load the saved model and plot the training loss:

import tensorflow as tf

import matplotlib.pyplot as plt

# Load the saved model from an .h5 file

model = tf.keras.models.load\_model('model.h5')

# Access the training history and get the loss values

loss = model.history.history['loss']

# Plot the loss values

plt.plot(loss)

plt.title('Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.show()

parameter display

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

# Load the dataset

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

# Normalize the data and convert to float32

X\_train = X\_train.astype(np.float32) / 255.0

X\_test = X\_test.astype(np.float32) / 255.0

# Reshape the data

X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1))

X\_test = X\_test.reshape((X\_test.shape[0], 28, 28, 1))

# Define the model architecture

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28,28,1)),

tf.keras.layers.MaxPooling2D((2,2)),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(100, activation='relu'),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model

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

# Define the callbacks to save the model and plot the metrics

checkpoint = tf.keras.callbacks.ModelCheckpoint("model.h5", save\_weights\_only=False, save\_best\_only=True, monitor='val\_loss', mode='min', verbose=1)

history = tf.keras.callbacks.History()

callbacks\_list = [checkpoint, history]

# Train the model

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, callbacks=callbacks\_list)

# Plot the metrics

fig, axs = plt.subplots(2, 3, figsize=(12, 8))

axs[0, 0].plot(history.history['loss'], label='Training Loss')

axs[0, 0].plot(history.history['val\_loss'], label='Validation Loss')

axs[0, 0].set\_xlabel('Epoch')

axs[0, 0].set\_ylabel('Loss')

axs[0, 0].legend()

axs[0, 1].plot(history.history['accuracy'], label='Training Accuracy')

axs[0, 1].plot(history.history['val\_accuracy'], label='Validation Accuracy')

axs[0, 1].set\_xlabel('Epoch')

axs[0, 1].set\_ylabel('Accuracy')

axs[0, 1].legend()

axs[0, 2].plot(history.history['precision'], label='Training Precision')

axs[0, 2].plot(history.history['val\_precision'], label='Validation Precision')

axs[0, 2].set\_xlabel('Epoch')

axs[0, 2].set\_ylabel('Precision')

axs[0, 2].legend()

axs[1, 0].plot(history.history['recall'], label='Training Recall')

axs[1, 0].plot(history.history['val\_recall'], label='Validation Recall')

axs[1, 0].set\_xlabel('Epoch')

axs[1, 0].set\_ylabel('Recall')

axs[1, 0].legend()

axs[1, 1].plot(history.history['f1\_score'], label='Training F1 Score')

axs[1, 1].plot(history.history['val\_f1\_score'], label='Validation F1 Score')

axs[1, 1].set\_xlabel('Epoch')

axs[1, 1].set\_ylabel('F1 Score

axs[1, 1].legend() axs[1, 2].plot(history.history['auc'], label='Training AUC') axs[1, 2].plot(history.history['val\_auc'], label='Validation AUC') axs[1, 2].set\_xlabel('Epoch') axs[1, 2].set\_ylabel('AUC') axs[1, 2].legend() plt.show()

python code to plot all six parameters you mention above using .h5 file

import h5py

import numpy as np

import matplotlib.pyplot as plt

# Load the model from the .h5 file

with h5py.File('model.h5', 'r') as f:

model\_weights = []

for i in range(f.attrs['nb\_layers']):

model\_weights.append(f['layer\_{}'.format(i)].attrs['weight'])

model = tf.keras.models.load\_model('model.h5')

# Load the training and validation metrics from the history object

history = model.history.history

# Plot the metrics

fig, axs = plt.subplots(2, 3, figsize=(12, 8))

axs[0, 0].plot(history['loss'], label='Training Loss')

axs[0, 0].plot(history['val\_loss'], label='Validation Loss')

axs[0, 0].set\_xlabel('Epoch')

axs[0, 0].set\_ylabel('Loss')

axs[0, 0].legend()

axs[0, 1].plot(history['accuracy'], label='Training Accuracy')

axs[0, 1].plot(history['val\_accuracy'], label='Validation Accuracy')

axs[0, 1].set\_xlabel('Epoch')

axs[0, 1].set\_ylabel('Accuracy')

axs[0, 1].legend()

axs[0, 2].plot(history['precision'], label='Training Precision')

axs[0, 2].plot(history['val\_precision'], label='Validation Precision')

axs[0, 2].set\_xlabel('Epoch')

axs[0, 2].set\_ylabel('Precision')

axs[0, 2].legend()

axs[1, 0].plot(history['recall'], label='Training Recall')

axs[1, 0].plot(history['val\_recall'], label='Validation Recall')

axs[1, 0].set\_xlabel('Epoch')

axs[1, 0].set\_ylabel('Recall')

axs[1, 0].legend()

axs[1, 1].plot(history['f1\_score'], label='Training F1 Score')

axs[1, 1].plot(history['val\_f1\_score'], label='Validation F1 Score')

axs[1, 1].set\_xlabel('Epoch')

axs[1, 1].set\_ylabel('F1 Score')

axs[1, 1].legend()

axs[1, 2].plot(history['auc'], label='Training AUC')

axs[1, 2].plot(history['val\_auc'], label='Validation AUC')

axs[1, 2].set\_xlabel('Epoch')

axs[1, 2].set\_ylabel('AUC')

axs[1, 2].legend()

plt.show()

plot validation loss, learning rate, mean squared error, weight and bias distribution from .h5 file

import h5py

import matplotlib.pyplot as plt

import numpy as np

# Load the model from the .h5 file

with h5py.File('model.h5', 'r') as f:

model\_weights = []

for i in range(f.attrs['nb\_layers']):

model\_weights.append(f['layer\_{}'.format(i)].attrs['weight'])

model = tf.keras.models.load\_model('model.h5')

# Load the training and validation metrics from the history object

history = model.history.history

# Plot the validation loss

plt.plot(history['val\_loss'])

plt.title('Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.show()

# Plot the learning rate

try:

lrs = history['lr']

plt.plot(lrs)

plt.title('Learning Rate')

plt.xlabel('Epoch')

plt.ylabel('Learning Rate')

plt.show()

except:

print('Learning rate not found in the history object.')

# Plot the mean squared error

try:

mse = history['mse']

val\_mse = history['val\_mse']

plt.plot(mse)

plt.plot(val\_mse)

plt.title('Mean Squared Error')

plt.xlabel('Epoch')

plt.ylabel('MSE')

plt.legend(['Training MSE', 'Validation MSE'])

plt.show()

except:

print('Mean squared error not found in the history object.')

# Plot the weight and bias distributions

for layer in model.layers:

if len(layer.get\_weights()) > 0:

w = layer.get\_weights()[0]

b = layer.get\_weights()[1]

plt.hist(w.flatten(), bins=50)

plt.title(layer.name + ' Weights')

plt.show()

plt.hist(b.flatten(), bins=50)

plt.title(layer.name + ' Biases')

plt.show()