Anslisis de resultados CHINESE-MNIST

Adrian Vargas Orellana
1606868

Indice

1. Introducción

En este documento se analizaran todos los aspectos necesarios del caso Kaggle CHINESE-MNIST
Este problema consiste en clasificar 15 caracteres numericos chinos apartir de imagenes de 64x64 pixeles, es el mismo tipo de problema que MNIST pero aplicado a caracteres chinos.
La base de datos se recolecto haciendo escribir a 100 chinos 15 numeros 10 veces cada numero.
Lo que da una muestra total de 15000 numeros, guardados de la siguiente forma para cada archivo:

`original name (example): Locate{1,3,4}.jpg`
`index extracted: suite_id: 1, sample_id: 3, code: 4`

`resulted file name: input_1_3_4.jpg`

Tambien se guardo un archivo llamado `chinese_mnist.csv`, este csv aparte de los datos proporcionados anteriormente, tambien tienen los datos de la *Figura 1*

	value	character	code
0	0	零	1
1	1	一	2
2	2	二	3
3	3	三	4
4	4	四	5
5	5	五	6
6	6	六	7
7	7	七	8
8	8	八	9
9	9	九	10
10	10	十	11
11	100	百	12
12	1000	千	13
13	10000	万	14
14	100000000	亿	15

Figura 1

2. Anàlisis base de datos

Antes de comenzar con el análisis hace falta entender el funcionamiento de la base de datos proporcionada, y ver que modelos se pueden adecuar más para resolver el problema que tenemos.

2.1. Explicació base de datos

La base de datos està formada por 15.000 imágenes y un csv de 15.000 filas, y 5 columnas

Cada fila está asociada a una imagen de 64x64 como se explicó previamente, mediante el **suite_id**, **sample_id** y el **code**, a partir **del cual se identifica la imagen y la columna value y character** lo relacionan con el retorno y el carácter que representan respectivamente, como se puede observar en la ***Figura 2***

Dimensionalitat de la BBDD: (15000, 5)

Tabla de la BBDD:

	suite_id	sample_id	code	value	character
0	1	1	10	9	九
1	1	10	10	9	九
2	1	2	10	9	九
3	1	3	10	9	九
4	1	4	10	9	九
...	...	...	...	...	...
14995	99	5	9	8	八
14996	99	6	9	8	八
14997	99	7	9	8	八
14998	99	8	9	8	八
14999	99	9	9	8	八

15000 rows × 5 columns

Figura 2. Base de datos

2.2. Tratamiento de la base de datos

Para comenzar con el tratamiento, primero comenzaré viendo si hay datos nulos en el CSV, y comprobando que están los 15.000 archivos y no hay valores nulos.

```
#percentatges nulls
print(dataset.isnull().sum()/len(dataset)*100)
```

```
suite_id     0.0
sample_id    0.0
code         0.0
value        0.0
character    0.0
dtype: float64
```

```
#caracters de sortida:
sortides = dataset['character'].unique()
print(sortides)
```

```
['九' '十' '百' '千' '万' '亿' '零' '一' '二' '三' '四' '五' '六' '七' '八']
```

Figura 3. comprobación nulos

Como se puede observar, no hay valores nulos, y están los 15 caracteres necesarios.
En este momento decidí imprimir unas cuantas imágenes para confirmar que cargan correctamente, y que son lo que corresponde, como se puede observar en la ***Figura 4.***

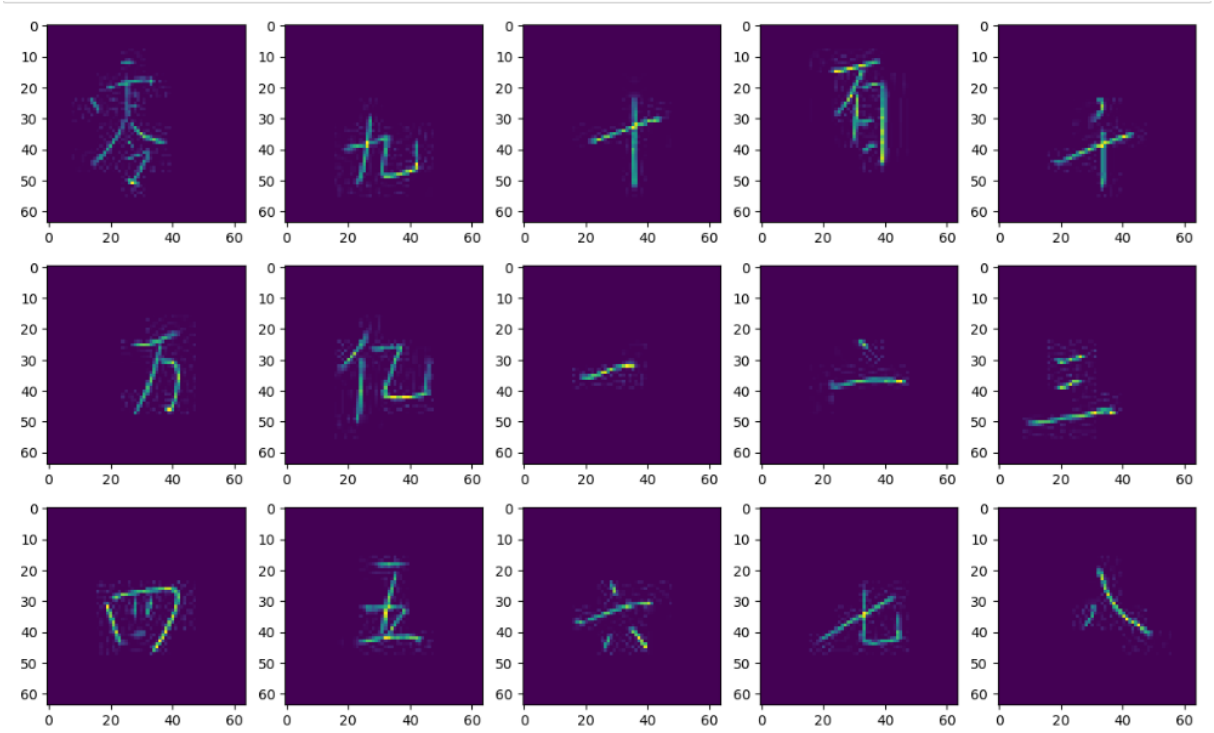


Figura 4. Muestra de 25 img 64x64

2.3. Preanalisisi para modelos

Como la base de datos es muy parecida a MNIST, en vez de hacer regresiones de cualquier tipo, comenzare haciendo modelos de Redes Neuronales, porque es lo que mejor se aplica a este tipo de problemas de “detección de imagenes”

3. Procesamiento

El procesamiento de datos consistira en lo siguiente:

1. Para cada fila del CSV identificar la imagen relacionada
2. Guardar la imagen en formato matricial de numpy de **64x64**
3. Guardar el valor de salida (la solución)

Apartir de aqui habra una divergencia en el procedimiento, habran **2 bases de datos**, una que tenga la salida en forma de **One Hot Encoder** y **otra que no lo tenga**.

4. Models bàsics

Primero creare todas las **redes neuronales con unas 50 neuronas** para **comparar** las **diferentes perdidas**, modelos etc.
Debido a que mientras mas neuronas, mas capas ocultas mas tarda, los parámetros decididos son:

-Una capa de neuronas de entrada tipo Flatten
-Una capa densa de 50 neuronas
-Una capa densa de salida

Los parámetros decididos para el entrenamiento inicial son:

-50 Épocas

Todos los modelos tendrán las siguientes caracteristicas, lo unico que cambiara es la función de coste/perdida:

```
Model: "sequential_54"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_54 (Flatten)        (None, 4096)              0

 dense_117 (Dense)           (None, 50)                204850

 dense_118 (Dense)           (None, 1)                 51

=================================================================
Total params: 204,901
Trainable params: 204,901
Non-trainable params: 0
```

Figura 5. Modelos RN sin One Hot Encoder

Los as diferentes **funciones de perdida** que aplicaremos son:

- **mean_squared_error**
- **MeanAbsolutePercentageError**
- **categorical_crossentropy**
- **MeanAbsoluteError**

4.1. Modelos Redes Neuronales con dataset Sin One hot Encode

En las Siguientes Imagenes observaremos las primeras y ultimas 3 epocas de cada modelo, para ver su "***LOSS***" y su "***ACCURACY***":

```
#print(Xn_train.shape)
historialMSE = mseModel.fit(Xn_train, yn_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 667395706322944.0000 - accuracy: 0.0684
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 662534944194560.0000 - accuracy: 0.0683
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 654346857480192.0000 - accuracy: 0.0683
Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 525088440975360.0000 - accuracy: 0.0880
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 523923229769728.0000 - accuracy: 0.0876
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 522788452433920.0000 - accuracy: 0.0872
```

Figura 6. Modelo con MSE

```
historialMAPE = mapeModel.fit(Xn_train, yn_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 7816463872.0000 - accuracy: 0.0679
Epoch 2/50
375/375 [==============================] - 1s 3ms/step - loss: 1192922752.0000 - accuracy: 0.0658
Epoch 3/50
375/375 [==============================] - 1s 3ms/step - loss: 104030144.0000 - accuracy: 0.0587
Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 340623648.0000 - accuracy: 0.0627
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 324989120.0000 - accuracy: 0.0594
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 340576064.0000 - accuracy: 0.0645
```

Figura 7. Modelo con MAPE

```
historialCCE = cceModel.fit(Xn_train, yn_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 2s 3ms/step - loss: 0.7968 - accuracy: 0.0563
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 0.7968 - accuracy: 0.0563
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 0.7968 - accuracy: 0.0563
Epoch 48/50
375/375 [==============================] - 2s 4ms/step - loss: 0.7968 - accuracy: 0.0563
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.7968 - accuracy: 0.0563
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.7968 - accuracy: 0.0563
```

Figura 8. Modelo con CCE

```
historialMAE = maeModel.fit(Xn_train, yn_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 6684087.5000 - accuracy: 0.0635
Epoch 2/50
375/375 [==============================] - 1s 3ms/step - loss: 6684061.5000 - accuracy: 0.0677
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 6684057.5000 - accuracy: 0.0742

Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 6684048.5000 - accuracy: 0.0641
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 6684054.0000 - accuracy: 0.0662
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 6684050.0000 - accuracy: 0.0653
```

Figura 9. Modelo con MAE

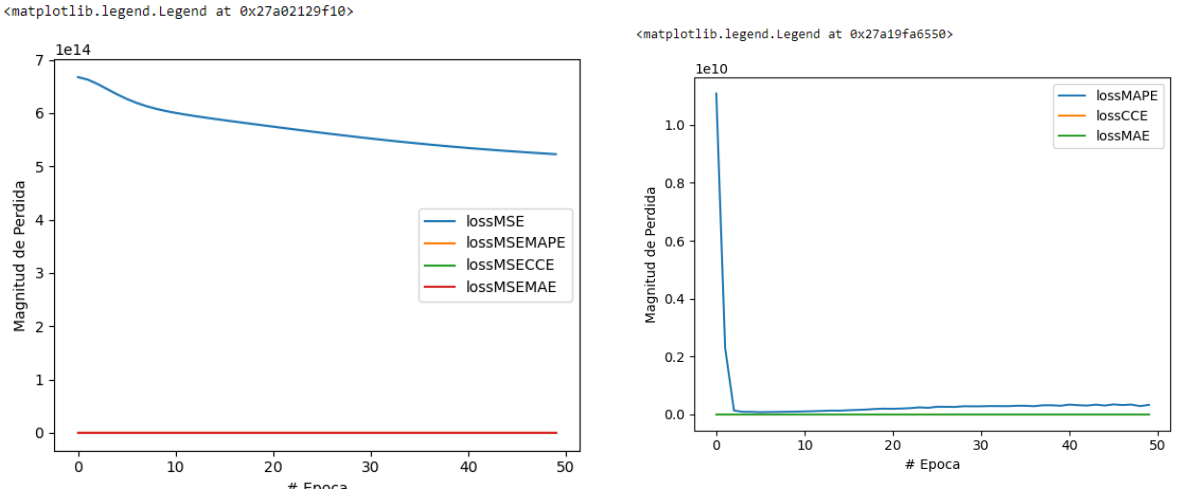


Figura 10. Loss/Epoca

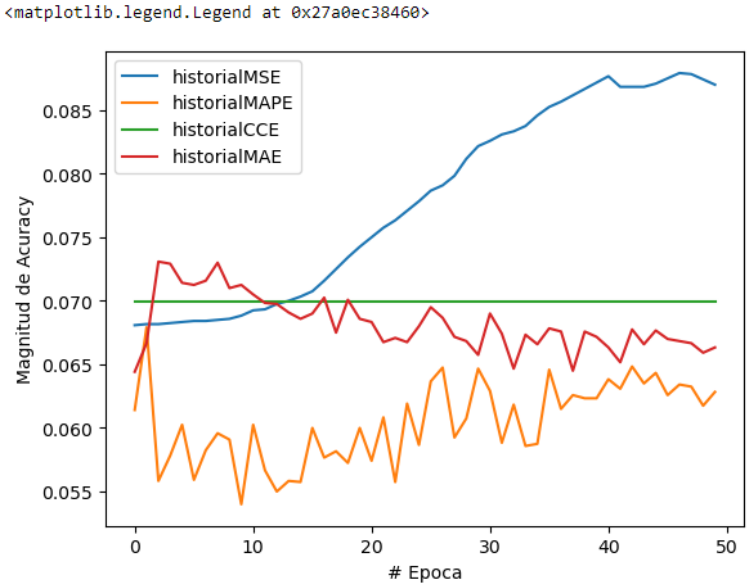


Figura 10. Accuracy/Epoca

```
Resultado en las pruebas MSE:
94/94 - 0s - loss: 532647214317568.0000 - accuracy: 0.0790 - 224ms/epoch - 2ms/step
Resultado en las pruebas MAPE:
94/94 - 0s - loss: 237110688.0000 - accuracy: 0.0600 - 322ms/epoch - 3ms/step
Resultado en las pruebas CCE:
94/94 - 0s - loss: 0.7869 - accuracy: 0.0637 - 356ms/epoch - 4ms/step
Resultado en las pruebas MAE:  0.06366666406393051
94/94 - 0s - loss: 6600810.5000 - accuracy: 0.0697 - 365ms/epoch - 4ms/step
```

Figura 11.Evaluacion de modelos

Como se puede observar en las ***figuras 6-10*** **la perdida en cada *Epoca* es inmensa**, en comparacion a los modelos que veremos mas adelante
Del mismo modo, el accuracy en todos estos modelos es decepcionante, el unico que se salva visualmente es el modelo que aplica ***MSE***, pero si se observa bien los ejes el valor maximo de accuraxy es **8%**, lo cual deja mucho que desear…
De la misma forma la evaluacion como es de esperar segun el acuracy, es muy decepcionante para los datos de testeo.

4.1. Modelos Redes Neuronales con dataset Con One hot Encode

En las Siguientes Imagenes observaremos las primeras y ultimas 3 epocas de cada modelo, para ver su "***LOSS***" y su "***ACCURACY***":

```
historialMSEOHE = mseModelOHE.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 0.1220 - accuracy: 0.0806
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 0.1202 - accuracy: 0.0973
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 0.1201 - accuracy: 0.0979
Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 0.1185 - accuracy: 0.1111
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1177 - accuracy: 0.1171
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1172 - accuracy: 0.1207
```

Figura 12. Modelo con MSE

```
historialMAPEOHE = mapeModelOHE.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 55518588.0000 - accuracy: 0.1671
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 50895524.0000 - accuracy: 0.2364
Epoch 3/50
375/375 [==============================] - 1s 3ms/step - loss: 49227652.0000 - accuracy: 0.2617
Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 37708376.0000 - accuracy: 0.4342
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 37125712.0000 - accuracy: 0.4430
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 37364840.0000 - accuracy: 0.4397
```

Figura 13. Modelo con MAPE

```
historialCCEOHE = cceModelOHE.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 3ms/step - loss: 4.8748 - accuracy: 0.1196
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 2.4861 - accuracy: 0.1668
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 2.3814 - accuracy: 0.2007

Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 1.2128 - accuracy: 0.5441
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 1.2198 - accuracy: 0.5450
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 1.2025 - accuracy: 0.5531
```

Figura 14. Modelo con CCE

```
historialMAEOHE = maeModelOHE.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
375/375 [==============================] - 2s 3ms/step - loss: 0.1194 - accuracy: 0.1043
Epoch 2/50
375/375 [==============================] - 1s 3ms/step - loss: 0.1139 - accuracy: 0.1455
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 0.1113 - accuracy: 0.1654

Epoch 48/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0847 - accuracy: 0.3649
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0844 - accuracy: 0.3672
Epoch 50/50
375/375 [==============================] - 1s 2ms/step - loss: 0.0855 - accuracy: 0.3585
```

Figura 15. Modelo con MAE

```
<matplotlib.legend.Legend at 0x27a0e4dce50>
```

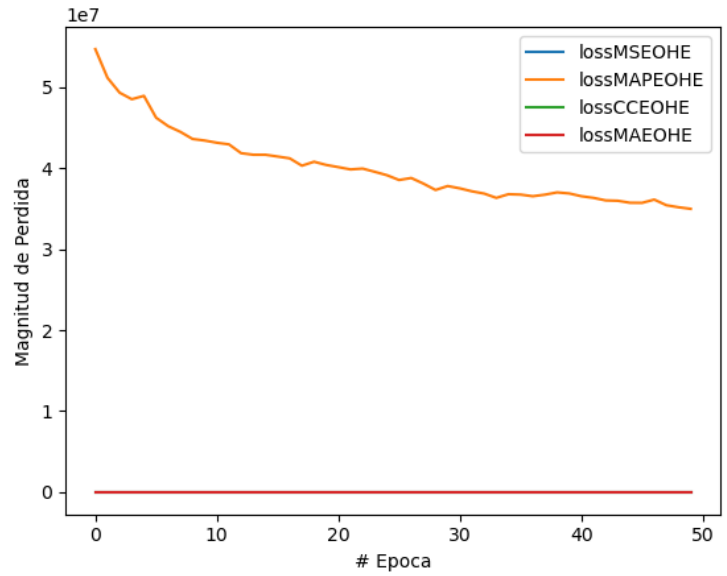


Figura 15. Grafica Loss/Epoca

```
<matplotlib.legend.Legend at 0x27a06ff4730>
```

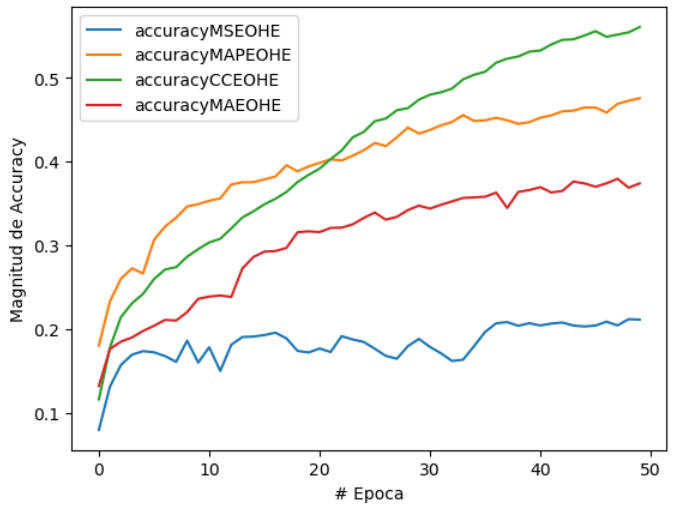


Figura 16. Grafica Accuracy/Epoca

```
Resultado en las pruebas MSE con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 0.1080 - accuracy: 0.1897
Resultado en las pruebas MAPE con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 38208112.0000 - accuracy: 0.4270
Resultado en las pruebas CCE con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 2.4464 - accuracy: 0.4367
Resultado en las pruebas MAE con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 0.0874 - accuracy: 0.3443
```

Figura 17.Evaluacion de modelos

Como se puede observar, el loss en todos los modelos loss<1 exceptuando en el modelo **MAPE** (***figuras 12-15)***
Por otro lado como se puede observar en la ***Figura 17*** La accuracy es mucho mejor en estos modelos que en los anteriores, todos superando el **8%** en pocas o en primeras Epocas.
Por otro lado Como se pude observar en la evaluacion, mejora muchisimo, aunque aun sigue siendo insuficiente, con la mayor accuracy siendo **43,67%** por parte del modelo con ***categorical_crossentropy***.

4.1. Comparación de Modelos

Como se puede observar en la ***Figura 18*****,** de todos los modelos entrenados, sin duda los modelos con *One Hot Encoder* son mejores con diferencia, y entre todos los modelos con esas caracteriscticas, el mejor es el modelo que usa **CCE** (***categorical_crossentropy***) aunque el modelo que usa **MAPE** parece que al principio sera mejor, llegado a un tiempo parece que tendera mas a una forma asintótica a diferencia del modelo **CCE**.

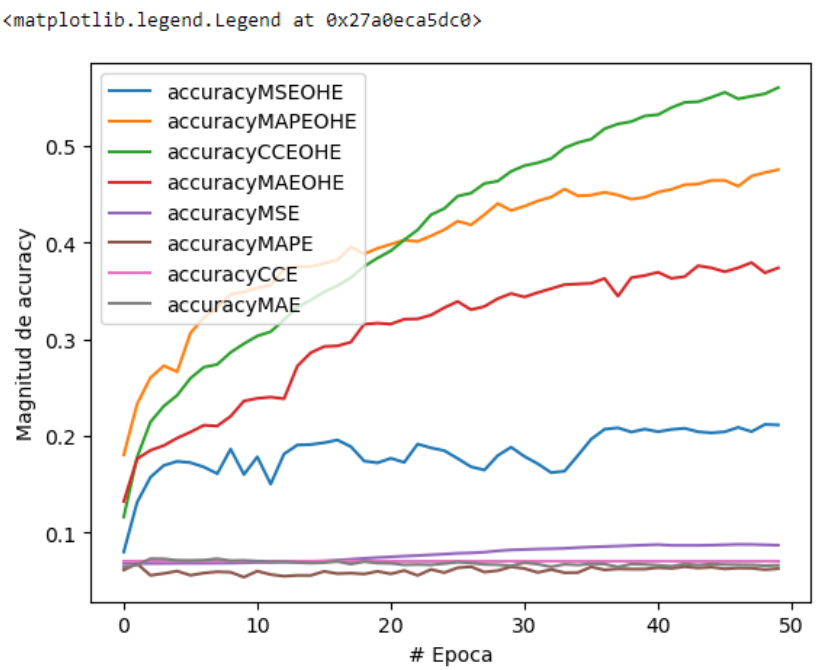


Figura 18

5. Mejoras en el Modelo

Como hemos observado, el mejor modelo es la red neuronal con una funcion de perdida segun **'categorical_crossentropy'**, por lo que nos encargaremos de intentar mejorar aun mas las accuracys con más neuronas por capa, mas capas o otras opciones que ofrece tensor flow

Compararemos modelos can las siguientes caracteristicas:

- **2 capas ocultas de 50 neuronas**, activacion relu y una capa de salida de 15 neuronas con activacion softmax
- **1 capa oculta de 128 neuronas**, activacion relu y una capa de salida de 15 neuronas con activacion softmax
- **2 capas ocultas de 128neuronas**, activacion relu y una capa de salida de 15 neuronas con activacion softmax
- **3 capas ocultas de 128 neuronas**, activacion relu y una capa de salida de 15 neuronas con activacion softmax

Obs: Para ver las caracteristicas especificas de las redes neuronales acceder al archivo chinese_mnist.ipynb o https://colab.research.google.com/drive/1IqZmHGJnHBsFdOwx4fves5li1aU0o4FY#scrollTo=g-FV1M-9kMon

Como en apartados anteriores mostraemos las 3 primeras y ultimas epocas de cada modelo, seguido de una imagen mostrando una grafica de la prediccion de ha hecho segun los costes, y por ultimo compararemos los modelos:

```
historialCCEOHE_1 = cceModelOHE_1.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
375/375 [==============================] - 1s 2ms/step - loss: 4.1012 - accuracy: 0.1614
Epoch 2/50
375/375 [==============================] - 1s 2ms/step - loss: 2.2386 - accuracy: 0.2778
Epoch 3/50
375/375 [==============================] - 1s 2ms/step - loss: 1.9884 - accuracy: 0.3357

Epoch 48/50
375/375 [==============================] - 1s 3ms/step - loss: 0.7026 - accuracy: 0.7371
Epoch 49/50
375/375 [==============================] - 1s 2ms/step - loss: 0.7218 - accuracy: 0.7365
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.7017 - accuracy: 0.7404
```

Figura 19. Modelo con CCE 2 capas y 50 neuronas

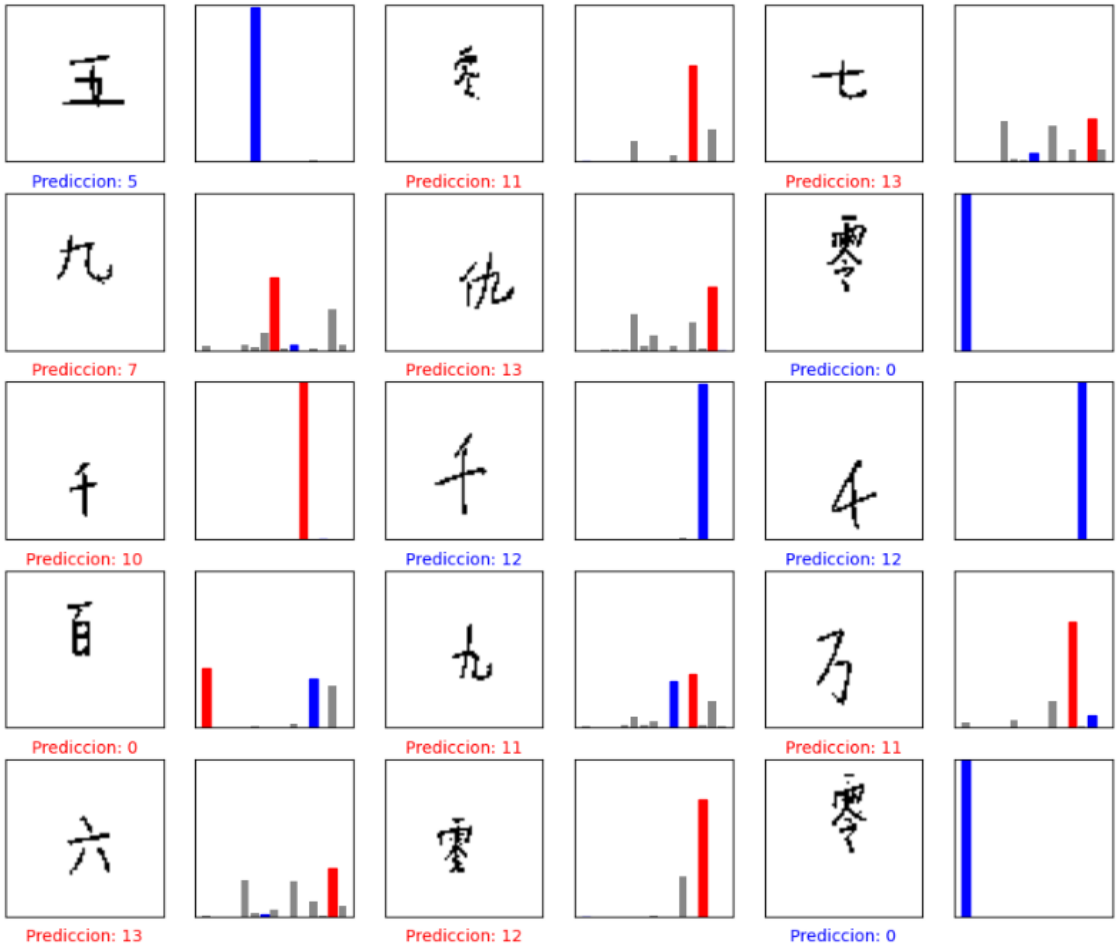


Figura 20. Predicciones modelo con CCE 2 capas y 50 neuronas

```
historialCCEOHE_2 = cceModelOHE_2.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
375/375 [==============================] - 2s 3ms/step - loss: 7.1150 - accuracy: 0.1713
Epoch 2/50
375/375 [==============================] - 1s 3ms/step - loss: 2.2013 - accuracy: 0.2864
Epoch 3/50
375/375 [==============================] - 1s 3ms/step - loss: 1.9657 - accuracy: 0.3545
...
Epoch 48/50
375/375 [==============================] - 1s 3ms/step - loss: 0.5549 - accuracy: 0.8170
Epoch 49/50
375/375 [==============================] - 1s 3ms/step - loss: 0.5648 - accuracy: 0.8189
Epoch 50/50
375/375 [==============================] - 1s 3ms/step - loss: 0.5516 - accuracy: 0.8243
```

Figura 21. Modelo con CCE 1 capa y 128 neuronas

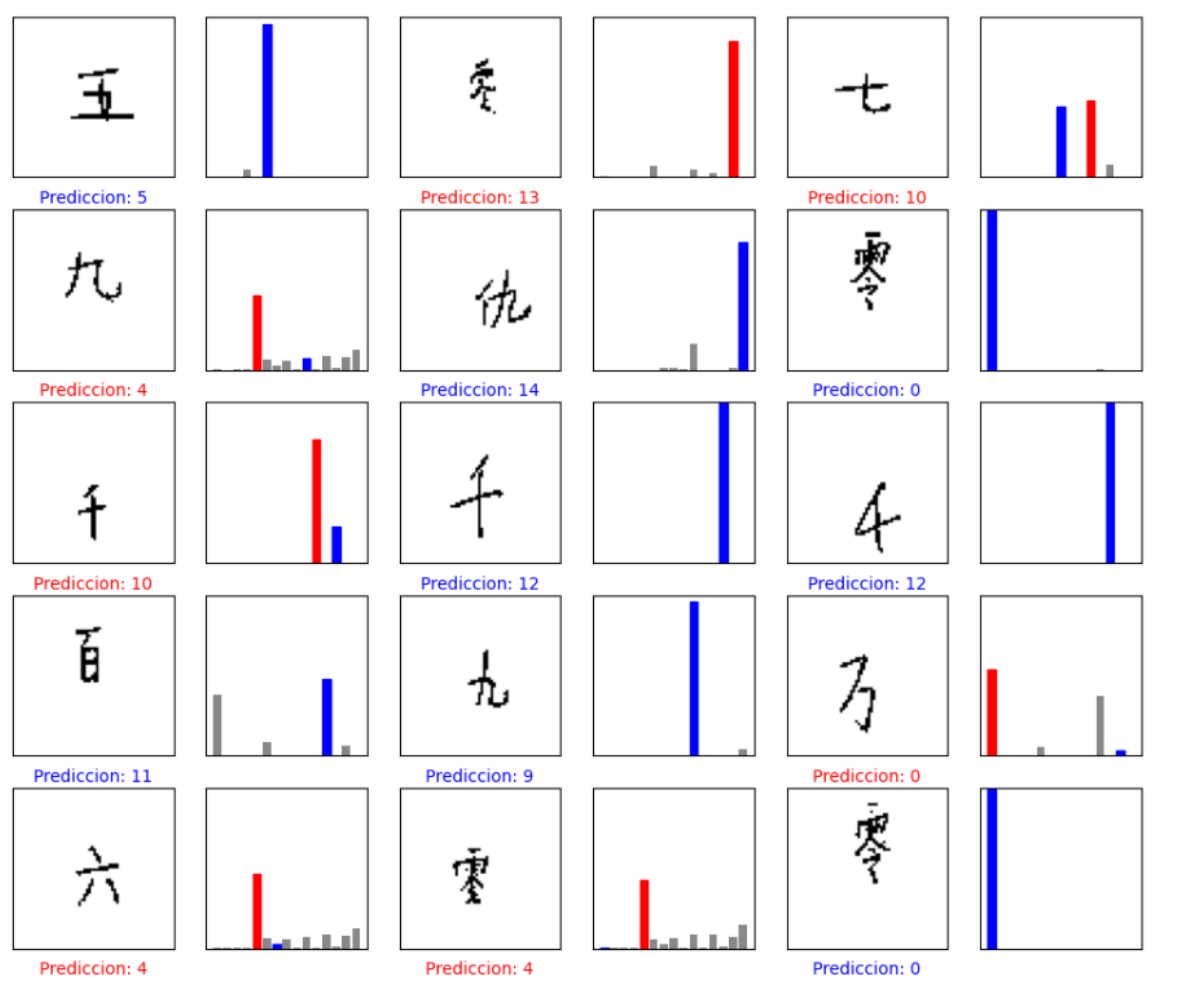


Figura 22. Predicciones modelo con CCE 1 capa y 128 neuronas

```
historialCCEOHE_3 = cceModelOHE_3.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
375/375 [==============================] - 2s 3ms/step - loss: 5.4999 - accuracy: 0.2449
Epoch 2/50
375/375 [==============================] - 1s 4ms/step - loss: 1.8124 - accuracy: 0.4212
Epoch 3/50
375/375 [==============================] - 1s 4ms/step - loss: 1.4972 - accuracy: 0.5086
Epoch 48/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1297 - accuracy: 0.9747
Epoch 49/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1090 - accuracy: 0.9764
Epoch 50/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1045 - accuracy: 0.9788
```

Figura 23. Modelo con CCE 2 capas y 128 neuronas

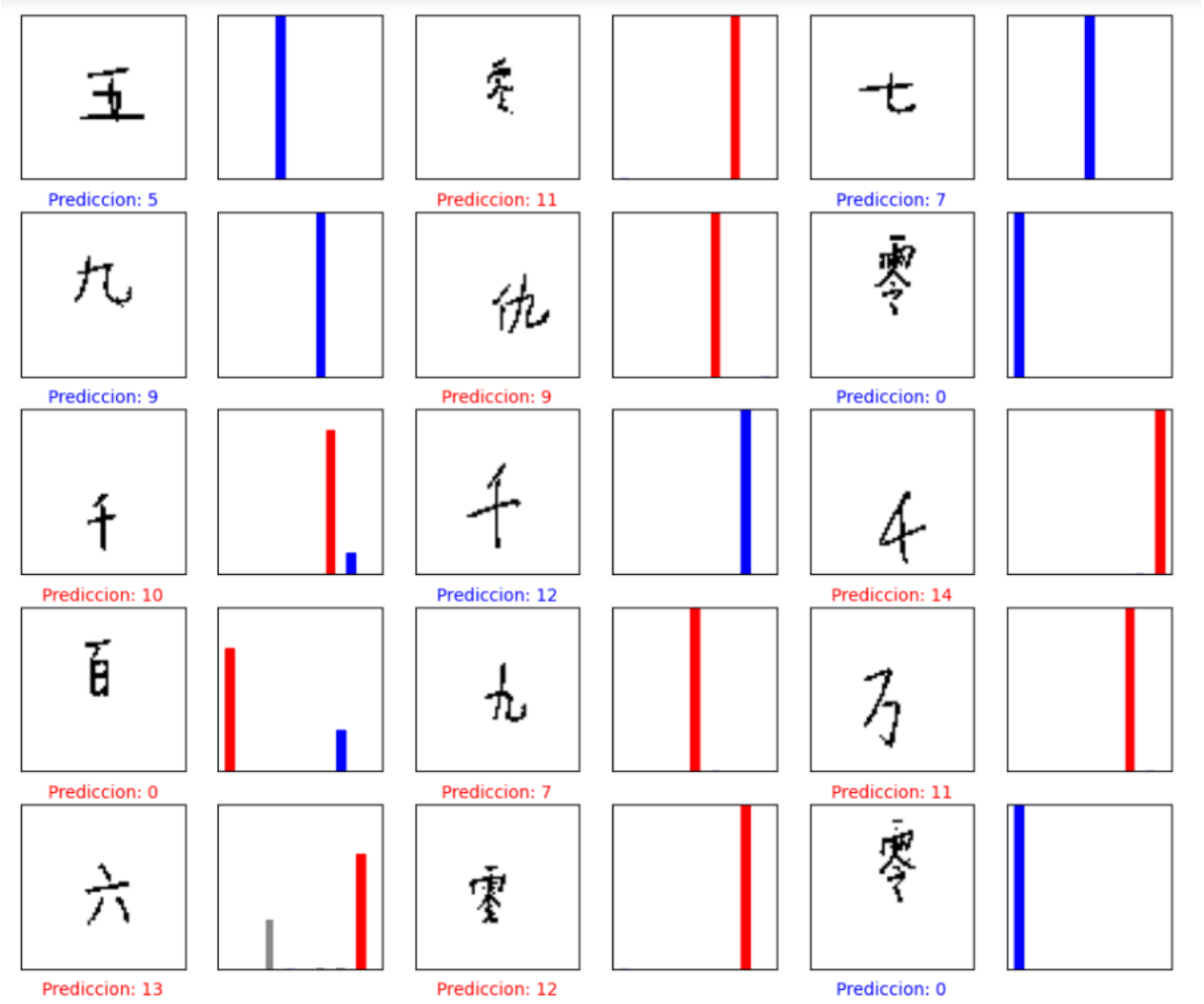


Figura 24. Predicciones modelo con CCE 2 capas y 128 neuronas

```
historialCCEOHE_4 = cceModelOHE_4.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
375/375 [==============================] - 2s 3ms/step - loss: 3.7299 - accuracy: 0.2759
Epoch 2/50
375/375 [==============================] - 1s 3ms/step - loss: 1.4691 - accuracy: 0.5194
Epoch 3/50
375/375 [==============================] - 1s 3ms/step - loss: 1.0035 - accuracy: 0.6607

Epoch 48/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1321 - accuracy: 0.9743
Epoch 49/50
375/375 [==============================] - 2s 4ms/step - loss: 0.1112 - accuracy: 0.9753
Epoch 50/50
375/375 [==============================] - 2s 4ms/step - loss: 0.0881 - accuracy: 0.9817
```

Figura 25. Modelo con CCE 3 capas y 128 neuronas

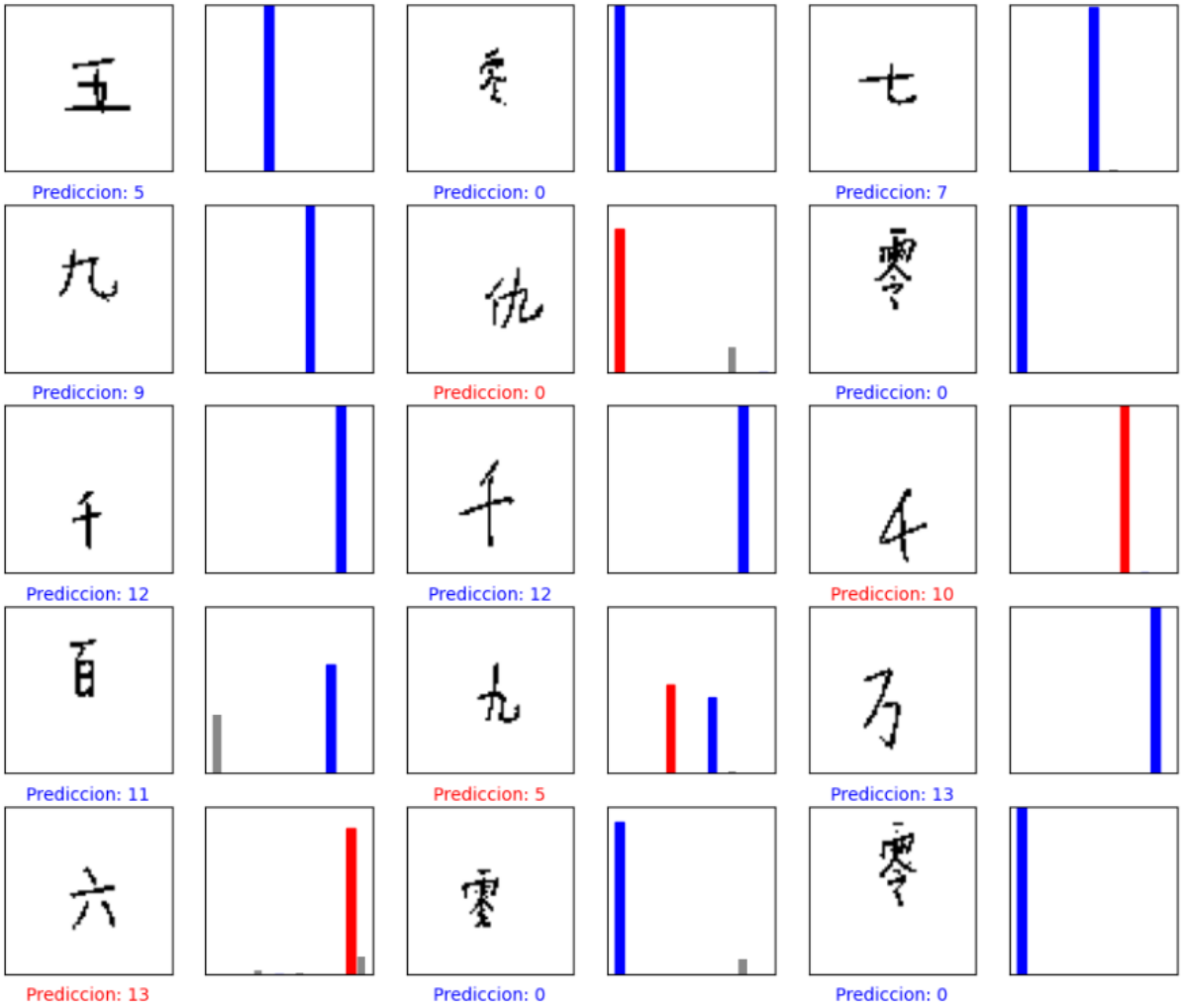


Figura 26. Predicciones modelo con CCE 3 capas y 128 neuronas

Como se pueden observar en las ***figuras 19-28*** La mejora es impresionante, en las figuras que sale la imagen de salida se puede observar en rojo en que casos ha fallado la prediccion, se ve que va mejorando poco a poco, aun asi, la imagen de la ***figura 24*** deberia reflejar aun mas aciertos que los otros casos ya que tiene mas neuronas y capas que los anteriores modelos, aun asi como solo escogemos 15 de todos los datos de test, cayo en la probabilidad que justo la mayoria de los test que se muestran los predice de forma incorrecta, aun asi viendo las ***figuras 27 y 28*** se puede observar claramente que **los mejores modelos son los modelos con 2 y 3 capas de 128 neuronas** respectivamente, siendo el de 3 capas el mejor

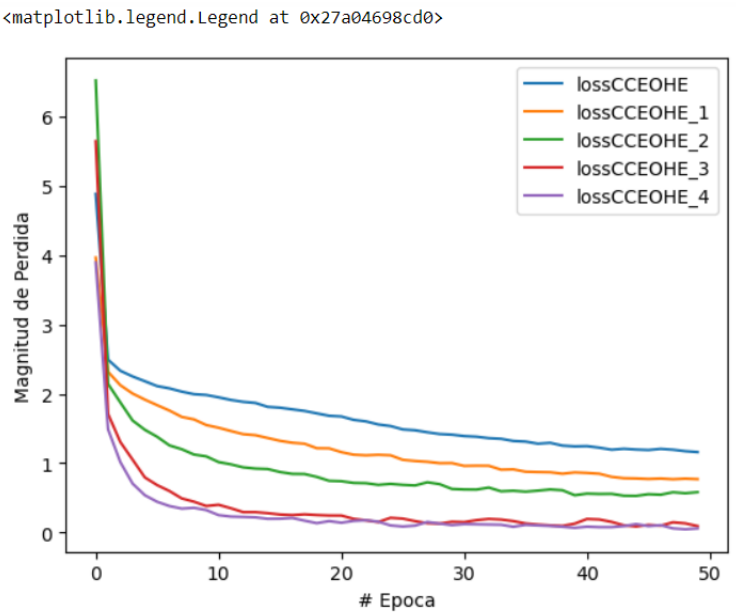


Figura 27. Grafica Loss/Epocas

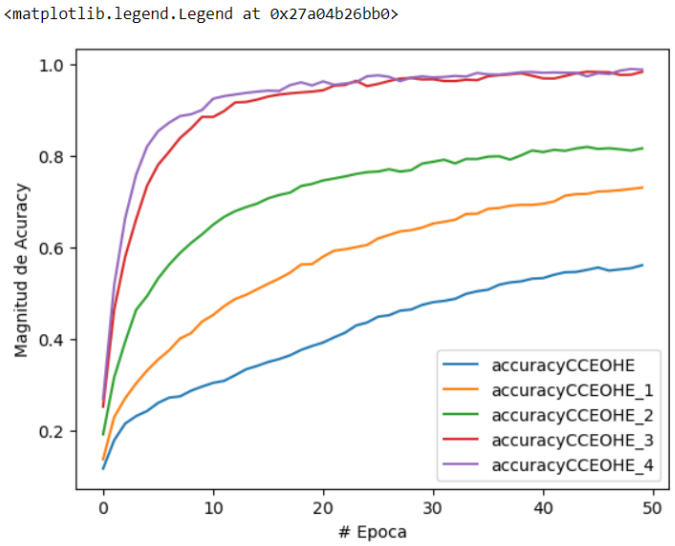


Figura 28. Grafica Accuracy/Epocas

```
Resultado en las pruebas CCE con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 2.4464 - accuracy: 0.4367
Resultado en las pruebas CCE_1 con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 2.4295 - accuracy: 0.5223
Resultado en las pruebas CCE_2 con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 4.0967 - accuracy: 0.5700
Resultado en las pruebas CCE_3 con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 4.6199 - accuracy: 0.6963
Resultado en las pruebas CCE_4 con OHE:
94/94 [==============================] - 0s 2ms/step - loss: 2.3689 - accuracy: 0.7593
```

Figura 29. Accuracy's modelos

Finalmente como se puede observar en la ***figura 28***, la mejor acuracy para los datos de test la tiene el ultimo modelo, con **3 capas de 128 neuronas**, y un acuracy del **75%**

6. Red Neuronal Convolucional

A Partir del mejor modelo mostrado, **haré un modelo aplicando una red neuronal convolucional CNN**, desde el principio de la practica queria llegar a hacer este tipo de red, ya que a medida que me iba informando de redes neuronales mas me daba cuenta de que es la que mejor se ajustaba al problema de CHINESE_MNIST.

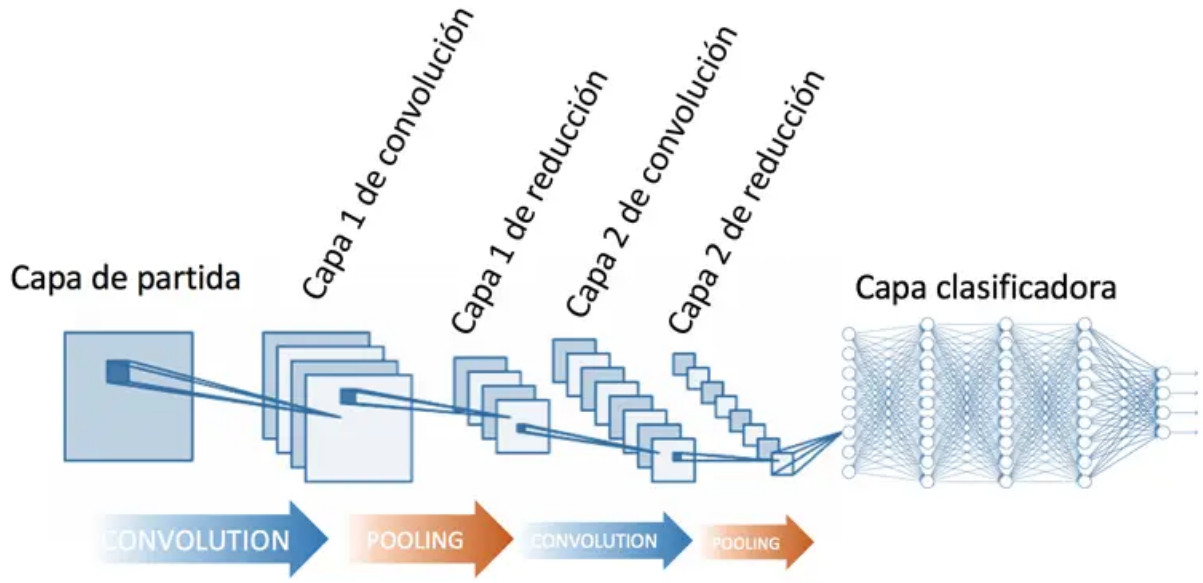


Figura 30. CNN

Una red neuronal Convolucional funciona como se puede visualizar en la Figura 30, no me centrare en explicar como funciona en la documentación asi que pasare a poner y explicar las figuras de las épocas, las graficas y tomare conclusiones

```
historialCNN = modelCNN.fit(X_train, y_train, epochs = 50)
```

```
Epoch 1/50
375/375 [==============================] - 33s 86ms/step - loss: 1.0598 - accuracy: 0.7280
Epoch 2/50
375/375 [==============================] - 34s 90ms/step - loss: 0.1958 - accuracy: 0.9361
Epoch 3/50
375/375 [==============================] - 34s 91ms/step - loss: 0.1078 - accuracy: 0.9652
Epoch 48/50
375/375 [==============================] - 37s 99ms/step - loss: 2.0886e-06 - accuracy: 1.0000
Epoch 49/50
375/375 [==============================] - 38s 102ms/step - loss: 1.6812e-06 - accuracy: 1.0000
Epoch 50/50
375/375 [==============================] - 38s 102ms/step - loss: 1.3502e-06 - accuracy: 1.0000
```

Figura 31. Modelo CNN

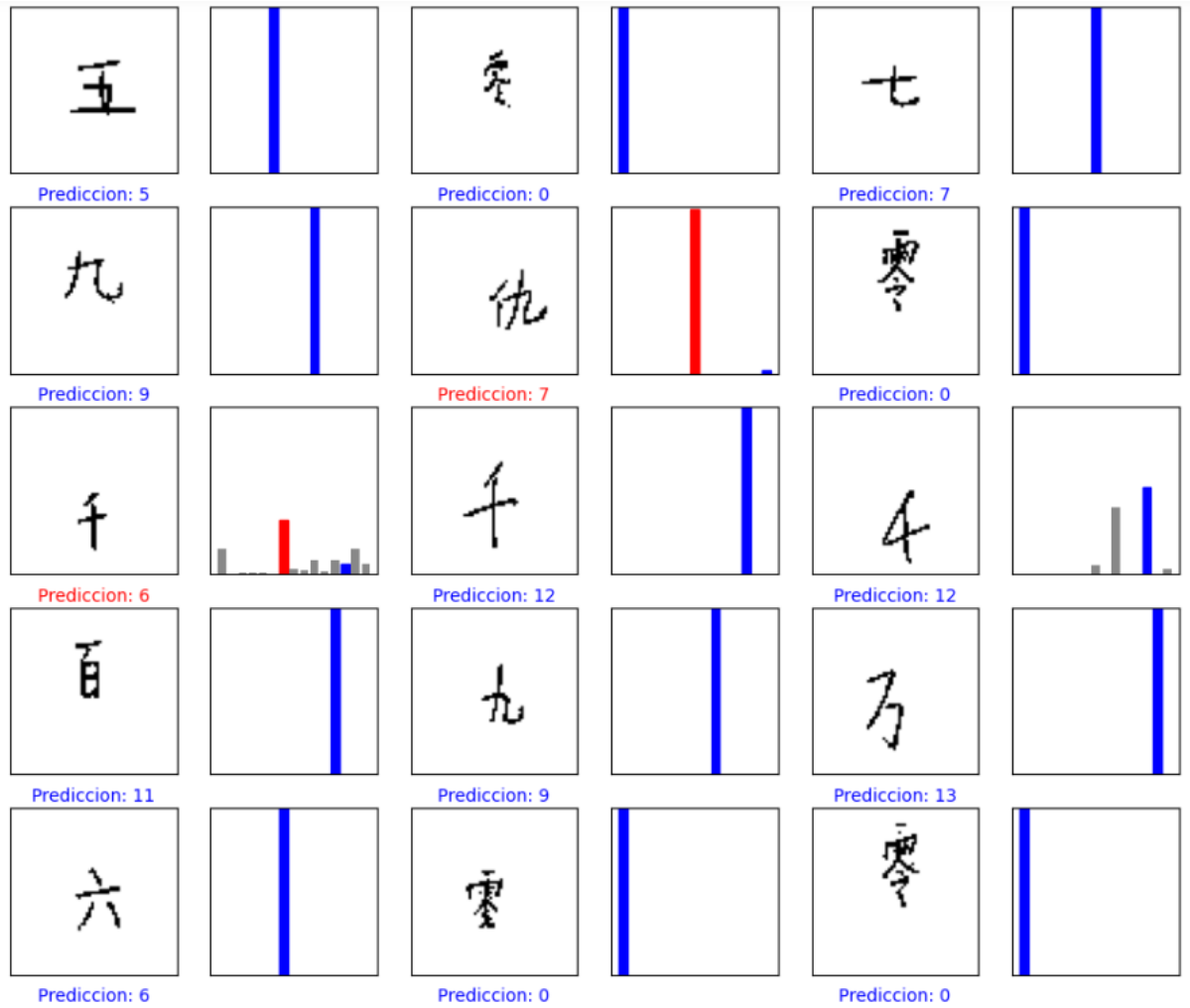
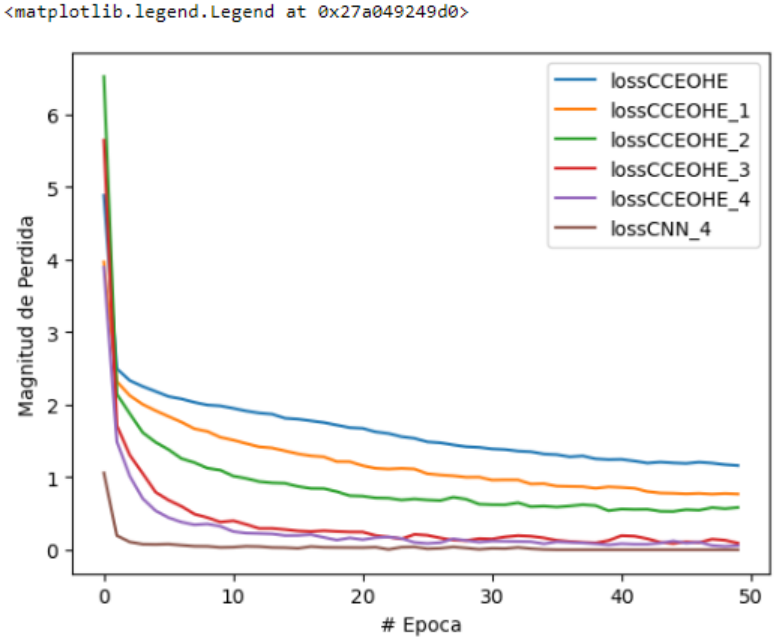


Figura 32. Predicciones modelo CNN

```
Resultado en las pruebas CNN con OHE:
94/94 [==============================] - 1s 12ms/step - loss: 0.2712 - accuracy: 0.9607
```

Figura 33. Evaluación modelo CNN

Figura 34. Grafica Loss/Epoca



Como se puede observar, en la ***figura 31*** **el modelo comienza con un 72%** de acuracy y en la 2a it ya **supera el 93%**, no se ve (en el notebook si) pero en la it 35 ya **alcanza accuracy's del 100%** por lo que perfectamente se podria haber detenido en ese momento el entrenamiento del modelo, aunque en el entrenamiento se haya conseguido un 100% de accuracy eso seguramente causa overfiting, ya que aunque tenga un 100% cuando se evalua con el set de test, como se puede observar en la ***figura***

33, el accuracy disminuye al **96,07%** seguramente si añadimos mas capas de convolucion y mas capas normales se podria mejorar mas el modelo, aun asi, el modelo ha tardado casi **30'** en ser entrenado, y **ha dado un muy buen resultado comparado con los otros modelos**, como se pude observar en las ***figuras 34 y 35.***

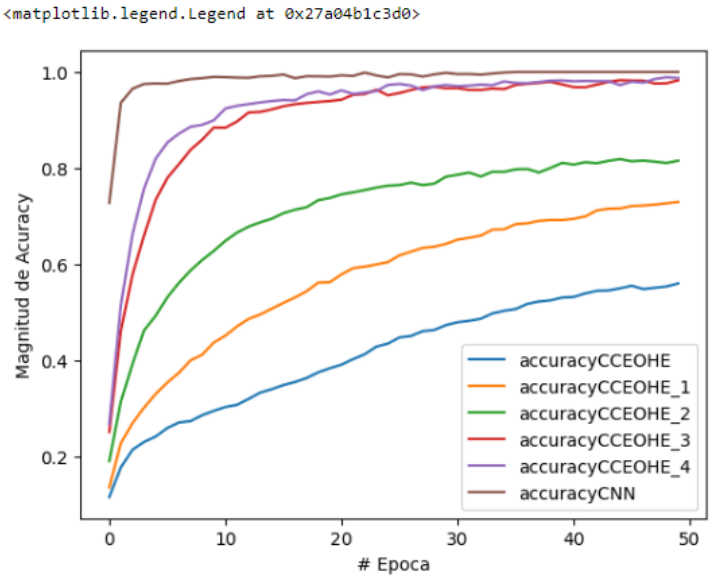


Figura 35. Grafica Accuracy/Epoca

5. Anàlisis complet del cas Kaggle

Al largo de este documento se puede observar todo el trabajo dedicado a este trabajo, el problema era muy simple y parecido al **"problema padre"** de las redes neuronales el **MNIST**, teniendo esto en cuenta como he comentado con anterioridad, he decidido **centrarme en mostrar diferentes tipos de modelos de redes neuronales**, con muchos parámetros diferentes, de perdida, con mas o menos capas, para experimentar y aprender, en vez de ponerme a hacer regresiones lo cual por otro lado ya he hecho mucho en la **PRACTICA 1** y en las **PRACTICA 2**.
Finalmente el problema ser reducida a un problema de **clasificación de imagenes** asi que simplemente me dedique a guardar las imágenes y aprender a programar una red neuronal que que se adapte a mi problema.
Finalmente el caso que **mejores resultados** dio, como era de esperar, era el caso de la **Red Neuronal Convolucional,** como comenté con anterioridad, tiene solo 2 capas convolucionales, y 3 capas normales como se puede observar en el siguiente fragmento de codigo:

```
#Crear el modelo, este caso tendra 3 capas de 128 neuronas

modelCNN = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), input_shape=[64, 64, 1], activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(2,2),#2,2 es el tamaño de la matriz

    tf.keras.layers.Conv2D(64, (3,3), input_shape=[64, 64, 1], activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(2,2),#2,2 es el tamaño de la matriz

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation=tf.nn.relu), #1a capa oculta activacion relu
```

```
    tf.keras.layers.Dense(128, activation=tf.nn.relu), #2a capa oculta activacion relu
    tf.keras.layers.Dense(128, activation=tf.nn.relu), #2a capa oculta activacion relu
    tf.keras.layers.Dense(15, activation=tf.nn.softmax), #capa de salida 15 salidas posibles
])
modelCNN.compile(
    optimizer = 'adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
modelCNN.summary()
```

Por ultimo decir que aparte de que suponia que seria el mejor modelo, también suponía que **el costo computacional y temporal aumentaría**, como ha pasado, paso de tardar como 10 min o menos a tardar casi 30, aunque seguramente **si en vez de tener 3 capas normales le hago tener solo 1 pasaria a durar 20 min**, de hecho lo intente y calcule que tardaría 22 min, pero no lo deje acabar, porque **preferí ponerle 3 capas y que tarde 30 min**, sabiendo que tendría mejor resultado de forma segura, quizá en una versión posterior hago mas pruebas para demostrar lo que comento.
Asi que como conclusión final repetir que CNN que es el mejor modelo

6. Observaciones

Comentar que para hacer este proyecto me he informado mucho, aparte de que sin saberlo al 100% llevo desde 1o de bach viendo videos en YouTube sobre Inteligencia Artificial, como comentario gracioso o anecdota, comentar que he usado la nueva red neuronal CHATGPT para resolver algunas dudas como por ejemplo para encontrar donde me habia equivocado al imprimir las imagenes de prediccion, ya que debido a un problema con los datos no se imprimian las imagenes, lo gracioso recae en el hecho de usar una red neuronal inmensa para hacer una nueva red neuronal, por otro lado tenia pensado aplicar lo del video que comento en el anexo llamado **“Crea una red neuronal que reconozca tu escritura”,** para hacer una pagina web que haga llamadas a las redes neuronales y probar en directo el funcionamiento, si me da tiempo antes de la presentación quiza lo hago, sino quiza lo hago como proyecto a futuro ya que me parece algo muy interesante.

7. Annexos

Documentacion propia:

- Kaggle Chinese MNIST: https://www.kaggle.com/datasets/gpreda/chinese-mnist
- Todo el notebook se ha guardado en el siguiente notebook en github:
- https://github.com/adriend1102/Practica3APC-Chinese-MNIST
- Documento google Colab: https://colab.research.google.com/drive/1_YhlY42X-tOzCUkfJwgotaR2v8pmBpIE#scrollTo=Ug1mfRaXKc9o

Consultas:

- Consultas para ver diferentes soluciones a problemas parecidos:
  - Crea una red neuronal que reconozca tu escritura
  - Tu primera red neuronal en Python y Tensorflow
  - ¿Cómo optimizamos en 300x los procesos de IA?
  - Funciones de activación a detalle (Redes neuronales)
  - Redes Neuronales Convolucionales - Clasificación avanzada de imágenes con …
  - TUTORIAL DE PYTHON: RECONOCIMIENTO DE PATENTES
  - Crea tu propia red neuronal que puede leer
  - Cómo funcionan las redes neuronales - Inteligencia Artificial
  - Tu primera red neuronal - Inteligencia Artificial
  - Tu primer clasificador de imágenes con Python y Tensorflow
  - Cómo funcionan las redes neuronales - Inteligencia Artificial
  - Haz un deepfake en 7 minutos - Inteligencia Artificial
  - ¿Pocos datos de entrenamiento? Prueba esta técnica
  - Aprende a PROGRAMAR una RED NEURONAL - Tensorflow, Keras, Sklearn
  - Descargar y procesar MNIST, sólo Python ! –Fundamentos de Deep Learning c…
- Solucion que sirvio de soporte para este caso (del mismo caso): https://www.kaggle.com/code/rudolfdebelak/chinese-mnist-fnn-cnn-and-lenet-5

- Consulta sobre Red Neuronal Prealimentada: https://es.wikipedia.org/wiki/Red_neuronal_prealimentada
- Consulta documentacion TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/
- Consultas resolucioon problema visualizacion de predicciones por incompatibilidad: https://chat.openai.com/chat https://python-para-impacientes.blogspot.com/2019/11/convertir-copiar-ordenar-unir-y-dividir.html https://numpy.org/doc/stable/reference/generated/numpy.take.html https://noemioocc.github.io/posts/Crear-una-imagen-BGR-openCV-Python/ https://numpy.org/doc/stable/user/basics.indexing.html https://stackoverflow.com/questions/47902295/what-is-the-use-of-verbose-in-keras-while-validating-the-model
- Consultas para la documentación:

https://www.youtube.com/watch?v=kV8se-C3tCw

- Fuente figura: https://www.diegocalvo.es/wp-content/uploads/2017/07/red-neuronal-convolucional-arquitectura.png