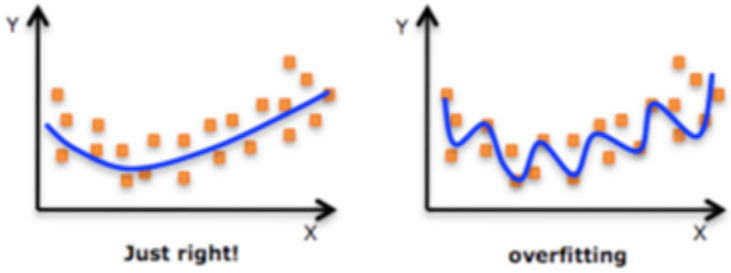
What is Overfitting?

Overfitting occurs when a model is too complex and learns irrelevant details from the training data, causing poor performance on new data.

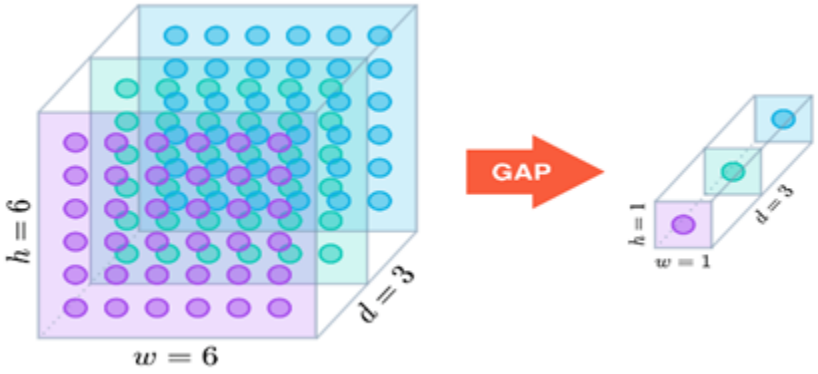


Methods to avoid overfitting

1. **Regularization:** L1/L2, Dropout reduces model complexity, and prevents overfitting.
2. **Early Stopping:** Stop training when validation performance declines.
3. **Data Augmentation:** Apply random image transformations to increase the dataset.
4. **Transfer Learning:** Use pre-trained models and fine-tune.
5. **Model Ensemble:** Combine multiple models for robust results.
6. **Reduce Network Complexity:** Use smaller filters and fewer layers.

Global Average pooling

Global Average Pooling 2D is a CNN pooling operation that averages feature maps, reducing spatial dimensions. It captures global spatial information, reducing overfitting and computational cost.



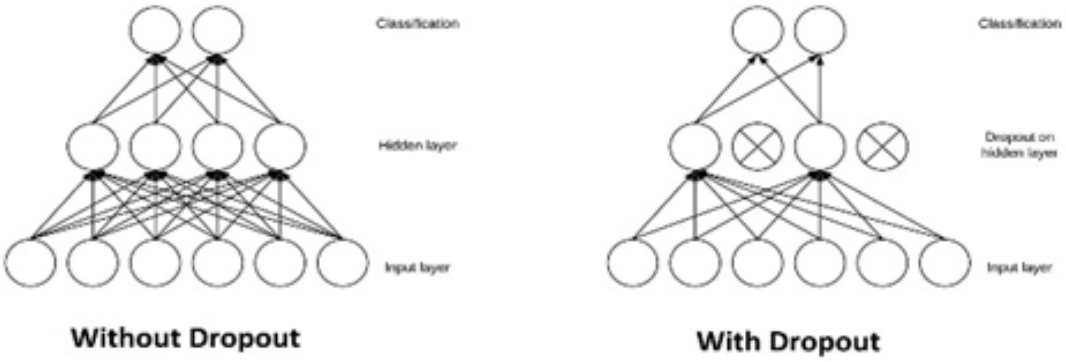
Batch Normalization

Batch normalization normalizes neuron activations within batches, improving training stability and speed. It transforms each batch's activations to zero mean and unit standard deviation before passing to the next layer.

$$\hat{x}^{(k)} \leftarrow \frac{x_i^{(k)} - \mu^{(k)}}{\sqrt{\sigma^{2(k)} + \epsilon}}$$

Dropout

Dropout is a deep learning regularization technique that randomly drops out some neurons during training to prevent overfitting, encouraging the network to learn multiple representations of data.



Without Dropout | With Dropout

When to use batch normalization and dropout?

- Empirically, dropout is most effective placed between Dense layers with 0.5 probability. Lower probability (0.1-0.25) dropout after MaxPooling improves performance.

- Normalizing feature distribution after Conv Layer and passing it to ReLU block clamps negative values to 0, defeating normalization's purpose.

L1 Regularization VS L2 Regularization

- L1 Regularization, aka Lasso, adds a penalty to the loss function based on weight magnitude. It reduces feature count in high-feature models.

- L2 Regularization, aka Ridge, adds a weight magnitude-based penalty to the loss function. It's often used in deep learning to prevent overfitting without inducing sparsity.

Ridge Regularization(L2)

$$loss = \sum_{i=0}^{n}(y_i - X_i\beta)^2 + \sum_{j=0}^{m}\beta_j^2$$

Lasso Regularization(L1)

$$loss = \sum_{i=0}^{n}(y_i - X_i\beta)^2 + \sum_{j=0}^{m}|\beta_j|$$

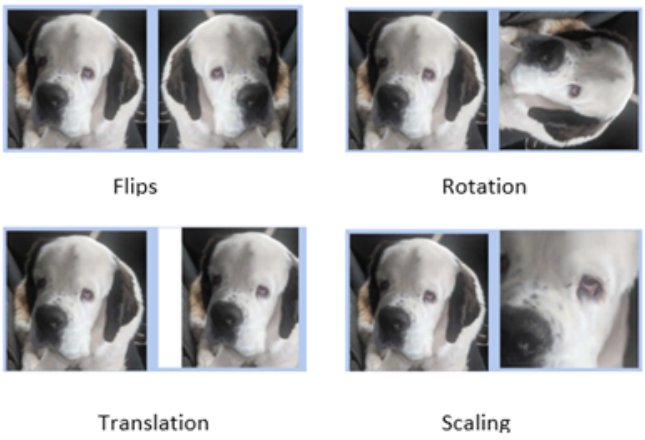
Data Augmentation

Data augmentation is a technique used in Convolutional Neural Networks (CNNs) to artificially increase the size of the training data by applying various transformations to the input images.

The purpose of data augmentation is to prevent overfitting, which occurs when the model is too complex and learns the training data too well, causing poor generalization to new, unseen data.

Examples of data augmentation techniques used in CNNs include:
Rotation: Rotating the images by a random angle.
Scaling: Resizing the images to a different scale.
Translation: Shifting the images horizontally or vertically by a random amount.
Flipping: Flipping the images horizontally or vertically.

Examples of Data Augmentation



Ways to Apply Data Augmentation

There are many ways to apply augmentation in Tensorflow/Keras, few of them are discussed here:

1. using the **Keras** Preprocessing Layers, just like preprocessing functions like resizing and rescaling, keras also provides data augmentation layers like **tf.keras.layers.RandomFlip**, **tf.keras.layers.RandomRotation**, etc. These can be used similarly as we used the preprocessing functions.

2. using **tf.image** methods like tf.image.stateless_random_flip_up_down, tf.image.stateless_random_brightness

3. using Keras **ImageDataGenerator** API - It provides quick, out-of-the-box suite of augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.