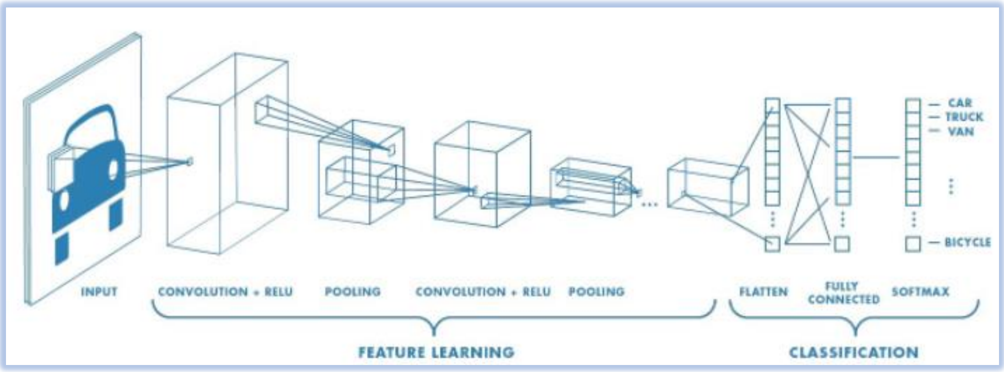


Convolutional Neural Network

- Deep learning algorithm
- Takes input image
- Assigns importance (learnable weights and biases) to aspects/objects in the image
- Able to differentiate aspects/objects from one another

Input Images

- Reduce each image to size of 64x64x3 (pixels)
- Converted image space from RGB to HSV (Hue, Saturation, Value)
    - Color spaces for images include: Grayscale, RGB, HSV, CMYK, etc.
    - Color space can affect model accuracy

1. Feature Learning

a. Convolution
b. Pooling
c. Dropout
d. Batch Normalization

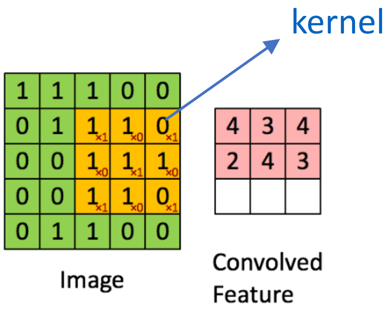
a. Convolution

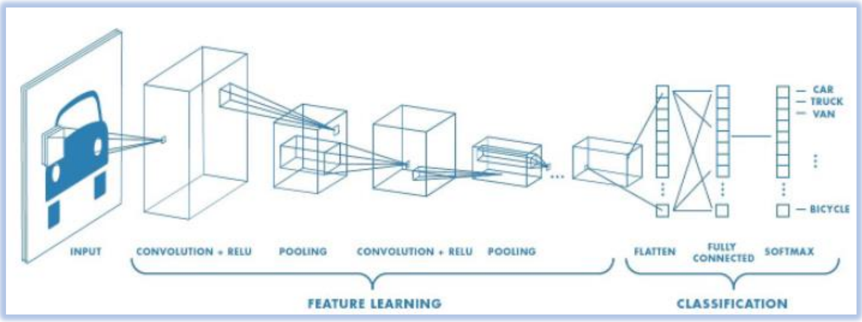
- mathematical operation
- Image Tensor x Filter (kernel) Tensor =
- Convolved Feature

Convolutional Layer – The Kernel

- Input: Image
- Filtered with Kernel
- Output: Convolved feature

- Specific kernels to extract low and high-level features
    - edges, color, gradients, etc.
    - Added layers produce higher-level features





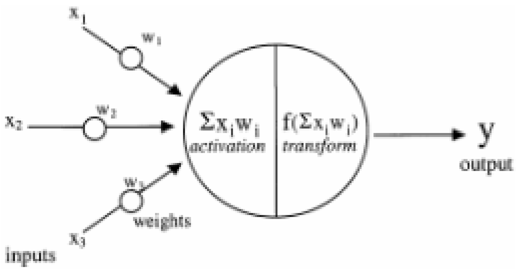

```
# First Convolutional layer with 32 filters and kernel size of 2. Use the 'same' padding and input shape of 64*64*3
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu",input_shape=(64,64,3)))
```

2 x 2 filter

"same" the output size of image = input size
padding types: zero, full, same

Activation Function

- Decides if a neuron should be activated
- Introduces non-linearity to neuron output
    - Sigmoid
    - Tanh
    - Rectified Linear (Relu)



b. Pooling Layer

- Applied after convolutional layer
- Reduce the size of the feature - minimize computation requirement)
- **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel.

```
# max-pooling layer with a pool size of 2
model.add(MaxPooling2D(pool_size=2))
```

2 x 2 filter

c. Dropout

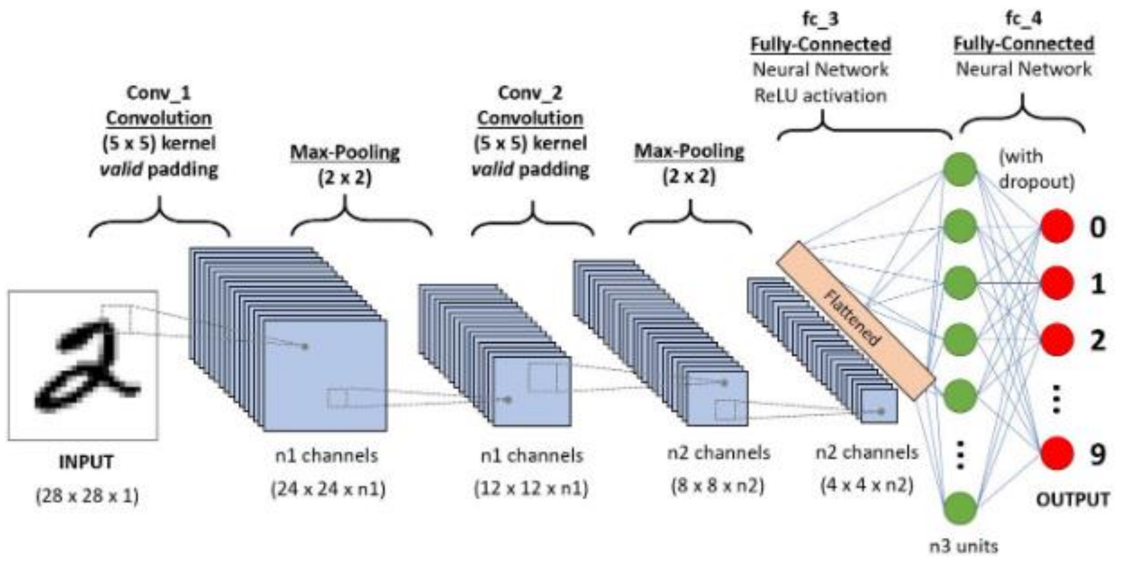
- To prevent overfitting of the training dataset
- Some number of layer outputs are randomly ignored or "*dropped out*."
- Implemented per-layer in a neural network.

```
# Add dropout to randomly switch off 20% neurons to reduce overfitting
model.add(Dropout(0.2))
```

d. Batch Normalization

- Accelerates the training of neural networks
    - During training, input from prior layers constantly changes after weight updates
    - In some way, standardizes the inputs to a layer for each pass.

```
#BatchNormalization layer
model2.add(BatchNormalization())
```



2. Classification (fully connected layer)

- a. Flatten
- b. Hidden Layers
- c. Dropout
- d. Optimizer

a. Flatten

- Output from feature learning is a matrix.
- Flatten the matrix into a vector to feed into the neural network
- Inputs into a fully connected network of neurons

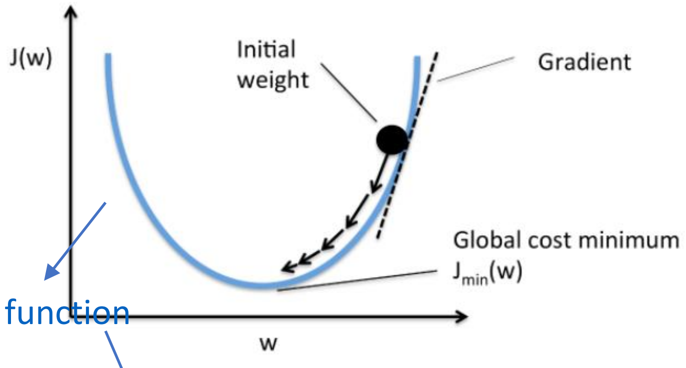
```
# Flatten the output from the previous layer
model2.add(Flatten())
```

b. Hidden Layers with c. Dropout

```
# Hidden Layer 1
model2.add(Dense(512, activation=activation_f))
model2.add(Dropout(0.2))

# Hidden Layer 2
model2.add(Dense(512, activation=activation_f))
model2.add(Dropout(0.1))

# Output layer with nodes equal to the number of classes and softmax activation
model2.add(Dropout(0.1))
model2.add(Dense(2,activation="softmax"))

# Define Optimizer
adam = optimizers.Adam(learning_rate=0.001)
```

a. Optimizer

- Algorithms used to change the attributes of your neural network (weights, learning rate, etc.)
- in order to reduce the losses (minimize loss function)
- Adam, Adamax, SGD, etc.



Loss function

```
model4.compile(loss='categorical_crossentropy',
               optimizer=optimizer,
               metrics=['accuracy'])
```