Object classification with CNNs using the keras Deep Learning Library.

A convolution multiplies a matrix of pixels with a filter matrix or ‘kernel’ and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered.

Dataset : CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60,000 photos divided into 10 classes (hence the name CIFAR-10). Classes include common objects such as airplanes, automobiles, birds, cats and so on. The dataset is split in a standard way, where 50,000 images are used for training a model and the remaining 10,000 for evaluating its performance.

The photos are in color with red, green and blue components, but are small measuring 32 by 32 pixel squares.

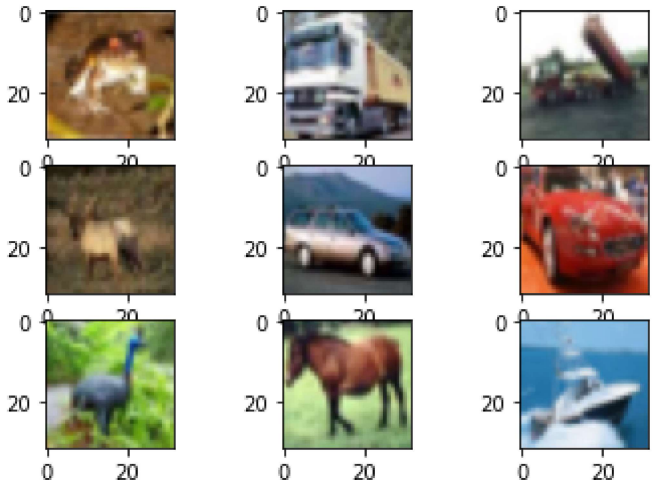
Source (https://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/)

Loading The CIFAR-10 Dataset in Keras

In [1]:
```
# Plot ad hoc CIFAR10 instances
from keras.datasets import cifar10
from matplotlib import pyplot
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# create a grid of 3x3 images
for i in range(0, 9):
        pyplot.subplot(330 + 1 + i)
        pyplot.imshow(X_train[i])
# show the plot
pyplot.show()
```

```
Using TensorFlow backend.

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
```



In [2]:
```
X_train[0].shape
```

Out[2]:
```
(32, 32, 3)
```

CNN basics

The Convolutional neural networks are regularized versions of multilayer perceptron (MLP). They were developed based on the working of the neurons of the animal visual cortex.

In a regular Neural Network there are three types of layers:

1) Input Layer: It's the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels incase of an image).

2) Hidden Layer: The input from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.

3) Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into probability score of each class.

The data is then fed into the model and output from each layer is obtained this step is called feedforward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

Source (https://www.geeksforgeeks.org/introduction-convolution-neural-network/)

Simple Convolutional Neural Network for CIFAR-10

In [3]:
```
# Simple CNN model for CIFAR-10
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
...
```

Out[3]:
```
Ellipsis
```

In [0]:
```
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [0]:
```
#Note, the data is loaded as integers, so we must cast it to floating point va
lues in order to perform the division.

# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
```

We can use a one hot encoding to transform them into a binary matrix in order to best model the classification problem. We know there are 10 classes for this problem, so we can expect the binary matrix to have a width of 10.

In [0]:
```
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

In [0]:
```
# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), activation='relu', paddi
ng='same'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

In [8]:
```
# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accura
cy'])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
```

In [10]:
```
# Fit the model
mnist_train=model.fit(X_train, y_train,validation_data=(X_test, y_test), epochs=3)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/3
50000/50000 [==============================] - 444s 9ms/step - loss: 1.1221 - accuracy: 0.5966 - val_loss: 1.0305 - val_accuracy: 0.6292
Epoch 2/3
50000/50000 [==============================] - 444s 9ms/step - loss: 1.0076 - accuracy: 0.6443 - val_loss: 0.9506 - val_accuracy: 0.6629
Epoch 3/3
50000/50000 [==============================] - 443s 9ms/step - loss: 0.9138 - accuracy: 0.6756 - val_loss: 0.8648 - val_accuracy: 0.6951
```

In [11]:
```
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 69.51%
```