UNIT – IV: Effective training of Deep Neural Networks

Early stopping, Dropout, Instance Normalization, Group Normalization, Transfer Learning, Data Augmentation.

4.1. Early Stopping:

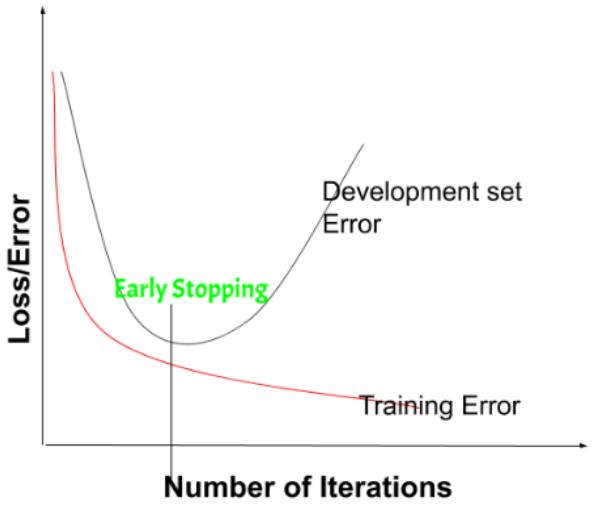
Regularization is a kind of regression where the learning algorithms are modified, to reduce overfitting. This may incur a higher bias but will lead to lower variance when compared to nonregularized models i.e. increases generalization of the training algorithm.

In a general learning algorithm, the dataset is divided into a **training set** and a **test set.** After each epoch of the algorithm, the parameters are updated accordingly after understanding the dataset. Finally, this trained model is applied to the test set.

Generally, the training set error will be less compared to the test set error. This is because of overfitting whereby the algorithm memorizes the training data and produces the right results on the training set. So, the model becomes highly exclusive to the training set and fails to produce accurate results for other datasets including the test set.

Regularization techniques are used in such situations to **reduce overfitting** and **increase the model's performance** on any general dataset.

In Regularization by Early Stopping, we stop training the model when the performance on the validation set is getting worse- increasing loss decreasing accuracy, or poorer scores of the scoring metric. By plotting the error on the training dataset and the validation dataset together, both the errors decrease with a number of iterations until the point where the model starts to overfit. After this point, the training error still decreases but the validation error increases. So, even if training is continued after this point, early stopping essentially returns the set of parameters that were used at this point and so is equivalent to stopping training at that point. So, the final parameters returned will enable the model to have low variance and better generalization. The model at the time the training is stopped will have a better generalization performance than the model with the least training error.



On the validation set is getting worse- increasing loss or decreasing accuracy or poorer scores. Early stopping can be thought of as **implicit regularization**, contrary to regularization via weight decay. This method is also efficient since it requires less amount of training data, which is not always available. Due to this fact, early stopping requires lesser time for training compared to other regularization methods. Repeating the early stopping process many times may result in the model overfitting the validation dataset, just as similar as overfitting occurs in the case of training data.

The number of iterations (i.e. epoch) taken to train the model can be considered a **hyperparameter**. Then the model has to find an optimum value for this hyperparameter (by hyperparameter tuning) for the best performance of the learning model.

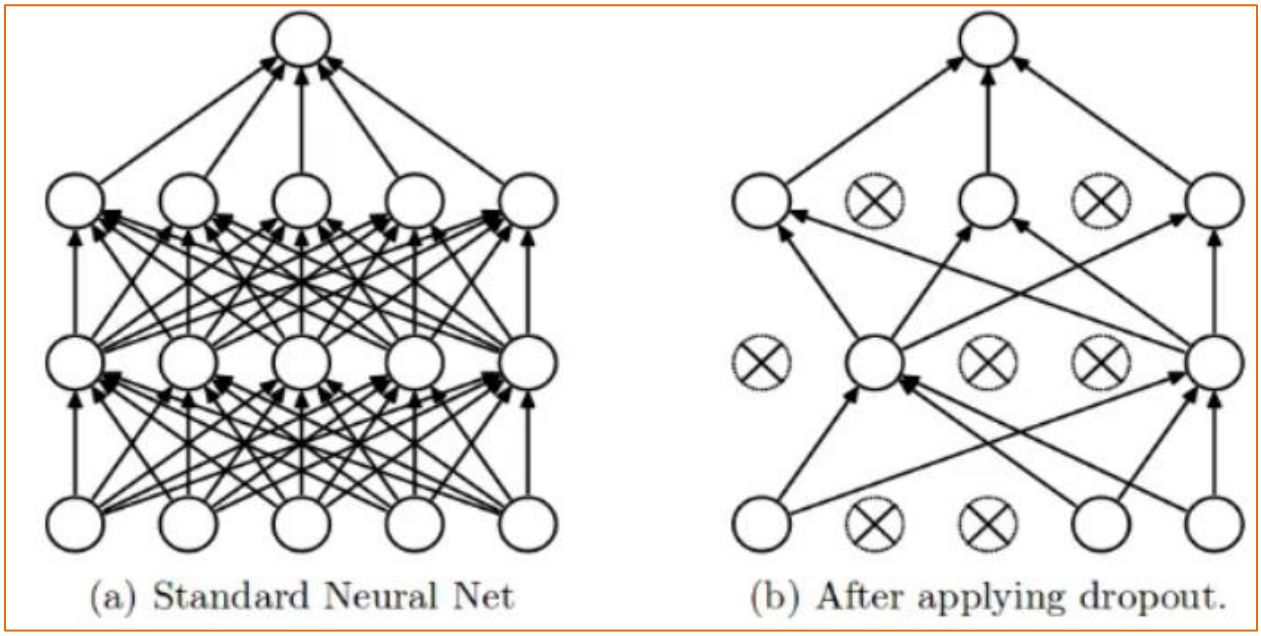
Benefits of Early Stopping:

- Helps in reducing overfitting.
- It improves generalization.
- It requires less amount of training data.
- Takes less time compared to other regularisation models.
- It is simple to implement.

Limitations of Early Stopping:

- If the model stops too early, there might be risk of underfitting.
- It may not be beneficial for all types of models.
- If validation set is not chosen properly, it may not lead to the most optimal stopping.

4.2. Dropout in Neural Networks:



The term "dropout" refers to dropping out the nodes (input and hidden layer) in a neural network (as seen in Figure). All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

Given a input x: {1, 2, 3, 4, 5} to the fully connected layer. Let's have a dropout layer with probability p = 0.2 (or keep probability = 0.8).

During the forward propagation (training) from the input x, 20% of the nodes would be dropped .The x could become {1, 0, 3, 4, 5} or {1, 2, 0, 4, 5} and so on.

Similarly, it is applied to the hidden layers ,if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch).

Generally, for the input layers, the keep probability, i.e. 1- drop probability, is closer to 1, 0.8 being the best .

For the hidden layers, the greater the drop probability more sparse the model, where 0.5 is the most optimised keep probability, that states dropping 50% of the nodes.

In the overfitting problem, the model learns the statistical noise. To be precise, the main motive of training is to decrease the loss function, given all the units (neurons). So in overfitting, a unit may change in a way that fixes up the mistakes of the other units. This

leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalise on the unseen dataset.

If we use dropout, it prevents these units to fix up the mistake of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable. So by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.

This ensures that the model is getting generalised and hence reducing the overfitting problem.

4.3. Normalization Techniques in Deep Neural Networks:

Normalization techniques can decrease our model's training time by a huge factor

1. It normalizes each feature so that they maintains the contribution of every feature, as some feature has higher numerical value than others. This way our network can be unbiased(to higher value features).
2. It reduces **Internal Covariate Shift**. It is the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift.
3. Batch Norm makes loss surface smoother(i.e. it bounds the magnitude of the gradients much more tightly)
4. It makes the Optimization faster because normalization doesn't allow weights to explode all over the place and restricts them to a certain range.

Batch Normalization

Batch normalization is a method that normalizes activations in a network across the mini-batch of definite size. For each feature, batch normalization computes the mean and variance of that feature in the mini-batch. It then subtracts the mean and divides the feature by its mini-batch standard deviation.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

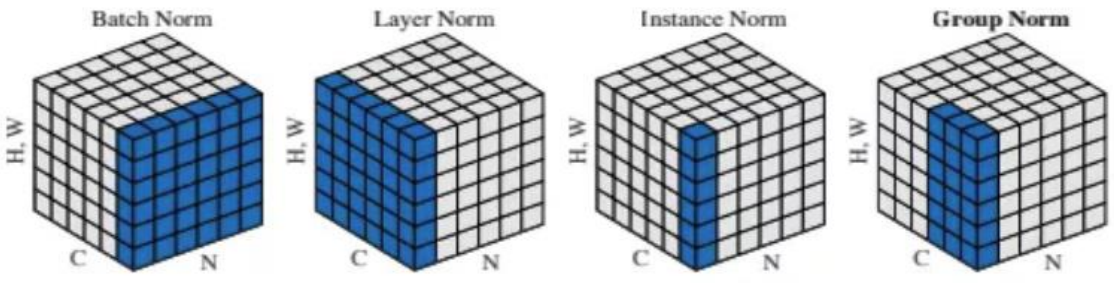
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Problems associated with Batch Normalization

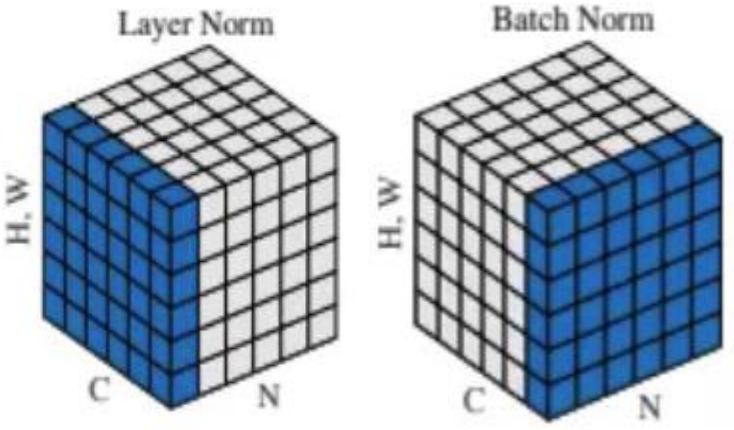
1. **Variable Batch Size →** If batch size is of 1, then variance would be 0 which doesn't allow batch norm to work. If we have small mini-batch size then it becomes too noisy and training might affect.
2. **Distributed Training:** if you are computing in different machines then you have to take same batch size because otherwise γ and β will be different for different systems.
3. **Recurrent Neural Network →** In an RNN, we have to fit a separate batch norm layer for each time-step. This makes the model more complicated and space consuming because it forces us to store the statistics for each time-step during training.



A visual comparison of various normalization methods

Layer Normalization

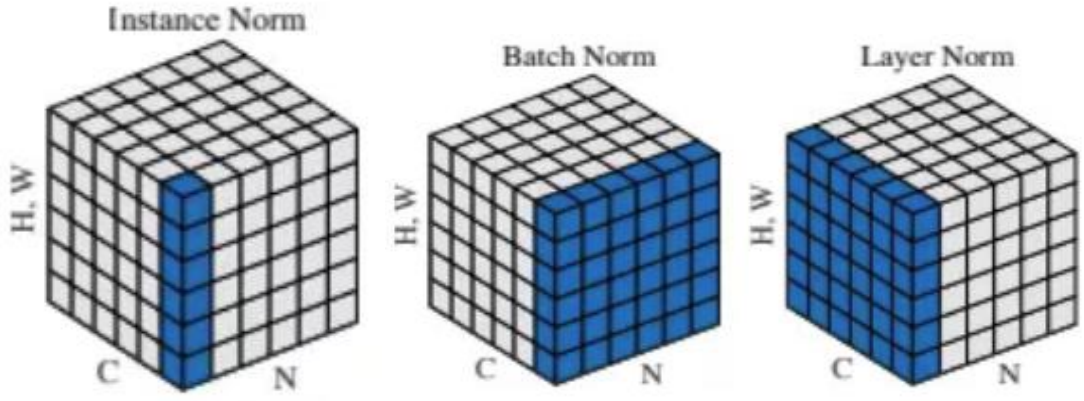
Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization.



Instance(or Contrast) Normalization

Layer normalization and instance normalization is very similar to each other but the difference between them is that instance normalization normalizes across each channel in each training example instead of normalizing across input features in an training example.

Unlike batch normalization, the instance normalization layer is applied at test time as well(due to non-dependency of mini-batch).



$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

Here, $\mathbf{x} \in \mathbb{R}$ T ×C×W×H be an input tensor containing a batch of T images. $\mathbf{x}_{tijk}$ denote its tijk-th element

where k and j span spatial dimensions(**H**eight and **W**idth of the image) i is the feature channel (color channel if the input is an RGB image) t is the index of the image in the batch.

$$\mu_i = \frac{1}{m}\sum_{k\in\mathcal{S}_i} x_k, \qquad \sigma_i = \sqrt{\frac{1}{m}\sum_{k\in\mathcal{S}_i}(x_k-\mu_i)^2 + \epsilon},$$

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i). \qquad y_i = \gamma\hat{x}_i + \beta,$$

Here, **x** is the feature computed by a layer, and i is an index.

Weight Normalization

We **normalize weights of a layer** instead of normalizing the activations directly Weight normalization reparameterizes the weights (**ω**) as :

$$\boldsymbol{w} = \frac{g}{\|\boldsymbol{v}\|}\boldsymbol{v}$$

4.4. Transfer learning:

Transfer learning is a technique in machine learning where a model trained on one task is used as the starting point for a model on a second task. This can be useful when the second task is similar to the first task, or when there is limited data available for the second task. By using the learned features from the first task as a starting point, the model can learn more quickly and effectively on the second task. This can also help to prevent overfitting, as the model will have already learned general features that are likely to be useful in the second task.

Many deep neural networks trained on images have a curious phenomenon in common: in the early layers of the network, a deep learning model tries to learn a low level of features, like detecting edges, colours, variations of intensities, etc. Such kind of features appear not to be specific to a particular dataset or a task because no matter what type of image we are processing either for detecting a lion or car. In both cases, we have to detect these low-level features. All these features occur regardless of the exact cost function or image dataset. Thus, learning these features in one task of detecting lions can be used in other tasks like detecting humans.

A general summary of how transfer learning works:

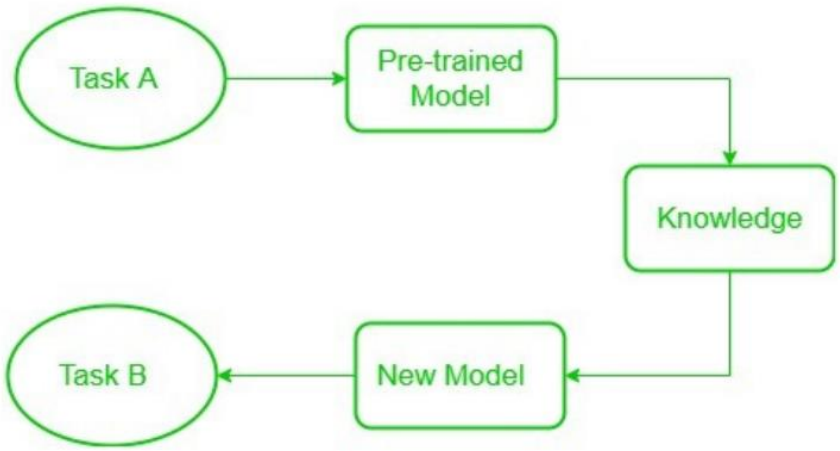
Pre-trained Model: Start with a model that has previously been trained for a certain task using a large set of data. Frequently trained on extensive datasets, this model has identified general features and patterns relevant to numerous related jobs.

Base Model: The model that has been pre-trained is known as the base model. It is made up of layers that have utilized the incoming data to learn hierarchical feature representations.

Transfer Layers: In the pre-trained model, find a set of layers that capture generic information relevant to the new task as well as the previous one. Because they are prone to learning low-level information, these layers are frequently found near the top of the network.

Fine-tuning: Using the dataset from the new challenge to retrain the chosen layers. We define this procedure as fine-tuning. The goal is to preserve the knowledge from the pre-training while enabling the model to modify its parameters to better suit the demands of the current assignment.

The Block diagram is shown below as follows:

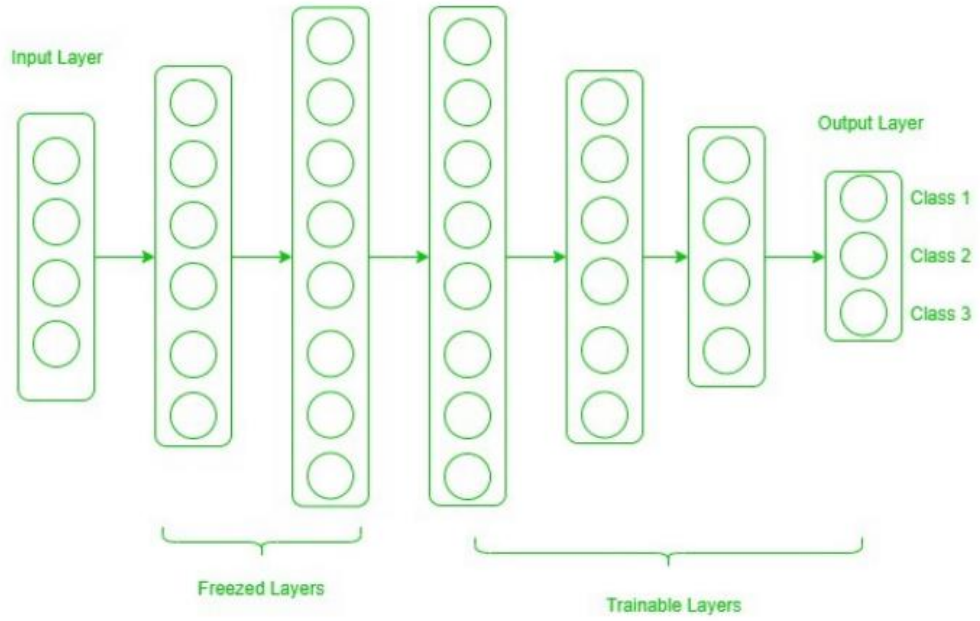


Transfer Learning

Low-level features learned for task A should be beneficial for learning of model for task B. When dealing with transfer learning, we come across a phenomenon called the freezing of layers. A layer, it can be a CNN layer, hidden layer, a block of layers, or any subset of a set of all layers, is said to be fixed when it is no longer available to train. Hence, the weights of freeze layers will not be updated during training. While layers that are not

frozen follows regular training procedure. When we use transfer learning in solving a problem, we select a pre-trained model as our base model. Now, there are two possible approaches to using knowledge from the pre-trained model. The first way is to freeze a few layers of the pre-trained model and train other layers on our new dataset for the new task. The second way is to make a new model, but also take out some features from the layers in the pre-trained model and use them in a newly created model. In both cases, we take out some of the learned features and try to train the rest of the model. This makes sure that the only feature that may be the same in both of the tasks is taken out from the pre-trained model, and the rest of the model is changed to fit the new dataset by training.

Freezed and Trainable Layers:



Now, one may ask how to determine which layers we need to freeze, and which layers need to train. The answer is simple, the more you want to inherit features from a pre-trained model, the more you have to freeze layers. For instance, if the pre-trained model detects some flower species and we need to detect some new species. In such a case, a new dataset with new species contains a lot of features similar to the pre-trained model. Thus, we freeze less number of layers so that we can use most of its knowledge in a new model. Now, consider another case, if there is a pre-trained model which detects humans in images, and we want to use that knowledge to detect cars, in such a case where the dataset is entirely different, it is not good to freeze lots of layers because freezing a large number of

layers will not only give low level features but also give high-level features like nose, eyes, etc which are useless for new dataset (car detection). Thus, we only copy low-level features from the base network and train the entire network on a new dataset.

Let's consider all situations where the size and dataset of the target task vary from the base network.

The target dataset is small and similar to the base network dataset: Since the target dataset is small, that means we can fine-tune the pre-trained network with the target dataset. But this may lead to a problem of overfitting. Also, there may be some changes in the number of classes in the target task. So, in such a case we remove the fully connected layers from the end, maybe one or two, and add a new fully connected layer satisfying the number of new classes. Now, we freeze the rest of the model and only train newly added layers.

The target dataset is large and similar to the base training dataset: In such cases when the dataset is large, and it can hold a pre-trained model there will be no chance of overfitting. Here, also the last full-connected layer is removed, and a new fully-connected layer is added with the proper number of classes. Now, the entire model is trained on a new dataset. This makes sure to tune the model on a new large dataset keeping the model architecture the same.

The target dataset is small and different from the base network dataset: Since the target dataset is different, using high-level features of the pre-trained model will not be useful. In such a case, remove most of the layers from the end in a pre-trained model, and add new layers a satisfying number of classes in a new dataset. This way we can use low-level features from the pre-trained model and train the rest of the layers to fit a new dataset. Sometimes, it is beneficial to train the entire network after adding a new layer at the end.

The target dataset is large and different from the base network dataset: Since the target network is large and different, the best way is to remove the last layers from the pre-trained network and add layers with a satisfying number of classes, then train the entire network without freezing any layer.

Transfer learning is a very effective and fast way, to begin with, a problem. It gives the direction to move, and most of the time best results are also obtained by transfer learning.

4.5. Data augmentation:

Data augmentation is the process of increasing the amount and diversity of data. We do not collect new data; rather we transform the already present data. I will be talking specifically about image data augmentation in this article. So we will look at various ways to transform and augment the image data.

This article covers the following articles –

1. Need for data augmentation
2. Operations in data augmentation
3. Data augmentation in Keras

1. Need for data augmentation: Data augmentation is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue. It helps us to increase the size of the dataset and introduce variability in the dataset.

2. Operations in data augmentation:

The most commonly used operations are

1. Rotation: Rotation operation as the name suggests, just rotates the image by a certain specified degree.

2. Shearing: Shearing is also used to transform the orientation of the image.

3. Zooming: Zooming operation allows us to either zoom in or zoom out.

4. Cropping: Cropping allows us to crop the image or select a particular area from an image.

5. Flipping: Flipping allows us to flip the orientation of the image. We can use horizontal or vertical flip. Flipping You should use this feature carefully as there will be scenarios where this operation might not make much sense e.g. suppose you are designing a facial recognition system, then it is highly unlikely that a person will stand upside down in front of a camera, so you can avoid using the vertical flip operation.

6. Changing the brightness level: This feature helps us to combat illumination changes.You can encounter a scenario where most of your dataset comprises of images having a similar brightness level e.g. collecting the images of employees entering the office, by augmenting the images we make sure that our model is robust and is able to detect the person even in different surroundings.

3. Data augmentation in Keras: Keras is a high-level machine learning framework build on top of TensorFlow. I won't go into the details of the working of Keras, rather I just want to introduce the concept of data augmentation in Keras. We can perform data augmentation by using the ImageDataGenerator class. It takes in various arguments like – rotation_range, brightness_range, shear_range, zoom_range etc.

****