```python
import tensorflow as tf

# Display the version
print(tf.__version__)

# other imports
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
```

```
2.19.0
```

```python
# Load in the data
cifar10 = tf.keras.datasets.cifar10

# Distribute it to train and test set
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

```python
# Reduce pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()
```

```python
# visualize data by plotting images
fig, ax = plt.subplots(5, 5)
k = 0

for i in range(5):
    for j in range(5):
        ax[i][j].imshow(x_train[k], aspect='auto')
        k += 1

plt.show()
```

```
# number of classes
K = len(set(y_train))

# calculate total number of classes
# for output layer
print("number of classes:", K)

# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)

# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
```

```
number of classes: 10
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 1024)	2,098,176
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10,250

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

```
# Fit
r = model.fit(
  x_train, y_train, validation_data=(x_test, y_test), epochs=5)
```

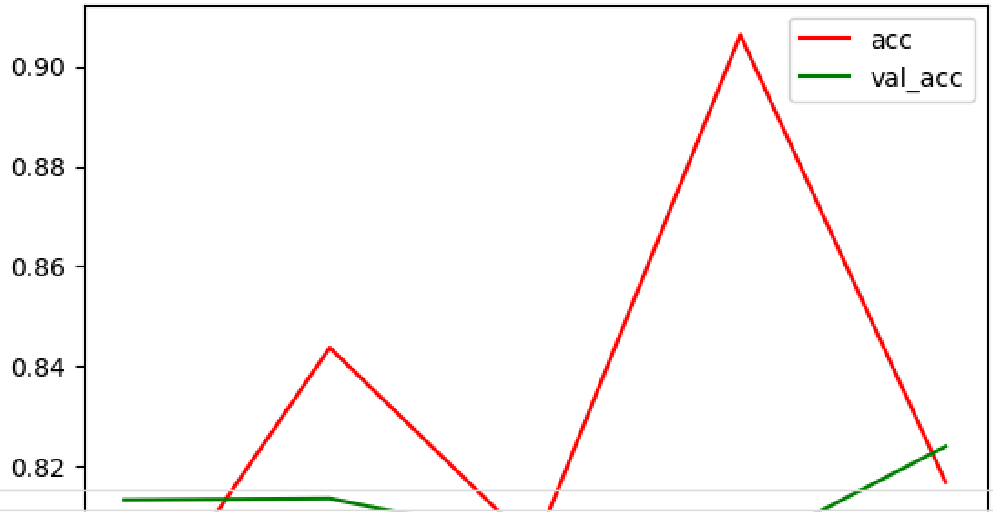
```
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 411s 263ms/step - accuracy: 0.7802 - loss: 0.6436 - val_accur
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 409s 261ms/step - accuracy: 0.8156 - loss: 0.5345 - val_accur
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 410s 262ms/step - accuracy: 0.8424 - loss: 0.4505 - val_accur
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 443s 263ms/step - accuracy: 0.8695 - loss: 0.3730 - val_accur
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 442s 263ms/step - accuracy: 0.8910 - loss: 0.3133 - val_accur
```

```
# Fit with data augmentation
# Note: if you run this AFTER calling
# the previous model.fit()
# it will CONTINUE training where it left off
batch_size = 32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(
  width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)

train_generator = data_generator.flow(x_train, y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size

r = model.fit(train_generator, validation_data=(x_test, y_test),
              steps_per_epoch=steps_per_epoch, epochs=5)
```

```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.
  self._warn_if_super_not_called()
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 454s 291ms/step - accuracy: 0.7795 - loss: 0.6734 - val_accur
Epoch 2/5
   1/1562 ━━━━━━━━━━━━━━━━━━━━ 5:50 225ms/step - accuracy: 0.8438 - loss: 0.4647/usr/local/l
  self._interrupted_warning()
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.8438 - loss: 0.4647 - val_accurac
Epoch 3/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 431s 276ms/step - accuracy: 0.8067 - loss: 0.5765 - val_accur
Epoch 4/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.9062 - loss: 0.3415 - val_accurac
Epoch 5/5
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 428s 274ms/step - accuracy: 0.8170 - loss: 0.5398 - val_accur
```

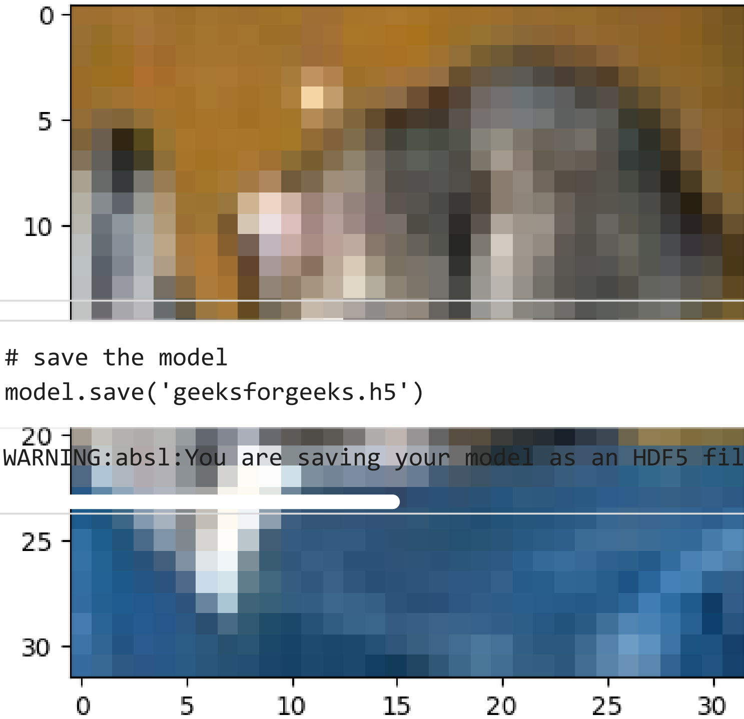
```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x789f01ccf890>
```




```
# label mapping

labels = '''airplane automobile bird cat deerdog frog horseship truck'''.split()

# select the image from our test dataset
image_number = 0

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape it
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print("Original label is {} and predicted label is {}".format(
    original_label, predicted_label))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
Original label is cat and predicted label is frog
```



```
# save the model
model.save('geeksforgeeks.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.s
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
import seaborn as sns
import os
from datetime import datetime

import warnings
warnings.filterwarnings("ignore")
```

```
import pandas as pd

data = pd.read_csv('/content/all_stocks_5yr.csv', delimiter=',', on_bad_lines='skip')
print(data.shape)
print(data.sample(7))
```

```
(152910, 7)
              date    open    high       low   close   volume  Name
50493   2014-06-02  108.56  108.95  107.8100  108.35  1270180  ANTM
74656   2015-03-16   67.76   68.48   67.7500   68.43  2966222   BAX
149681  2016-09-22   27.08   27.72   26.9500   27.61  5094979   CTL
72524   2016-09-23   46.43   46.87   46.3100   46.61  1957032     A
27624   2017-10-23  100.04  100.04   99.1200   99.49   463664   AIZ
42411   2017-04-26  165.61  165.61  164.3397  164.61  4175536  AMGN
76393   2017-02-06  162.42  164.08  162.3800  163.98  3110494    BA
```

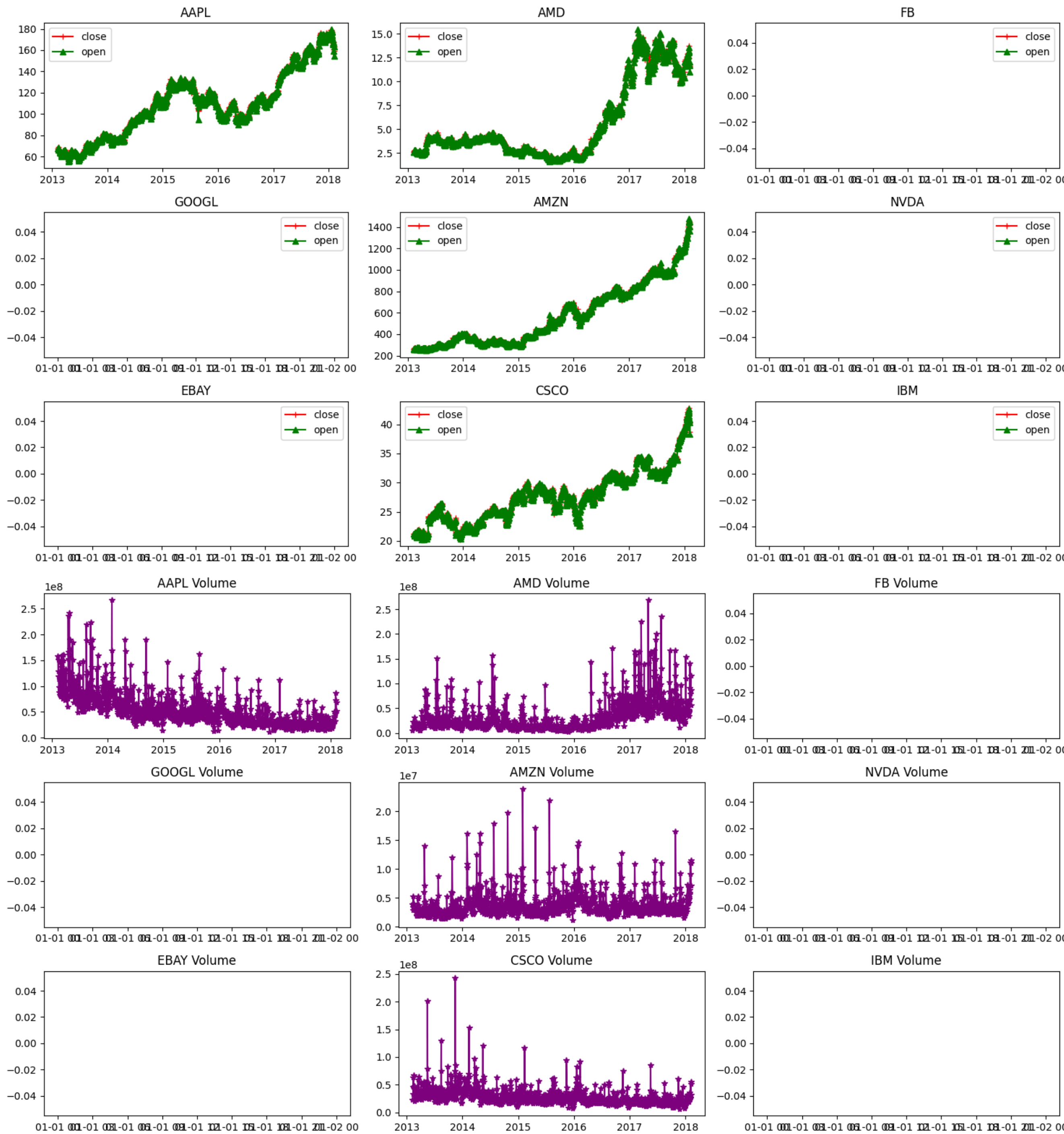
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152910 entries, 0 to 152909
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   date    152910 non-null  object 
 1   open    152909 non-null  float64
 2   high    152909 non-null  float64
 3   low     152909 non-null  float64
 4   close   152910 non-null  float64
 5   volume  152910 non-null  int64  
 6   Name    152909 non-null  object 
dtypes: float64(4), int64(1), object(2)
memory usage: 8.2+ MB
```

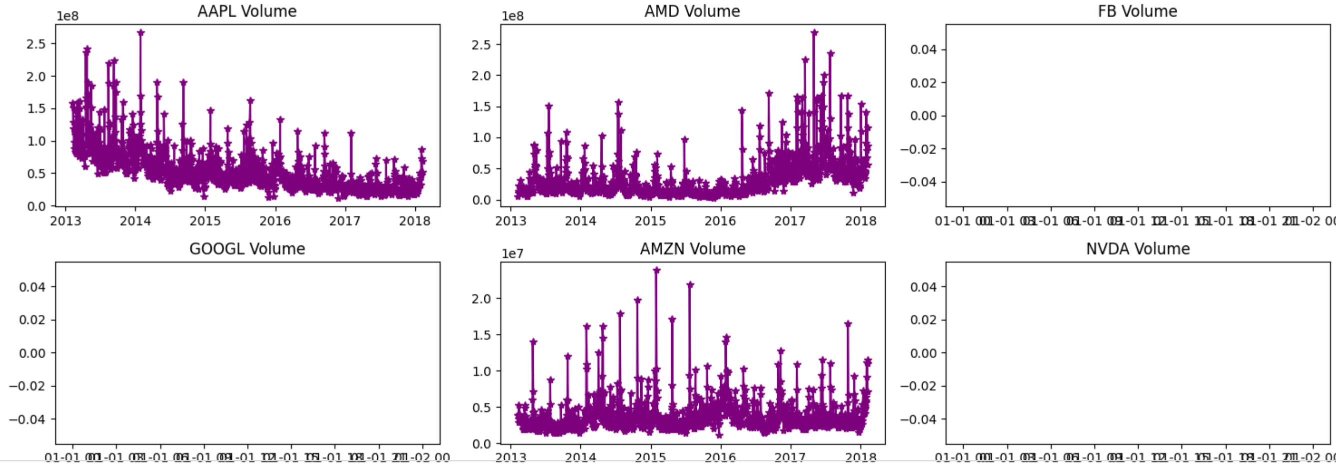
```
data['date'] = pd.to_datetime(data['date'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152910 entries, 0 to 152909
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype         
---  ------  --------------   -----         
 0   date    152910 non-null  datetime64[ns]
 1   open    152909 non-null  float64       
 2   high    152909 non-null  float64       
 3   low     152909 non-null  float64       
 4   close   152910 non-null  float64       
 5   volume  152910 non-null  int64         
 6   Name    152909 non-null  object        
dtypes: datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 8.2+ MB
```

```
companies = ['AAPL', 'AMD', 'FB', 'GOOGL', 'AMZN', 'NVDA', 'EBAY', 'CSCO', 'IBM']

plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['close'], c="r", label="close", marker="+")
    plt.plot(c['date'], c['open'], c="g", label="open", marker="^")
    plt.title(company)
    plt.legend()
    plt.tight_layout()

plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['volume'], c='purple', marker='*')
    plt.title(f"{company} Volume")
    plt.tight_layout()
```




```
plt.figure(figsize=(15, 8))
for index, company in enumerate(companies, 1):
    plt.subplot(3, 3, index)
    c = data[data['Name'] == company]
    plt.plot(c['date'], c['volume'], c='purple', marker='*')
    plt.title(f"{company} Volume")
    plt.tight_layout()
```




```
apple = data[data['Name'] == 'AAPL']
prediction_range = apple.loc[(apple['date'] > datetime(2013,1,1))
 & (apple['date']<datetime(2018,1,1))]
plt.plot(apple['date'],apple['close'])
plt.xlabel("Date")
plt.ylabel("Close")
plt.title("Apple Stock Prices")
plt.show()
```


```
close_data = apple.filter(['close'])
dataset = close_data.values
training = int(np.ceil(len(dataset) * .95))
print(training)
```

```
1197
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

train_data = scaled_data[0:int(training), :]
# prepare feature and labels
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

```
model = keras.models.Sequential()
model.add(keras.layers.LSTM(units=64,
                            return_sequences=True
```

```
                            return_sequences=True,
                            input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(units=64))
model.add(keras.layers.Dense(32))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(1))
model.summary
```

```
keras.src.models.model.Model.summary
def summary(line_length=None, positions=None, print_fn=None, expand_nested=False,
show_trainable=False, layer_range=None)
```

/usr/local/lib/python3.12/dist-packages/keras/src/models/model.py

```
Prints a string summary of the network.

Args:
    line_length: Total length of printed lines
        (e.g. set this to adapt the display to different
```

```
model.compile(optimizer='adam',
              loss='mean_squared_error')
history = model.fit(x_train,
                    y_train,
                    epochs=10)
```

```
Epoch 1/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - loss: 0.0533
Epoch 2/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 0.0087
Epoch 3/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 92ms/step - loss: 0.0090
Epoch 4/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - loss: 0.0090
Epoch 5/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - loss: 0.0083
Epoch 6/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - loss: 0.0075
Epoch 7/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step - loss: 0.0071
Epoch 8/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - loss: 0.0065
Epoch 9/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0061
Epoch 10/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 0.0066
```

```
test_data = scaled_data[training - 60:, :]
x_test = []
y_test = dataset[training:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

mse = np.mean(((predictions - y_test) ** 2))
rmse = np.sqrt(mse)

print("MSE", mse)
print("RMSE", np.sqrt(mse))
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 425ms/step
MSE 45.63328004905825
RMSE 6.755240931977056
```

```
train = apple[:training]
test = apple[training:]
test['Predictions'] = predictions

plt.figure(figsize=(10, 8))
plt.plot(train['date'], train['close'])
plt.plot(test['date'], test[['close', 'Predictions']])
plt.title('Apple Stock Close Price')
plt.xlabel('Date')
plt.ylabel("Close")
plt.legend(['Train', 'Test', 'Predictions'])
```

```
<matplotlib.legend.Legend at 0x7dc867a727e0>
```