**Practice Keras with code examples for Timeseries, reinforcement learning, Support Vector Machine(SVM) etc.**

**What is Keras?**

Keras is a neural network library in python that generally uses TensorFlow, Microsoft CNTK or Theano as its backend.

* Keras AI offers a high-level and user-friendly interface for developing and refining deep learning models.
* Additionally, Keras provides an extensive library of pre-built layers, activation functions, and optimisation methods, greatly streamlining the process of building sophisticated neural networks.

You can install Keras and its backend (preferably TensorFlow) from PyPI as:

pip install Keras

pip install tensorFlow

Keras Features

Let us see some of the top features of Keras that make it worth learning:

1. Prelabeled Datasets

* Keras provides a ton of prelabeled datasets that you can directly import and load.  
  **Example:** CIFAR10 small image classification, IMDB movie review sentiment classification, Reuters newswire topics classification, MNIST handwritten digit dataset, and few others (these are the examples of some famous datasets that are available in Keras)
* To import and load this MNIST dataset (a dataset):

from Keras.datasets import mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

2. Numerous implemented layers and parameters

Keras contains numerous implemented layers and parameters like loss functions, optimizers, evaluations metric.

You can use these layers and parameters for construction, configuration, training, and evaluation of neural networks.

* You would load the required layers to build your digit classifier.

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Flatten

from keras.optimizers import Adam

from keras.utils import np\_utils

Keras also has support for 1D and 2D convolutions and recurrent neural nets and for our digit classifier, you would use Convolution neural nets(Conv2D layer).

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

3. Multiple methods for Data Preprocessing

Keras also has a ton of methods for data preprocessing, here you would use Keras.np\_utils.to\_categorical() method for one-hot encoding of y\_train and y\_test.

one-hot encoding in AI keras:

One-hot encoding is a technique used to convert categorical variables into a binary vector representation for machine learning models, including those implemented with Keras. In Keras, you can use the **to\_categorical** function from the **tensorflow.keras.utils** module to perform one-hot encoding.

import numpy as np

from tensorflow.keras.utils import to\_categorical

# Example categorical labels

labels = ['cat', 'dog', 'bird', 'dog', 'cat']

# Perform one-hot encoding

encoded\_labels = to\_categorical(labels)

print(encoded\_labels)

Before that, reshape and normalize the dataset for your requirements.

#reshape in form of (60000, 28, 28, 1)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

#normalize to get data in range of 0-1

X\_train/=255

X\_test/=255

number\_of\_classes = 10

y\_train = np\_utils.to\_categorical(y\_train, number\_of\_classes)

y\_test = np\_utils.to\_categorical(y\_test, number\_of\_classes)

4. .add() Method in Keras

To add layers imported above by specifying parameters to build your digit classifier, it is done using .add() method.

model = Sequential()

model.add(Conv2D(32, (5, 5), input\_shape=(X\_train.shape[1], X\_train.shape[2], 1), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(number\_of\_classes, activation='softmax'))

5. .compile() Method in Keras

Before training, you need to configure your learning process which is done using .compile() method.

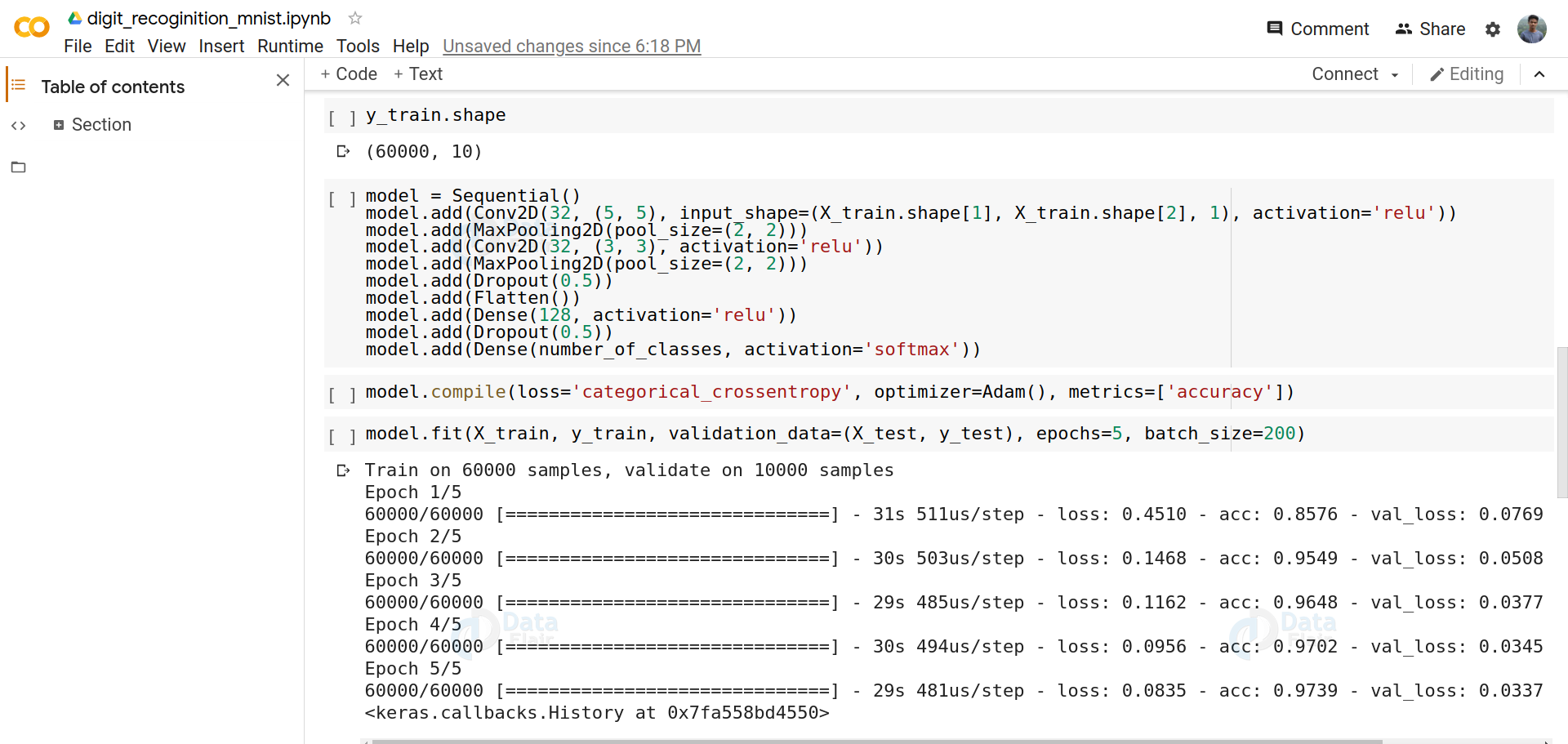
model.compile(loss='categorical\_crossentropy', optimizer=Adam(), metrics=['accuracy'])

6. .fit() method

You can train Keras models on numpy arrays using .fit().

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=5, batch\_size=200)

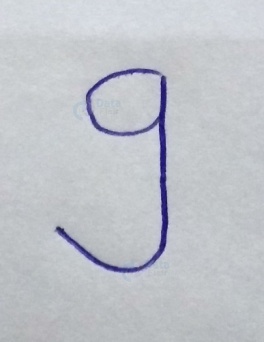
The training may take some time, here I have used only 5 epochs but you can increase the epoch count as per your systems.



#### 7. Model Evaluation

After training your model, you need to test your results on unseen data or you can evaluate your model using .predict\_classes() or .evaluate().

You can test your model on your own handwritten digits. I tested it on the following handwritten digit.



Sample Test Data.

But before giving it as the input, you need to convert it in the form of MNIST dataset digits.

MNIST dataset digits are grayscale images of (28\*28\*1) dimensions.

import cv2

img = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

# resize image

resized = cv2.resize(img, (28,28), interpolation = cv2.INTER\_AREA)

img = np.resize(resized, (28,28,1))

im2arr = np.array(img)

im2arr = im2arr.reshape(1,28,28,1)

y\_pred = model.predict\_classes(im2arr)

print(y\_pred)

**Output:**

9

8. Modularity

As discussed above, Keras is modular. You can save the model you train and use this model later by loading it.

This is done as:

model.save('model.h5')

Keras with Timeseries:

For timeseries data, we will use a simple LSTM (Long Short-Term Memory) network to predict the next value in the timeseries sequence.

import numpy as np

from keras.models import Sequential

from keras.layers import LSTM, Dense

# Generate synthetic timeseries data

def generate\_timeseries\_data(n\_steps):

time = np.linspace(0, 2\*np.pi, n\_steps)

data = np.sin(time) + np.random.normal(0, 0.1, n\_steps)

return data

n\_steps = 100

data = generate\_timeseries\_data(n\_steps)

# Prepare data for LSTM (input and target)

X = data[:-1]

y = data[1:]

# Reshape data for LSTM (samples, timesteps, features)

X = X.reshape(-1, 1, 1)

# Create LSTM model

model = Sequential()

model.add(LSTM(32, input\_shape=(1, 1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

# Train the model

model.fit(X, y, epochs=100, batch\_size=1)

Keras with reinforcement learning:

For reinforcement learning, we'll use a simple Q-learning algorithm to solve the classic CartPole environment from OpenAI Gym.

import gym

import numpy as np

from keras.models import Sequential

from keras.layers import Dense

# Create the CartPole environment

env = gym.make('CartPole-v1')

# Q-learning algorithm

def q\_learning(env, episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):

n\_actions = env.action\_space.n

n\_states = env.observation\_space.shape[0]

q\_table = np.zeros((n\_states, n\_actions))

for episode in range(episodes):

state = env.reset()

done = False

while not done:

if np.random.rand() < epsilon:

action = env.action\_space.sample()

else:

action = np.argmax(q\_table[state, :])

next\_state, reward, done, \_ = env.step(action)

q\_next = np.max(q\_table[next\_state, :])

q\_table[state, action] = (1 - alpha) \* q\_table[state, action] + alpha \* (reward + gamma \* q\_next)

state = next\_state

return q\_table

# Train Q-learning agent

q\_table = q\_learning(env)

# Test the agent

state = env.reset()

done = False

while not done:

action = np.argmax(q\_table[state, :])

next\_state, \_, done, \_ = env.step(action)

env.render()

state = next\_state

env.close()

Keras with Support Vector Machine(SVM):

For SVM, we will use the classic Iris dataset and the **SVC** class from scikit-learn, which integrates well with Keras.

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from keras.models import Sequential

from keras.layers import Dense

# Load Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the 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)

# Create the SVM model using SVC from scikit-learn

svm\_model = SVC(kernel='linear')

svm\_model.fit(X\_train, y\_train)

# Evaluate the SVM model

accuracy = svm\_model.score(X\_test, y\_test)

print("SVM accuracy:", accuracy)

# Alternatively, you can use Keras to build an SVM-like model

model = Sequential()

model.add(Dense(1, input\_dim=4, activation='linear'))

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

# Train the SVM-like model

model.fit(X\_train, y\_train, epochs=100, batch\_size=32)

# Evaluate the SVM-like model

\_, accuracy = model.evaluate(X\_test, y\_test)

print("SVM-like model accuracy:", accuracy)

**Loading Data, defining Training models and predict results in Keras**

We'll use the classic Iris dataset for a multiclass classification task.

1. **Load Data**:

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the 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)

1. **Preprocess Data (optional)**:

For this example, we'll skip preprocessing since the Iris dataset is already clean and well-formatted. However, in real-world scenarios, you might need to perform data preprocessing, such as scaling, normalization, handling missing values, etc.

1. **Define the Model**:

We'll create a simple Multilayer Perceptron (MLP) model using Keras.

from keras.models import Sequential

from keras.layers import Dense

# Create the MLP model

model = Sequential()

model.add(Dense(10, activation='relu', input\_shape=(X\_train.shape[1],)))

model.add(Dense(3, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

**Train the Model**:

# Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.1)

**Evaluate the Model**:

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy)

**Make Predictions**:

# Make predictions on new data

new\_data = [[5.1, 3.5, 1.4, 0.2], [6.4, 3.2, 4.5, 1.5], [7.3, 2.9, 6.3, 1.8]]

predictions = model.predict(new\_data)

# Convert predictions to class labels

predicted\_classes = [iris.target\_names[pred.argmax()] for pred in predictions]

print("Predicted Classes:", predicted\_classes)