CNN - Convolutional Neural Networks

Convolutional neural networks, or CNNs, are specialised architectures that work particularly well with visual data, i.e., images and videos. A CNN is a multistack layer of simple modules: convolution layers, activation functions, Max pooling, Dense and output predictions.

Common Interview Questions

1. What is the job of the convolution layers?
2. What is the key job of activation functions?
3. What are the different activation functions?
4. What is the max pooling operation?
5. What are hyperparameters in fine-tuning CNNs?
6. Why should all weights not be initialised to zero or to the same value?
7. What does batch normalisation do?
8. What are the different augmentation techniques ?
9. What are the different optimisers used in building a CNN?
10. What are the differences between Conv1D, Conv2D and Conv3D?

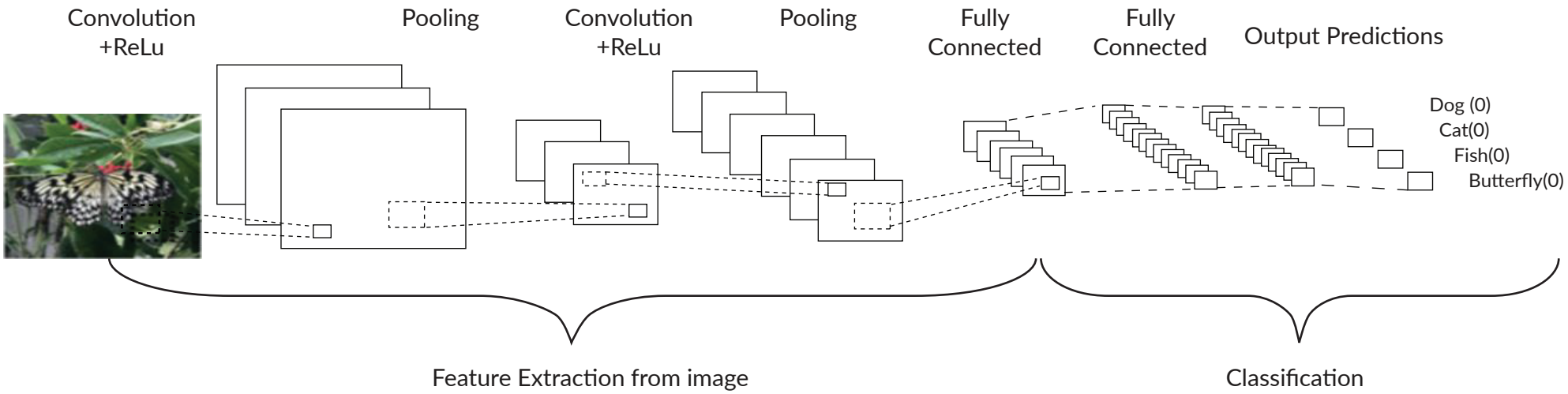
Common Interview Questions

- Convolution
- Activation Function
- Pooling
- Fully Connected Layer
- Classification Layer
- Regularisation
- Optimisers
- Hyperparameters
- Loss Function

A convolutional neural network (CNN) is a stack of layers. These include feature extraction layers (convolution, activation, max pooling[AC1]) and classification layers, which comprise [AC2] fully connected, dense[AC3] and softmax layers. The key learning steps are as follows:

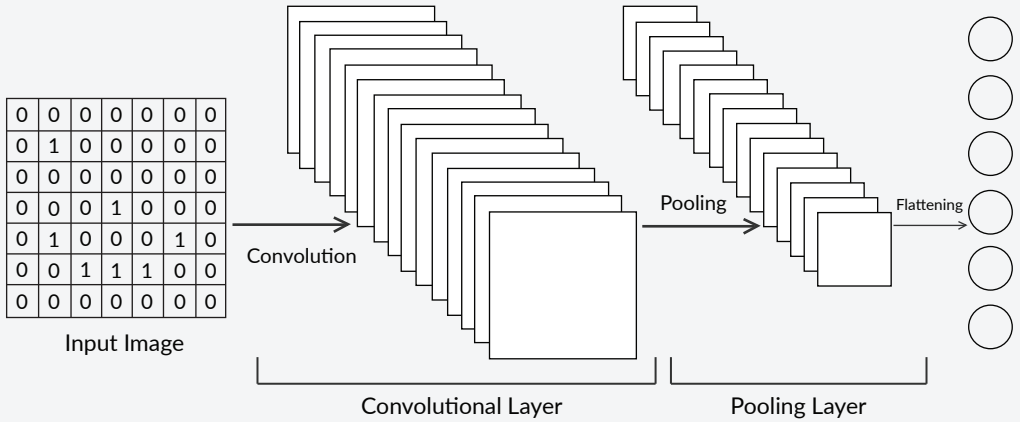
- **Initialisation** — Initial weights are applied to all neurons.
- **Forward propagation** — The inputs from a training set are passed through a neural network and an output is computed.
- **Error function** — It captures the delta between the correct output and the actual output of the model.
- **Backpropagation** — The objective of backpropagation is to change the weights for the neurons. The amount of error in the neurons in the output layer is propagated back to the preceding layers.
- **Weight updating** — Weights are changed to the optimal values according to the results of the backpropagation algorithm.
- **Iteration until convergence** — Because weights are updated a small delta step at a time, the network requires several iterations to learn.

Stages of CNN



Convolution - Key Layers

- CNN is a sliding window process. Linear filter applied in the sliding window manner.
- Its hyperparameters include the filter size, F, and stride S.
- Convolve the filter with an image (Slide over the image spatially computing dot products).
- Convolution Filters – are usually stacks of 3 x 3 convolutions.
- The Convolution layers handle the complex nonlinearly separable relations between input and output by introducing an activation function.
- The resulting output is called a feature map or activation map.
- The output from the convolutional layers represents high-level features in the data.



Advantages of CNN

- CNNs are very good feature extractors, which automatically detect important features,
- CNNs are primarily credited for their role in image processing.

Stride

- How the filter is moved across the image
- Stride = 1 move by one position, Stride =2 move by two positions

Convolution Formula

$$\text{output size} = \left[\frac{(\text{intput size}) + 2^{*}\text{padding} - (\text{kernel size} - 1) - 1}{\text{Stride}} + 1 \right]$$

ACTIVATION FUNCTIONS

- The activation function serves as a gate between the current neuron input and its output
- The activation Function is used to introduce non-linearity into a model
- We need non-linearity, to capture more complex features and model more complex variations that simple linear models can not capture.

Softmax -

- The essential goal of softmax is to turn numbers into probabilities that sum up to 1 across all predictions.
- This is normally used for obtaining classification predictions/probabilities

```
import numpy as np
#softmax of y_i is simply the exponential divided by the sum
of exponential of the whole Y vector
def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    return np.exp(x) / np.sum(np.exp(x), axis=0)


#Input values
scores = [7.0, 2.0, 1.2]
#Scores distribution resaled to sum up to 1
print(softmax(scores))

[0.99032894 0.00667278 0.00299828]
```

```
import tensorflow as tf

#input tensorvalues
vector [0.,-1.5,1.5,2.5,-3.5,-0.1]

#use relu to linear data by replacing negative values with o
r = tf.nn.relu (vector)

#o to 1 range for input vector values
s = tf.nn.sigmoid(vector)

#-1 to 1 range for input vector values
t = tf.nn.tanh (vector)

#print input vector
print(vector)

#Relu Vector
print('Relu Activation Output')
print(r)

#Sigmoid output
print('Sigmoid Activation Output')
print(s)

#Tanh output
print('Tanh Activation Output')
print(t)
```

```
[0.0, 1.5, 1.5, 2.5, 3.5, -0.1]
Relu Activation Output
tf.Tensor([0. 0. 1.5 2.5 0. 0. ], shape=(6,), dtype=float32)
Sigmoid Activation Output
tf.Tensor([0.5         0.18242551 0.8175745 0.9241418 0.02931223
0.47502083], shape=(6,), dtype=float32)
Tanh Activation Output tf.Tensor([ 0.    .9051482 0.9051482
0.9866143 -0.9981779 -0.09966799], shape=(6,), dtype=float32)
```

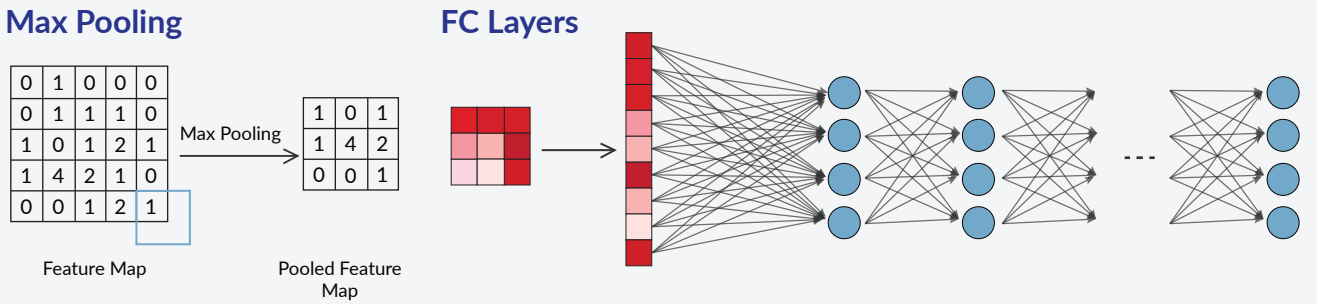
POOLING OPERATION AND OPTIMIZERS USED IN TRAINING

Pooling - Pooling does not preserve the features, rather it sends only important data in the next layer

MAX Pooling - It sends the maximum value of regions across the image to the next layeronly important data in the next layer

Flatten - It converts literally any input to a 1D array to submit it to a dense layer.

Dense / fully connected layer (FC) - Every neuron of the dense layer is connected to every element of its input.



Optimizer

- Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses.
- Momentum takes into account past gradients to smooth out the update.
- RMSProp - Root Mean Square Prop (RMSProp). RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations.
- Adaptive Moment Estimation (Adam) combines ideas from both RMSProp and Momentum.
- Adagrad (short for adaptive gradient) adaptively sets the learning rate according to a parameter
- Adadelta is probably short for 'adaptive delta', where delta here refers to the difference between the current weight and the newly updated weight.

Loss Functions

Probabilistic Losses	Regression Losses	Hinge Losses
	Mean Squared Error (MSE)	Hinge
	Mean Absolute Error (MAE)	Squared Hinge
	Mean Absolute Percentage Error (MAPE)	Categorical Hinge
	Mean Squared Logarithmic Error	
	Cosine Similarity	
	Huber	
	Log-Cosh	

REGULARIZATION TECHNIQUES

Regularization techniques ---

Dropouts : Dropout() is an effective (and fast) way to reduce overfitting. Randomly replaces some values with 0, and increases the remaining values to compensate.

Data Augmentations: By rotating/shifting/zooming/flipping/padding etc. an image you make sure that your model is forced to train its parameters better nd not overfit to the existing dataset.

Early Stopping: This is a quite common technique for avoiding training your model too much

L1 penalties: L1 loss function stands for Least Absolute Deviations. This tries to ensure that all parameters will be equally taken into consideration when classifying an input.

L2 penalties: Called Euclidean norm. Simply the Euclidean distance between the origin and the point x. L2 loss function stands for Least square errors

```
#Dropout is a regularization technique for reducing overfitting
in neural networks
import tensorflow as tf
x = [1.,2.,1.,4,1]
#0.1 is with probability rate elements of x are set to 0
result = tf.nn.dropout (x,0.1)
print((result))
tf.Tensor([1.1111112 2.2222223 0.    4.4444447 0.   ],
shape=(5,), dtype=float32)
```

$$Cost\ function = Loss + \frac{\lambda}{2m} * \Sigma \|w\|$$

$$Cost\ function = Loss + \frac{\lambda}{2m} * \Sigma \|w\|^2$$

HYPERPARAMETER TUNING

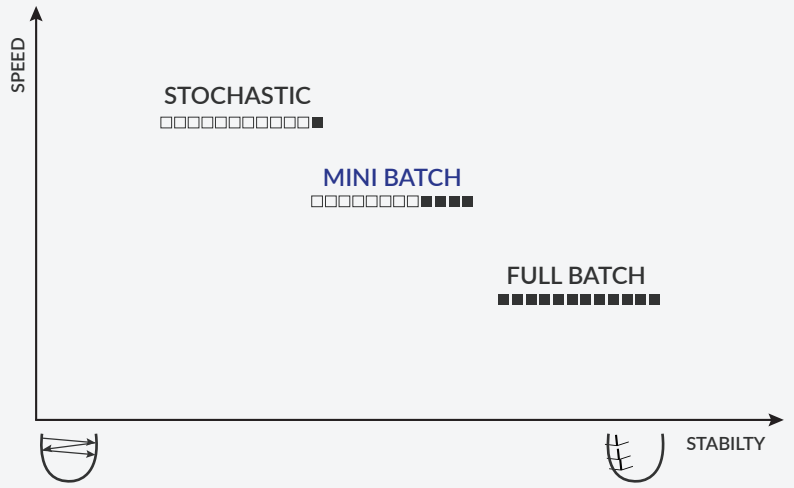
Key Parameters

- he number of hidden layers, the number of units in each layer,
- Regularization techniques, network weight initialization,
- Activation functions, learning rate, momentum values,
- Number of epochs, batch size (minibatch size), decay rate, optimization algorithms

Gradient Descent

Most common optimization algorithm.

- The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network.
- This is normally done using backpropagation.
- Backward propagation = chain rule differentiation. AI is a nothing but usage of math applied creatively for computing.
- Batch Gradient Descent
- Mini-batch Gradient Descent
- Stochastic Gradient Descent



Hyperparameters tuning - Key Parameters

- he number of hidden layers, the number of units in each layer,
- Regularization techniques, network weight initialization,
- Activation functions, learning rate, momentum values,
- Number of epochs, batch size (minibatch size), decay rate, optimization algorithms

Gradient Descent

Most common optimization algorithm.

- The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network.
- This is normally done using backpropagation.
- Backward propagation = chain rule differentiation. AI is a nothing but usage of math applied creatively for computing.
- Batch Gradient Descent
- Mini-batch Gradient Descent
- Stochastic Gradient Descent