Deep Learning: Convolutional Neural Networks № 5

Daniela Pinto Veizaga, dpintove@itam.mx
Diego Villa Lizárraga, dvillali@itam.mx

26/02/2020

Introducción

Para responder la pregunta *1*, empleamos una base de datos de imágenes con dos clases: perros y gatos. El objetivo es diseñar redes neuronales de convolución para clasificar perros de gatos. Este problema es mucho más complejo que los anteriores, ya que: 1) las características visuales que comparten perros y gatos pueden ser altas; y 2) las imágenes pueden contener background común; estas dos características hace difícil separar las imágenes.

Para responder el resto de las preguntas, importamos imágenes de dígitos resguardados en la librería *MNIST* para resolver un problema de clasificación de números del 0 al 9

Pregunta 1

Usando la siguiente red, pude conseguir una pérdida de 0.2485 y exactitud de 0.5707. Diseña una red que te permita mejorar este desempeño. ¿Puedes lograr exactitud >= 0.8? ¿Cuál es la pérdida asociada? ¿Cuántos parámetros tiene tu modelo? ¿Qué otros hiper-parámetros definiste?

Con el fin de lograr una exactitud >= a 0.8, implementamos la siguiente arquitectura de red:

Listing 1: Arquitectura de la Red Neuronal Implementada

```
1
2  CNN = Sequential([
3      Conv2D(32, (3,3), activation='relu', input_shape=(IMG_HEIGHT, ↩
           IMG_WIDTH, 3)),
4      MaxPooling2D(pool_size=(2,2)),
5      Conv2D(64, (3,3), activation='relu'),
6      MaxPooling2D(pool_size=(2,2)),
7      Conv2D(128, (3,3), activation='relu'),
8      MaxPooling2D(pool_size=(2,2)),
9      Flatten(),
10     Dense(128, activation='relu'),
11     Dropout(0.9),
12     Dense(1, activation='sigmoid')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_131 (Conv2D)          (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_128 (MaxPoolin (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_132 (Conv2D)          (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_129 (MaxPoolin (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_133 (Conv2D)          (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_130 (MaxPoolin (None, 17, 17, 128)       0
_________________________________________________________________
flatten_43 (Flatten)         (None, 36992)             0
_________________________________________________________________
dense_85 (Dense)             (None, 128)               4735104
_________________________________________________________________
dropout_117 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_86 (Dense)             (None, 1)                 129
=================================================================
Total params: 4,828,481
Trainable params: 4,828,481
Non-trainable params: 0
```

Figura 1. Modelo 1, Precisión y Pérdida con 40 épocas

Como se puede observar, nuestra red consta de $4,828,481$ parámetros; los hiperparámetros empleados fueron un *batch size*= 40, 40 épocas y un dropout, especificado después de la capa densa, igual a 0.9.

Training Loss	*Training Accuracy*	*Validation Loss*	*Validation Accuracy*
0.0170	0.9740	0.1764	0.7967

Table 1: Resultados en la Época 40.

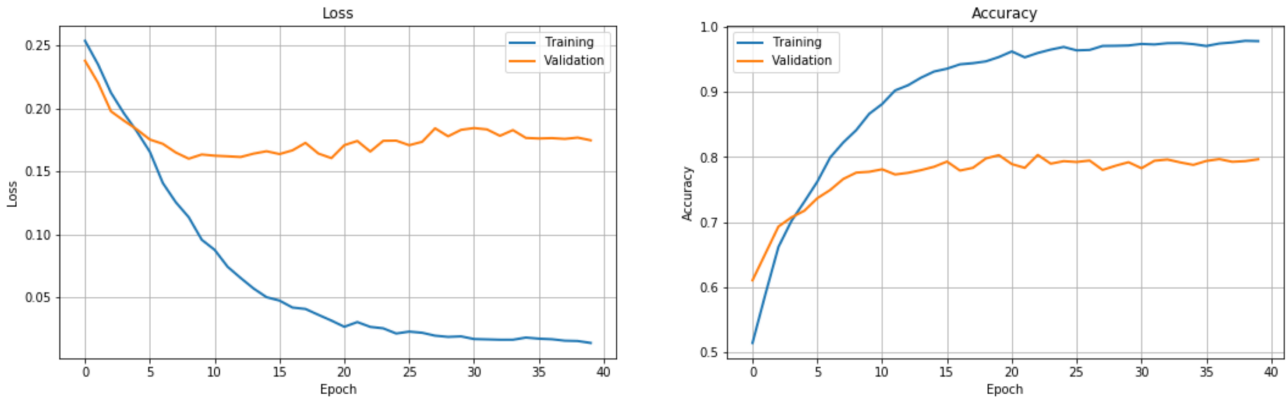


Figura 1. Modelo 1, Precisión y Pérdida con 40 épocas

Pregunta 2

Revisa esta y las tres celdas siguientes, y explica por qué crees que en esta celda, al importar las imágenes, las asignamos a una variable llamada ***y*** *(en vez de a x como lo hicimos en alguna tarea anterior).*

Listing 2: Generando "y".

```
from keras.datasets import mnist
(y_train, _), (y_test, _) = mnist.load_data()
y_train = y_train / 255.
y_test  = y_test  / 255.
y_train = np.reshape(y_train, (y_train.shape[0], y_train.shape[1], y_train.shape[2], 1))
y_test  = np.reshape(y_test, (y_test.shape[0], y_test.shape[1], y_test.shape[2], 1))

print("Shapes")
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)
```

El presente problema de clasificación de imágenes tiene una naturaleza distinta al problema de clasificación de perros y gatos: ahora, a fin de entrenar un red neuronal para clasificar los números de 0 al 9, importamos las imágenes originales con mnist.load.data(), tal y como se encuentran, a *"y train"* y *"y test"*, prescindiendo de *"x train"* y *"x test"*. Lo anterior, porque en este problema, como lo sugiere el código de abajo, haremos las predicciones de y con los mismos datos de entrada de y, pero perturbados por la función **"a function"**; esto nos permitirá emplear la técnica de clasificación **Upsampling+Convolutional.**

Listing 3: Generando "x".

```

x_train = np.array([a_function(IMG) for IMG in y_train])
x_test  = np.array([a_function(IMG) for IMG in y_test ])

print("Shapes")
print("x_train:", x_train.shape)
print("x_test:", x_test.shape)
```

Pregunta 3

Aquí defino una función. ¿Puedes explica qué es lo que hace?

Listing 4: Function "a function".

```
1
2  # define function
3  from skimage.transform import resize
4  def a_function(IMG):
5    img = resize(IMG.copy(), (7, 7))
6    for row in range(7):
7      for col in range(7):
8        thresh = np.random.rand()
9        if thresh > 0.9:
10         img[row, col] = np.abs(img[row, col] - 1)
11   return img
```

La función desarrollada en el código de arriba realiza lo siguiente:

a. Recibe como argumento una imagen;

b. Cambia las dimensiones de la imagen que recibe;

c. Perturba aleatoriamente la composición de los pixeles de la imagen.

d. Devuelve la imagen perturbada.

Pregunta 4

¿Qué problema está resolviendo esta red?

En el presente ejercicio se está resolviendo un problema de clasificación de números del 0 al 9: la red recibe imágenes de números como entrada y predice qué número son.

El andamiaje detrás de la red implementada es el siguiente: los datos de entrada de x, generados a partir de y con perturbaciones aleatorias y dimensiones menores (7x7), son empleados para predecir y (con dimensión 28x28). Para resolver el problema de dimensionalidad, se emplea la técnica de upsampling+ Conv2D.

Pregunta 5

¿Puedes mejorar el modelo para aumentar la exactitud a más de 0.95? Reporta el modelo resultante.

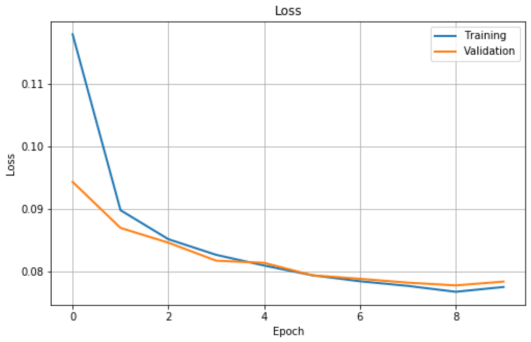
Muchas fueron las implementaciones y propuestas de arquitecturas realizadas para mejorar la exactitud de la predicción de la red; sin embargo, ninguna arquitectura implementada superó el umbral de 95 precisión.

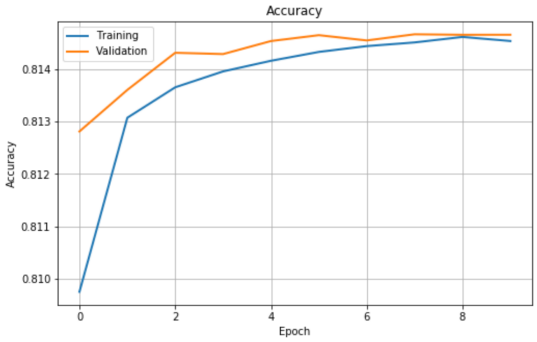
A continuación, se presentan algunas propuestas de arquitecturas y sus respectivos desempeños.

Modelo 1

```
Model: "sequential_38"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_240 (Conv2D)          (None, 7, 7, 64)          640
_________________________________________________________________
conv2d_241 (Conv2D)          (None, 7, 7, 64)          36928
_________________________________________________________________
up_sampling2d_62 (UpSampling (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_242 (Conv2D)          (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_243 (Conv2D)          (None, 14, 14, 32)        36896
_________________________________________________________________
up_sampling2d_63 (UpSampling (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_244 (Conv2D)          (None, 28, 28, 128)       36992
_________________________________________________________________
conv2d_245 (Conv2D)          (None, 28, 28, 32)        36896
_________________________________________________________________
conv2d_246 (Conv2D)          (None, 28, 28, 32)        9248
_________________________________________________________________
conv2d_247 (Conv2D)          (None, 28, 28, 32)        9248
_________________________________________________________________
conv2d_248 (Conv2D)          (None, 28, 28, 1)         289
=================================================================
Total params: 240,993
Trainable params: 240,993
Non-trainable params: 0
```

```
Epocas: 10
Batch size: 64
Optimizador:  Adam
Loss function: Binary Crossentropy
Training Loss: 0.077575157503287
Validation Loss: 0.07842957244316737
Accuracy 0.8145438398431849
Validation Accuracy 0.8146636921564738
```

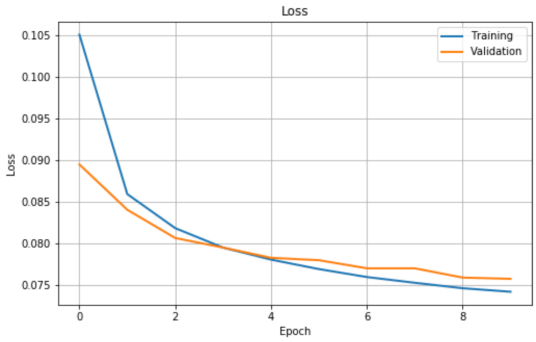


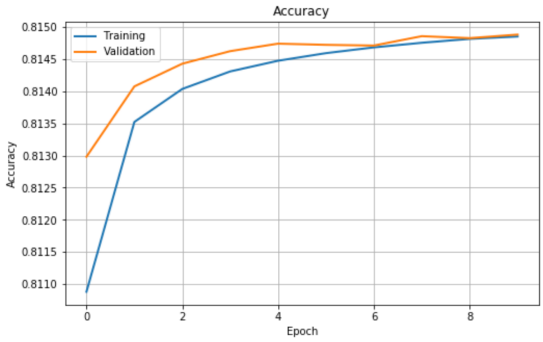


Modelo 2

```
Model: "sequential_39"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_249 (Conv2D)          (None, 7, 7, 64)          640
_________________________________________________________________
conv2d_250 (Conv2D)          (None, 7, 7, 64)          36928
_________________________________________________________________
up_sampling2d_64 (UpSampling (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_251 (Conv2D)          (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_252 (Conv2D)          (None, 14, 14, 128)       147584
_________________________________________________________________
up_sampling2d_65 (UpSampling (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_253 (Conv2D)          (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_254 (Conv2D)          (None, 28, 28, 64)        73792
_________________________________________________________________
conv2d_255 (Conv2D)          (None, 28, 28, 1)         577
=================================================================
Total params: 480,961
Trainable params: 480,961
Non-trainable params: 0
```

```
Epocas: 10
Batch size: 30
Optimizador:  Adam
Loss function: Binary Crossentropy
Training Loss: 0.07420010910265976
Validation Loss: 0.07574893835932016
Accuracy 0.8148527016904619
Validation Accuracy 0.8148822250962258
```





Modelo 3

```
Model: "sequential_40"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_256 (Conv2D)          (None, 7, 7, 64)          640
_________________________________________________________________
conv2d_257 (Conv2D)          (None, 7, 7, 64)          36928
_________________________________________________________________
up_sampling2d_66 (UpSampling (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_258 (Conv2D)          (None, 14, 14, 128)       73856
_________________________________________________________________
up_sampling2d_67 (UpSampling (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_259 (Conv2D)          (None, 28, 28, 64)        73792
_________________________________________________________________
conv2d_260 (Conv2D)          (None, 28, 28, 1)         577
=================================================================
Total params: 185,793
Trainable params: 185,793
Non-trainable params: 0
```

```
Epocas: 10
Batch size: 15
Optimizador:  Adam
Loss function: Binary Crossentropy
Training Loss: 0.07554045096867615
Validation Loss: 0.07693064297549426
Accuracy 0.8147076013187567
Validation Accuracy 0.8147189611196518
```