Exercise 45

One of the most common problems data science professionals face is to avoid overfitting. Avoiding overfitting can single-handedly improve our model's performance. Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. **Regularization techniques** help in overcoming overfitting.

- modifying training procedure
  - dropout layer
  - early stopping – early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model.
  - dynamic learning rate – changing learning rate during training
  - bias regularizer – regularizer to apply a penalty on the layer's bias (see L1 and L2)
  - activity regularizer – regularizer to apply a penalty on the layer's output (see L1 and L2)
- modifying loss function
  - L2 and L1 regularization – L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

    Cost function = Loss + Regularization term

    Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

    L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

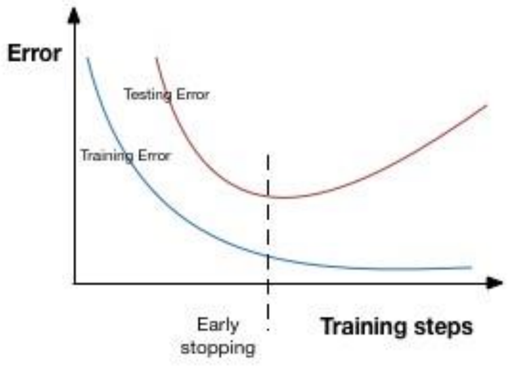
    $$\text{Cost function} = \text{Loss} + \frac{\lambda}{m} \cdot \sum ||w||^2$$

    L1, unlike L2, the weights may be reduced to zero.

    $$\text{Cost function} = \text{Loss} + \frac{\lambda}{m} \cdot \sum ||w||$$

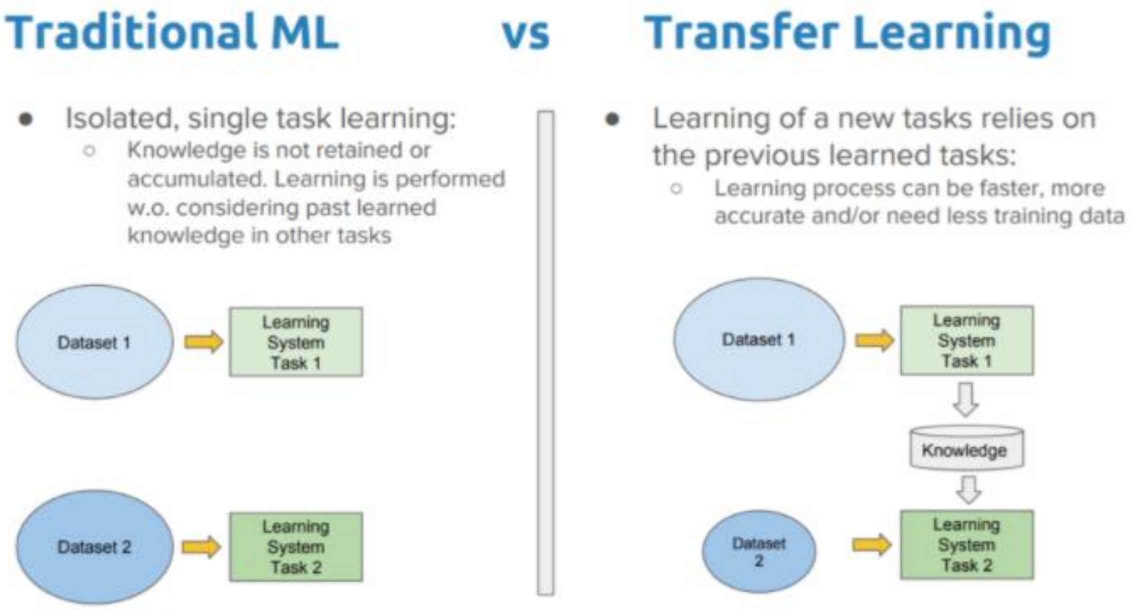
    Lambda is the regularization parameter (I don't know that *m* is, maybe number of connections in layer).
- generating 'new' training data from data you have
  - resizing (images)?

- data augmentation
- noise – add statistical noise to inputs during training



Exercise 46

The intuition behind **transfer learning** for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.



Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

Traditional Learning vs Transfer Learning

Fine-Tuning is a technique that unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to ’fine-tune’ the higher-order feature representations in the base model in order to make them more relevant for the specific task

This is an optional last step that can potentially give you incremental improvements. It could also potentially lead to quick overfitting -- keep that in mind.

Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify racoons may be useful to kick-start a model meant to identify tanukis.

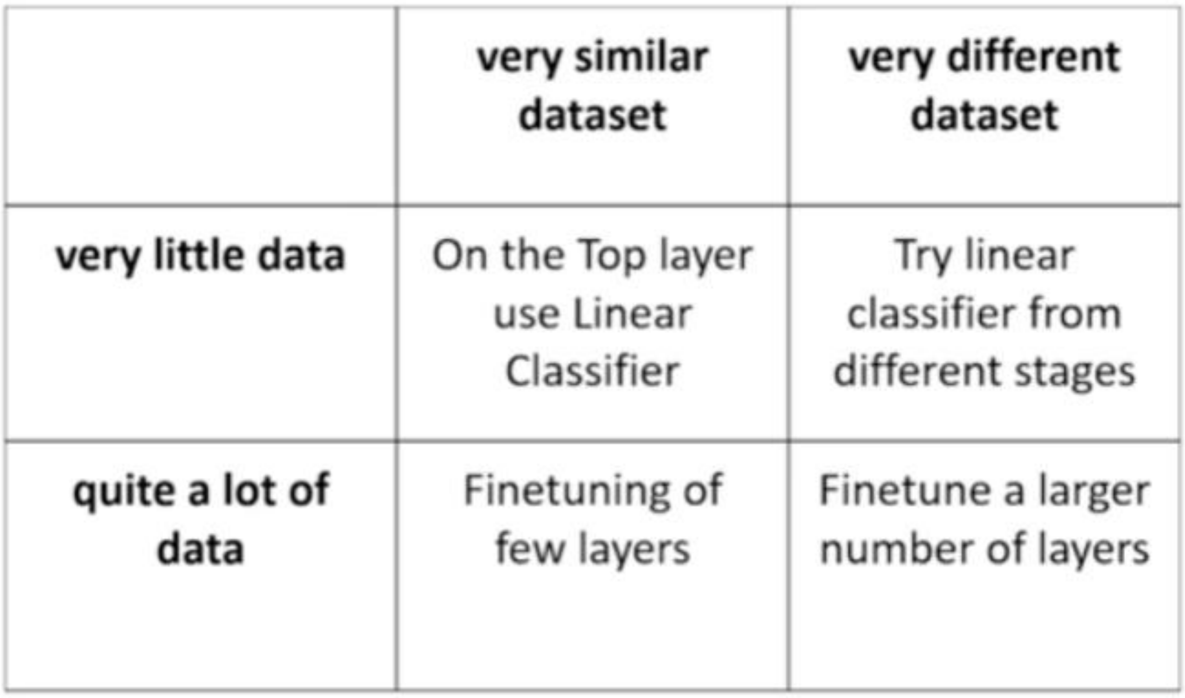
Transfer learning is usually done for tasks where your dataset has too little data to train a full-scale model from scratch.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

1. Take layers from a previously trained model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.

A last, optional step, is **fine-tuning**, which consists of unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

Exercise 47



	very similar dataset	**very different dataset**
very little data	On the Top layer use Linear Classifier	Try linear classifier from different stages
quite a lot of data	Finetuning of few layers	Finetune a larger number of layers

Scenario 1 – Size of the Data set is small while the Data similarity is very high – In this case, since the data similarity is very high, we do not need to retrain the model. All we need to do is to customize and modify the output layers according to our problem statement. We use the pretrained model as a feature extractor.

Scenario 2 – Size of the data is small as well as data similarity is very low – In this case we can freeze the initial (let's say k) layers of the pretrained model and train just the remaining(n-k) layers again. The top layers would then be customized to the new data set. Since the new data set has low similarity it is significant to retrain and customize the higher layers according to the new dataset. The small size of the data set is compensated by the fact that the initial layers are kept pretrained(which have been trained on a large dataset previously) and the weights for those layers are frozen.

Scenario 3 – Size of the data set is large however the Data similarity is very low – In this case, since we have a large dataset, our neural network training would be effective. However, since the data we have is very different as compared to the data used for training our pretrained models. The predictions made using pretrained models would not be effective. Hence, its best to train the neural network from scratch according to your data.

Scenario 4 – Size of the data is large as well as there is high data similarity – This is the ideal situation. In this case the pretrained model should be most effective. The best way to use the model is to retain the architecture of the model and the initial weights of the model. Then we can retrain this model using the weights as initialized in the pre-trained model.