# Section5: Keras

# Tensor Flow and Keras library

- **TensorFlow:** TensorFlow is a versatile open-source machine learning framework developed by Google, offering flexibility and scalability for building, training, and deploying various machine learning models.
- **Keras:** Keras is a high-level neural networks API written in Python, designed for quick and easy experimentation with deep learning models, often used in conjunction with TensorFlow for its simplicity and user-friendly interface.
- **Google Colab**: Google Colab is a cloud-based platform that provides free access to Jupyter notebooks with pre-installed machine learning libraries, including TensorFlow and Keras, allowing collaborative development and execution of machine learning projects directly in the browser.

# MNIST Dataset: First data set

**Some key features of the MNIST dataset**:

- **Images**: The dataset contains images of handwritten digits, each measuring 28 pixels by 28 pixels. Therefore, each image is represented as a 28x28 matrix of pixel values.
- **Labels**: Each image in the dataset is associated with a label indicating the digit it represents, ranging from 0 to 9.
- **Training and Test Sets**: The MNIST dataset is typically divided into two subsets: a training set and a test set. The training set is used to train machine learning models, while the test set is used to evaluate the performance of the trained models.

# Load the MNIST dataset and access the training and test sets:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()

print('X Train Shape is : ', x_train.shape) # X Train Shape is :  (60000, 28, 28)
print('X Train  is : ', x_train[5]) # print the image number 5 in the training set
print('---------------------------------------- ')
print('X Test Shape is : ', x_test.shape) # X Test Shape is :  (10000, 28, 28)
print('X Test  is : ', x_test[5]) # print the image number 5 in the testing set
print('---------------------------------------- ')
print('y Train Shape is : ', y_train.shape) # y Train Shape is :  (60000,)
print('y Train is : ', y_train[5]) # y Train is (label) :  2
print('---------------------------------------- ')
print('y Test Shape is : ', y_test.shape) # y Test Shape is :  (10000,)
print('y Test  is : ', y_test[5]) # y Test  is (label): 1

import matplotlib.pyplot as plt
plt.matshow(x_test[5])
plt.matshow(x_train[5])
plt.show()
```

- The **train_test_split** function from scikit-learn is typically used to split a dataset into training and testing sets.
- However, since the MNIST dataset is already pre-split into training and testing sets, you wouldn't normally use **train_test_split** for this purpose.

# Building Neural networks

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
#-----------------------------------------------------------------
model = tf.keras.models.Sequential()
# you create an empty sequential neural network model object.
#-----------------------------------------------------------------
model.add(tf.keras.layers.Flatten())
'''
- You can then add layers to this model using the .add() method,
- Flatten() is used as the first layer (input layer) to convert the input image (2D)
  into a one-dimensional array before passing it to the next layer (hidden layer)
```

- For one image: The number of neurons in the input layer =28*28=784

# Building Neural networks (Cont'd)

```
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
'''
- Dense represents adding a hidden layer,Each neuron in a dense layer is connected to every neuron
  in the previous layer.
- The output of a dense layer is calculated as: output= activation(input*kernel + bias)
- 128 is the number of neurons in the hidden layer, it is often chosen as 2^n
  (8, 16, 32, 64, 128, 256,...)
- If you want to create a neural network with three hidden layers, you can add three instances of
  "tf.keras.layers.Dense" to your model.Here's an example:
   >> model.add(tf.keras.layers.Dense(1024, activation=tf.nn.relu))
   >> model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
   >> model.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
- As you can observe, there is a preference for a decrease in the number of neurons from
  one hidden layer to the next (1024 then 256 then 64).
'''
```

# Building Neural networks (Cont'd)

```
model.add(tf.keras.layers.Dropout(0.2))
'''
- Dropout is typically applied to hidden layers.
- It Randomly removes a fraction of the neurons in a layer during training.
- Dropout is a regularization technique commonly used in neural networks to prevent overfitting.
- Overfitting occurs when a machine learning model learns the training data too well,
  capturing noise in the data rather than the underlying patterns.
  As a result, the model performs well on the training data but fails to generalize to
  unseen data or performs poorly on new, unseen examples.
'''
```

# Building Neural networks (Cont'd)

```
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
'''
- Number of classes in mnist dataset=10(0-9)
- Last dense layer adds an output layer to your neural network model.
- Dense layer in this example (output layer) with 10 neurons, corresponds to
  the number of classes or categories in your classification task.
- Softmax activation function is only applied in the last layer.
- Softmax transforms the raw output of each neuron in the output layer into a probability score
  between 0 and 1, ensuring that the sum of all probabilities for each sample equals 1.
'''
```

# Building Neural networks (Cont'd)

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
'''
- model.compile configures your neural network for training.
- optimizer='adam': This specifies the optimization algorithm to use during training (update weight).
- In this case, you're using the Adam optimizer, which is a popular optimization algorithm
  for training deep learning models. If you want to use plain stochastic gradient descent (SGD)
  as your optimizer instead of Adam, you can specify it like this: optimizer='sgd'.
  However, it's important to note that plain SGD is less commonly used in practice compared
  to more advanced optimization algorithms like Adam.
- loss='sparse_categorical_crossentropy', This specifies the loss function to use during training.
- There are various options for loss functions in TensorFlow/Keras,
 >> MSE (mean_squared_error Suitable for regression problems where the model predicts continuous values.)
 >> binary_crossentropy (Suitable for binary classification problems where there are two classes.)
 >> categorical_crossentropy (Suitable for multi-class classification problems where there are more than
    two exclusive classes.)
- metrics=['accuracy']: This specifies the evaluation metric to monitor during training.
'''
```

# Building Neural networks (Cont'd)

```
model.fit(x_train, y_train, epochs=3)
```

Loss: decrease from epoch to epoch

accuracy: increase from epoch to epoch

```
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 2ms/step - accuracy: 0.7256 - loss: 7.4692
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8432 - loss: 0.5784
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8785 - loss: 0.4602
```

- **Epoch 1/3**: Indicates that the model is currently on the first epoch out of a total of 3 epochs. An epoch is one pass through the entire training dataset.
- **1875**: if you have 60,000 samples in the MNIST training dataset and you choose a batch size of 32, then the total number of batches would be 60,000 / 32 = 1875. This means that the training process will iterate over 1875 batches, with each batch containing 32 samples. 1 epoch has 1875 iterations.
- **Note:** In a neural network processing images with a batch size of 32 and each image being a 28x28 grayscale image, the input layer would typically have 25,088 (28*28*32) neurons.
- **11s**: Indicates the elapsed time for the current epoch.
- **2ms/step**: Represents the average time taken per step (or batch) during training.

# Building Neural networks (Cont'd)

- you can change the batch size when training your neural network model.

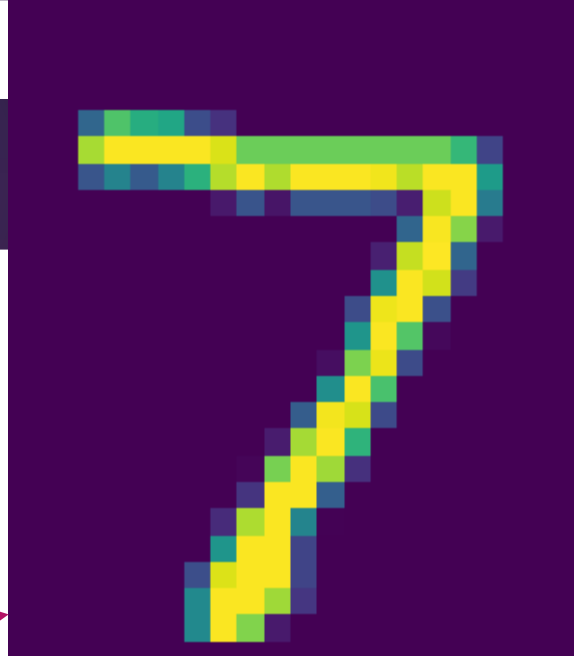
```
model.fit(x_train, y_train,batch_size=64, epochs=3)
```

```
Epoch 1/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7050 - loss: 9.8777
Epoch 2/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8384 - loss: 0.6277
Epoch 3/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8738 - loss: 0.4499
```

# Building Neural networks (Cont'd)

```
y_predict = model.predict(x_test)
# ----------------------------------
print(y_predict.shape)
print(y_predict[0])
print(y_test[0])
import matplotlib.pyplot as plt
plt.matshow(x_test[0])
plt.show()
```

*The dimension of y_predict (neural)= the dimension of y_test (actual)*



- The first dimension (10,000) corresponds to the number of samples in y_predict.
- The second dimension (10) corresponds to the number of classes for which the model is making predictions.

Output:

```
(10000, 10)
[9.9859330e-24 7.0550485e-37 9.2510721e-22 3.5256131e-25 1.3065527e-36
 2.7608122e-32 0.0000000e+00 1.0000000e+00 0.0000000e+00 1.6051290e-18]
7
```

(10000, 10) → y_predict.shape

y_predict[0]

10 numbers in each row (each sample or each image) representing the probabilities for each of the 10 classes.

7 → y_test[0]

# Building Neural networks (Cont'd)

- If you want to convert the predicted probabilities to integers, you can round the values to the nearest integer using:

```
import numpy as np
y_predict=model.predict(x_test)
y_predict = np.round(y_predict).astype(int)
#-----------------------------------------
print(y_predict.shape)
print(y_predict[0])
for i in range (10):
 print(y_predict[i])
```

Print(y_predict.shape)

Print(y_predict[0])

```
(10000, 10)
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
```

For loop: print the first 10 predictions

# Building Neural networks (Cont'd)

```
val_loss, val_acc = model.evaluate(x_test, y_test)
'''
- model.evaluate(x_test, y_test), generates predictions for the input data x_test using the model's current weights,
  and then compares these predictions to the ground truth labels y_test to compute the loss and any specified metrics.
- calling model.evaluate(x_test, y_test) automatically performs the prediction step as part of the evaluation process.
  You don't need to explicitly call model.predict(x_test) before calling model.evaluate(x_test, y_test).


'''
print(val_loss) #0.284
print(val_acc)  #0.935
```

# Finally: You can merge three layers in one step

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(128, activation=  tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
                                   ])
```

# First neural network (Full Code)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
#-------------------------------------------------------------
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#-------------------------------------------------------------------------------------------
model.fit(x_train, y_train,batch_size=32, epochs=3)
#-------------------------------------------------------------------------------------------
model.predict(x_test)
val_loss, val_acc = model.evaluate(x_test, y_test)
#-------------------------------------------------------------------------------------------
print(val_loss) #0.284
print(val_acc)  #0.935
```

# Scale the dataset to raise the accuracy using MinMaxScaler

- In the context of the MNIST dataset, the images are initially stored as a 3D array (the shape of x_train=60000*28*28) where each image is represented as a 28x28 grid of pixel values.

- However, some operations, like scaling with MinMaxScaler, might require the input to be in a different shape. reshape allows us to reshape the 3D array representing the images into a 2D array where each row represents one image. So the new shape of x_train is (60000*784)

# Scale the dataset (MinMaxScaler)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
#-------------------------------------------------------
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
x_train = x_train.reshape(-1, 28*28).astype(float)
x_test = x_test.reshape(-1, 28*28).astype(float)
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
#-------------------------------------------------------
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#-------------------------------------------------------------------------------------------
model.fit(x_train, y_train,batch_size=32, epochs=3)
#-------------------------------------------------------------------------------------------
model.predict(x_test)
val_loss, val_acc = model.evaluate(x_test, y_test)
#-------------------------------------------------------------------------------------------
print(val_loss) #0.08954
print(val_acc)  #0.97409
```

Scaling

# Scale the dataset (tensor flow(normalize))

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
#-----------------------------------------------------
x_train = tf.keras.utils.normalize(x_train,axis=1)
x_test = tf.keras.utils.normalize(x_test,axis=1)
#-----------------------------------------------------
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#-----------------------------------------------------------------------------------------
model.fit(x_train, y_train, batch_size=32,epochs=3)
#-----------------------------------------------------------------------------------------
model.predict(x_test)
val_loss, val_acc = model.evaluate(x_test, y_test)
#-----------------------------------------------------------------------------------------
print(val_loss) #0.1142
print(val_acc)  #0.9661
```

MinMaxScaler is more suitable for your dataset and the neural network model you are using.

# Try to increase the accuracy by increasing the number of hidden layers

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#-------------------------------------------------------------------------------
model.fit(x_train, y_train, batch_size=32,epochs=3)
#-------------------------------------------------------------------------------
model.predict(x_test)
val_loss, val_acc = model.evaluate(x_test, y_test)
#-------------------------------------------------------------------------------
print(val_loss) #0.074
print(val_acc)  #0.9785
```

# Second dataset: fashion_mnist

- The Fashion-MNIST dataset is a popular benchmark dataset used in machine learning and computer vision research. It is similar in structure to the classic MNIST dataset but contains images of fashion items instead of handwritten digits.

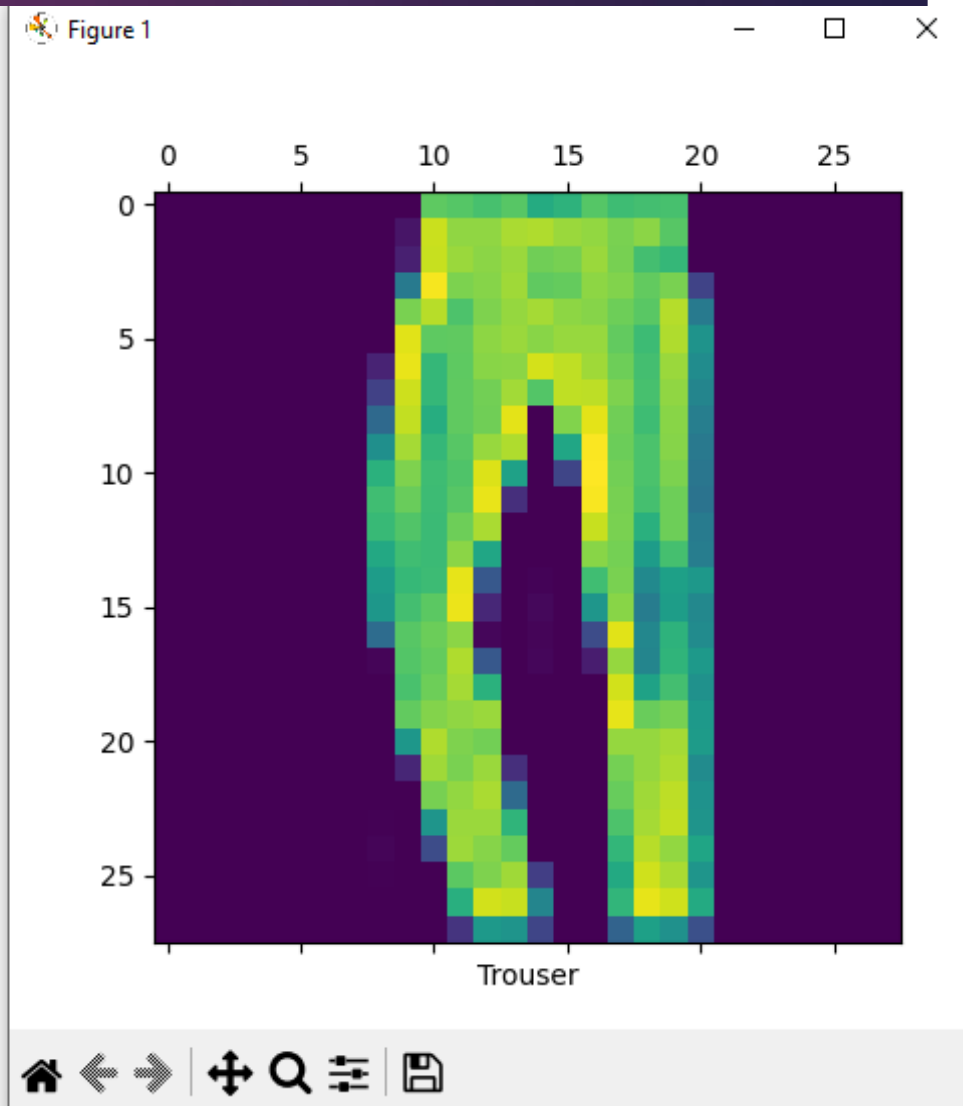
- Here are some key details about the Fashion-MNIST dataset:
  - **Contents**: The dataset consists of 60,000 training images and 10,000 test images.
  - **Image Size**: Each image is grayscale and has a size of 28x28 pixels, similar to the MNIST dataset.
  - **Classes**: There are 10 classes of fashion items in the dataset, each represented by a unique label:

    0: T-shirt/top, 1: Trouser, 2: Pullover, 3: Dress, 4: Coat, 5: Sandal, 6: Shirt, 7: Sneaker, 8: Bag, 9: Ankle boot
  - Fashion-MNIST provides a more challenging classification task compared to MNIST due to the greater variability and complexity of fashion item images. It serves as a useful benchmark for evaluating the performance of machine learning models, especially in the context of image classification and deep learning.

# Second dataset: fashion_mnist

```python
import tensorflow as tf
fashion = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = fashion.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(x_train.shape) # (60000, 28, 28)
print(x_train[5])
print(y_train.shape) # (60000,)
print(y_train[5]) # 2 (class number)
print(x_test.shape) # (10000, 28, 28)
print(x_test[5])
print(y_test.shape) #(10000,)
print(y_test[5]) # 1 (class number)

import matplotlib.pyplot as plt
plt.matshow(x_test[5])
plt.xlabel(class_names[y_test[5]])
plt.show()
#-----------------------------------------------------------
```



# Neural Network For Fashion_mnist

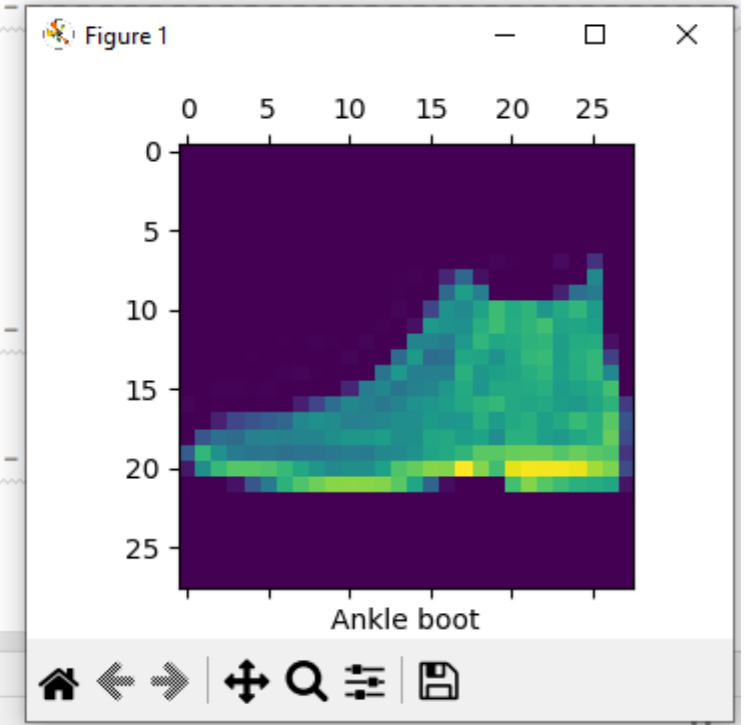
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#------------------------------------------------------------------------------
model.fit(x_train, y_train, batch_size=32,epochs=3)
#------------------------------------------------------------------------------
val_loss, val_acc = model.evaluate(x_test, y_test)
#------------------------------------------------------------------------------
print(val_loss) #0.6483
print(val_acc)  #0.7419
```

# Scale the dataset

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
x_train = x_train.reshape(-1, 28*28).astype(float)
x_test = x_test.reshape(-1, 28*28).astype(float)
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
#-------------------------------------------------------
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#-------------------------------------------------------------------------------------------
model.fit(x_train, y_train, batch_size=32,epochs=3)
#-------------------------------------------------------------------------------------------
```

# Evaluate the dataset

```
import numpy as np
y_predict=model.predict(x_test) # x is the predictions of x_test, x=y_predict(neural)
print(y_predict.shape) # print the dimension of y_predict ((10000, 10))
print(np.round(y_predict[0]).astype(int)) # print the output (prediction of the first image in the
# [0 0 0 0 0 0 0 0 0 1]
#-----------------------------------------------------------------
# to confirm the result we can display the first image
plt.matshow(x_test[0].reshape(28,28))
plt.xlabel(class_names[y_test[0]])
plt.show()
#-----------------------------------------------------------------
val_loss, val_acc = model.evaluate(x_test, y_test)
#-----------------------------------------------------------------
print(val_loss) #0.3924
print(val_acc)  #0.8560
```



# Classification of dogs and cats project (PetImages dataset)

- https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765
- **PetImages Dataset (12500 dogs and 12500 cats)**
- **First we import the following libraries:**

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import os # to interact with the file system in Python
import cv2 # OpenCV lib
from tqdm import tqdm
'''
- tqdm adds progress bars to your Python loops or tasks
  making it easier to monitor the progress of your code.
- tqdum in Arabic, which means "progress" or "progression.
'''
```

# Read the images from the directory

```
DATADIR = 'PetImages'
CATEGORIES = ["Dog", "Cat"] # dog has index 0, cat has index 1
for category in CATEGORIES:  # dogs and cats
    path = os.path.join(DATADIR,category)  # create path to dogs and cats (PetImages\Dog)
    x=0
    for img in os.listdir(path):  # iterate over each image per dogs and cats
        x+=1
        img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)# imread convert to array
        # os.path.join(path,img): create path to images PetImages\Dog\0.jpg
        plt.imshow(img_array,cmap='gray')  # graph it
        plt.show()  # display!
        if x==3 :
            break
```

# The last image we displayed:

```
23  print(img_array)
24  print(img_array.shape)
25
```

```
test
2024-05-14 00:47:32.044931: I tensorflow/core/util/port.cc:113] oneDNN custom
2024-05-14 00:47:33.415712: I tensorflow/core/util/port.cc:113] oneDNN custom
[[ 29  27  23 ... 149 148 147]
 [ 33  32  29 ... 149 148 147]
 [ 35  38  39 ... 149 149 146]
 ...
 [143 142 136 ... 152 149 144]
 [141 141 138 ... 153 150 143]
 [144 143 142 ... 153 151 144]]
(500, 489)
```

If you read images in RGB not Grayscale the dimension will be (500, 489, 3)

# Try to Resize one image:

- The primary challenge with the PetImages dataset lies in the inconsistency of image sizes.
- First we will try only with one image (last image we displayed (img_array)):

```
IMG_SIZE = 80


new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```

# Resizing the dataset:

```
dataset = [
    [image_data_1, class_label_1],
    [image_data_2, class_label_2],
    ...
]
```

```python
dataset = []
def create_dataset():
    for category in CATEGORIES:  # dogs and cats

        path = os.path.join(DATADIR,category)  # create path to dogs and cats
        class_num = CATEGORIES.index(category)  # get the classification  (0 or a 1). 0=dog 1=cat
        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                dataset.append([new_array, class_num])  # add (image,class label) to our training_data
            except Exception as e:
                pass
'''try-except block is used to handle exceptions that may occur during image reading or resizing.
       Any exceptions encountered are simply ignored (with pass), allowing the function to continue
       processing the remaining images.'''
create_dataset()
print(len(dataset))
```

# Shuffle the dataset

```
import random
random.shuffle(dataset)


#print the class labels of the first and fifth images,
#each element in training_data is list [image, class label]
print(dataset[0][1])
print(dataset[5][1])
```

# Reshaping the dataset

```
X = []
y = []

for features,label in dataset:
    X.append(features)
    y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE) #999,80,80
y=np.array(y) # 999,1
```

# Finally: create the neural network

```
from sklearn.model_selection import train_test_split
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(1, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
#------------------------------------------------------------------------------
model.fit(X_train, y_train, batch_size=32,epochs=3)
#------------------------------------------------------------------------------
val_loss, val_acc = model.evaluate(X_test,y_test)
#------------------------------------------------------------------------------
print(val_loss) #8.927
print(val_acc)  #0.4399
```

# Convolution neural network (CNN) with petimages dataset

```
from sklearn.model_selection import train_test_split
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(200, kernel_size=(3, 3), activation=tf.nn.relu, input_shape=(80, 80,1)))
model.add(tf.keras.layers.Conv2D(150, kernel_size=(3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(4, 4)))
model.add(tf.keras.layers.Conv2D(200, kernel_size=(3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D(150, kernel_size=(3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(4, 4)))
#-----------------------------------------------------------------------------------
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(1, activation=tf.nn.softmax))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
#-----------------------------------------------------------------------------------
model.fit(X_train, y_train, batch_size=32,epochs=3)
#-----------------------------------------------------------------------------------
val_loss, val_acc = model.evaluate(X_test,y_test)
#-----------------------------------------------------------------------------------
print(val_loss) #7.891
print(val_acc)  #0.5049
```

200 filters will have a size of 3x3,