Share if you like it!
And especially if you don't!

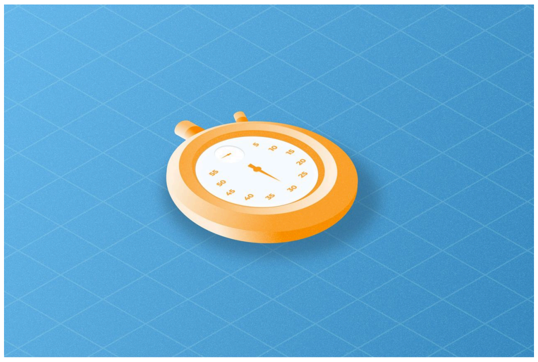
in

Research & Development November 8, 2023 • 22 min read

# Transfer Learning Applied: 97% Accuracy in 30 Seconds

Gökhan Şimşek

**For new startups that need to quickly validate their product idea, machine learning has one significant obstacle. The cost of building the dataset and training a new model can be a showstopper. That's where transfer learning comes in — get the ML functionality needed to prove your concept without breaking the bank.**



In this blog post, we will take a real-life machine learning (ML) problem and solve it with the power of **transfer learning** — achieving **97% image prediction accuracy** with **30 seconds of training** and just **200 images**. Training from scratch would have taken hours, require thousands of images and much more additional work.

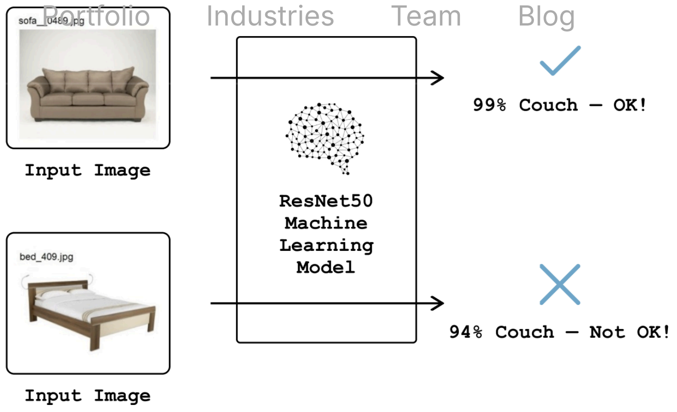
## Problem & Solution

### Problem: Proper Classification

The ResNet50 pre-trained image classification ML model does not support images of 'beds'. For a product that relies on the correct identification and classification of furniture items, this was a serious shortcoming.

This was one of our problems in the project we outlined here: Image Similarity Search — from Research to Production. Because of this we were unable to support bed images in the MVP phase.

We were using the ResNet50 model for image similarity search and we wanted to use the same model for image classification as well. But ResNet50, which supports 1000+ image classes, **did not support** furniture class 'bed' — simply because it was not trained to do so.



## Solution: Transfer Learning

In order to classify 'bed' images, the first two options that come to mind are "transfer learning" and "learning from scratch". We chose **transfer learning** since it requires less time to train, less data, less engineering time, less code/design complexity and thus less problems — I think I made my point :)

Let's check the results right away then we will dive into more technical details later on.

The section below is the output of our train & test python script transfer_learn.py. The output is simplified, in the first step we trained our model with transfer learning techniques using only 100 images per class (200 in total, this number is very small since we are using transfer learning).

In the second step, we tested the accuracy of our trained model with 200 images which this model has never seen. **Finally,** we get **97% test prediction accuracy** (and 95% training accuracy) after **30 seconds** of training.

(Note: Validation step for training is omitted for simplicity.)

```
> transfer_learn.py
```

```

--- 1. Training Model (with Transfer Learning)...
------------------------------------------
Found 200 images belonging to 2 classes.   # <--- 100 sofa + 100 bed im
Classes found: {'bed': 0, 'sofa': 1}

Epoch 1/5
7/7 [==============================] - 9s 1s/step - loss: 1.5591 - accu
Epoch 2/5
7/7 [==============================] - 9s 1s/step - loss: 0.2288 - accu
Epoch 3/5
7/7 [==============================] - 11s 2s/step - loss: 0.1520 - acc
...

--- 2. Testing Model...
==========================================
Passed  bed        sofa      Image
-------------------------------------------
No      0.23562    0.76438   bed_400.jpg
Yes     0.98450    0.01550   bed_401.jpg
Yes     0.99747    0.00253   bed_402.jpg
....
Yes     0.99399    0.00601   bed_497.jpg
Yes     0.99406    0.00594   bed_498.jpg
No      0.16362    0.83638   bed_499.jpg      # <--- 100 th bed image
Yes     0.00002    0.99998   sofa__0400.jpg
Yes     0.00012    0.99988   sofa__0401.jpg
Yes     0.14652    0.85348   sofa__0402.jpg
...
Yes     0.00012    0.99988   sofa__0497.jpg
Yes     0.00045    0.99955   sofa__0498.jpg
Yes     0.00009    0.99991   sofa__0499.jpg  # <---100 th sofa image
-------------
Passed: 195
Total : 200
-------------------------------------------
Test Accuracy: 0.97                          <--- 97% accuracy
-------------------------------------------
```

view raw

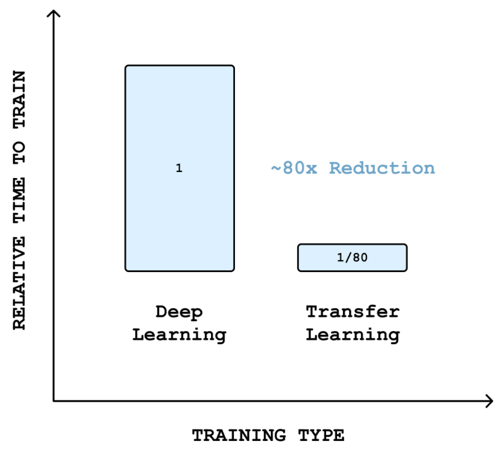
transfer_learn_result.py
hosted with ❤ by GitHub

## Why Transfer Learning & why is it so important?

With transfer learning the training time can be reduced dramatically — up to 80x. Hours instead of days, minutes/seconds instead of hours, depending on the model, dataset and use case. Other advantages are that the training task, code and model design becomes less complex. These all reduced engineering time and effort.

With transfer learning, the knowledge from a pre-trained model is carried over so we can re-use proven pre-trained models for a similar task.

Let's see some numbers from Intel — a comparison of transfer learning vs training from scratch (aka traditional training):

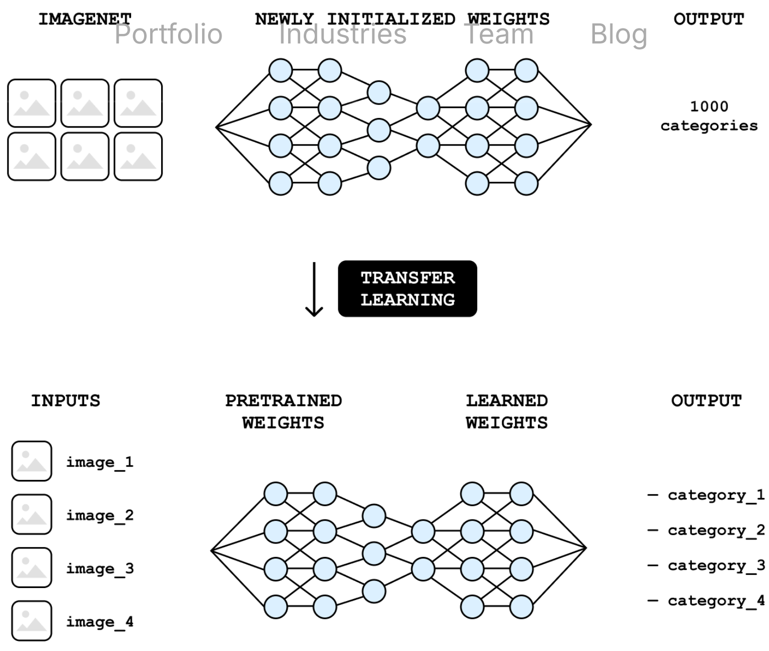


## What is Transfer Learning?

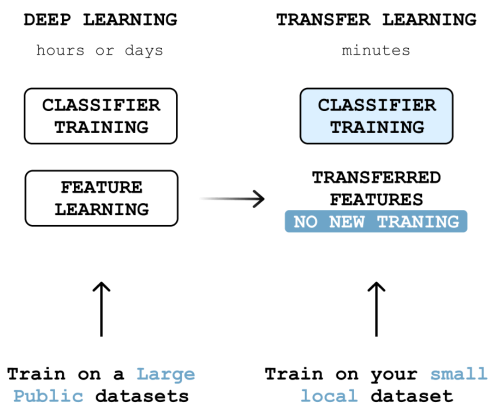
Transfer learning lets us utilize gained knowledge of an existing model — in our case the ResNet50 image classifier — and apply it to a similar problem.

Transfer learning is basically transferring knowledge (by re-training certain parts of a ML model with a smaller dataset) from one machine learning model to another instead of training a model from scratch. For example, knowledge gained while learning to recognize cars could be transferred to recognize trucks.

First let's show the application of the transfer learning algorithm to categorize images:



Now we can see how transfer learning gives us faster training times with a smaller dataset:



## The Code & Our Transfer Learning Strategy

In this section we will summarize our transfer learning strategy and explain its code.

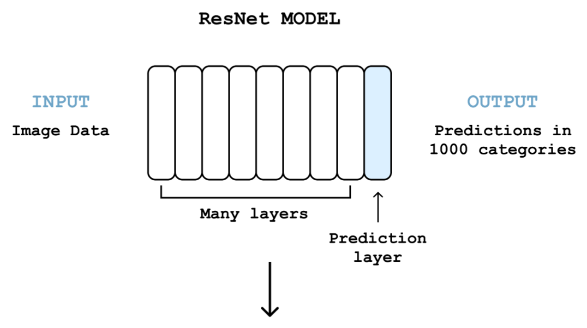
**ResNet50:** We are using the famous deep neural network ResNet50 as the pre-trained base model for our Transfer Learning. The ResNet50 model has already been trained on ImageNet and is quite a beast — 150 GB, ~1.2 million images for training and validation, 1,000 classes.

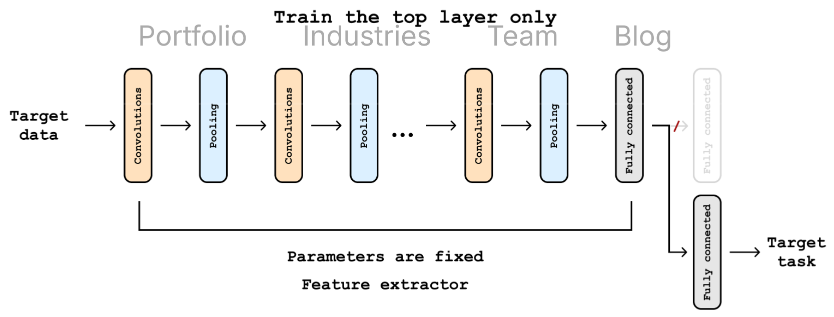
But, no beds! So we're going to repurpose it to classify new images of beds and sofas.

**Transfer Learning Strategy:**

- Load and remove the last predicting layer of the pre-trained ResNet50 model and replace it with our new dense layers.
- Freeze (make non-trainable) the weights of ResNet50 pre-trained model and use it as a feature extractor (Weights of the pre-trained model are frozen and will not be updated during the training).
- Only train the newly added dense layers, which are created from randomly initializing the weights

So, in the end the new model we'll create will consist of a base pre-trained network (from ResNet50) and a new dense network classifier.





Sources and more info: Transfer Learning | Kaggle & Transfer Learning | Lecture 9

The related python code can be seen below and it is heavily commented. Other parts of the code are not shown but can be found in the Github repository linked below.

```python
### ---- 1. Create and configure new model based on ResNet50
def create_model():
  ## Load pre-trained model “Resnet50” without the final(top) layer
  base_model = ResNet50(weights='imagenet', include_top = False)
  output = base_model.output
  ## Condense feature maps from the output
  output = tf.keras.layers.GlobalAveragePooling2D()(output)
  ## Add dense fully connected artificial neural network at the end
  output = tf.keras.layers.Dense(1024, activation='relu')(output)
  output = tf.keras.layers.Dense(1024, activation='relu')(output)
  output = tf.keras.layers.Dense(1024, activation='relu')(output)
  output = tf.keras.layers.Dense(512, activation='relu')(output)
  ## Final layer has 2 output neurons since we're classifying beds and
  final_output = tf.keras.layers.Dense(2, activation ='softmax')(output

  ## Create our own network/model
  model = tf.keras.models.Model(inputs=base_model.input, outputs=final_

  ## Freeze the layers up to 174 (pre-trained layers from ResNet50)
  ## For layer 175 and up set them trainable
  for layer in model.layers[:175]: layer.trainable = False
  for layer in model.layers[175:]: layer.trainable = True
  return model
```

view raw

create_model.py hosted with ❤ by GitHub

```python
### ---- 2. Train the model
def train(learn_path):
  model = create_model() # Use the function from previous code section

  ## Setup training parameters
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function = tf.keras.applications.resnet50.preprocess_
  )
  learn_path = '/ML/bed_sofa/learn' # inside two subfolders “bed” and “
  train_generator = train_datagen.flow_from_directory(
    learn_path,
    target_size = (224, 224),
    color_mode = 'rgb',
    batch_size = 32,
    class_mode = 'categorical',
    shuffle = True
  )

  ## Optional. e.g. label_map -> {'bed': 0, 'sofa': 1}
  label_map = (train_generator.class_indices)
  print(f'\n--- Classes found: {label_map}\n')

  ## Train on 5 epochs
  LOSS_FUNCTION = 'categorical_crossentropy'
  model.compile(optimizer='Adam', loss=LOSS_FUNCTION, metrics=['accurac
  history = model.fit_generator(generator = train_generator, epochs = 5

  ## Optionally save the trained model to disk to use later
  if save_the_model:
    model.save(save_model_name)
    print('--- Saved model: ', save_model_name)

  return model
```

view raw

transfer_learn.py
hosted with ❤ by GitHub

## Source code on Github

So, in the end we’ve got the proper classification of beds for our furniture image recognition project. By using the right transfer learning model along with the

ResNet50 classifier, we saved a bunch of time without sacrificing accuracy.

Don’t let the estimated cost of model training hold you back from launching an ML-enabled product! As we’ve shown, there are often ways to get the functionality you need without training from scratch.

**Thanks for reading 😜**

Other Sources and further reading:

- Use Transfer Learning For Efficient Deep Learning Training On Intel®
- Introduction to Transfer Learning with TensorFlow 2.0
- Tutorial Keras: Transfer Learning with ResNet50

Previous Article

It's a Blockchain — 101

Next Article

Object Detection from Image and Video using HSV Color Space