Understanding Transfer Learning for Deep Learning

ADVANCED DEEP LEARNING PYTHON

Transfer learning is a powerful technique used in Deep Learning. By harnessing the ability to reuse existing models and their knowledge of new problems, transfer learning has opened doors to training deep neural networks even with limited data. This breakthrough is especially significant in data science, where practical scenarios often need more labeled data. In this article, we delve into the depths of transfer learning, unraveling its concepts and exploring its applications in empowering data scientists to tackle complex challenges with newfound efficiency and effectiveness.

This article was published as a part of the Data Science Blogathon

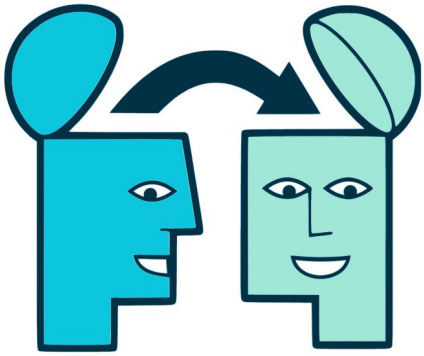
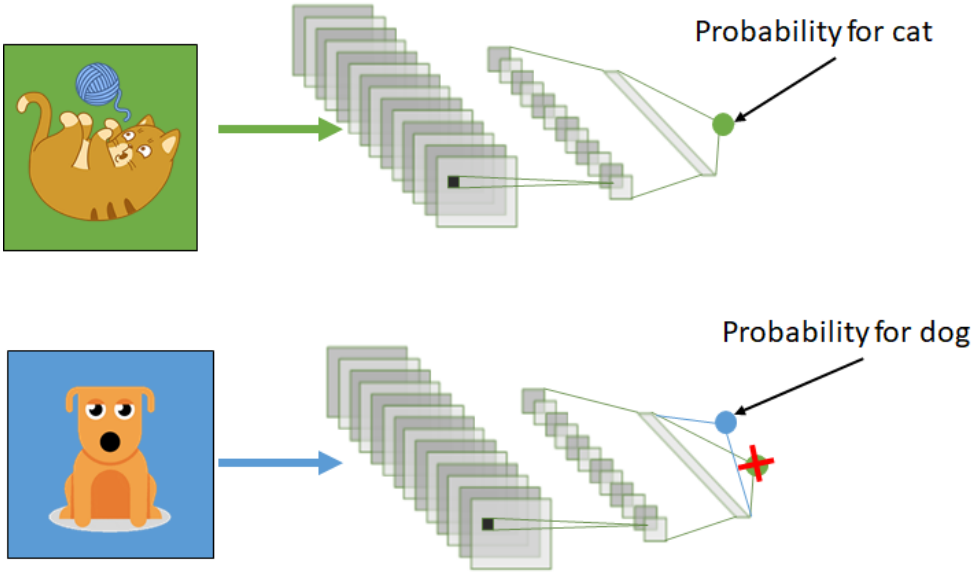


Table of contents

What Is Transfer Learning?

The reuse of a pre-trained model on a new problem is known as transfer learning in machine learning. A machine uses the knowledge learned from a prior assignment to increase prediction about a new task in transfer learning. You could, for example, use the information gained during training to distinguish beverages when training a classifier to predict whether an image contains cuisine.

The knowledge of an already trained machine learning model is transferred to a different but closely linked problem throughout transfer learning. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the model's training knowledge to identify other objects such as sunglasses.



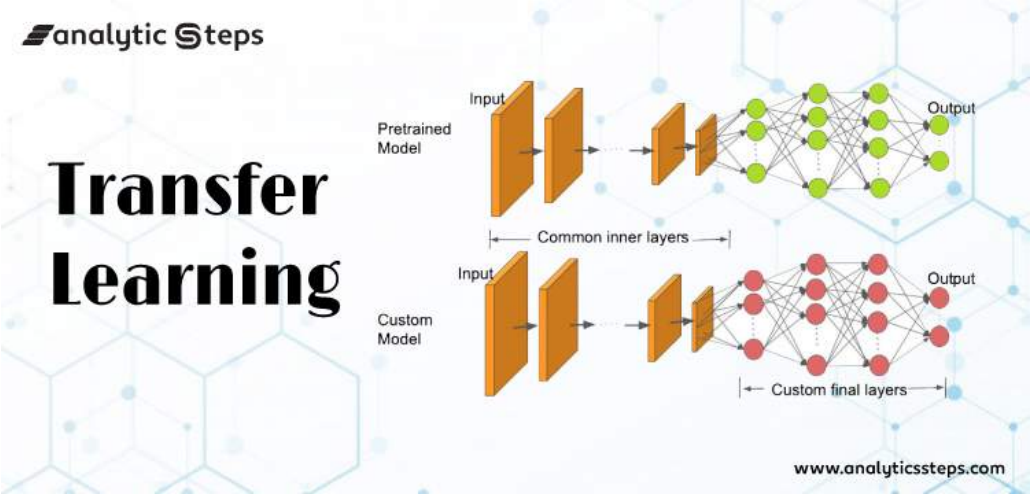
With transfer learning, we basically try to use what we've learned in one task to better understand the concepts in another. weights are being automatically being shifted to a network performing "task A" from a network that performed new "task B."

Because of the massive amount of CPU power required, transfer learning is typically applied in computer vision and natural language processing tasks like sentiment analysis.

How Transfer Learning Works?

In computer vision, neural networks typically aim to detect edges in the first layer, forms in the middle layer, and task-specific features in the latter layers.

The early and central layers are employed in transfer learning, and the latter layers are only retrained. It makes use of the labelled data from the task it was trained on.

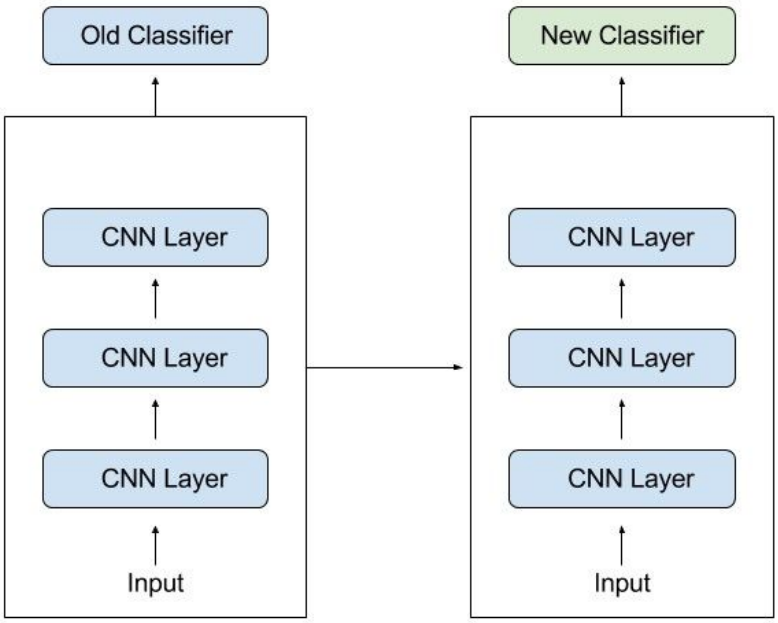


Let's return to the example of a model that has been intended to identify a backpack in an image and will now be used to detect sunglasses. Because the model has trained to recognise objects in the earlier levels, we will simply retrain the subsequent layers to understand what distinguishes sunglasses from other objects.

Why Should You Use Transfer Learning?

Transfer learning offers a number of advantages, the most important of which are reduced training time, improved neural network performance (in most circumstances), and the absence of a large amount of data.

To train a neural model from scratch, a lot of data is typically needed, but access to that data isn't always possible – this is when transfer learning comes in handy.



Because the model has already been pre-trained, a good machine learning model can be generated with fairly little training data using transfer learning. This is especially useful in natural language processing, where huge labelled datasets require a lot of expert knowledge. Additionally, training time is decreased because building a deep neural network from the start of a complex task can take days or even weeks.

Steps to Use Transfer Learning

Time needed: 20 minutes

When we don't have enough annotated data to train our model with and there is a pre-trained model that has been trained on similar data and tasks. If you used TensorFlow to train the original model, you might simply restore it and retrain some layers for your job. Transfer learning, on the other hand, only works if the features learnt in the first task are general, meaning they can be applied to another activity. Furthermore, the model's input must be the same size as it was when it was first trained.

If you don't have it, add a step to resize your input to the required size:

1. **Training a Model to Reuse it**

   Consider the situation in which you wish to tackle Task A but lack the necessary data to train a deep neural network. Finding a related task B with a lot of data is one method to get around this.

   Utilize the deep neural network to train on task B and then use the model to solve task A. The problem you're seeking to solve will decide whether you need to employ the entire model or just a few layers.

   If the input in both jobs is the same, you might reapply the model and make predictions for your new input. Changing and retraining distinct task-specific layers and the output layer, on the other hand, is an approach to investigate.

2. **Using a Pre Trained Model**

   The second option is to employ a model that has already been trained. There are a number of these models out there, so do some research beforehand. The number of layers to reuse and retrain is determined by the task.

   Keras consists of nine pre-trained models used in transfer learning, prediction, fine-tuning. These models, as well as some quick lessons on how to utilise them, may be found here. Many research institutions also make trained models accessible.

   The most popular application of this form of transfer learning is deep learning.

3. **Extraction of Features**

   Another option is to utilise deep learning to identify the optimum representation of your problem, which comprises identifying the key features. This method is known as representation learning, and it can often produce significantly better results than hand-designed representations.

   Feature creation in machine learning is mainly done by hand by researchers and domain specialists. Deep learning, fortunately, can extract features automatically. Of course, this does not diminish the importance of feature engineering and domain knowledge; you must still choose which features to include in your network.

4. **Extraction of Features in Neural Networks**

   Neural networks, on the other hand, have the ability to learn which features are critical and which aren't. Even for complicated tasks that would otherwise necessitate a lot of human effort, a representation learning algorithm can find a decent combination of characteristics in a short amount of time.

   The learned representation can then be applied to a variety of other challenges. Simply utilise the initial layers to find the appropriate feature representation, but avoid using the network's output because it is too task-specific. Instead, send data into your network and output it through one of the intermediate layers.

   The raw data can then be understood as a representation of this layer.
   This method is commonly used in computer vision since it can shrink your dataset, reducing computation time and making it more suited for classical algorithms.

Models That Have Been Pre-Trained

There are a number of popular pre-trained machine learning models available. The Inception-v3 model, which was developed for the ImageNet "Large Visual Recognition Challenge," is one of them." Participants in this challenge had to categorize pictures into 1,000 subcategories such as "zebra," "Dalmatian," and "dishwasher."

Code Implementation of Transfer Learning with Python

Importing Libraries

```
import tensorflow as tf import pandas as pd import matplotlib.pyplot as plt from tensorflow.keras import Model from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D from tensorflow.keras.models import Sequential from tensorflow.keras.preprocessing.image import ImageDataGenerator from tensorflow.keras.callbacks import ReduceLROnPlateau from tensorflow.keras.layers import Input, Lambda, Dense, Flatten from tensorflow.keras.models import Model from tensorflow.keras.applications.inception_v3 import InceptionV3 from tensorflow.keras.applications.inception_v3 import preprocess_input from tensorflow.keras.preprocessing import image from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img from tensorflow.keras.models import Sequential import numpy as np from glob import glob
```

Uploading Data via Kaggle API

```
from google.colab import files files.upload()
```

Saving kaggle.json to kaggle.json

```
!mkdir -p ~/.kaggle !cp kaggle.json ~/.kaggle/ !chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d mohamedhanyyy/chest-ctscan-images #downloading data from kaggle API of Dataset
```

```
from zipfile import ZipFile file_name = "chest-ctscan-images.zip" with ZipFile(file_name,'r') as zip: zip.extractall() print('Done')
```

Designing Our CNN Model with help of Pre–Trained Model

```
InceptionV3_model = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
from tensorflow.keras import Model from tensorflow.keras.layers import Conv2D, Dense, MaxPooling2D, Dropout, Flatten,GlobalAveragePooling2D from tensorflow.keras.models import Sequential # The last 15 layers fine tune for layer in InceptionV3_model.layers[:-15]: layer.trainable = False x = InceptionV3_model.output x = GlobalAveragePooling2D()(x) x = Flatten()(x) x = Dense(units=512, activation='relu')(x) x = Dropout(0.3)(x) x = Dense(units=512, activation='relu')(x) x = Dropout(0.3)(x) output = Dense(units=4, activation='softmax')(x) model = Model(InceptionV3_model.input, output) model.summary()
```

Image Augmentation(For preventing the issue of Overfitting)

```
# Use the Image Data Generator to import the images from the dataset from tensorflow.keras.preprocessing.image import ImageDataGenerator train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True) test_datagen = ImageDataGenerator(rescale = 1./255) #no flip and zoom for test datase
```

```
# Make sure you provide the same target size as initialied for the image size training_set = train_datagen.flow_from_directory('/content/Data/train', target_size = (224, 224), batch_size = 32, class_mode = 'categorical')
```

Training Our Model

```
# fit the model # Run the cell. It will take some time to execute r = model.fit_generator( training_set, validation_data=test_set, epochs=8, steps_per_epoch=len(training_set), validation_steps=len(test_set) )
```

```
# plot the loss plt.plot(r.history['loss'], label='train loss') plt.plot(r.history['val_loss'], label='val loss') plt.legend() plt.show() plt.savefig('LossVal_loss') # plot the accuracy plt.plot(r.history['accuracy'], label='train acc') plt.plot(r.history['val_accuracy'], label='val acc') plt.legend() plt.show() plt.savefig('AccVal_acc')
```

Making Predictions

```
import numpy as np y_pred = np.argmax(y_pred, axis=1) y_pred
```

The above code is being executed, and respective output for classification using Transfer Learning is being shown below the embedded notebook:

Loss plot

accuracy plot

You can access the Github link for Google Colab notebook here!

Conclusion

In conclusion, understanding transfer learning is crucial for data scientists venturing into deep learning. It equips them to leverage pre-trained models and extract valuable knowledge from existing data, enabling them to solve complex problems with limited resources. Consider exploring our Blackbelt program to further enhance your expertise in transfer learning and propel your data science journey. With its comprehensive curriculum and practical hands-on approach, the program offers a unique opportunity to master transfer learning and unlock the full potential of deep learning in your data science endeavors.

Frequently Asked Questions

Q1. What is transfer learning in a CNN?

A. Transfer learning in a CNN refers to using a pre-trained model on a similar task as a starting point for training a new model on a different task.

Q2. What is an example of learning transfer?

A. An example of learning transfer is using a pre-trained image classification model to build a model for a specific image recognition task.

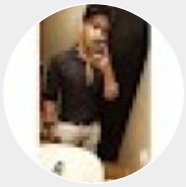
Q3. What type of learning is transfer learning?

A. Transfer learning is supervised learning where knowledge gained from one task is transferred to another related task to improve performance.

Q4. What is transfer learning in RL?

A. In RL, transfer learning involves using knowledge learned from one RL task to improve learning and performance on another related RL task, accelerating the learning process and enhancing performance.

Article Url - https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/



Pranshu Sharma