SENATI

Apellidos y Nombres: Huamani Feria Claudio

ID: 001597435

Dirección Zonal/CFP: Arequipa/Arequipa/Puno

Carrera: Ingeniería de software con Inteligencia artificial

Semestre: 4to Semestre

Instructor: Jesús Romero Villanueva

Curso: seminario de complementación práctica I

TAREAS Y OPERACIONES COVERTURADAS

-TH1_Realiza operaciones con las Librerías Pandas y Numpy

- Estudia los fundamentos de vectores y matrices.
- Define la librería Pandas y Numpy.
- Manipula y analiza estructuras de datos.
- Lee archivos CSV con Numpy y Pandas.

-TH2_Estudia el uso de las Librerías Scikit-learn y Pytorch

- Define el concepto de Machine Learning
- Define la librería Scikit-Learn y Pytorch.
- Identifica principales aplicaciones.

-TH3_Estudia el uso de las librerías SciPy y Nltk

- Define que el procesamiento de lenguaje natural (NLP)
- Define la librería SciPy y Nltk.
- Identifica principales aplicaciones del NLP

-TH4_Estudia el uso de las Librerías Tensorflow y Keras

- Define el concepto de Deep Learning
- Define la librería Tensorflow y Keras.
- Identifica principales aplicaciones

-TH5_Realiza operaciones con las librerías Matplotlib y Seaborn

- Define los histogramas y la importancia de la visualización de datos.
- Define la librería Matplotlib y Seaborn.
- Crea ejemplos de aplicación

-TH6_Crea programas con algoritmos de aprendizaje supervisado

- Describe los tipos de algoritmos del aprendizaje supervisado.
- Define la regresión lineal simple y múltiple.
- Implementa algoritmo de regresión lineal simple con Python.

-TH6_Crea programas con algoritmos de no aprendizaje supervisado

- Describe los tipos de algoritmos del aprendizaje supervisado.
- Define la regresión lineal simple y múltiple.
- Implementa algoritmo K-Means con Python.

_TH7_ Crea programas con algoritmos de aprendizaje supervisado

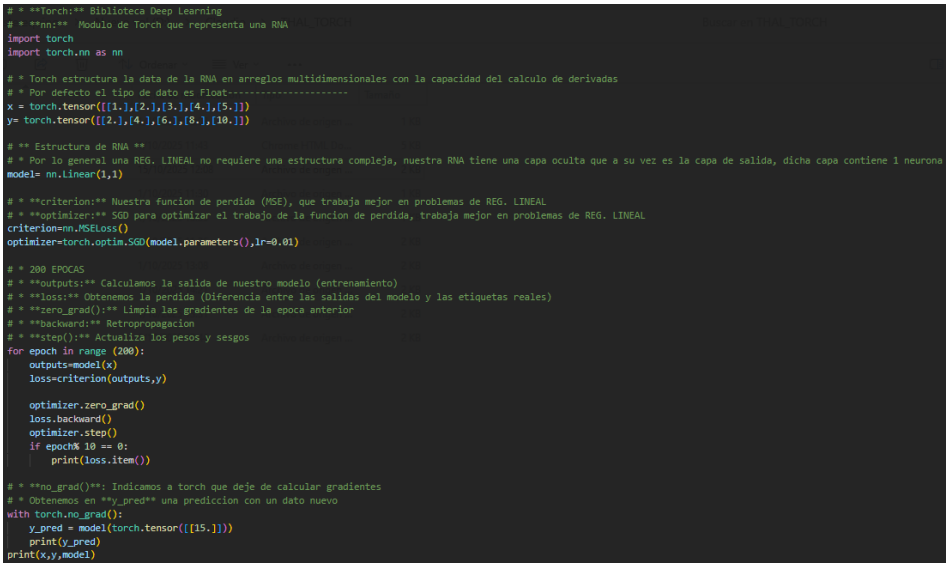
- Describe los tipos de algoritmos del aprendizaje supervisado.
- Define la regresión lineal simple y múltiple.
- Implementa algoritmo de regresión lineal simple con Python.

_TH8_ Define la estructura y crea una red neuronal artificial

- Define la red neuronal artificial y su importancia en al IA.
- Describe la estructura de una red neuronal artificial.
- Identifica los tipos de redes neuronales artificiales.
- Crea una red neuronal con Tensorflow y Keras

Tarea 1

Se crea un modelo de regresión lineal simple con PyTorch usando una capa lineal.
Se entrena con datos de entrada y salida, optimizando el error cuadrático medio con SGD durante 200 épocas.



```
# * **Torch:** Biblioteca Deep Learning
# * **nn:**  Modulo de Torch que representa una RNA
import torch
import torch.nn as nn

# * Torch estructura la data de la RNA en arreglos multidimensionales con la capacidad del calculo de derivadas
# * Por defecto el tipo de dato es Float----------------------
x = torch.tensor([[1.],[2.],[3.],[4.],[5.]])
y= torch.tensor([[2.],[4.],[6.],[8.],[10.]])

# ** Estructura de RNA **
# * Por lo general una REG. LINEAL no requiere una estructura compleja, nuestra RNA tiene una capa oculta que a su vez es la capa de salida, dicha capa contiene 1 neurona
model= nn.Linear(1,1)

# * **criterion:** Nuestra funcion de perdida (MSE), que trabaja mejor en problemas de REG. LINEAL
# * **optimizer:** SGD para optimizar el trabajo de la funcion de perdida, trabaja mejor en problemas de REG. LINEAL
criterion=nn.MSELoss()
optimizer=torch.optim.SGD(model.parameters(),lr=0.01)

# * 200 EPOCAS
# * **outputs:** Calculamos la salida de nuestro modelo (entrenamiento)
# * **loss:** Obtenemos la perdida (Diferencia entre las salidas del modelo y las etiquetas reales)
# * **zero_grad():** Limpia las gradientes de la epoca anterior
# * **backward:** Retropropagacion
# * **step():** Actualiza los pesos y sesgos
for epoch in range (200):
    outputs=model(x)
    loss=criterion(outputs,y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch% 10 == 0:
        print(loss.item())

# * **no_grad()**: Indicamos a torch que deje de calcular gradientes
# * Obtenemos en **y_pred** una prediccion con un dato nuevo
with torch.no_grad():
    y_pred = model(torch.tensor([[15.]]))
    print(y_pred)
print(x,y,model)
```

Tarea 2

Se generan datos con dos entradas usando NumPy, luego se convierten a tensores de PyTorch para entrenar una red simple con una capa lineal que aprende a sumar las dos entradas.

```
# 1) Ajustar el código del Ejercicio1, para que la celda Nro. 2, trabaje con NUMPY.
# 2) Implementar una RNA torch con 2 entradas y una salida, debe calcular la suma de entradas

import torch
import torch.nn as nn
import numpy as np

x = np.array([[1.,2.],[3.,4.],[5.,6],[7.,8.],[9.,10.]])
y= np.array([x[:,0]+x[:,1]])

ToTensor_x=torch.tensor(x,dtype=torch.float)
ToTensor_y=torch.tensor(y,dtype=torch.float).reshape(-1,1)
model= nn.Linear(2,1)

criterion=nn.MSELoss()
optimizer=torch.optim.SGD(model.parameters(),lr=0.01)

for epoch in range (1000):
    outputs=model(ToTensor_x)
    loss=criterion(outputs,ToTensor_y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if epoch% 10 == 0:
        print('Perdida: \n',loss.item() ,'\n')

with torch.no_grad():
    y_pred = model(torch.tensor([[11.,12.]]))
    print('Prediccion: \n', y_pred ,'\n')

print('tensor X \n',ToTensor_x,'\n')
print('tensor Y \n',ToTensor_y)
```

Tarea 3

Se crean y entrenan modelos de redes neuronales en PyTorch para tareas de regresión y lógica usando funciones de pérdida como MSE y BCELoss junto con optimizadores como SGD y Adam, preparando datos con NumPy o tensores y realizando predicciones precisas.

```python
import torch
import torch.nn as nn

x = torch.tensor([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
y = torch.tensor([[0.0], [0.0], [0.0], [1.0]])


class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN,self).__init__()
        self.hidden = nn.Linear(2, 3)
        self.output = nn.Linear(3, 1)
        self.sigmoid= nn.Sigmoid()
    def forward(self,x):
        x = self.sigmoid(self.hidden(x))
        x =self.sigmoid(self.output(x))
        return x

model = SimpleNN()
criterion=nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(),lr=0.01)

for epoch in range (1000):
    outputs=model(x)
    loss=criterion(outputs,y)
    optimizer.zero_grad()
    loss.backward
    optimizer.step()
    if epoch%100==0:
        print(loss.item())
        print(outputs.round())

with torch.no_grad():
    y_pred = model(x)
    print(y_pred.round())
```

Tarea 4

Se crea y entrena una red neuronal simple con una sola neurona y activación sigmoide para clasificar valores negativos y positivos usando BCELoss y Adam.

```python
import torch
import torch.nn as nn
import numpy as np
x = np.linspace(-10,10,200).reshape(-1,1)
y = np.array([[1 if i < 0 else 0 for i in x]] ).reshape(-1,1)
ToTensor_x = torch.tensor(x,dtype=torch.float32)
ToTensor_y= torch.tensor(y,dtype=torch.float32).reshape(-1,1)

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN,self).__init__()
        self.output = nn.Linear(1, 1)
        self.sigmoid= nn.Sigmoid()
    def forward(self,ToTensor_x):
        x =self.sigmoid(self.output(ToTensor_x))
        return x

model = SimpleNN()
criterion=nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(),lr=0.01)

for epoch in range (1000):
    outputs=model(ToTensor_x)
    loss=criterion(outputs,ToTensor_y)
    optimizer.zero_grad()
    loss.backward
    optimizer.step()
    # if epoch%100==0:
    #     print(loss.item())
    #     print(outputs.round())

with torch.no_grad():
    y_pred = model(torch.tensor([[-20.],[-5.],[2.],[8.],[2.],[10.],[0.],]))
    print(y_pred.round())
```

Tarea 5

El código prepara datos con experiencia y rango codificados luego crea una red neuronal simple con PyTorch que usa una capa lineal y sigmoid para predecir la variable contratado optimiza con Adam y calcula la pérdida con BCELoss y finalmente predice un ejemplo

```python
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from sklearn.preprocessing import StandardScaler, LabelEncoder

datos = pd.DataFrame(
    {
        'exp': [1, 2, 3, 4, 5],
        'rango': ['j', 'j', 'j', 's', 's'],
        'contratado': [1, 1, 1, 0, 0],
    }
)

x=datos[['exp','rango']]

scaler=StandardScaler()
x_num=scaler.fit_transform(x[['exp']])

encoder=LabelEncoder()
x_cat=encoder.fit_transform(x['rango']).reshape(-1,1)

x=np.hstack([x_num,x_cat])
y=np.array(datos[['contratado']])

ToTorch_x= torch.tensor(x,dtype=torch.float32)
ToTorch_y= torch.tensor(y,dtype=torch.float32)

class SimpleNN(nn.Module):
    def __init__ (self):
        super(SimpleNN,self).__init__()
        self.output=nn.Linear(2,1)
        self.sigmoid=nn.Sigmoid()
    def forward (self,ToTorch_x):
        x=self.sigmoid(self.output(ToTorch_x))
        return x
model=SimpleNN()
criterion=nn.BCELoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.01)

for epochs in range(200):
    output=model(ToTorch_x)
    loss=criterion(output,ToTorch_y)
    loss.backward
    optimizer.step
with torch.no_grad():
    y_pred=model(torch.tensor([[3.,0.]]))
    print(y_pred)
```

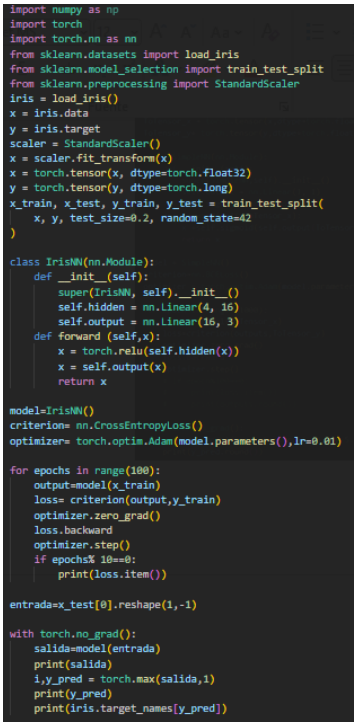
Tarea 6

El código codifica variables categóricas y ordinales prepara datos para PyTorch crea una red neuronal con capa oculta y salida con sigmoid usa BCELoss y Adam para entrenar predice probabilidades y muestra resultados redondeados junto con etiquetas reales.

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, LabelBinarizer
import numpy as np
import torch
import torch.nn as nn

datos = pd.DataFrame(
    {
        "leng": ["Java", "PHP", "Java", "PHP", "Java"],  # NOMINAL
        "cert": ["NO", "NO", "SI", "SI", "NO"],  # NOMINAL
        "rang": ["J", "J", "S", "S", "S"],  # ORDINAL
        "contratado": [0, 0, 1, 0, 0],  # TARGET
    }
)

encoder = LabelEncoder()
x_num = encoder.fit_transform(datos["rang"]).reshape(-1, 1)

one_encoder = OneHotEncoder(sparse_output=False)
x_cat = one_encoder.fit_transform(datos[["cert", "leng"]])

label_bin = LabelBinarizer()
y_num = label_bin.fit_transform(datos["contratado"])

x = np.hstack([x_num, x_cat])
y = np.array(y_num)

ToTorch_x = torch.tensor(x, dtype=torch.float32)
ToTorch_y = torch.tensor(y, dtype=torch.float32)


class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.hidden = nn.Linear(5, 3)
        self.output = nn.Linear(3, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.hidden(x))
        x = self.sigmoid(self.output(x))
        return x


model = SimpleNN()
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epochs in range(1000):
    output = model(ToTorch_x)
    loss = criterion(output, ToTorch_y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
with torch.no_grad():
    y_pred = model(ToTorch_x)
    print(y_pred.round())

print(ToTorch_y)
```

Tarea 7

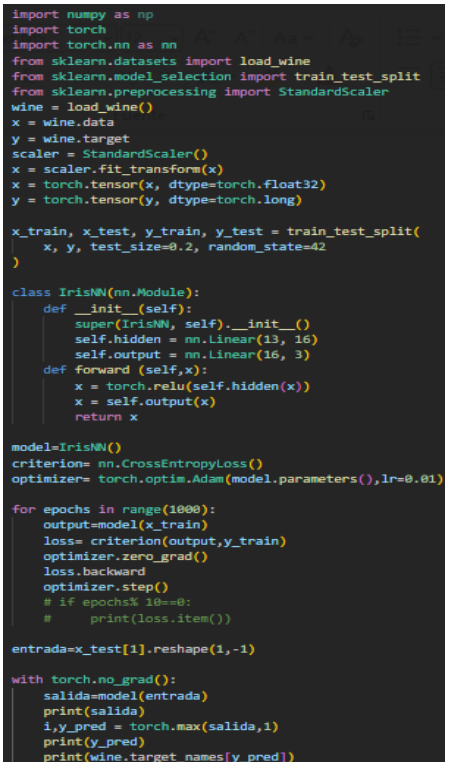
El código carga Iris, escala datos, crea red neuronal con capa oculta usa ReLU y CrossEntropyLoss entrena con Adam durante 100 épocas imprime pérdida cada 10 épocas y predice clase para una muestra mostrando su nombre corregir faltan paréntesis en backward y step.



```python
import numpy as np
import torch
import torch.nn as nn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
iris = load_iris()
x = iris.data
y = iris.target
scaler = StandardScaler()
x = scaler.fit_transform(x)
x = torch.tensor(x, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.long)
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)

class IrisNN(nn.Module):
    def __init__(self):
        super(IrisNN, self).__init__()
        self.hidden = nn.Linear(4, 16)
        self.output = nn.Linear(16, 3)
    def forward (self,x):
        x = torch.relu(self.hidden(x))
        x = self.output(x)
        return x

model=IrisNN()
criterion= nn.CrossEntropyLoss()
optimizer= torch.optim.Adam(model.parameters(),lr=0.01)

for epochs in range(100):
    output=model(x_train)
    loss= criterion(output,y_train)
    optimizer.zero_grad()
    loss.backward
    optimizer.step()
    if epochs% 10==0:
        print(loss.item())

entrada=x_test[0].reshape(1,-1)

with torch.no_grad():
    salida=model(entrada)
    print(salida)
    i,y_pred = torch.max(salida,1)
    print(y_pred)
    print(iris.target_names[y_pred])
```

Tarea 8

El código carga y escala datos de vino crea red neuronal con capa oculta usa ReLU y CrossEntropyLoss entrena con Adam durante 1000 épocas pero faltan paréntesis en loss.backward() y optimizer.step() impidiendo entrenamiento correcto y predicción final.



```python
import numpy as np
import torch
import torch.nn as nn
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
wine = load_wine()
x = wine.data
y = wine.target
scaler = StandardScaler()
x = scaler.fit_transform(x)
x = torch.tensor(x, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.long)

x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)

class IrisNN(nn.Module):
    def __init__(self):
        super(IrisNN, self).__init__()
        self.hidden = nn.Linear(13, 16)
        self.output = nn.Linear(16, 3)
    def forward (self,x):
        x = torch.relu(self.hidden(x))
        x = self.output(x)
        return x

model=IrisNN()
criterion= nn.CrossEntropyLoss()
optimizer= torch.optim.Adam(model.parameters(),lr=0.01)

for epochs in range(1000):
    output=model(x_train)
    loss= criterion(output,y_train)
    optimizer.zero_grad()
    loss.backward
    optimizer.step()
    # if epochs% 10==0:
    #     print(loss.item())

entrada=x_test[1].reshape(1,-1)

with torch.no_grad():
    salida=model(entrada)
    print(salida)
    i,y_pred = torch.max(salida,1)
    print(y_pred)
    print(wine.target_names[y_pred])
```

Tarea 9

El código carga y preprocesa cats_vs_dogs crea CNN con capas convolucionales y pooling entrena con Adam y binary crossentropy durante cinco épocas grafica precisión y muestra predicción para una imagen etiquetando como perro o gato según el resultado.

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt

(ds_train, ds_test), ds_info= tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]','train[80%:]'],
    as_supervised= True,
    with_info= True
)

def preprocessing(img, label):
  img= tf.image.resize(img, (128, 128))
  img= tf.cast(img, tf.float32)/255.0
  return img, label

train_ds= ds_train.map(preprocessing).shuffle(10000).batch(32).prefetch(1)
test_ds= ds_test.map(preprocessing).batch(32).prefetch(1)

for imgs, labels in train_ds.take(1):
  for i in range(6):
    plt.subplot(2,3, i+1)
    plt.imshow(imgs[i])
    plt.title('dog' if labels[i] else 'cat')
    plt.axis('off')

model= tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(128,128,3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history= model.fit(train_ds, epochs=5, validation_data= test_ds)

plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión del Modelo')
plt.grid()
plt.legend()
plt.show()

for img, label in test_ds.take(1):
  pred= model.predict(img)
  plt.imshow(img[13])
  plt.title('dog' if pred[13]>0.5 else 'cat')
  plt.axis('off')
  plt.show()
```

Tarea 10

El código carga MNIST mal divide datos usa mismo split sin canal extra en imágenes no compila ni entrena modelo y predice sin ajustar capas Conv2D ni mostrar imagen con escala de grises faltan pasos para entrenamiento y evaluación correcta.

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
from keras.datasets import mnist

(ds_train,ds_test),ds_info =tfds.load(
    'mnist',
    shuffle_files=True,
    split=['train[:80%]','train[:80%]'],
    as_supervised=True,
    with_info=True
)

def preprocessing(img,label):
    img = tf.image.resize(img,(28,28))
    img = tf.cast(img,tf.float32)/255.0
    return img,label

train_ds = ds_train.map(preprocessing).shuffle(10000).batch(32).prefetch(1)
test_ds = ds_test.map(preprocessing).batch(32).prefetch(1)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16,(3,3),activation='relu',input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64,(3,3),activation='relu',input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])

for img ,label in test_ds.take(1):
  pred = model.predict(img)
  plt.imshow(img[2])
  plt.axis('off')
```

Tarea 11

El código carga imágenes en escala de grises de FER_2013 desde carpetas prepara datasets normalizando crea CNN con dos capas convolucionales y pooling capa densa y salida softmax usa sparse_categorical_crossentropy entrena 15 épocas predice y muestra una imagen con su etiqueta y grafica precisión entrenamiento y validación.

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'ROSTROS_DIGIT/archive(6)/train', image_size =(48,48), color_mode = 'grayscale', batch_size = 32
)
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'ROSTROS_DIGIT/archive(6)/test', image_size =(48,48), color_mode = 'grayscale', batch_size = 32
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'ROSTROS_DIGIT/archive(6)/validation', image_size =(48,48), color_mode = 'grayscale', batch_size = 32
)

class_names= train_ds.class_names
print(class_names)


train_ds= train_ds.map(lambda x,y: (x/255.0, y))
test_ds= test_ds.map(lambda x,y: (x/255.0, y))
val_ds= val_ds.map(lambda x,y: (x/255.0, y))

for imgs, labels in train_ds.take(1):
  for i in range(6):
    plt.subplot(2,3,i+1)
    eti=labels[i]
    plt.title(class_names[eti])
    plt.imshow(imgs[i])
    plt.axis('off')
plt.show()

model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(48,48,1)),
    tf.keras.layers.Conv2D(32,(3,3),activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64,(3,3),activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(len(class_names),activation='softmax')
])

print(model.summary())

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_ds, epochs=15, validation_data= val_ds)
for images, labels in test_ds.take(1):
    img = images[5]
    true_label = labels[5].numpy()
    img_array= np.expand_dims(img, 0)
    pred = np.argmax(model.predict(img_array),1)[0]
    plt.imshow(img)
    plt.title(class_names[pred])
    plt.show()

plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión del Modelo')
plt.grid()
plt.legend()
plt.show()
```