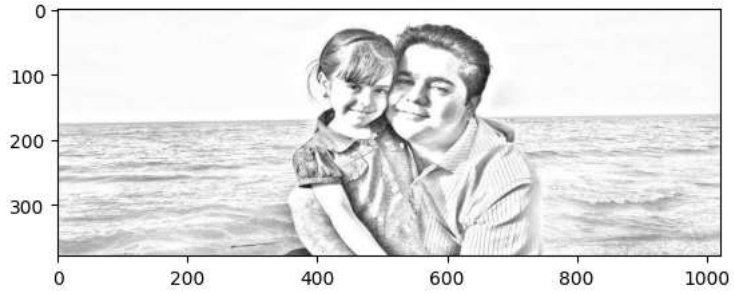
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow import keras
from tensorflow.keras import layers

# Load the image
data = np.array(plt.imread('/content/FB_IMG_1699980572276.jpg'))

# Display the image
plt.imshow(data)
plt.show()

# Get image dimensions
height, width, channels = data.shape
print(f"Image dimensions: height = {height}, width = {width}, channels = {channels}")

# Convert image to RGB format
data = cv2.cvtColor(data, cv2.COLOR_BGR2RGB)

# Resize image to 224x224
data_resized = cv2.resize(data, (224, 224))

# Define the model
model = keras.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, 3, activation="relu"),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, activation="relu"),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax")
])

# Compile the model
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.Adam(),
              metrics=["accuracy"])

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Preprocess the data
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# Reshape the data to add channel dimension
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Train the model
history = model.fit(x_train, y_train, batch_size=64, epochs=10, validation_split=0.2)

# Evaluate the model
test_scores = model.evaluate(x_test, y_test, verbose=2)
print("Test loss:", test_scores[0])
print("Test accuracy:", test_scores[1])

# Save the model
model.save("path_to_my_model")
```




```
Image dimensions: height = 379, width = 1022, channels = 3
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/10
750/750 [==============================] - 43s 56ms/step - loss: 0.1789 - accuracy: 0.9467 - val_loss: 0.0660 - val_accuracy: 0.9809
Epoch 2/10
750/750 [==============================] - 41s 55ms/step - loss: 0.0511 - accuracy: 0.9842 - val_loss: 0.0473 - val_accuracy: 0.9858
Epoch 3/10
750/750 [==============================] - 41s 54ms/step - loss: 0.0359 - accuracy: 0.9888 - val_loss: 0.0432 - val_accuracy: 0.9868
Epoch 4/10
750/750 [==============================] - 41s 54ms/step - loss: 0.0274 - accuracy: 0.9910 - val_loss: 0.0391 - val_accuracy: 0.9887
Epoch 5/10
750/750 [==============================] - 40s 54ms/step - loss: 0.0191 - accuracy: 0.9937 - val_loss: 0.0397 - val_accuracy: 0.9895
Epoch 6/10
750/750 [==============================] - 41s 54ms/step - loss: 0.0148 - accuracy: 0.9951 - val_loss: 0.0338 - val_accuracy: 0.9907
Epoch 7/10
750/750 [==============================] - 40s 54ms/step - loss: 0.0107 - accuracy: 0.9964 - val_loss: 0.0349 - val_accuracy: 0.9905
Epoch 8/10
750/750 [==============================] - 41s 55ms/step - loss: 0.0097 - accuracy: 0.9966 - val_loss: 0.0434 - val_accuracy: 0.9898
Epoch 9/10
750/750 [==============================] - 41s 55ms/step - loss: 0.0087 - accuracy: 0.9970 - val_loss: 0.0395 - val_accuracy: 0.9909
Epoch 10/10
750/750 [==============================] - 41s 55ms/step - loss: 0.0069 - accuracy: 0.9976 - val_loss: 0.0499 - val_accuracy: 0.9894
313/313 - 2s - loss: 0.0373 - accuracy: 0.9908 - 2s/epoch - 7ms/step
Test loss: 0.03725796565413475
Test accuracy: 0.9908000230789185
```