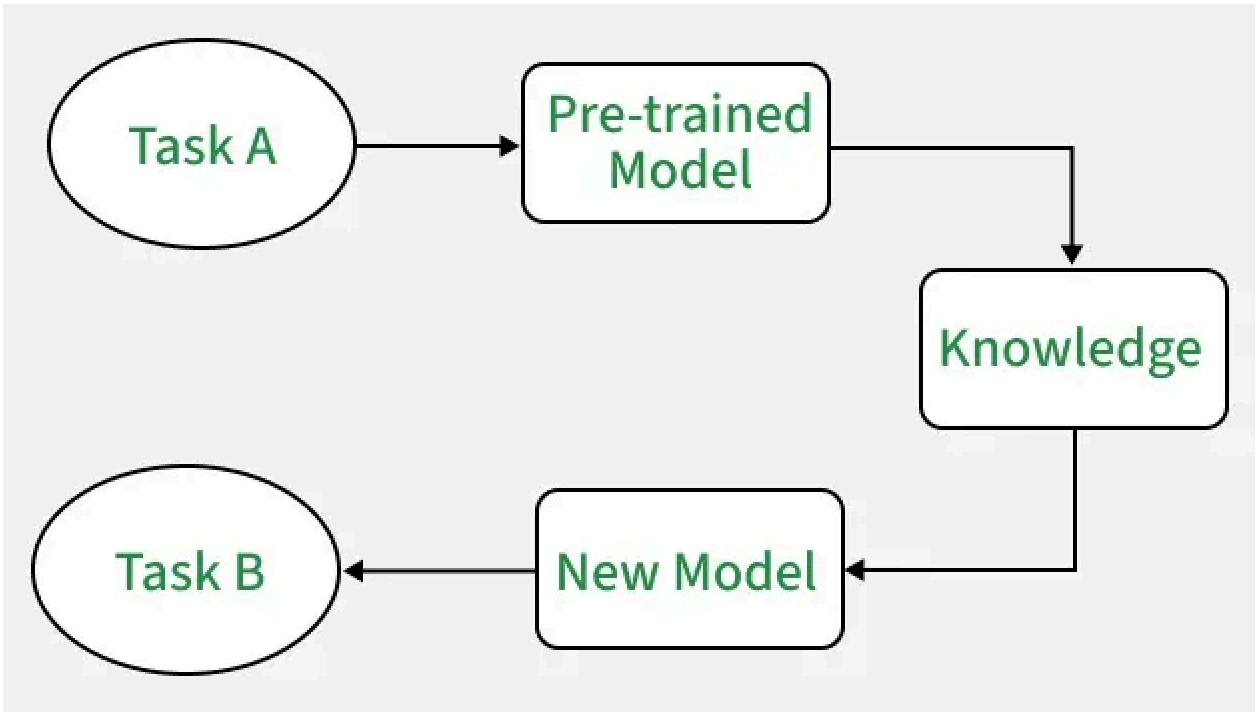
What is Transfer Learning?

geeksforgeeks.org/machine-learning/ml-introduction-to-transfer-learning

GeeksforGeeks

November 23, 2019



Transfer learning is a machine learning technique where a model trained on one task is repurposed as the foundation for a second task. This approach is beneficial when the second task is related to the first or when data for the second task is limited.

1 / 1

Using learned features from the initial task, the model can adapt more efficiently to the new task, accelerating learning and improving performance. Transfer learning also reduces the risk of overfitting, as the model already incorporates generalizable features useful for the second task.

Importance of Transfer Learning

Transfer learning offers solutions to key challenges like:

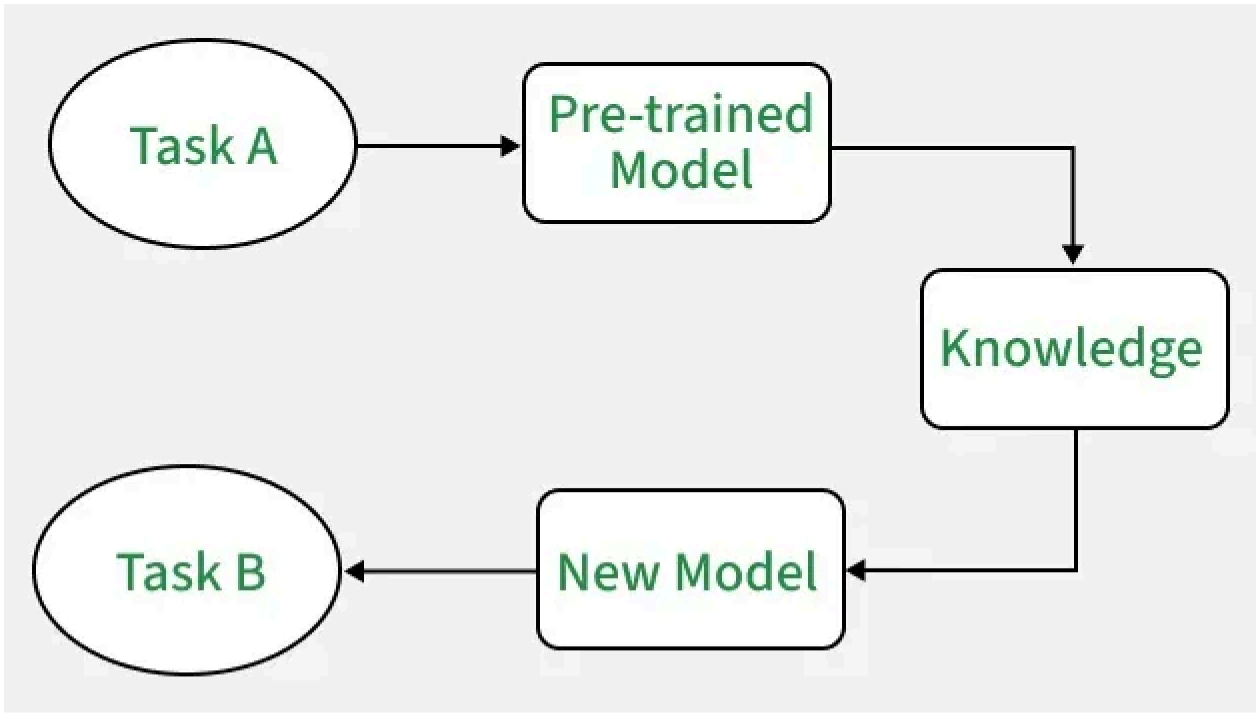
1. **Limited Data:** Acquiring extensive labelled data is often challenging and costly. Transfer learning enables us to use pre-trained models, reducing the dependency on large datasets.
2. **Enhanced Performance:** Starting with a pre-trained model which has already learned from substantial data allows for faster and more accurate results on new tasks ideal for applications needing high accuracy and efficiency.
3. **Time and Cost Efficiency:** Transfer learning shortens training time and conserves resources by utilizing existing models hence eliminating the need for training from scratch.

4. **Adaptability:** Models trained on one task can be fine-tuned for related tasks making transfer learning versatile for various applications from image recognition to natural language processing.

Working of Transfer Learning

Transfer learning involves a structured process to use existing knowledge from a pre-trained model for new tasks:

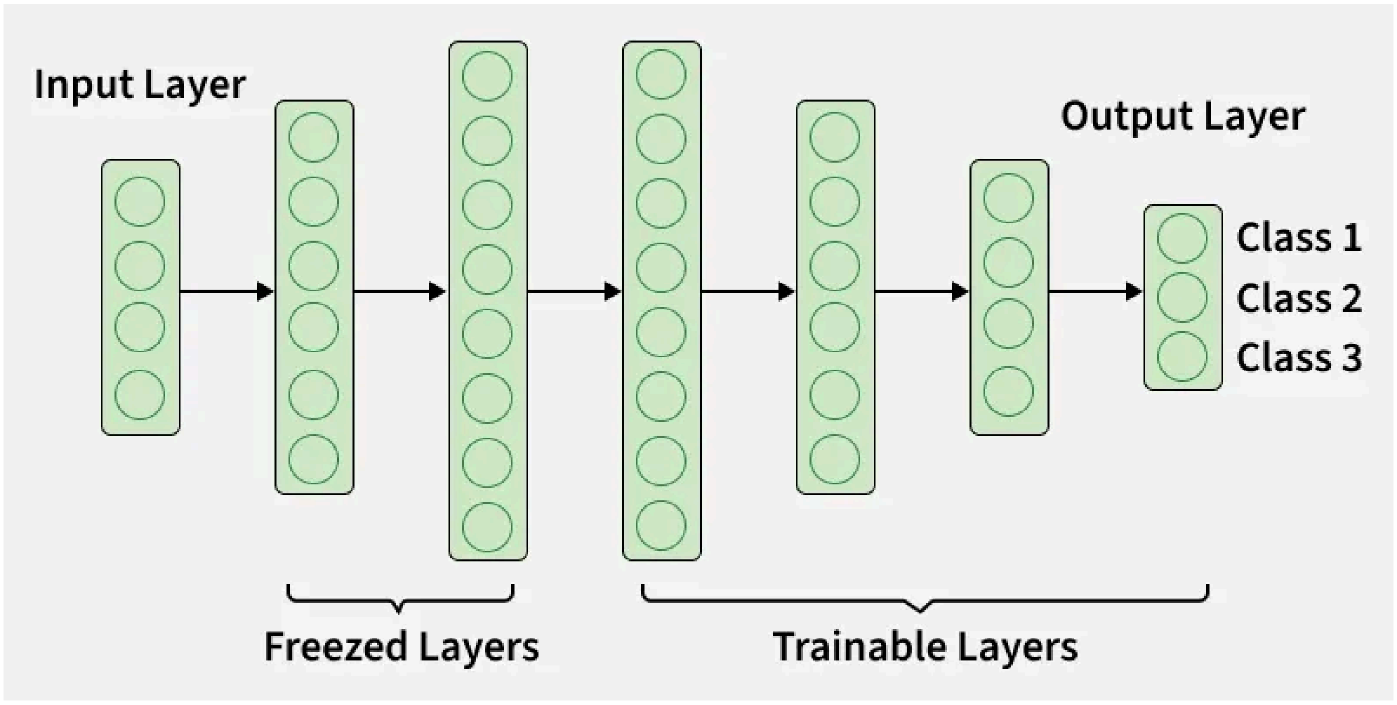
1. **Pre-trained Model:** Start with a model already trained on a large dataset for a specific task. This pre-trained model has learned general features and patterns that are relevant across related tasks.
2. **Base Model:** This pre-trained model, known as the base model, includes layers that have processed data to learn hierarchical representations, capturing low-level to complex features.
3. **Transfer Layers:** Identify layers within the base model that hold generic information applicable to both the original and new tasks. These layers often near the top of the network capture broad, reusable features.
4. **Fine-tuning:** Fine-tune these selected layers with data from the new task. This process helps retain the pre-trained knowledge while adjusting parameters to meet the specific requirements of the new task, improving accuracy and adaptability.



Low-level features learned for task A should be beneficial for learning of model for task B.

Frozen Vs Trainable Layers in Transfer Learning

Aspect	Frozen Layers	Trainable Layers
Definition	Layers whose weights are kept fixed and not updated during training	Layers whose weights are updated during training
Purpose	Preserve general features learned from large pre-trained datasets	Adapt to task-specific features of the new dataset
Learning Process	No backpropagation updates; remain constant	Updated through backpropagation based on new data
Use Case	Used when new dataset is small or similar to the original dataset	Used when new dataset is large or significantly different from the original task
Computation Cost	Lower, since fewer parameters are trained	Higher, as more parameters need to be updated
Example in CNN	Early convolutional layers that capture edges, textures and basic shapes	Later fully connected layers or deeper convolutional layers for fine-tuned features



How to Decide Which Layers to Freeze or Train?

The extent to which you freeze or fine-tune layers depends on the similarity and size of your target dataset:

- **Small, Similar Dataset**: For smaller datasets that resemble the original dataset, you freeze most layers and only fine-tune the last one or two layers to prevent overfitting.
- **Large, Similar Dataset**: With large, similar datasets you can unfreeze more layers allowing the model to adapt while retaining learned features from the base model.
- **Small, Different Dataset**: For smaller, dissimilar datasets, fine-tuning layers closer to the input layer helps the model learn task-specific features from scratch.
- **Large, Different Dataset**: In this case, fine-tuning the entire model helps the model adapt to the new task while using the broad knowledge from the pre-trained model.

Transfer Learning with MobileNetV2 for MNIST Classification

In this section, we'll explore transfer learning by fine-tuning a MobileNetV2 model pre-trained on ImageNet for classifying MNIST digits.

1. Preparing the Dataset

We start by loading theMNIST dataset. Since MobileNetV2 is pre-trained on three-channel RGB images of size 224x224, we make a few adjustments to match its expected input shape:

- Reshape the images from grayscale (28x28, 1 channel) to RGB (28x28, 3 channels).
- Resize images to 32x32 pixels, aligning with our model's configuration.
- Normalize pixel values to fall between 0 and 1 by dividing by 255.

2. Building the Model

We load MobileNetV2 with pre-trained weights from ImageNet excluding the fully connected top layers to customize for our 10-class classification task:

- Freeze the base model to retain learned features and avoid overfitting.
- Add a global average pooling layer to reduce model complexity.
- Add a dense layer with softmax activation for the output classes.

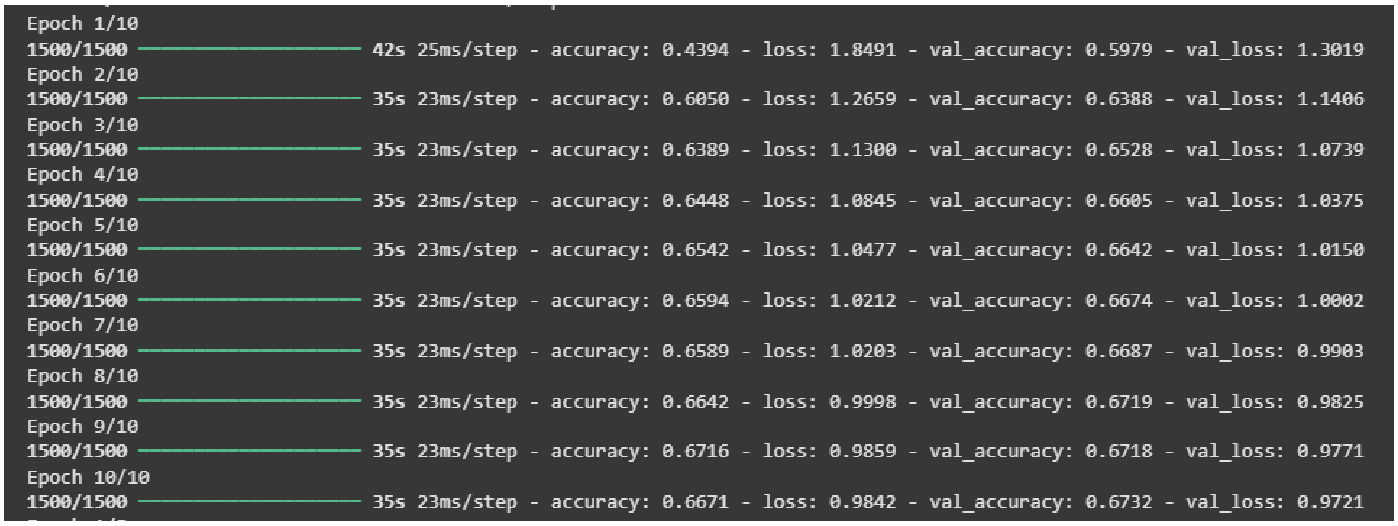
Output:

> base_model=MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

3. Compiling and Training the Model

The model is compiled with categorical cross-entropy as the loss function and accuracy as the evaluation metric. Using Adam optimizer we train the model on the MNIST training data for ten epochs.

Output:



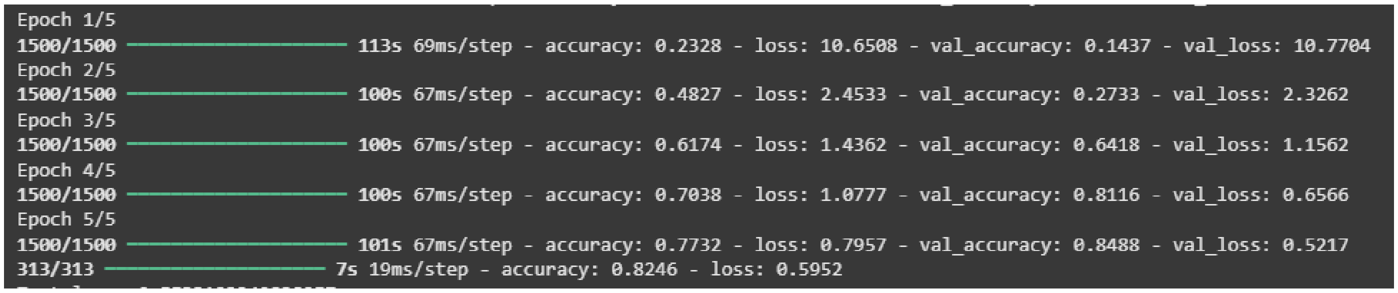
```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 42s 25ms/step - accuracy: 0.4394 - loss: 1.8491 - val_accuracy: 0.5979 - val_loss: 1.3019
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6050 - loss: 1.2659 - val_accuracy: 0.6388 - val_loss: 1.1406
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6389 - loss: 1.1300 - val_accuracy: 0.6528 - val_loss: 1.0739
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6448 - loss: 1.0845 - val_accuracy: 0.6605 - val_loss: 1.0375
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6542 - loss: 1.0477 - val_accuracy: 0.6642 - val_loss: 1.0150
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6594 - loss: 1.0212 - val_accuracy: 0.6674 - val_loss: 1.0002
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6589 - loss: 1.0203 - val_accuracy: 0.6687 - val_loss: 0.9903
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6642 - loss: 0.9998 - val_accuracy: 0.6719 - val_loss: 0.9825
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6716 - loss: 0.9859 - val_accuracy: 0.6718 - val_loss: 0.9771
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.6671 - loss: 0.9842 - val_accuracy: 0.6732 - val_loss: 0.9721
```

Training the model

4. Fine-Tuning the Model

After initial training we unfreeze the last few layers of the base model to perform fine-tuning. This allows the model to adjust high-level features for the MNIST data while retaining its foundational knowledge.

Output:



```
Epoch 1/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 113s 69ms/step - accuracy: 0.2328 - loss: 10.6508 - val_accuracy: 0.1437 - val_loss: 10.7704
Epoch 2/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 100s 67ms/step - accuracy: 0.4827 - loss: 2.4533 - val_accuracy: 0.2733 - val_loss: 2.3262
Epoch 3/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 100s 67ms/step - accuracy: 0.6174 - loss: 1.4362 - val_accuracy: 0.6418 - val_loss: 1.1562
Epoch 4/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 100s 67ms/step - accuracy: 0.7038 - loss: 1.0777 - val_accuracy: 0.8116 - val_loss: 0.6566
Epoch 5/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 101s 67ms/step - accuracy: 0.7732 - loss: 0.7957 - val_accuracy: 0.8488 - val_loss: 0.5217
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 19ms/step - accuracy: 0.8246 - loss: 0.5952
```

Fine-Tuning the model

5. Model Evaluation

Once the model has been trained and fine-tuned we evaluate it on the test set, measuring its loss and accuracy. This step assesses how well the transfer learning model has adapted to the MNIST dataset and demonstrates its effectiveness in digit classification.

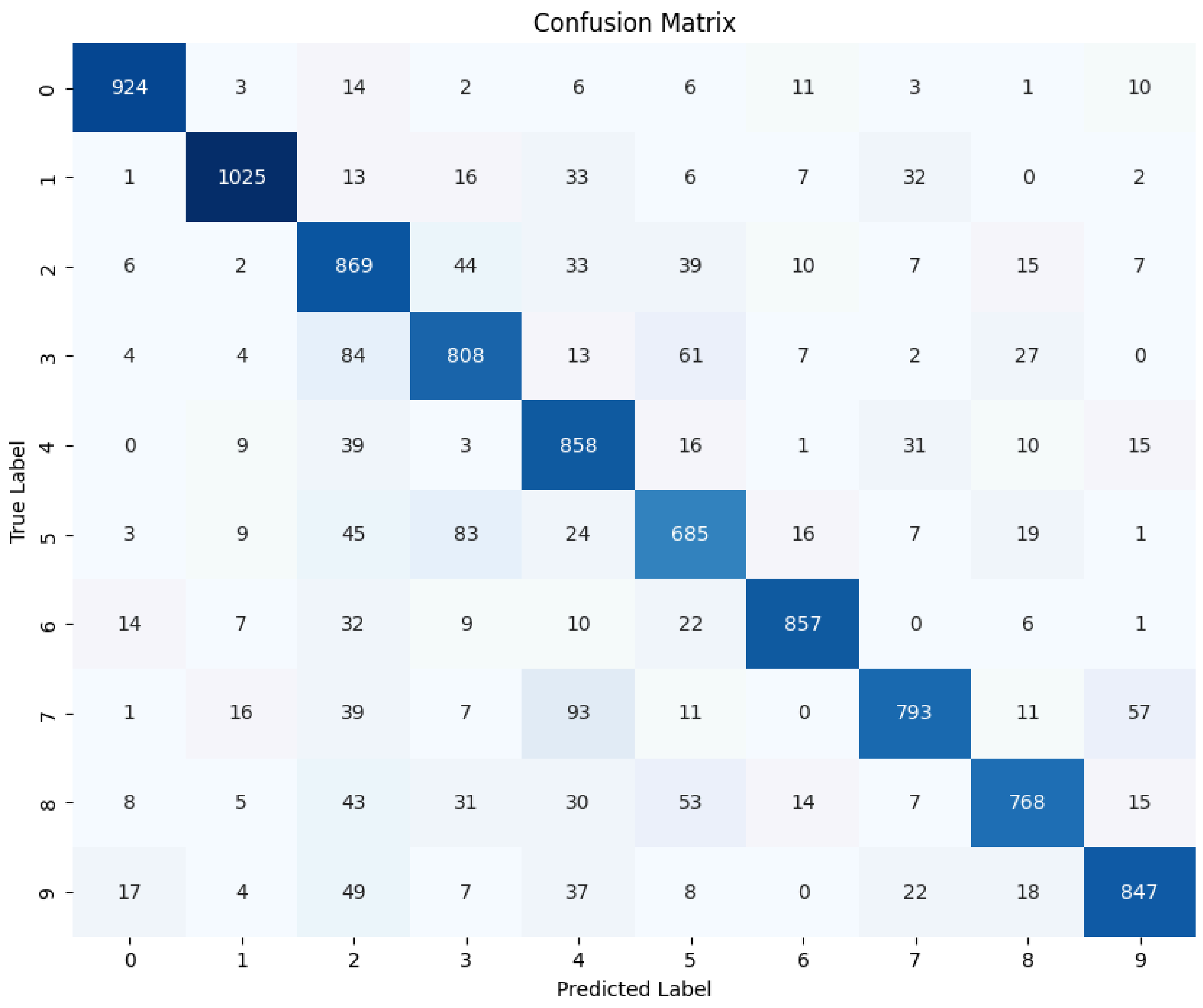
Output:

> Test loss: 0.5697252154350281
> Test accuracy: 0.8434000015258789

6. Visualizing Model Performance

To visualize the performance further a confusion matrix provides a breakdown of correct and incorrect classifications.

Output:

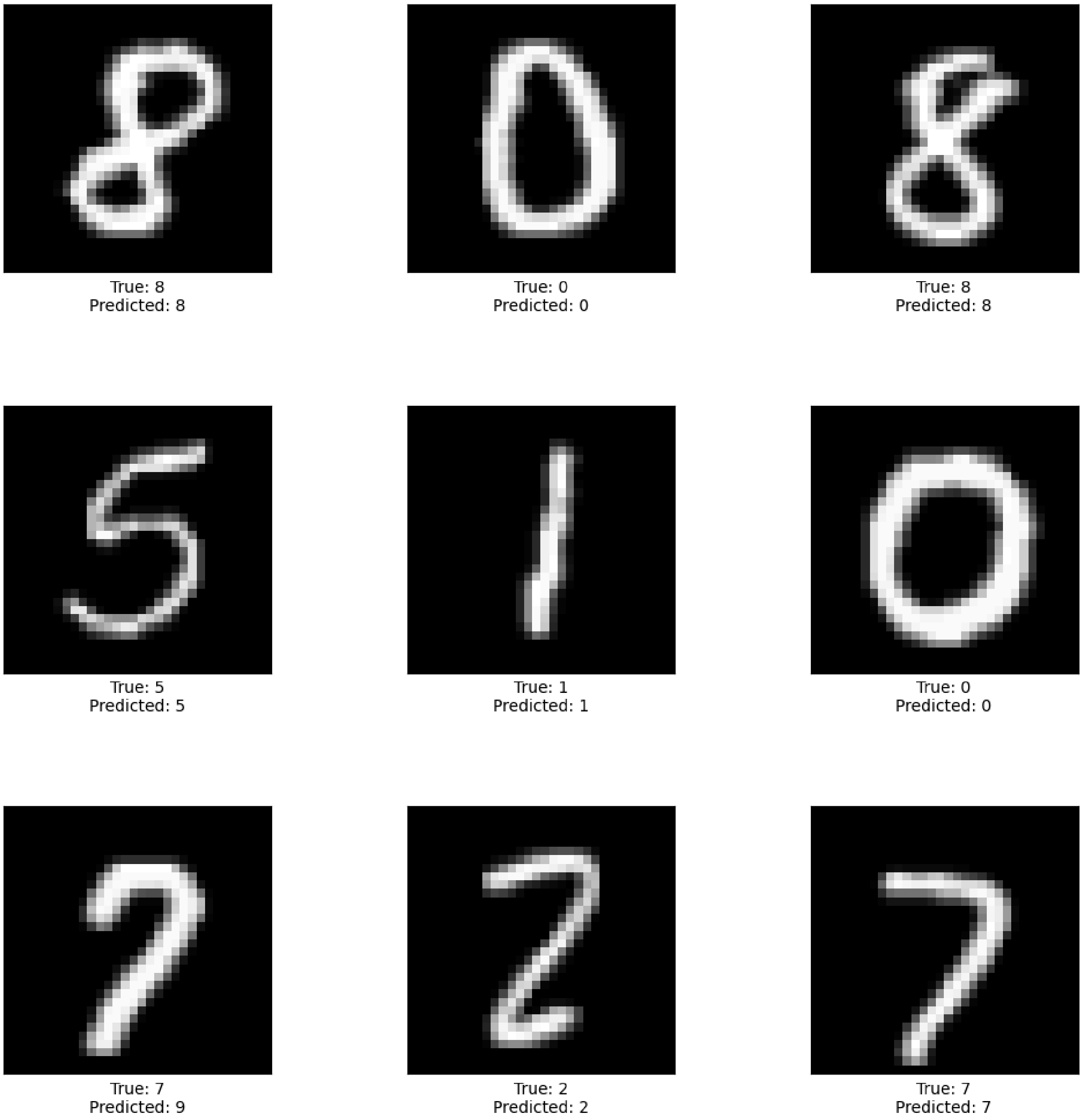


	0	1	2	3	4	5	6	7	8	9
0	924	3	14	2	6	6	11	3	1	10
1	1	1025	13	16	33	6	7	32	0	2
2	6	2	869	44	33	39	10	7	15	7
3	4	4	84	808	13	61	7	2	27	0
4	0	9	39	3	858	16	1	31	10	15
5	3	9	45	83	24	685	16	7	19	1
6	14	7	32	9	10	22	857	0	6	1
7	1	16	39	7	93	11	0	793	11	57
8	8	5	43	31	30	53	14	7	768	15
9	17	4	49	7	37	8	0	22	18	847

7. Sample Image Visualization

Finally we select a few test images to visualize the model's predictions against their true labels.

Output:



Labelled Images Output

> You can get the complete source code from here.

Applications

Transfer learning is widely used across multiple domains including:

- **Computer Vision**: Transfer learning is prevalent in image recognition tasks where models pre-trained on large image datasets are adapted to specific tasks such as medical imaging, facial recognition and object detection.
- **Natural Language Processing (NLP)**: In NLP models like BERT, GPT or ELMo are pre-trained on vast text corpora and later fine-tuned for specific tasks such as sentiment analysis, machine translation and question-answering.
- **Healthcare**: Transfer learning helps develop medical diagnostic tools using knowledge from general image recognition models to analyze medical images like X-rays or MRIs.
- **Finance**: Transfer learning in finance assists in fraud detection, risk assessment and credit scoring by transferring patterns learned from related financial datasets.

Advantages

- **Speed up the training process:** By using a pre-trained model the model can learn more quickly and effectively on the second task, as it already has a good understanding of the features and patterns in the data.
- **Better performance:** Transfer learning can lead to better performance on the second task, as the model can use the knowledge it has gained from the first task.
- **Handling small datasets:** When there is limited data available for the second task, transfer learning can help to prevent overfitting as the model will have already learned general features that are likely to be useful in the second task.

Disadvantages

- **Domain mismatch:** The pre-trained model may not be well-suited to the second task if the two tasks are vastly different or the data distribution between the two tasks is very different.
- **Overfitting**: Transfer learning can lead to overfitting if the model is fine-tuned too much on the second task, as it may learn task-specific features that do not generalize well to new data.
- **Complexity**: The pre-trained model and the fine-tuning process can be computationally expensive and may require specialized hardware.