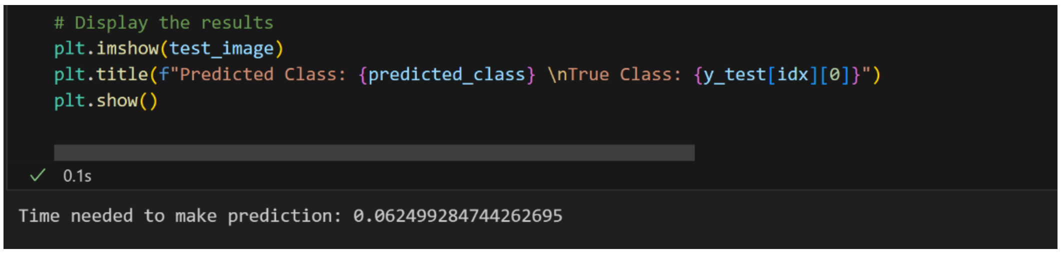
Discussion:



```
# Display the results
plt.imshow(test_image)
plt.title(f"Predicted Class: {predicted_class} \nTrue Class: {y_test[idx][0]}")
plt.show()
```

```
✓ 0.1s
```

```
Time needed to make prediction: 0.062499284744262695
```

Based on the screenshot, the latency of making a prediction using the model deployed with TensorFlow serving is approximately 62.4 milliseconds.

When serving machine learning models, latency refers to the round-trip time it takes to send a POST request with input data to the model API and receive a prediction. This time includes network transmission time, the time the server takes to process the request, and the time to send the response back. Here are a few considerations on latency when querying a model:

Model Complexity: It is obvious that there is large number of layers and filters in the model. Larger, more complex models will generally take longer to make predictions due to the greater number of computations required.

Network Speed: Because TensorFlow serving is remote server, the request and response times are affected by the network speed between the client and the server.

Batching: Batch size is another crucial factor may influence the latency. Larger batch sizes may improve overall efficiency and throughput but can lead to higher latency for individual predictions.

Optimization: Models that are not optimized for inference may have higher latency. To mitigate the latency, the structure and parameters of the model should be adjected. Techniques like quantization, pruning, and using efficient model architectures can reduce latency.

Concurrency: The number of concurrent requests the server is handling can also impact the response time for each individual request.

System Resources: The hardware on which the model is being served, including CPU, GPU, memory, can significantly impact latency.

Compare to Flask application vs TensorFlow Serving

When comparing Flask applications and TensorFlow Serving, we can explore the biggest differences between the two in terms of their design goals, performance, scalability, and specialized support for serving machine learning models.

In terms of specialization and purpose, Flask is a flexible general-purpose web framework for building a wide range of web applications. While it can be used for serving machine learning models, this is not its primary design purpose. Developers need to write additional code to load models, process requests, and return predictions. TensorFlow Serving, on the other hand, is designed to serve TensorFlow models, and it has built-in specialized features for machine learning model servicing such as model version control, hot updates, and efficient inference.

Regarding to performance and scalability, Flask provides great development flexibility, but by default, it has no special performance optimizations for high concurrency or machine learning model inference. While its performance can be improved through scaling and optimization, this can require quite a bit of work. TensorFlow Serving was designed from the start to be a high-performance model serving system, and by optimizing the model loading and inference process, as well as supporting features such as GPU acceleration, it is able to efficiently handle a large number of inference requests, making it particularly suited for large-scale deployments in production environments.

Therefore, the biggest difference lies in their respective areas of specialization and performance optimization. TensorFlow Serving provides an efficient and scalable solution designed specifically for TensorFlow model serving, including model versioning, efficient inference performance, and support for large-scale concurrent requests. Flask, on the other hand, offers a more general and flexible approach to web service development, allowing developers to implement customized service logic by writing Python code, but is less specialized than TensorFlow Serving in terms of performance optimization and specific features dedicated to machine learning model serving.