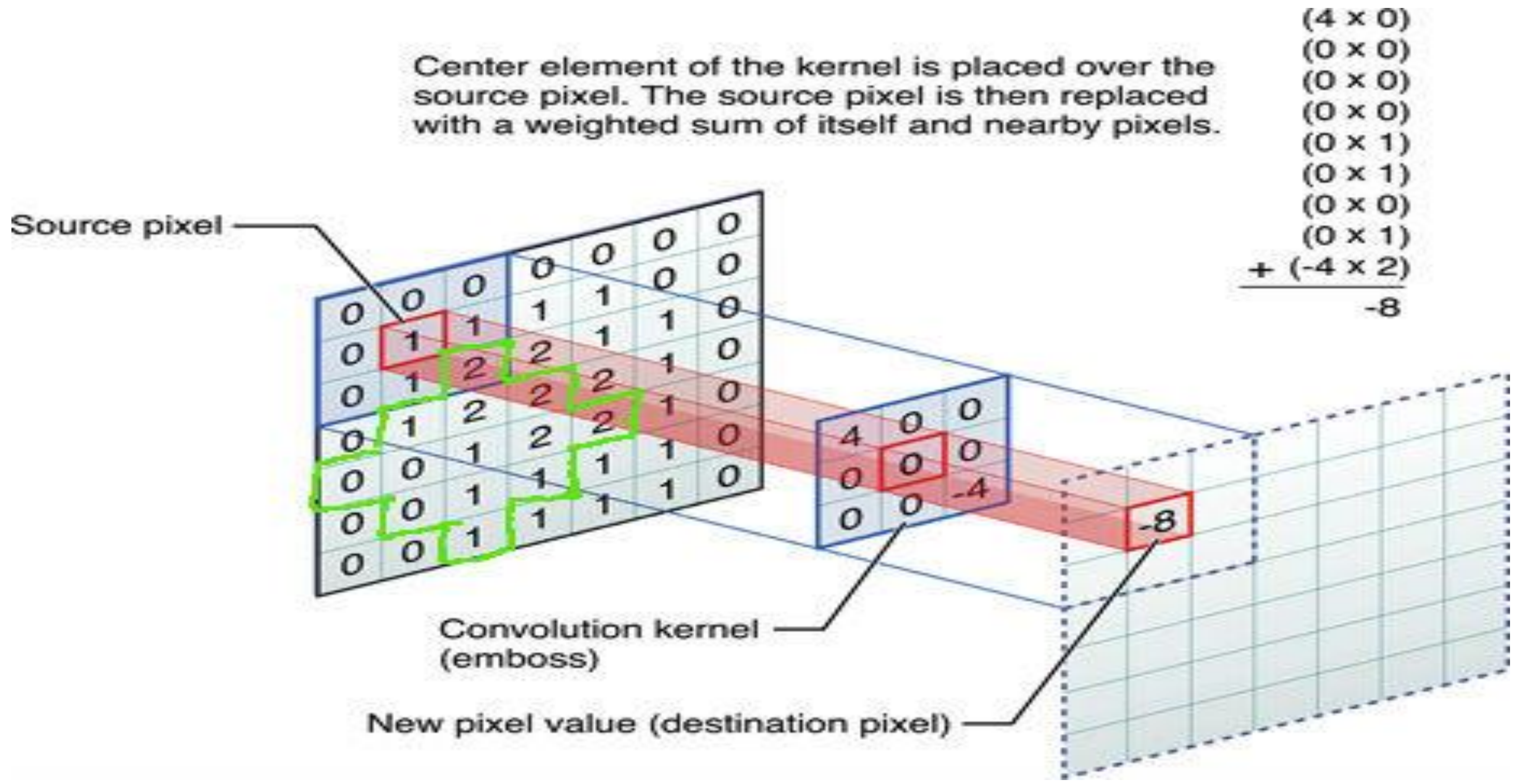
Convolutional Neural Networks

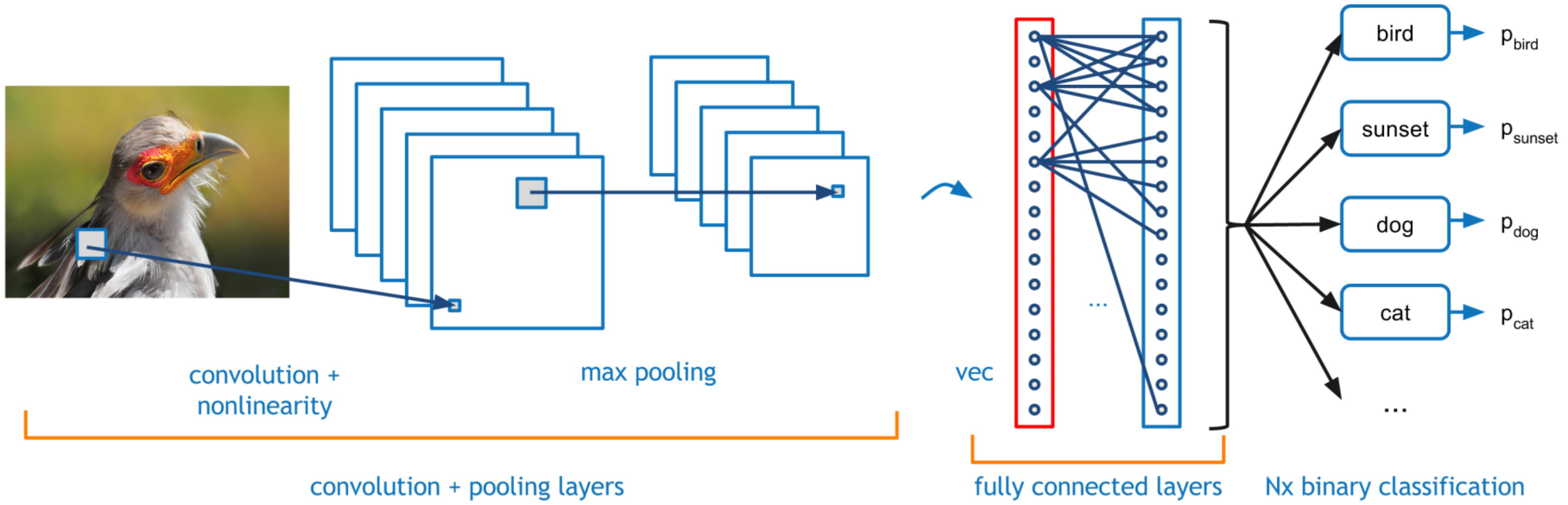
Convolution

- Convolution Operation involves a matrix arithmetic operations



Pooling

- After the convolution, there is another operation called pooling
- So, in chain, convolution and pooling is applied sequentially on the data in the interest of extracting some features from the data
- After the sequential convolutional and pooling layers, the data is flattened into feed forward neural network



Keras Library

- Most popular library for CNN implementation is keras developed by Francois Chollet.
- Library has been written in Python and also has a corresponding R package

Setting the sequential run

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Convolution2D(32,( 3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

(Optional) Add a second convnet layer

```
# Adding a second convolutional layer
classifier.add(Convolution2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Flattening the convnet layers

- After the convnet layers, we flatten the data and then apply feed forward neural network
- We setting the activations for various hidden layers with Dense()
- Specify the loss function, optimizer and metrics

```
# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Compilation Specs

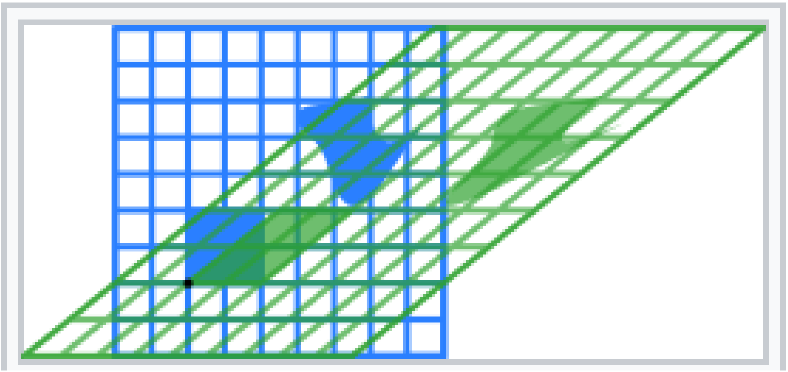
- A loss function— How the network will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction.
- An optimizer— The mechanism through which the network will update itself based on the data it sees and its loss function.
- Metrics to monitor during training and testing— Here, we'll only care about accuracy (the fraction of the images that were correctly classified).

Image Processing

- We specify the shear and zoom ratios
- Shear Intensity is Shear angle in counter-clockwise direction in degrees

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```



Scaling and specifying the images path

```
test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

Actually Building the model

```
classifier.fit_generator(training_set,
                         samples_per_epoch = 8000,
                         nb_epoch = 25,
                         validation_data = test_set,
                         nb_val_samples = 2000)
```

Program Run

```
Epoch 1/25
250/250 [==============================] - 492s 2s/step - loss:
0.6773 - acc: 0.5789 - val_loss: 0.6487 - val_acc: 0.6106
Epoch 2/25
250/250 [==============================] - 304s 1s/step - loss:
0.6175 - acc: 0.6598 - val_loss: 0.6014 - val_acc: 0.6770
Epoch 3/25
250/250 [==============================] - 305s 1s/step - loss:
0.5784 - acc: 0.7006 - val_loss: 0.5519 - val_acc: 0.7151
Epoch 4/25
250/250 [==============================] - 300s 1s/step - loss:
0.5406 - acc: 0.7262 - val_loss: 0.5268 - val_acc: 0.7356
Epoch 5/25
250/250 [==============================] - 310s 1s/step - loss:
0.5184 - acc: 0.7418 - val_loss: 0.5068 - val_acc: 0.7560
Epoch 6/25
250/250 [==============================] - 307s 1s/step - loss:
0.4993 - acc: 0.7574 - val_loss: 0.4832 - val_acc: 0.7680
```

Program Run

```
Epoch 21/25
250/250 [==============================] - 302s 1s/step - loss: 0.3198 -
acc: 0.8615 - val_loss: 0.4913 - val_acc: 0.7963
Epoch 22/25
250/250 [==============================] - 1286s 5s/step - loss: 0.2990 -
acc: 0.8714 - val_loss: 0.5009 - val_acc: 0.7900
Epoch 23/25
250/250 [==============================] - 305s 1s/step - loss: 0.2970 -
acc: 0.8736 - val_loss: 0.5132 - val_acc: 0.8018
Epoch 24/25
250/250 [==============================] - 306s 1s/step - loss: 0.2771 -
acc: 0.8846 - val_loss: 0.5175 - val_acc: 0.7982
Epoch 25/25
250/250 [==============================] - 305s 1s/step - loss: 0.2685 -
acc: 0.8858 - val_loss: 0.5206 - val_acc: 0.7973
```

Storing the Built Model

```
from keras.models import load_model
## Serializing
classifier.save('dog_cat_Identifier.h5')
```

Recalling the Stored Model

```
## Deserializing
model = load_model('dog_cat_Identifier.h5')
```