

# Image Classification documentation

---

- For the image classification modeling, we are using the TensorFlow framework

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks

To use TensorFlow in python for machine learning we import it using the given code

```
import tensorflow as tf
```

To check the version of TensorFlow we use the code given below

```
# Display the version  
print(tf.__version__)
```

We are using Numpy Library to convert the List into an Array in this project

```
import numpy as np
```

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Matplotlib library is being used to plot the graph of the accuracy and loss of the model

```
import matplotlib.pyplot as plt
```

Matplotlib is a comprehensive library for **creating static, animated, and interactive visualizations in Python.**

We are using the Keras layers API and using it we are importing

Input, Conv2D , Dense, Flatten, Dropout

```
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
```

**Input** : this method take shape of the input dataset

**Conv2D** : 2D convolution layer (e.g. spatial convolution over images).

**Dense** : it decides the number of neurons and connects the Neural network

**Flatten** : Flattens the input. Does not affect the batch size.

**Dropout** : It deactivates some parts of the neurons, which helps prevent overfitting

After we are done importing the required library, we can write a code to load the dataset  
For the image classification, we are using CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

```
cifar10 = tf.keras.datasets.cifar10
```

After the dataset is loaded we split the dataset into Train and Test sets. This is done using the code given below.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()  
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

### Training Dataset

x\_train is the feature of the train dataset

y\_train is the label of the train dataset

### Test Dataset

x\_test is the feature of the test dataset

y\_test is the label of the test dataset

### Visualize The Data

Now we can visualize the data we imported from CIFAR-10 using matplotlib library

```
# visualize data by plotting images  
fig, ax = plt.subplots(10, 10)
```

plt.subplot(10,10) decides the size of each image plotted  
This for loop creates an image of a 5x5 matrix

```
k = 0  
for i in range(5):  
    for j in range(5):  
        ax[i][j].imshow(x_train[k], aspect='auto')  
        k += 1  
plt.show()  
  
# number of classes
```

The below code displays the number of classes in the dataset

```
K = len(set(y_train))
# calculate the total number of classes
# for the output layer
print("number of classes:", K)
```

Python code to create a model for the image classification

```
# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dropout(0.2)(x)
# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
```

The below code is the last hidden layer that is using softmax as an activation

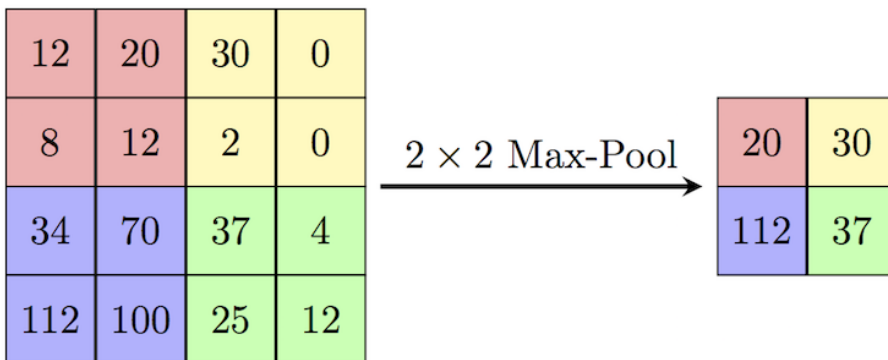
```
# last hidden layer i.e output layer
x = Dense(K, activation='softmax')(x)
model = Model(i, x)
```

`model.summary()` display the architecture of the model

```
# model description
```

```
model.summary()
```

- **Batch normalization** is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem
- **Max pooling** is a type of operation that is typically added to CNNs following individual convolutional layers. When added to a model, max pooling **reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer.**



- **Padding** is simply a process of adding layers of zeros to our input images so as to avoid the problems

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 |   |   |   |   |   |   | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Zero-padding added to image

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase.

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

## Model Training

```
r = model.fit(  
    x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model.

We have a limited number of datasets, so we can increase the dataset amount by rotating, moving, shifting zooming the same image and creating multiple variants of it



Now we can check how our model performs after the Train, it can be determined by plotting the graph of accuracy and val\_accuracy.

```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

### Testing with the labels displayed

```
# label mapping
labels = '''airplane automobile bird cat deerdog frog horseship
truck'''.split()
# select the image from our test dataset
image_number = 0
# display the image
plt.imshow(x_test[image_number])
# load the image in an array
n = np.array(x_test[image_number])
# reshape it
p = n.reshape(1, 32, 32, 3)
# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]
# load the original label
original_label = labels[y_test[image_number]]
# display the result
print("Original label is {} and predicted label is {}".format(
    original_label, predicted_label))
```

### Save the Model with .h5 extension

```
# save the model
model.save('imgclassification.h5')
```