Neural networks can be implemented in various applications. In the case of this code we will be performing image classifications using 2-D convolution layers to extract various features. The dataset is Cifar-10 which contains 10 classifications of objects that are 32x32 in size of the type RGD. A kernel within the convolutional layers which serves the purpose of identifying pixel characters, in the case of this model a consistent 3x3 kernel was used with zero padding meaning the kernel covers every individual pixel with no exclusions.

**Importing CIFAR-10 Dataset**

**Graphical user interface, text, application

Description automatically generated**

Below is the code to load cifar10 to training and testing set

**Graphical user interface, text, application

Description automatically generated**

**Verifying data**

Below is a method of classifying and showing the 10 image types based off their labels.

Text

Description automatically generated

**Baseline Model + dropout**

It is important to understand the configuration of the baseline model as it establishes the foundation for the variations built on top of this baseline model using increasing dropout layers.

At the beginning of the hidden layers we see the first convolutional layers. There is a duo 2-D convolutional layer with 32 feature extractions each, using a 3x3 kernal for feature identification, where the kernel has ‘same’ or zero padding; this allows for none of the features on the edges to be missed. The kernel is uniformly initialized. The first input is designed to handle 32x32 data from the Cifar-10 dataset and the “3” parameter within the “input\_shape” definition is used to signify RGB type input. If your dataset were greyscale and 64x64 in size the input\_shape=(64,64,3).

Following the dual convolutional layers the is a max pooling layer which essential takes the maximum pixel values within the matrix of the set pooling size. In this model we use 2x2 pooling and a stride of 0 meaning the pooling shifts over by 1 after the maximum is selected from the designated pooling.

Lastly there is a dropout layer following the maxpooling. This dropout essentially drops features passed through neurons. A dropout of .2 is equivalent to 20% dropout of neurons

**Table

Description automatically generated with medium confidence**

Cool! Now you have a basic understanding of the convolutional and pooling layers and their trend throughout the model we must finally cover the fully connected layer

**Graphical user interface, application

Description automatically generated**

There is a flattening layer also known as vectorization following that is a pooling layer of 128 neurons with a 20% dropout layer that is fully connected to the 10 outputs (10 for 10 classifications). The output neuron uses a SoftMax function because softmax is the ideal activation to use on image classification where as ReLu is used throughout the hidden layer and the final pooling layer at dense 128 neurons.

Below we can see the training process of the model. We can note that the model is working well when the loss value decreases and val\_accuracy aka test accuracy increases

Table

Description automatically generated

**Model Evaluation**

To evaluate the model, use the code below which generates a plot of accuracy and validation accuracy vs epoch. Also the values for loss of features and accuracy are shown.

Graphical user interface, application

Description automatically generated

**Baseline + Dropout + Data augmentation**

The trend remains the same. If you would like to make any modifications to the layers, consider the information provided in the previous section. Here we will discuss data augmentation.

Data augmentation helps to better train the model with variety in data. The code for data augmentation is quite self-explanatory. Various changes such as rotation, width shift, heigh shift and horizontal flipping are present at their respective scales

Text

Description automatically generated

**Baseline + Dropout + Data augmentation + Batch normalization**

Batch normalization is a technique for training that standardizes the inputs to a layer for each mini batch thus stabilizing the learning process and lowering the number of epochs required to train the network.

Some slight modifications were made with the models fully connected layer to gain better accuracy.

A picture containing text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Training your model**

When training the model you should use adam optimizer given the efficiency of it on the model and dataset however SGD is also reasonable if the network is tweaked in favor of it.

Here were a running the compilation using model. Compile where the batch size is 64 and there are 100 epochs.

In our case we only used the name “model” for all models. Take note of this when trying to compile certain models.

**Text

Description automatically generated**

**To show plots**

**Graphical user interface, text, application, email

Description automatically generated**