Universidad Zaragoza
1542

ANEXO

Título del trabajo : Aplicación de algoritmos “Machine Learning” a procesos de localización de objetos.

Autor/es

José Paulo Gonzales Parvina

Director/es

Ana María López Torres

Escuela Universitaria Politécnica de Teruel

2020

Indice Anexo

1.Funciones De Coste

A continuación se describirán las distintas funciones de coste.

1.1 Raíz Cuadrada Media – RMSE

La raíz cuadrada media, es una medida precisión calculada como la raíz cuadrada media de los residuos. Se entiende como residuos la diferencia entre el valor previsto (correcto) y el valor real obtenido.

Características del RMSE:

- Penaliza los valores que son muy grandes.
- No es fácilmente interpretable.
- Funciona muy bien para optimizar regresiones en general.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

1.2 Error Absoluto Medio – MAE

El error absoluto medio, es una medida de precisión y se calcula como la salida media de los valores absolutos de los errores.

Características del MAE:

- Mas difícil diferenciación y convergencia.
- Penaliza menos los valores grandes.
- Es mas fácil de interpretar.

$$MAE = \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{n}$$

1.3 Error Absoluto Medio Escalado -MASE

El error absoluto medio escalado, es una medida de precisión similar al MAE pero escalado.

Características del MASE:

- Mas difícil diferenciación y convergencia.
- Escala univariante.
- Simétrica
- Es mas fácil de interpretar.

$$MASE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{\frac{n}{n-1}\sum_{i=2}^{n}|\hat{y}_n - y_{n-1}|}$$

1.4 Entropía Cruzada Binaria – Binary Cross – Entropy

La entropía cruzada binaria, es una medida de precisión para variables binarias.

Características de Entropía cruzada binaria:

- Mas difícil diferenciación y convergencia.
- Escala univariante.
- Simétrica.
- Es mas fácil de interpretar.

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(p_i + (1-y)\log(1-pi))$$

2. Optimzadores

Se describirán los optimizadores mas comunes.

2.1 Descenso Por Gradiente

El descenso por gradiente es un algoritmo de optimización iterativo de primer orden para encontrar un mínimo local de una funcion diferenciable. Esta técnica se basa en la asunción de que si la funcion a ser minimizada *f,* esta definida y es diferenciable en el entorno de x. Dicha funcion va a decrecer mas rápidamente si

tomamos la dirección del gradiente negativo de x desde el punto x, -$\nabla f(x)$. (ver Ilustración 17)

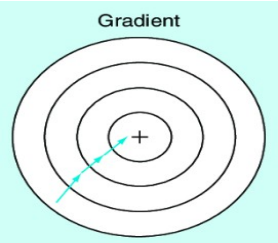


Ilustración 1: Descenso por gradiente

En una red neuronal convolucional, podemos simplificar la red como una función L=F(X,W) donde X son los datos de entrada y W los pesos correspondiente de cada capa, que tiene como salida una perdida L. Usando la perdida de la salida esto se conoce como forward pass, podemos computar los gradientes para cada una de las capas mediante el proceso de backpropagation y asi actualizar los pesos usando dichos gradientes. Esto se conoce como el backward pass.

Podemos definir este proceso de la siguiente manera

$$W_{t+1} = W_t - \alpha \nabla f(W_t, X),$$

donde Wt son el conjunto de los pesos calculados en el instante t. Quiere decir que son los pesos calculados en ese momento, por lo que para calcular el gradiente que es el decremento dependerá de la entrada Wt-1 que sera los peso calculados en el paso anterior. La tasa de aprendizaje α determinara el tamaño del paso en la dirección del gradiente negativo.

2.2 Stochastic Gradient Descent (SGD)

El descenso estocástico o gradiente de descenso incremental, es una aproximación estocástico del gradiente descendiente usado para minimizar una funcion objetivo que se escribe como una suma de funciones diferenciables. Este optimizador trata de encontrar mínimos y máximos por iteración.

Al igual que la función del gradiente descendiente, el gradiente indica la dirección en la que la función tiene el ratio de aumento mas pronunciada aunque no indica hasta donde se debe avanzar en esa dirección (ver Ilustración 18). Como este optimizador no nos fija cuanto avanzar, existe la tasa de aprendizaje que nos determina cuanto avanzar en cada iteración en la dirección del gradiente. En el descenso por gradiente original, se deben calcular los gradientes para todos lo datos de entrenamiento para poder realizar una iteración. Esto supone un grave problema al entrenar una red neuronal convolucional puesto que se necesita grandes cantidades de datos para evitar el overfitting. El SGD soluciona este problema calculando una estimación entrenando por lotes de n numero de muestras de entrenamiento.

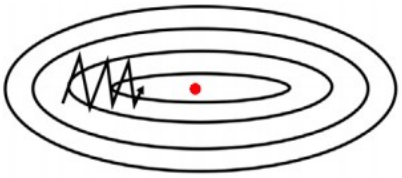


Ilustración 2: Descenso por gradiente estocástico

2.3 Momentum

El SGD tiene problemas para navegar por áreas en las que la curva se inclina mas a una dimensión que otra. Estas zonas son comunes cerca de un óptimo local. El momentum es un método que ayuda a acelerar el SGD en la dirección correcta. Podemos definir el termino como

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_w L(W, X, y)$$
$$W = W - \alpha V_t$$

Donde L es una función de perdida, con el decremento del gradiente del peso alfa la tasa de aprendizaje. La otra forma, que es la mas popular de escribir esta formula pero es menos intuitiva y simplemente omite el termino (1-β). Usando este termino Vt podemos actualizar los peso de la siguiente forma:

$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

Como podemos deducir de esta expresión matemática, el momentum incrementara para los gradientes que apunten hacia una misma dirección de forma consistente. En caso contrario se reducirá como se observa en la Ilustración 19.

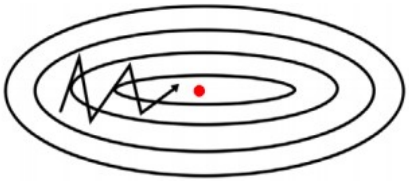


Ilustración 3: Descenso de gradiente estocástico con momentum

2.4 Adagrad

El optimizador Adagrad es un algoritmo basado en gradiente que adapta la tasa de aprendizaje a los parámetros. Realiza grandes actualizaciones cuando los parámetros son poco frecuentes y pequeñas actualización cuando son muy frecuentes.

Esta actualización de pesos puede definirse de la siguiente manera:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon + \Sigma g_t^2}} \odot g_t,$$

donde θ es el parámetro W a actualizar por lo que sera el peso, η es la taza de aprendizaje inicial ya que usara una tasa de aprendizaje distinta para cada peso, ε es una pequeña cantidad que se usa para evitar la división de cero ya que podría causar una indeterminación , $\Sigma\ g^2t$ es la suma de la estimación del gradiente cuadrado durante el entrenamiento y gt es la estimación del gradiente en el intervalo de tiempo t.

2.5 Adadelta

Adadelta es una extensión de Adagrad que busca reducir su tasa de aprendizaje agresiva y monótamente decreciente. En lugar de acumular todos los gradientes cuadrados pasados, Adadelta restringe la ventana de los gradientes pasados acumulados a un tamaño fijo v.

En lugar de almacenar de manera ineficiente w gradientes cuadrados anteriores, la suma de los gradientes se define de forma recursiva como un promedio decreciente de todos los gradientes cuadrados anteriores. El promedio E[g2]t depende de la medida anterior y del gradiente actual.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)\nabla f^2(W_t, X).$$

3. Código completo

3.1 Entrenamiento

```
#Entrenamiemto de una red nueronal

#Importacion de librerias
import matplotlib.pyplot as plt
import numpy as np
import os
import sys
import PIL
import tensorflow as tf
import pathlib
import seaborn as sn
import plotly.graph_objs as go
import plotly.figure_factory as ff


from PIL import Image
from keras import backend as K
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import EarlyStopping,ReduceLROnPlateau
from tensorflow import keras
from tensorflow.python.keras import optimizers
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

#Matriz de confusion
def plot_confusion_matrix(y_true, y_pred, class_names):
  global confusion_matrix
  confusion_matrix = confusion_matrix(y_true, y_pred)
  confusion_matrix = confusion_matrix.astype(int)

  layout = {
      "title": "Confusion Matrix",
      "xaxis": {"title": "Predicted value"},
      "yaxis": {"title": "Real value"}
  }

  z_text = [[str(class_names) for class_names in class_names] for class_names in confusion_matrix]
  fig = ff.create_annotated_heatmap(z=confusion_matrix, x=class_names, y=class_names, annotation_text=z_text, colorscale='Blues', showscale=True)
  fig.show()


#Eliminamos pesos guardados de un entenamiento anterior
K.clear_session()

#ruta del set de datos de imagenes
data_dir = '/home/paulo/Escritorio/tfg/imagenes3'
data_dir = pathlib.Path(data_dir)
```

```
44
45    #ruta del set de datos de imagenes
46    data_dir = '/home/paulo/Escritorio/tfg/imagenes3'
47    data_dir = pathlib.Path(data_dir)
48
49    #imprimos por pantalla dicha ruta
50    print(data_dir)
51
52    #numero total de imagenes
53    image_count = len(list(data_dir.glob('*/*.jpg')))
54    print(image_count)
55
56    #Parametros del entrenamiento
57    batch_size = 16  #lote de imagenes a procesar en cada iteracion
58    img_height = 150 # altura de la imagen
59    img_width = 150  #ancho de la imagen
60
61    #Cargamos nuestro set de datos de entrenamiento desde un directorio
62    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
63      data_dir,   #ruta de las imagenes
64      validation_split=0.2, #20% de las imagenes las usaremos para validacion
65      subset="training", #entrenamiento
66      seed=123,  #Semilla aleatoria opcional para barajar y transformaciones
67      image_size=(img_height, img_width), #tamaño de las imagenes
68      batch_size=batch_size) #lote de imagenes
69
```

```
70
71    #Cargamos nuestro set de datos de validacion
72    val_ds = tf.keras.preprocessing.image_dataset_from_directory(
73      data_dir,
74      validation_split=0.2,
75      subset="validation",
76      seed=123,
77      image_size=(img_height, img_width),
78      batch_size=batch_size)
79
80
81    #Representamos la imagenes de entrenamiento
82    plt.figure(figsize=(10, 10))
83    for images, labels in train_ds.take(1):
84        for i in range(9):
85            ax = plt.subplot(3, 3, i + 1)
86            plt.imshow(images[i].numpy().astype("uint8"))
87            plt.title(int(labels[i]))
88            plt.axis("off")
89
90    plt.show()
```

```
93
94    #Capturamos las etiquetas de cada clase
95    class_names = train_ds.class_names
96    print(class_names)
97
98    #Dimensiones de la imgen y numero de etiquetas
99    for image_batch, labels_batch in train_ds:
100     print(image_batch.shape)
101     print(labels_batch.shape)
102     break
103
104
105   #Precargamos los datos desde la cache.
106   AUTOTUNE = tf.data.experimental.AUTOTUNE
107   train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
108   val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
109
110
111   #Aumento de datos
112   data_augmentation = keras.Sequential(
113     [
114       layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width,3)),
115       layers.experimental.preprocessing.RandomRotation(0.4),
116       layers.experimental.preprocessing.RandomContrast(0.2),
117       layers.experimental.preprocessing.RandomZoom(0.2),
118     ]
119   )
120
121   #Imagenes con el aumento de datos
122   plt.figure(figsize=(10, 10))
123   for images, _ in train_ds.take(1):
124       for i in range(9):
125           augmented_images = data_augmentation(images)
126           ax = plt.subplot(3, 3, i + 1)
127           plt.imshow(augmented_images[0].numpy().astype("uint8"))
128           plt.axis("off")
129
130   plt.show()
131
132
133   #numero de clases de monedas
134   num_classes=3
```

```python
#Red neuronal convolocuional
model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(256, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(512, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.3),
  layers.Flatten(),
  layers.Dense(512, activation='relu'),
  layers.Dropout(0.5),
  layers.Dense(num_classes, activation='softmax')
])


#Optimizacion del error y metricas para la precision.
model.compile(optimizer='Adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#mostranmos por pantalla nuestra red neuronal con el numero de parametros
model.summary()

#numero de epocas
epochs = 500

#Paramos el entrenamiento si no mejora .
early_stop = EarlyStopping(monitor='val_loss', patience=100, verbose=1, mode='min', restore_best_weights=True)

#Reducimos la tasa de aprendizaje si no mejora
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)

#entrenamos nuestro modelo
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs, verbose=2, callbacks=[early_stop,reduce_lr])


#graficamos por pantalla la precision y el error
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```python

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(acc, label='train')
plt.plot(val_acc, label='test')
plt.title('Training and Validation Accuracy')
plt.legend()


plt.subplot(1, 2, 2)
plt.plot(loss, label='train')
plt.plot(val_loss, label='test')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()


#Construimos matriz de confusion
y_pred = model.predict(val_ds)
predicted_categories = tf.argmax(y_pred, axis=1)
true_categories = tf.concat([y for x, y in val_ds], axis=0)
print('\n')
print('\n')
print('Confusion Matrix')
cm=confusion_matrix(predicted_categories, true_categories)
print(cm)

#clasification Report
print('\n')
print('\n')
print('Clasification Report')
print(classification_report(predicted_categories, true_categories, target_names=class_names))

#Matriz de confusion con mapa de calor
plot_confusion_matrix(predicted_categories, true_categories, class_names)

```

```
#guardamos nuestro modelo
target_dir = './modelo/'
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
model.save('./graficas_redes_3class/modelo13.h5')
model.save_weights('./graficas_redes_3class/pesos13.h5')
```

3.2 Detección y predicación del objeto.

```
#Deteccion de objetos y prediccion

#importacion de librerias
import tensorflow as tf
from keras.preprocessing.image import load_img, img_to_array
from keras.models import load_model
import matplotlib.pyplot as plt
import cv2
import numpy as np

#caragmos nuestro modelo
longitud, altura = 150, 150
modelo = './graficas_redes_3class/modelo11.h5'
pesos_modelo = './graficas_redes_3class/pesos11.h5'
new_model = load_model(modelo)
new_model.load_weights(pesos_modelo)

#variables para guradar el resultado y la posicion
location1 =[]
location2 =[]
answers= []

#iniciamos la camara
cam = cv2.VideoCapture(2)

#Kernel para generar la mascara
kernel = np.ones((11,11),np.uint8)
kernel1 = np.ones((15,15),np.uint8)
i=0

```

```
#bucle infinito para cada fotograma
while (True):
    ret,frame = cam.read()

    frame1=frame.copy()


    #Rangos de colores HSV
    rangomax = np.array([100,255,255])
    rangomin = np.array([0,50,20])

    #Binarizamos la imagen
    mascara = cv2.inRange(frame, rangomin, rangomax)

    #operaciones moroflogicas
    opening = cv2.morphologyEx(mascara, cv2.MORPH_OPEN, kernel)
    cv2.imshow('opening',opening)
    cv2.imwrite('opening.jpg',opening)
    close1= cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
    cv2.imshow('close1',close1)
    cv2.imwrite('close1.jpg',close1)
    close2= cv2.morphologyEx(close1, cv2.MORPH_CLOSE, kernel)
    cv2.imshow('close2',close2)
    cv2.imwrite('close2.jpg',close2)
    #dilatamos
    dil2 = cv2.dilate(close2,kernel1,iterations = 1)
    cv2.imwrite('dil2.jpg',dil2)
    #operaciones morfologicas
    close3= cv2.morphologyEx(dil2, cv2.MORPH_CLOSE, kernel1)
    cv2.imshow('close3',close3)
    cv2.imwrite('close3.jpg',close3)
    close4= cv2.morphologyEx(close3, cv2.MORPH_CLOSE, kernel1)
    cv2.imshow('close4',close4)
    cv2.imwrite('close4.jpg',close4)


    #Buscamos contornos
    (contornos,_) = cv2.findContours(close4, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    print("He encontrado {} objetos".format(len(contornos)))
```

```
    #recorremos cada contorno
    for c in contornos:
        area = cv2.contourArea(c)
        if area > 1200 and area < 1000000:
            (x, y, w, h) = cv2.boundingRect(c)
            #deteccion de obejtos
            cv2.circle(frame,(x+w//2,y+h//2),6,(0,0,100),-1)
            cv2.rectangle(frame, (x, y), (x + w + 4, y + h + 4), (0, 255, 0), 3, cv2.LINE_AA)
            #segmentacion de cada objeto
            recorte = frame1[y:y+h+3,x:x+w+3]
            cv2.imwrite("/home/paulo/Escritorio/tfg/cap/"+str(i)+".jpg", recorte)
            #Cargamos cada objeto detectado y iniciamos la prediccion
            foto = load_img('/home/paulo/Escritorio/tfg/cap/'+str(i)+'.jpg', target_size=(longitud, altura))
            foto = img_to_array(foto)
            foto = np.expand_dims(foto, axis=0)
            array = new_model.predict(foto)
            result = array[0]
            #resultado
            answer = np.argmax(result)


            #guardamos las posiciones de cada objeto en una lista
            location1.append(x)
            location2.append(y)
            answers.append(answer)

            if answer == 0:
                cv2.putText(frame,'cinco', (x,y-5),1,1.5,(0,0,255),2)
            if answer == 1:
                cv2.putText(frame,'cincuenta', (x,y-5),1,1.5,(0,0,255),2)
            if answer == 2:
                cv2.putText(frame,'euro', (x,y-5),1,1.5,(0,0,255),2)
```

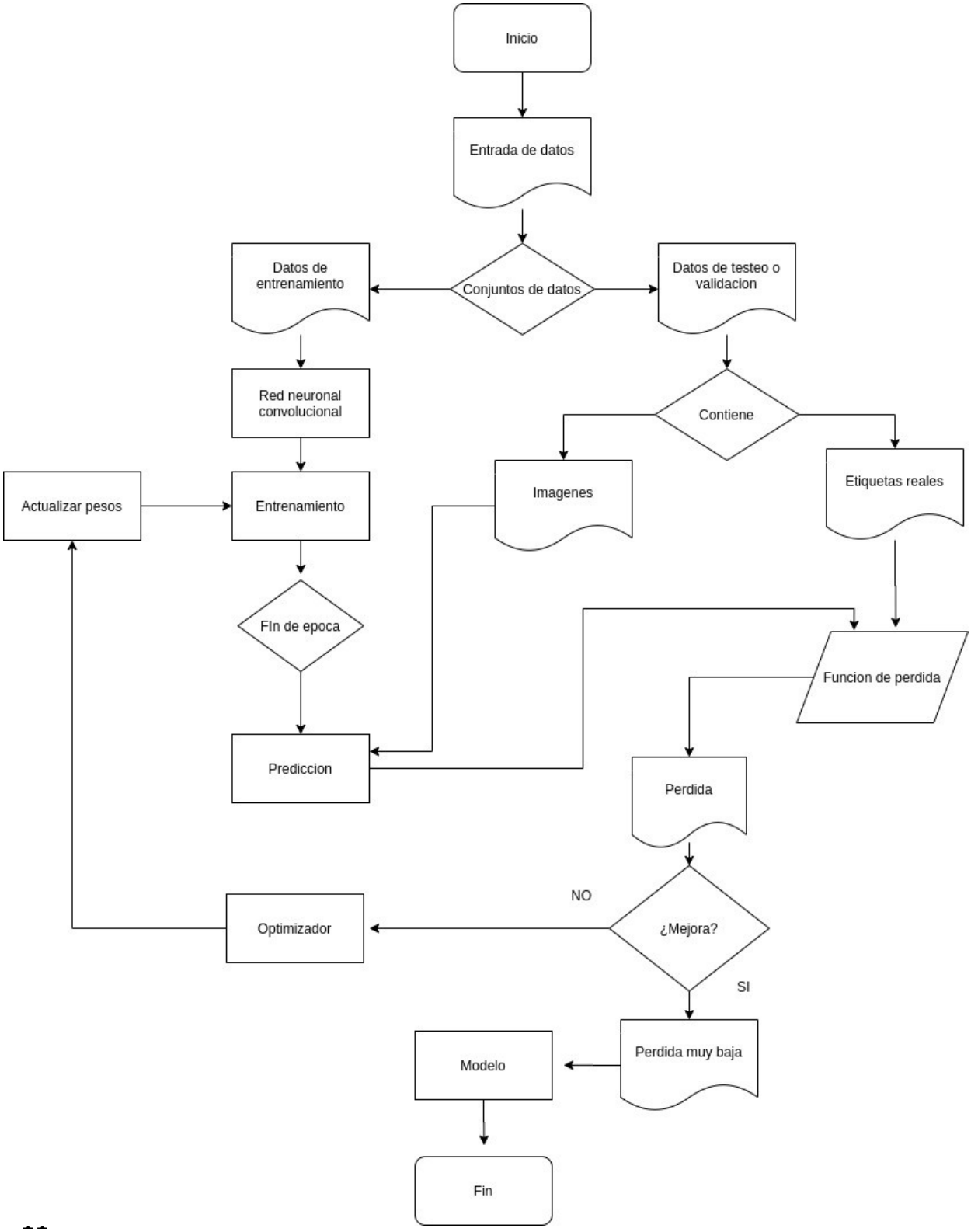
```
        #mostramos por pantalla las lista
        print(answers)
        print(location1)
        print(location2)
        print(len(answers))



        #mostramos la camara por pantalla cada fotograma
        cv2.imshow('camara1',frame)
        #cerramos nuestra camara
        k = cv2.waitKey(1) & 0xFF
        if k==27:
            break
```

4. Diagrama de Flujo

4.1 Entrenamiento



4.2 Predicción y Detección de objetos