About CIFAR – 10 Dataset

19 October 2024 20:10

CIFAR-10 dataset is one of the well-Known datasets in deep learning. & image classification tasks

It is often used as a benchmark dataset for testing and evaluating algorithms, especially in the context of CNNs.

overview of CIFAR – 10 Dataset

Dataset Size

60,000 color images

- 50,000 images for training
- 10,000 images for testing

Image Size:

all images are of size 32X32 pixels → 1024 pixels → X_i's vector having that many neurons.

32*32 =

images are colored → (RGB) → meaning each image has 3 channels.

Number of categories / classes:

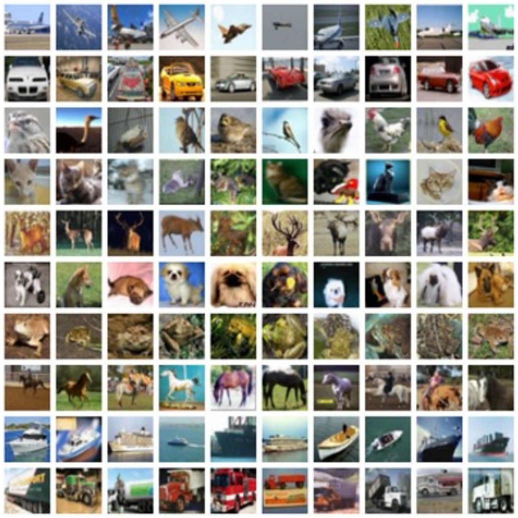
CIFAR- 10 dataset has 10 classes → representing objects or scenes or living things.

dataset is balanced → meaning there are 6000 images per class.

1 → 6000
2 → 6000
3 → 6000
⋮
10 → 6000

6000 x 10 → 60,000.

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



→ Multi-class classification task

The CIFAR-100 dataset

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 5(
Here is the list of classes in the CIFAR-100:

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Yes, I know mushrooms aren't really fruit or vegetables and bears aren't really carnivores.

Concept of dimensions

Array → ndim

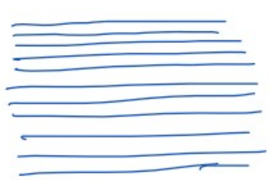
n-dimensions

1-D

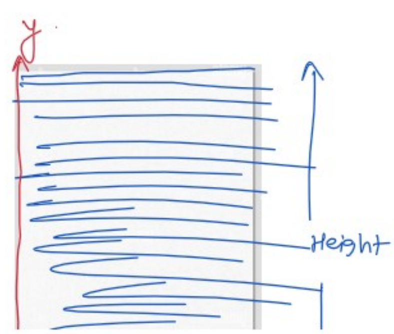
line

x axis

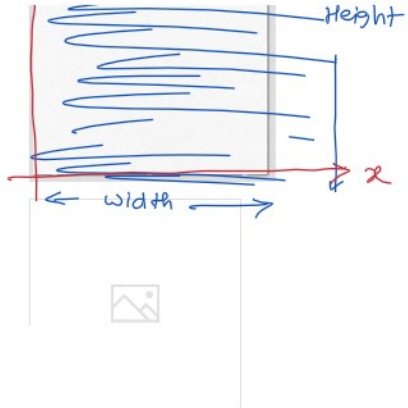
2-D

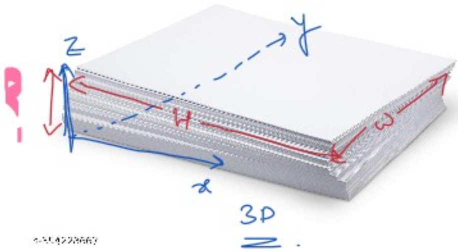


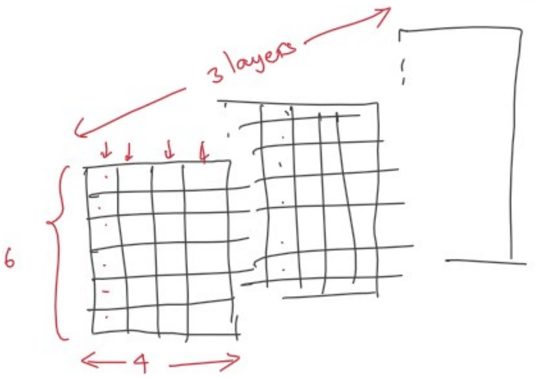
∞ times



∞ times







→ (3, 6, 4)

no. of filters #32

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (32,32,3))) ## added a 2D convolutional layer with 32 filters, each of size: 3 x 3, activ
model.add(layers.MaxPooling2D((2,2))) ### adds a max pooling layer with pool size of 2 x 2, to reduce the spatial dimension -- by half
model.add(layers.Conv2D(64, (3,3), activation = 'relu')) ### added a 2D convolutional layer with 64 filters, each of size: 3x3, activation function is 'ReLU'
model.add(layers.MaxPooling2D((2,2))) ### Adds another max pooling layer of similar size 2x2
model.add(layers.Conv2D(64, (3,3), activation='relu')) #added a 2D convolutional layer with 64 filters, each of size, 3x3, activation function is 'ReLU'
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_5 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_6 (Conv2D)	(None, 4, 4, 64)	36,928

Total params: 56,320 (220.00 KB)
Trainable params: 56,320 (220.00 KB)
Non-trainable params: 0 (0.00 B)

First layer: Conv2D

Input shape. (32, 32, 3)

32X32 image for 3 channels

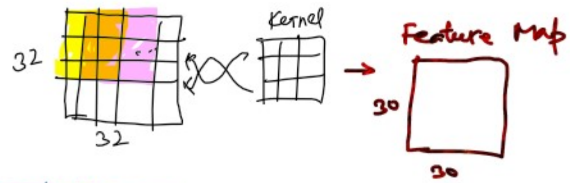
R, G, B.

?

output shape (30, 30, 32)

32 − 3 + 1 = 30   32 − 3 + 1 = 30

Filter size or kernel



- no padding
- default stride # 1

Number of parameters

Formula for the number of parameters in a conv2D layer is:

$$(\text{kernel height} \times \text{kernel width} \times \text{input channels} + 1) * \text{no. of filters}$$

bias

$$= ((3 \times 3 \times 3) + 1) * 32$$

$$= 28 * 32 = 28 * 32 = 896$$

Note: The additional '1' accounts for the bias term.

Second Layer: MaxPooling2D

Input shape: (30, 30, 32)

output shape: (15, 15, 32)

# of kernels | filters.

spatial dimensions are reduced by half because of 2X2 max pooling operation with stride # 2

No parameters are learnt in the MaxPooling layer, so param # = 0.

Compile and train the model

that uses adaptive learning rates for each parameter

(adaptive moment estimation)

− it is a variant of SGD − [Stochastic Gradient Descent]

```
model.compile(optimizer = 'adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

− for multi-class classification task

Class labels

Label Encoded: 0, 1, 2, 3, ⋮

sparse

```
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics = ['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

- for multi-class classification task

Label Encoded 2 3 ... 9

Sparse

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 44ms/step - accuracy: 0.2989 - loss: 2.5090 - val_accuracy: 0.4936 - val_loss: 1.3968
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 42ms/step - accuracy: 0.5361 - loss: 1.3004 - val_accuracy: 0.5417 - val_loss: 1.2944
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 41ms/step - accuracy: 0.6034 - loss: 1.1293 - val_accuracy: 0.5986 - val_loss: 1.1711
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 41ms/step - accuracy: 0.6429 - loss: 1.0336 - val_accuracy: 0.6170 - val_loss: 1.0992
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 65s 41ms/step - accuracy: 0.6760 - loss: 0.9368 - val_accuracy: 0.6638 - val_loss: 1.0036
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 41ms/step - accuracy: 0.6966 - loss: 0.8654 - val_accuracy: 0.6565 - val_loss: 1.0141
```

→ this is the function that the model is trying to minimize during training

$\boxed{\text{from\_logits = True}}$ → indicates that the model's output is raw logits (unnormalized output from the last layer)

↓

Keras will apply a softmax operation internally to convert logits into probabilities.

metrics = 'accuracy'

```
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

- model.fit() method is used to train the model.
- it adjusts the model's weights based on the training data using the optimizer and loss function specified in compile().

- Key parameters

train_images: input data (50,000) of shape (32,32,3)

train_labels: contains the correct class labels for each image

epochs = 10

validation data → unseen data

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 44ms/step - accuracy: 0.2989 - loss: 2.5090 - val_accuracy: 0.4936 - val_loss: 1.3968
```

time taken per batch

Total time taken | epoch

no. of batches = 1563 per epoch

training accuracy

training loss

(testing) Validation accuracy

validation (testing) loss

Total time taken | epoch — training accuracy, training loss, (testing) validation accuracy - validation (testing) loss

no. of batches = 1563 per epoch

training image = 50,000

Std. batch size = 32

$$\frac{50000}{32} \triangleq 1563$$

one image

Horse