [29]:
```
directory ='C:\\Users\\pablo.walters\\Desktop\\clase\\taller 3\\rockpaperscissors'
labels = ['paper','scissors','rock']
nb = len(labels)
X, y = input_target_split(directory,labels)
```

```
Completed: rocksors
```

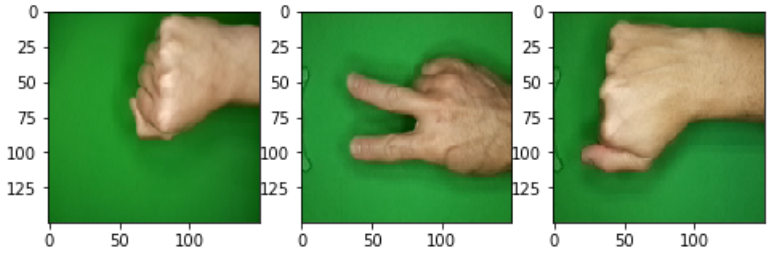
2.2 Mostrar 1 imagen de cada categoría: Piedra, Papel y Tijera.

In [50]:
```
np.unique(y,return_counts=True)
```

Out[50]:
```
(array([0, 1, 2]), array([712, 750, 726], dtype=int64))
```

In [51]:
```
fig, axs = plt.subplots(1, 3,figsize=(8,8))

axs[0].imshow(X[0])
axs[1].imshow(X[1])
axs[2].imshow(X[2])
```

Out[51]:
```
<matplotlib.image.AxesImage at 0x1bb2b2dfc10>
```



2.3 Separar los datos en conjunto de entranamiento y Test. Considerar 70% de los datos para entrenamiento.

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

2.4. Implementar una red neuronal convolucional

In [33]:
```
datagen = ImageDataGenerator(horizontal_flip=True,vertical_flip=True,rotation_range=15,zoom_range=0.1,width_shi
testgen = ImageDataGenerator()

datagen.fit(X_train)
testgen.fit(X_test)

XF=X_test
```

```
YF=y_test

y_train= np.eye(nb)[y_train]
y_test = np.eye(nb)[y_test]
```

In [34]:
```
model = Sequential()
model.add(Conv2D(16, (3,3), input_shape=(150,150,3), activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(16, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=3, activation='softmax'))

model.compile(optimizer = tf.keras.optimizers.Adam(lr = 0.001), loss = 'categorical_crossentropy', metrics=['ac
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 16)        2320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               2654336
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 387
=================================================================
Total params: 2,657,491
Trainable params: 2,657,491
Non-trainable params: 0
_________________________________________________________________
```

2.5 Ajustar la red neuronal convolucional con el conjunto de entranamiento

In [35]:
```
filepath= "model_cnn_final.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max', save_

early_stopping = EarlyStopping(monitor='val_loss',min_delta = 0, patience = 5, verbose = 1, restore_best_weights


callbacks_list = [
        checkpoint,
        early_stopping,]
hist = model.fit_generator(datagen.flow(X_train,y_train,batch_size=32),
                                        validation_data=testgen.flow(X_test,y_test,batch_size=32),
                                        epochs=50,
                                        callbacks=callbacks_list)
```

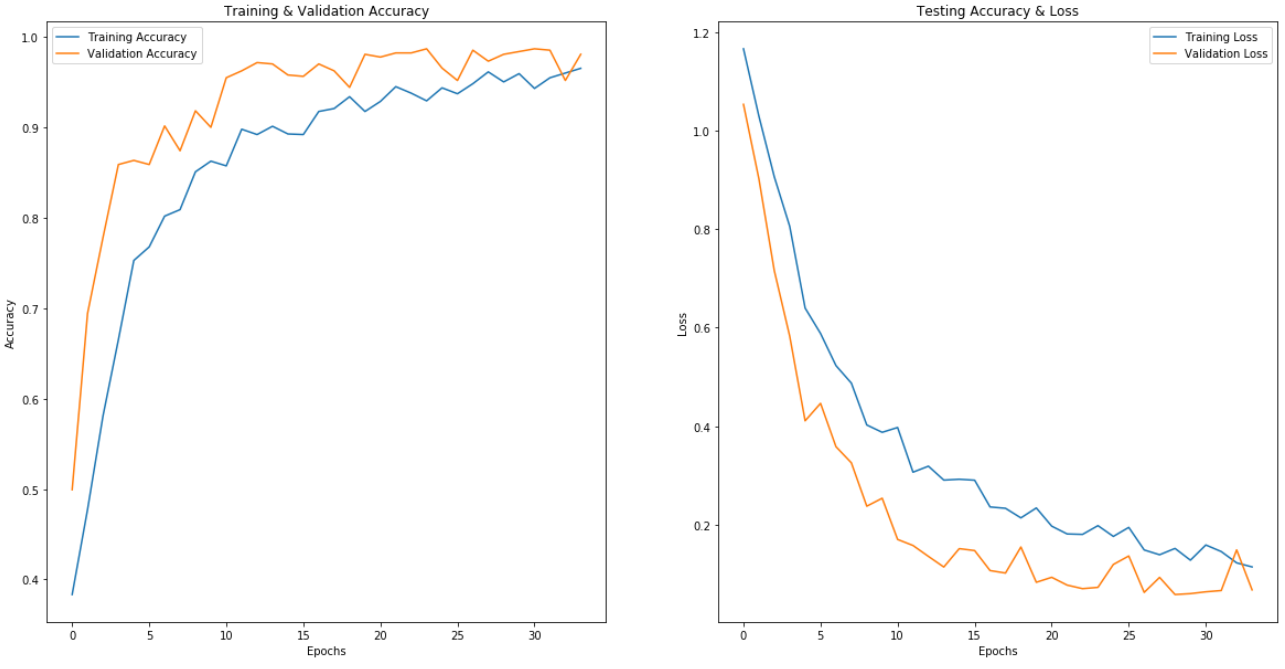
```
WARNING:tensorflow:From C:\Users\pablo.walters\AppData\Local\Temp\ipykernel_15036\3904658576.py:10: Model.fit_g
enerator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/50
48/48 [==============================] - ETA: 0s - loss: 1.1670 - accuracy: 0.3834
Epoch 00001: val_accuracy improved from -inf to 0.49924, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 301ms/step - loss: 1.1670 - accuracy: 0.3834 - val_loss: 1.0541 -
val_accuracy: 0.4992
Epoch 2/50
48/48 [==============================] - ETA: 0s - loss: 1.0300 - accuracy: 0.4775
Epoch 00002: val_accuracy improved from 0.49924 to 0.69406, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 283ms/step - loss: 1.0300 - accuracy: 0.4775 - val_loss: 0.9025 -
val_accuracy: 0.6941
Epoch 3/50
48/48 [==============================] - ETA: 0s - loss: 0.9067 - accuracy: 0.5807
Epoch 00003: val_accuracy improved from 0.69406 to 0.77778, saving model to model_cnn_final.h5
48/48 [==============================] - 15s 310ms/step - loss: 0.9067 - accuracy: 0.5807 - val_loss: 0.7158 -
val_accuracy: 0.7778
Epoch 4/50
48/48 [==============================] - ETA: 0s - loss: 0.8060 - accuracy: 0.6649
Epoch 00004: val_accuracy improved from 0.77778 to 0.85845, saving model to model_cnn_final.h5
48/48 [==============================] - 17s 347ms/step - loss: 0.8060 - accuracy: 0.6649 - val_loss: 0.5826 -
val_accuracy: 0.8584
Epoch 5/50
48/48 [==============================] - ETA: 0s - loss: 0.6399 - accuracy: 0.7524
Epoch 00005: val_accuracy improved from 0.85845 to 0.86301, saving model to model_cnn_final.h5
48/48 [==============================] - 15s 318ms/step - loss: 0.6399 - accuracy: 0.7524 - val_loss: 0.4106 -
val_accuracy: 0.8630
Epoch 6/50
48/48 [==============================] - ETA: 0s - loss: 0.5882 - accuracy: 0.7675
Epoch 00006: val_accuracy did not improve from 0.86301
48/48 [==============================] - 14s 291ms/step - loss: 0.5882 - accuracy: 0.7675 - val_loss: 0.4461 -
val_accuracy: 0.8584
Epoch 7/50
48/48 [==============================] - ETA: 0s - loss: 0.5229 - accuracy: 0.8014
Epoch 00007: val_accuracy improved from 0.86301 to 0.90107, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 285ms/step - loss: 0.5229 - accuracy: 0.8014 - val_loss: 0.3581 -
val_accuracy: 0.9011
Epoch 8/50
48/48 [==============================] - ETA: 0s - loss: 0.4872 - accuracy: 0.8086
Epoch 00008: val_accuracy did not improve from 0.90107
48/48 [==============================] - 15s 313ms/step - loss: 0.4872 - accuracy: 0.8086 - val_loss: 0.3252 -
val_accuracy: 0.8737
Epoch 9/50
48/48 [==============================] - ETA: 0s - loss: 0.4022 - accuracy: 0.8504
Epoch 00009: val_accuracy improved from 0.90107 to 0.91781, saving model to model_cnn_final.h5
48/48 [==============================] - 15s 321ms/step - loss: 0.4022 - accuracy: 0.8504 - val_loss: 0.2370 -
val_accuracy: 0.9178
Epoch 10/50
48/48 [==============================] - ETA: 0s - loss: 0.3872 - accuracy: 0.8622
Epoch 00010: val_accuracy did not improve from 0.91781
48/48 [==============================] - 15s 319ms/step - loss: 0.3872 - accuracy: 0.8622 - val_loss: 0.2532 -
val_accuracy: 0.8995
Epoch 11/50
48/48 [==============================] - ETA: 0s - loss: 0.3970 - accuracy: 0.8570
Epoch 00011: val_accuracy improved from 0.91781 to 0.95434, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 299ms/step - loss: 0.3970 - accuracy: 0.8570 - val_loss: 0.1694 -
val_accuracy: 0.9543
Epoch 12/50
48/48 [==============================] - ETA: 0s - loss: 0.3062 - accuracy: 0.8975
Epoch 00012: val_accuracy improved from 0.95434 to 0.96195, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 291ms/step - loss: 0.3062 - accuracy: 0.8975 - val_loss: 0.1571 -
val_accuracy: 0.9619
Epoch 13/50
48/48 [==============================] - ETA: 0s - loss: 0.3184 - accuracy: 0.8916
Epoch 00013: val_accuracy improved from 0.96195 to 0.97108, saving model to model_cnn_final.h5
48/48 [==============================] - 15s 315ms/step - loss: 0.3184 - accuracy: 0.8916 - val_loss: 0.1350 -
val_accuracy: 0.9711
Epoch 14/50
48/48 [==============================] - ETA: 0s - loss: 0.2901 - accuracy: 0.9007
Epoch 00014: val_accuracy did not improve from 0.97108
48/48 [==============================] - 16s 331ms/step - loss: 0.2901 - accuracy: 0.9007 - val_loss: 0.1133 -
val_accuracy: 0.9696
Epoch 15/50
```

```
48/48 [==============================] - ETA: 0s - loss: 0.2918 - accuracy: 0.8922
Epoch 00015: val_accuracy did not improve from 0.97108
48/48 [==============================] - 16s 333ms/step - loss: 0.2918 - accuracy: 0.8922 - val_loss: 0.1508 -
val_accuracy: 0.9574
Epoch 16/50
48/48 [==============================] - ETA: 0s - loss: 0.2899 - accuracy: 0.8916
Epoch 00016: val_accuracy did not improve from 0.97108
48/48 [==============================] - 15s 309ms/step - loss: 0.2899 - accuracy: 0.8916 - val_loss: 0.1468 -
val_accuracy: 0.9559
Epoch 17/50
48/48 [==============================] - ETA: 0s - loss: 0.2355 - accuracy: 0.9170
Epoch 00017: val_accuracy did not improve from 0.97108
48/48 [==============================] - 15s 308ms/step - loss: 0.2355 - accuracy: 0.9170 - val_loss: 0.1062 -
val_accuracy: 0.9696
Epoch 18/50
48/48 [==============================] - ETA: 0s - loss: 0.2328 - accuracy: 0.9203
Epoch 00018: val_accuracy did not improve from 0.97108
48/48 [==============================] - 18s 368ms/step - loss: 0.2328 - accuracy: 0.9203 - val_loss: 0.1011 -
val_accuracy: 0.9619
Epoch 19/50
48/48 [==============================] - ETA: 0s - loss: 0.2132 - accuracy: 0.9334
Epoch 00019: val_accuracy did not improve from 0.97108
48/48 [==============================] - 17s 350ms/step - loss: 0.2132 - accuracy: 0.9334 - val_loss: 0.1540 -
val_accuracy: 0.9437
Epoch 20/50
48/48 [==============================] - ETA: 0s - loss: 0.2335 - accuracy: 0.9170
Epoch 00020: val_accuracy improved from 0.97108 to 0.98021, saving model to model_cnn_final.h5
48/48 [==============================] - 14s 294ms/step - loss: 0.2335 - accuracy: 0.9170 - val_loss: 0.0823 -
val_accuracy: 0.9802
Epoch 21/50
48/48 [==============================] - ETA: 0s - loss: 0.1964 - accuracy: 0.9282
Epoch 00021: val_accuracy did not improve from 0.98021
48/48 [==============================] - 15s 302ms/step - loss: 0.1964 - accuracy: 0.9282 - val_loss: 0.0924 -
val_accuracy: 0.9772
Epoch 22/50
48/48 [==============================] - ETA: 0s - loss: 0.1806 - accuracy: 0.9445
Epoch 00022: val_accuracy improved from 0.98021 to 0.98174, saving model to model_cnn_final.h5
48/48 [==============================] - 16s 324ms/step - loss: 0.1806 - accuracy: 0.9445 - val_loss: 0.0766 -
val_accuracy: 0.9817
Epoch 23/50
48/48 [==============================] - ETA: 0s - loss: 0.1797 - accuracy: 0.9373
Epoch 00023: val_accuracy did not improve from 0.98174
48/48 [==============================] - 15s 323ms/step - loss: 0.1797 - accuracy: 0.9373 - val_loss: 0.0693 -
val_accuracy: 0.9817
Epoch 24/50
48/48 [==============================] - ETA: 0s - loss: 0.1975 - accuracy: 0.9288
Epoch 00024: val_accuracy improved from 0.98174 to 0.98630, saving model to model_cnn_final.h5
48/48 [==============================] - 15s 307ms/step - loss: 0.1975 - accuracy: 0.9288 - val_loss: 0.0720 -
val_accuracy: 0.9863
Epoch 25/50
48/48 [==============================] - ETA: 0s - loss: 0.1756 - accuracy: 0.9432
Epoch 00025: val_accuracy did not improve from 0.98630
48/48 [==============================] - 14s 290ms/step - loss: 0.1756 - accuracy: 0.9432 - val_loss: 0.1186 -
val_accuracy: 0.9650
Epoch 26/50
48/48 [==============================] - ETA: 0s - loss: 0.1940 - accuracy: 0.9366
Epoch 00026: val_accuracy did not improve from 0.98630
48/48 [==============================] - 15s 314ms/step - loss: 0.1940 - accuracy: 0.9366 - val_loss: 0.1358 -
val_accuracy: 0.9513
Epoch 27/50
48/48 [==============================] - ETA: 0s - loss: 0.1481 - accuracy: 0.9477
Epoch 00027: val_accuracy did not improve from 0.98630
48/48 [==============================] - 16s 325ms/step - loss: 0.1481 - accuracy: 0.9477 - val_loss: 0.0617 -
val_accuracy: 0.9848
Epoch 28/50
48/48 [==============================] - ETA: 0s - loss: 0.1383 - accuracy: 0.9608
Epoch 00028: val_accuracy did not improve from 0.98630
48/48 [==============================] - 15s 320ms/step - loss: 0.1383 - accuracy: 0.9608 - val_loss: 0.0923 -
val_accuracy: 0.9726
Epoch 29/50
48/48 [==============================] - ETA: 0s - loss: 0.1512 - accuracy: 0.9497
Epoch 00029: val_accuracy did not improve from 0.98630
48/48 [==============================] - 15s 304ms/step - loss: 0.1512 - accuracy: 0.9497 - val_loss: 0.0574 -
val_accuracy: 0.9802
Epoch 30/50
```

```
48/48 [==============================] - ETA: 0s - loss: 0.1272 - accuracy: 0.9589
Epoch 00030: val_accuracy did not improve from 0.98630
48/48 [==============================] - 14s 298ms/step - loss: 0.1272 - accuracy: 0.9589 - val_loss: 0.0593 -
val_accuracy: 0.9833
Epoch 31/50
48/48 [==============================] - ETA: 0s - loss: 0.1582 - accuracy: 0.9425
Epoch 00031: val_accuracy did not improve from 0.98630
48/48 [==============================] - 16s 328ms/step - loss: 0.1582 - accuracy: 0.9425 - val_loss: 0.0630 -
val_accuracy: 0.9863
Epoch 32/50
48/48 [==============================] - ETA: 0s - loss: 0.1448 - accuracy: 0.9543
Epoch 00032: val_accuracy did not improve from 0.98630
48/48 [==============================] - 16s 332ms/step - loss: 0.1448 - accuracy: 0.9543 - val_loss: 0.0655 -
val_accuracy: 0.9848
Epoch 33/50
48/48 [==============================] - ETA: 0s - loss: 0.1218 - accuracy: 0.9595
Epoch 00033: val_accuracy did not improve from 0.98630
48/48 [==============================] - 15s 305ms/step - loss: 0.1218 - accuracy: 0.9595 - val_loss: 0.1482 -
val_accuracy: 0.9513
Epoch 34/50
48/48 [==============================] - ETA: 0s - loss: 0.1135 - accuracy: 0.9647
Epoch 00034: val_accuracy did not improve from 0.98630
Restoring model weights from the end of the best epoch.
48/48 [==============================] - 14s 294ms/step - loss: 0.1135 - accuracy: 0.9647 - val_loss: 0.0669 -
val_accuracy: 0.9802
Epoch 00034: early stopping
```

In [ ]:

In [36]:
```
# Grafica de entrenamiento
epochs = [i for i in range(len(hist.epoch))]
fig , ax = plt.subplots(1,2)
fig.set_size_inches(20,10)

ax[0].plot(epochs, hist.history['accuracy'],     label='Training Accuracy')
ax[0].plot(epochs, hist.history['val_accuracy'], label= 'Validation Accuracy')
ax[0].set_title('Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs, hist.history['loss'],     label = 'Training Loss')
ax[1].plot(epochs, hist.history['val_loss'], label = 'Validation Loss')
ax[1].set_title('Testing Accuracy & Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```

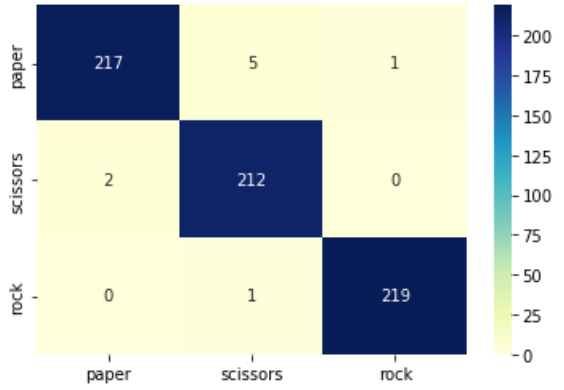


2.6 Obtener la matriz de confusión y el accuracy del clasificador para el conjunto de test.

In [37]:
```
model_saved = tf.keras.models.load_model('model_cnn_final.h5')
y_pred = model_saved.predict(X_test)
pred = np.argmax(y_pred,axis=1)
y_test_data = np.argmax(y_test,axis=1)
```

In [38]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm  = pd.DataFrame(confusion_matrix(y_test_data, pred),index=labels,columns=labels)
sns.heatmap(cm, annot=True,fmt="d",cmap="YlGnBu")
```

Out[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bb242a2280>
```

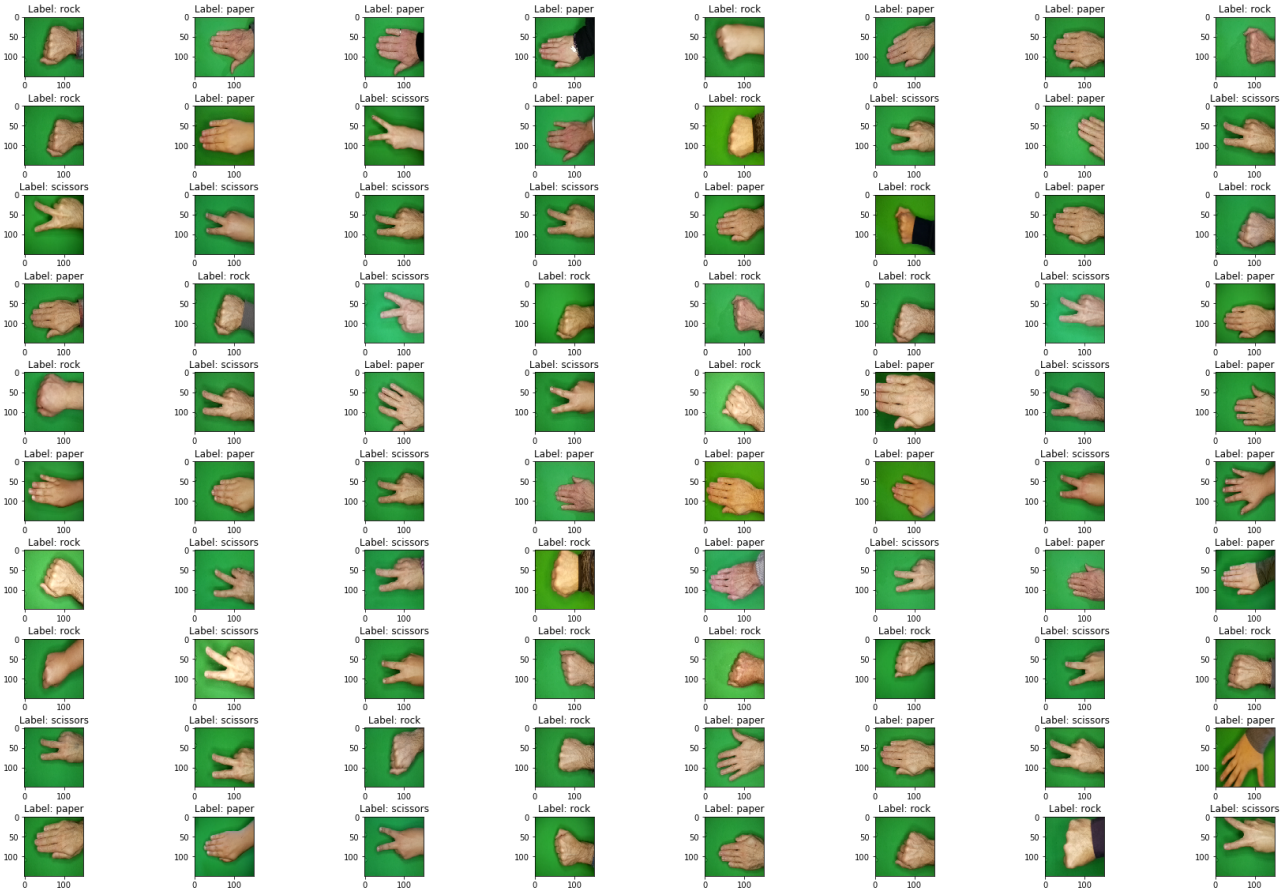


In [39]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test_data, pred,target_names = labels))
```

```
              precision    recall  f1-score   support

       paper       0.99      0.97      0.98       223
    scissors       0.97      0.99      0.98       214
        rock       1.00      1.00      1.00       220

    accuracy                           0.99       657
   macro avg       0.99      0.99      0.99       657
weighted avg       0.99      0.99      0.99       657
```

2.7 Mostrar algunos ejemplos de predicción.

In [40]:
```
plt.figure(figsize = (30 , 20))
n = 0
for i in range(80):
    n+=1
    plt.subplot(10 , 8, n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.3)
    plt.imshow(XF[i])
    if pred[i] == YF [i]:
        plt.title(f'Label: {labels[pred[i]]}')
    else:
        plt.title('@@@ MALA:'+labels[pred[i]])
```



Conclusión de la actividad:

Para la clasificación de imágenes “rock paper scissors”, se crea una red neuronal convolucional con 2 niveles convolucionales que parten por una capa con 16 filtros de 3 por 3, con función de activación relu. Seguida por 1 capa de maxpooling y finalmente 2 capas densas con función de activación relu y softmax. Resultando en 2,657,491 parámetros a entrenar.

Se ajustó el modelo con el conjunto de entrenamiento (70%) con 50 épocas y 32 de batch, obteniendo una alta exactitud (0.9802).

Al observar las gráficas del historial de entrenamiento del modelo, se puede notar un rápido aumento de la exactitud, en las primeras 5 épocas, de igual forma se observa una rápida disminución del error, en el mismo periodo. Por otro lado, los datos de validación obtuvieron un mejor desempeño a lo largo de casi todo el proceso. Finalmente, al evaluar el mejor modelo de clasificación con el conjunto de test, se obtienen desempeños altísimos, cercanos al 100%.

Se concluye que los modelos de redes neuronales convolucionales, por su alta complejidad, tienen performances superiores a los modelos multicapa utilizados anteriormente.

In [ ]: