CNN for CIFAR10 with Tensorflow 2[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#CNN-for-CIFAR10-with-Tensorflow-2)

This program trains a simple [Convolutional Neural Network](https://developers.google.com/machine-learning/glossary/#convolutional_neural_network) (CNN) to classify [CIFAR-10 images](https://www.cs.toronto.edu/~kriz/cifar.html). It uses [tf.keras](https://www.tensorflow.org/guide/keras), a high-level API to build and train models in TensorFlow, and [Keras Sequential API](https://www.tensorflow.org/guide/keras/overview), therefore creating and training the model will not take many lines of code.

Install required packages[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Install-required-packages)

In [1]:

*# Uncomment and run only once, the packages will remain.*  
*# Restart the kernel for installed packages to work.*  
*#pip install -U scikit-learn*  
*#pip install mlxtend*

Import TensorFlow and other libraries[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Import-TensorFlow-and-other-libraries)

In [2]:

*# TensorFlow and tf.keras*  
**import** tensorflow **as** tf  
  
*# Python standard library*  
**import** math  
  
*# Helper libraries*  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** sklearn.metrics **import** confusion\_matrix , classification\_report  
**from** sklearn.utils **import** shuffle  
**from** mlxtend.plotting **import** plot\_confusion\_matrix  
  
print(tf**.**\_\_version\_\_)

2.4.1

Define functions[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Define-functions)

In [3]:

*# calculate the softmax of a vector*  
*# in order to convert vector of outputs*  
*# to vector of probabilities*  
**def** softmax(vector):  
 e **=** np**.**exp(vector)  
 **return** e **/** e**.**sum()

Download and prepare the CIFAR-10 dataset[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Download-and-prepare-the-CIFAR-10-dataset)

The [CIFAR-10](https://www.cs.toronto.edu/~kriz/cifar.html) dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

In [4]:

*# Import and load the CIFAR-10 data directly from tf.keras*  
(train\_images, train\_labels), (test\_images, test\_labels) **=** tf**.**keras**.**datasets**.**cifar10**.**load\_data()

Loading the dataset returns four NumPy arrays:

* The train\_images and train\_labels arrays are the *training set*—the data the model uses to learn.
* The model is tested against the *test set*, the test\_images, and test\_labels arrays.

The images are 32x32 NumPy arrays, with pixel values ranging from 0 to 255. The *labels* are an array of arrays containing an integer, ranging from 0 to 9. These correspond to the *class* the image represents:

|  |  |
| --- | --- |
| **Label** | **Class** |
| 0 | Airplane |
| 1 | Automobile |
| 2 | Bird |
| 3 | Cat |
| 4 | Deer |
| 5 | Dog |
| 6 | Frog |
| 7 | Horse |
| 8 | Ship |
| 9 | Truck |

Each image is mapped to a single label. Since the *class names* are not included with the dataset, store them here to use later when plotting the images:

In [5]:

class\_names **=** ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',  
 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

Explore the data[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Explore-the-data)

Let's explore the format of the dataset before training the model. The following shows there are 50,000 images in the training set, with each image represented as 32 x 32 pixels:

In [6]:

train\_images**.**shape

Out[6]:

(50000, 32, 32, 3)

Likewise, there are 50,000 labels in the training set:

In [7]:

len(train\_labels)

Out[7]:

50000

Each label is an array containing an integer between 0 and 9

In [8]:

test\_labels

Out[8]:

array([[3],  
 [8],  
 [8],  
 ...,  
 [5],  
 [1],  
 [7]], dtype=uint8)

There are 10,000 images in the test set. Again, each image is represented as 32 x 32 pixels:

In [9]:

test\_images**.**shape

Out[9]:

(10000, 32, 32, 3)

And the test set contains 10,000 images labels:

In [10]:

len(test\_labels)

Out[10]:

10000

Choose a custom ratio between training and testing data[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Choose--a-custom-ratio-between-training-and-testing-data)

First, we shuffle the data randomly:

In [11]:

all\_images **=** np**.**concatenate((train\_images, test\_images))  
all\_labels **=** np**.**concatenate((train\_labels, test\_labels))  
  
all\_images, all\_labels **=** shuffle(train\_images, train\_labels, random\_state**=**0)

Then, we choose the ratio of test and train data:

In [12]:

*# choose the percentage of test images*  
test\_images\_percentage **=** 10  
  
train\_images\_percentage **=** 100 **-** test\_images\_percentage  
train\_images\_count **=** math**.**ceil(len(all\_images)**\***(train\_images\_percentage**/**100))  
  
train\_images, test\_images **=** np**.**split(all\_images, [train\_images\_count])  
train\_labels, test\_labels **=** np**.**split(all\_labels, [train\_images\_count])  
  
print("train images:", train\_images**.**shape)  
print("test images:", test\_images**.**shape)

train images: (45000, 32, 32, 3)  
test images: (5000, 32, 32, 3)

Preprocess the data[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Preprocess-the-data)

The data must be preprocessed before training the network. If we inspect the first image in the training set, we will see that the pixel values fall in the range of 0 to 255:

In [13]:

plt**.**figure()  
plt**.**imshow(train\_images[0])  
plt**.**colorbar()  
plt**.**grid(**False**)  
plt**.**show()



Scale (normalize) these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the training set and the testing set be preprocessed in the same way:

In [14]:

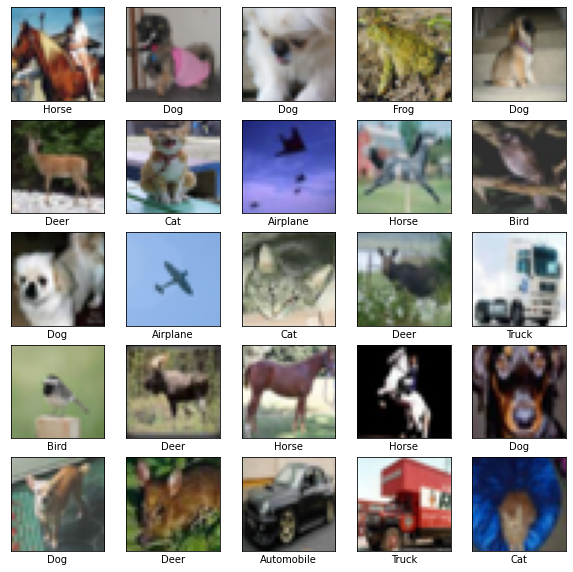
train\_images, test\_images **=** train\_images **/** 255.0, test\_images **/** 255.0

Verify the data[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Verify-the-data)

To verify that the data is in the correct format and that we're ready to build and train the network, let's display the first 25 images from the training set and display the class name below each image.

In [15]:

plt**.**figure(figsize**=**(10,10))  
**for** i **in** range(25):  
 plt**.**subplot(5,5,i**+**1)  
 plt**.**xticks([])  
 plt**.**yticks([])  
 plt**.**grid(**False**)  
 plt**.**imshow(train\_images[i], cmap**=**plt**.**cm**.**binary)  
 *# The CIFAR labels happen to be arrays,*   
 *# which is why you need the extra index*  
 plt**.**xlabel(class\_names[train\_labels[i][0]])  
plt**.**show()



Build the model[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Build-the-model)

Building the neural network requires configuring the layers of the model, then compiling the model.

Set up architecture[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Set-up-architecture)

The basic building block of a neural network is the [*layer*](https://www.tensorflow.org/api_docs/python/tf/keras/layers). Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers have parameters that are learned during training.

We use the [Keras Sequential API](https://www.tensorflow.org/guide/keras/overview) to create the model.

In [16]:

model **=** tf**.**keras**.**Sequential()

Create the convolutional base[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Create-the-convolutional-base)

The lines of code below define the convolutional base using a common pattern: a stack of [Conv2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D) and [MaxPooling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D) layers.

As input, a CNN takes tensors of shape (image\_height, image\_width, color\_channels), ignoring the batch size. color\_channels refers to (R,G,B). We will configure our CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. This can be done by passing the argument input\_shape to our first layer.

In [17]:

model**.**add(tf**.**keras**.**layers**.**Conv2D(32, (3, 3), activation**=**'relu', input\_shape**=**(32, 32, 3)))  
model**.**add(tf**.**keras**.**layers**.**MaxPooling2D((2, 2)))  
model**.**add(tf**.**keras**.**layers**.**Conv2D(64, (3, 3), activation**=**'relu'))  
model**.**add(tf**.**keras**.**layers**.**MaxPooling2D((2, 2)))  
model**.**add(tf**.**keras**.**layers**.**Conv2D(64, (3, 3), activation**=**'relu'))

Let's display the architecture of our model so far.

In [18]:

model**.**summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
conv2d (Conv2D) (None, 30, 30, 32) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 13, 13, 64) 18496   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 6, 6, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_2 (Conv2D) (None, 4, 4, 64) 36928   
=================================================================  
Total params: 56,320  
Trainable params: 56,320  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Above, we can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as we go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 or 64). Typically, as the width and height shrink, we can afford (computationally) to add more output channels in each Conv2D layer.

Add Dense layers on top[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Add-Dense-layers-on-top)

To complete our model, we will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor.

First, we will flatten (or unroll) the 3D output to 1D. The layer tf.keras.layers.Flatten transforms the format of the images from a 3D arrays containing two-dimensional arrays (of 32 by 32 pixels) to a one-dimensional array (of 32 \* 32 = 1024 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two tf.keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 64 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes of the CIFAR-10 dataset.

In [19]:

model**.**add(tf**.**keras**.**layers**.**Flatten())  
model**.**add(tf**.**keras**.**layers**.**Dense(64, activation**=**'relu'))  
model**.**add(tf**.**keras**.**layers**.**Dense(10))

Here's the complete architecture of our model.

In [20]:

model**.**summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
conv2d (Conv2D) (None, 30, 30, 32) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 13, 13, 64) 18496   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 6, 6, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_2 (Conv2D) (None, 4, 4, 64) 36928   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten (Flatten) (None, 1024) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 64) 65600   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 10) 650   
=================================================================  
Total params: 122,570  
Trainable params: 122,570  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We can see that our (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

Compile the model[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Compile-the-model)

Before the model is ready for training, it needs a few more settings. These are added during the model's [*compile*](https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile) step:

* [*Loss function*](https://www.tensorflow.org/api_docs/python/tf/keras/losses) —This measures how accurate the model is during training. We want to minimize this function to "steer" the model in the right direction.
* [*Optimizer*](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers) —This is how the model is updated based on the data it sees and its loss function.
* [*Metrics*](https://www.tensorflow.org/api_docs/python/tf/keras/metrics) —Used to monitor the training and testing steps. We use *accuracy*, the fraction of the images that are correctly classified.

In [21]:

model**.**compile(optimizer**=**'adam',  
 loss**=**tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(from\_logits**=True**),  
 metrics**=**['accuracy'])

Train the model[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Train-the-model)

Training the neural network model requires the following steps:

1. Feeding the training data to the model. In this case, the training data is in the train\_images and train\_labels arrays.
2. The model learns to associate images and labels.
3. We ask the model to make predictions about a test set—in this case, the test\_images array.
4. Verify that the predictions match the labels from the test\_labels array.

Feed the model[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Feed-the-model)

To start training, we call the [model.fit](https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) method—so called because it "fits" the model to the training data. We also pass testing data as validation\_data parameter on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data.

In [22]:

epoch\_count **=** 20  
history **=** model**.**fit(train\_images, train\_labels, batch\_size**=**32, epochs**=**epoch\_count, validation\_data**=**(test\_images, test\_labels))

Epoch 1/20  
1407/1407 [==============================] - 9s 4ms/step - loss: 1.7697 - accuracy: 0.3446 - val\_loss: 1.2934 - val\_accuracy: 0.5382  
Epoch 2/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 1.2300 - accuracy: 0.5614 - val\_loss: 1.1802 - val\_accuracy: 0.5828  
Epoch 3/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 1.0579 - accuracy: 0.6254 - val\_loss: 1.0168 - val\_accuracy: 0.6402  
Epoch 4/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.9364 - accuracy: 0.6719 - val\_loss: 0.9734 - val\_accuracy: 0.6540  
Epoch 5/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.8438 - accuracy: 0.7038 - val\_loss: 0.9274 - val\_accuracy: 0.6688  
Epoch 6/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.7920 - accuracy: 0.7195 - val\_loss: 0.8817 - val\_accuracy: 0.6888  
Epoch 7/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.7419 - accuracy: 0.7383 - val\_loss: 0.8764 - val\_accuracy: 0.7012  
Epoch 8/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.6757 - accuracy: 0.7646 - val\_loss: 0.8940 - val\_accuracy: 0.7002  
Epoch 9/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.6499 - accuracy: 0.7748 - val\_loss: 0.8658 - val\_accuracy: 0.7010  
Epoch 10/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.6045 - accuracy: 0.7855 - val\_loss: 0.9099 - val\_accuracy: 0.6882  
Epoch 11/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.5711 - accuracy: 0.8024 - val\_loss: 0.9041 - val\_accuracy: 0.7006  
Epoch 12/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.5346 - accuracy: 0.8111 - val\_loss: 0.8800 - val\_accuracy: 0.7052  
Epoch 13/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.4888 - accuracy: 0.8274 - val\_loss: 0.9669 - val\_accuracy: 0.6928  
Epoch 14/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.4738 - accuracy: 0.8307 - val\_loss: 0.9234 - val\_accuracy: 0.7102  
Epoch 15/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.4392 - accuracy: 0.8466 - val\_loss: 0.9736 - val\_accuracy: 0.6974  
Epoch 16/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.4098 - accuracy: 0.8552 - val\_loss: 0.9979 - val\_accuracy: 0.7042  
Epoch 17/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.3800 - accuracy: 0.8650 - val\_loss: 1.0754 - val\_accuracy: 0.7038  
Epoch 18/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.3580 - accuracy: 0.8710 - val\_loss: 1.0797 - val\_accuracy: 0.6942  
Epoch 19/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.3331 - accuracy: 0.8805 - val\_loss: 1.1744 - val\_accuracy: 0.6850  
Epoch 20/20  
1407/1407 [==============================] - 6s 4ms/step - loss: 0.3109 - accuracy: 0.8887 - val\_loss: 1.1478 - val\_accuracy: 0.6974

As the model trains, the loss and accuracy metrics are displayed (as well as the loss and accuracy of testing data ("val\_accuracy" here). This model reaches an accuracy of about 0.89 (or 89%) on the training data.

Evaluate the model[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Evaluate-the-model)

Next, compare how the model performs on the test dataset:

In [23]:

test\_loss, test\_acc **=** model**.**evaluate(test\_images, test\_labels, verbose**=**2)  
  
print('\nTest accuracy:', test\_acc)

157/157 - 0s - loss: 1.1478 - accuracy: 0.6974  
  
Test accuracy: 0.6973999738693237

It turns out that the accuracy on the test dataset is less than the accuracy on the training dataset. This gap between training accuracy and test accuracy represents *overfitting*. Overfitting happens when a machine learning model performs worse on new, previously unseen inputs than it does on the training data. An overfitted model "memorizes" the noise and details in the training dataset to a point where it negatively impacts the performance of the model on the new data. For more information, see the following:

* [Demonstrate overfitting](https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#demonstrate_overfitting)
* [Strategies to prevent overfitting](https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#strategies_to_prevent_overfitting)

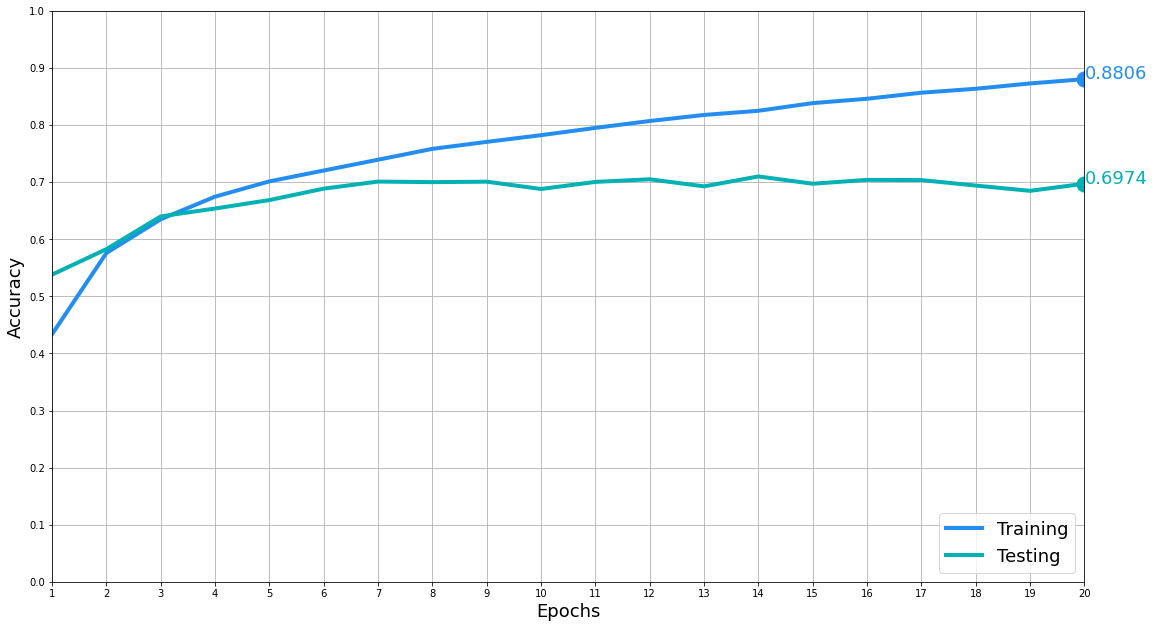
We can also plot the graph of the accuracy of both training and testing data, which changes after every epoch.

In [24]:

fig, ax **=** plt**.**subplots()  
fig**.**set\_size\_inches(18.5, 10.5)  
  
ax**.**xaxis**.**set\_ticks(np**.**arange(0, epoch\_count**+**1))  
ax**.**yaxis**.**set\_ticks(np**.**arange(0, 1.1, 0.1))  
ax**.**grid()  
  
train\_acc **=** history**.**history['accuracy']  
test\_acc **=** history**.**history['val\_accuracy']  
  
color\_train**=**'#238ef0'  
color\_test**=**'#01b1b3'  
  
plt**.**rcParams**.**update({'font.size': 18})  
plt**.**plot(np**.**arange(1,epoch\_count**+**1), train\_acc, label**=**'Training', c**=**color\_train, linewidth**=**4)  
plt**.**plot(np**.**arange(1,epoch\_count**+**1), test\_acc, label**=**'Testing', c**=**color\_test, linewidth**=**4)  
  
plt**.**plot(epoch\_count, train\_acc[**-**1],'co', c**=**color\_train, markersize**=**15)  
plt**.**text(epoch\_count, train\_acc[**-**1], "{:.4f}"**.**format(train\_acc[**-**1]), c**=**color\_train)  
  
plt**.**plot(epoch\_count, test\_acc[**-**1], 'co', c**=**color\_test, markersize**=**15)  
plt**.**text(epoch\_count, test\_acc[**-**1], "{:.4f}"**.**format(test\_acc[**-**1]), c**=**color\_test)  
  
plt**.**xlabel('Epochs', fontsize**=**18)  
plt**.**ylabel('Accuracy', fontsize**=**18)  
  
plt**.**xlim([1, epoch\_count])  
plt**.**ylim([0.0, 1.0])  
  
plt**.**legend(loc**=**'lower right')

Out[24]:

<matplotlib.legend.Legend at 0x7f18391dbc50>

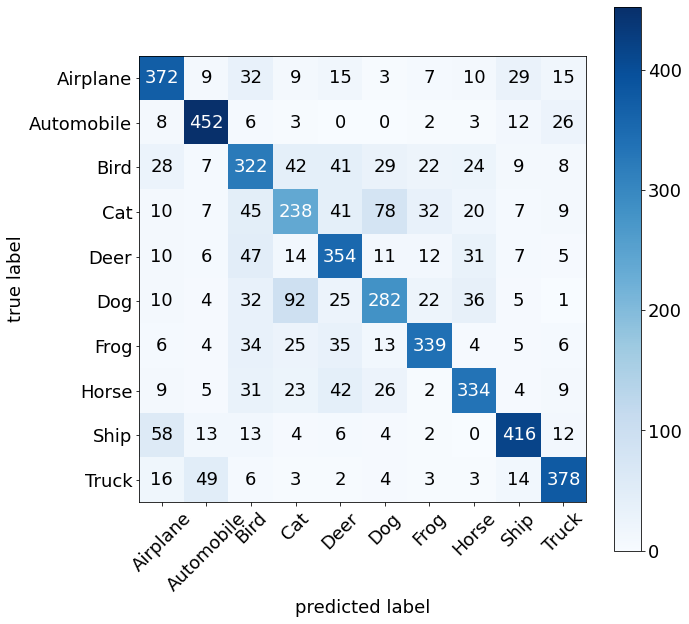


Classification report and a confusion matrix can also be generated:

In [25]:

predicted\_test\_images **=** model**.**predict(test\_images)  
predicted\_test\_image\_classes **=** [np**.**argmax(element) **for** element **in** predicted\_test\_images]  
  
print("\nClassification Report: \n\n",  
 classification\_report(test\_labels, predicted\_test\_image\_classes))  
  
cmat\_multiclass **=** confusion\_matrix(test\_labels, predicted\_test\_image\_classes)  
  
print("\n\nConfusion Matrix:")  
  
fig, ax **=** plot\_confusion\_matrix(conf\_mat**=**cmat\_multiclass,  
 colorbar**=True**,  
 show\_absolute**=True**,  
 show\_normed**=False**,  
 class\_names**=**class\_names,  
 figsize**=**(10,10))  
  
plt**.**show()

Classification Report:   
  
 precision recall f1-score support  
  
 0 0.71 0.74 0.72 501  
 1 0.81 0.88 0.85 512  
 2 0.57 0.61 0.59 532  
 3 0.53 0.49 0.51 487  
 4 0.63 0.71 0.67 497  
 5 0.63 0.55 0.59 509  
 6 0.77 0.72 0.74 471  
 7 0.72 0.69 0.70 485  
 8 0.82 0.79 0.80 528  
 9 0.81 0.79 0.80 478  
  
 accuracy 0.70 5000  
 macro avg 0.70 0.70 0.70 5000  
weighted avg 0.70 0.70 0.70 5000  
  
  
  
Confusion Matrix:



Our simple CNN has achieved a test accuracy of 69%. For another CNN style, we could use the Keras subclassing API and a tf.GradientTape. For that, see instructions [here](https://www.tensorflow.org/tutorials/quickstart/advanced).

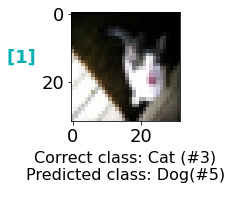
There is also a version of this program with changeable parameters and network architecture for further testing and model refinement.

Show classification results of some images[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#Show-classification-results-of-some-images)

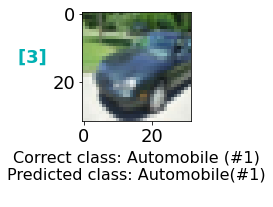
In [26]:

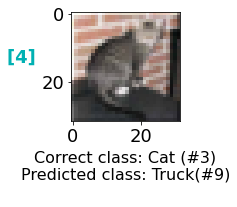
*# select how many images to present*  
sample\_img\_count **=** 30  
  
test\_image\_predictions **=** model**.**predict(test\_images[:sample\_img\_count])  
*#print(test\_image\_predictions)*  
  
*# convert vectors of outputs to vectors of probabilities*  
test\_image\_predictions **=** [softmax(x) **for** x **in** test\_image\_predictions]  
  
*# get the classes our CNN predicted*  
predicted\_classes **=** [np**.**argmax(element) **for** element **in** test\_image\_predictions]  
  
*# get the correct classes*  
correct\_classes **=** list(test\_labels[:sample\_img\_count]**.**reshape(**-**1,))  
  
**for** i **in** range(sample\_img\_count):  
 ylabel **=** f'[{i**+**1}]'  
   
 plt**.**figure(figsize **=** (15,2))  
 plt**.**imshow(test\_images[i])  
  
 xlabel **=** (  
 f'Correct class: {class\_names[correct\_classes[i]]} (#{correct\_classes[i]})\n'  
 f'Predicted class: {class\_names[predicted\_classes[i]]}(#{predicted\_classes[i]})\n'  
 )  
 ylabel **=** f'[{i**+**1}] '  
   
 plt**.**xlabel(xlabel, fontsize**=**16)  
 plt**.**ylabel(ylabel, fontsize**=**18, fontweight**=**'bold', color**=**'#01b1b3', rotation**=**0)

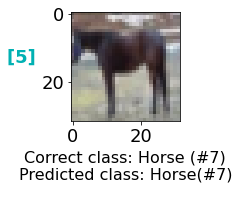
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:19: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max\_open\_warning`).

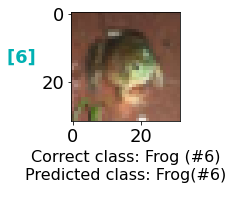


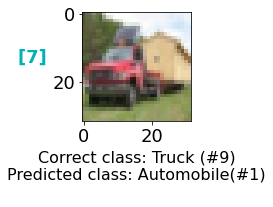




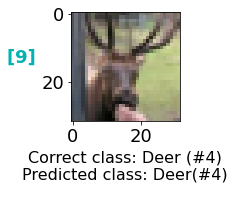






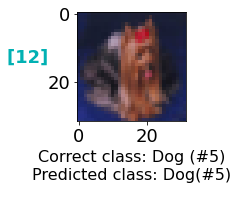




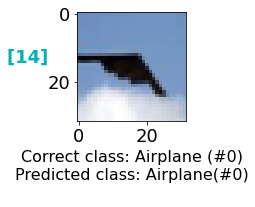


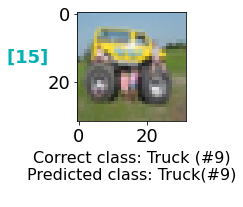




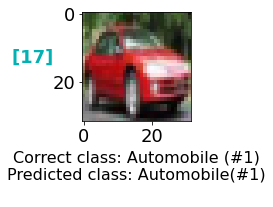


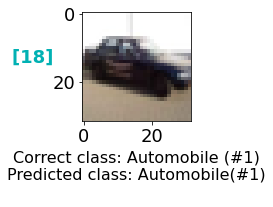


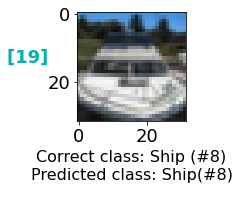


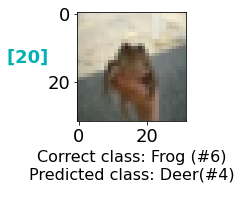


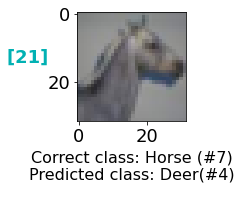


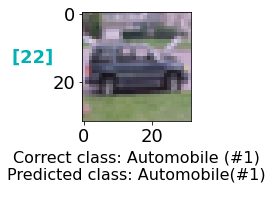


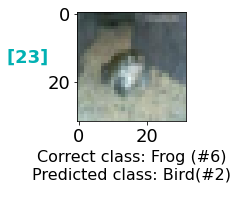


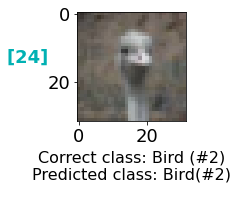


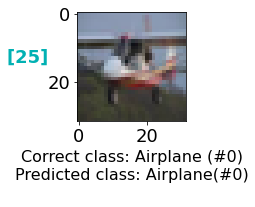


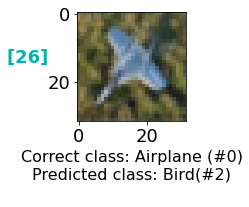


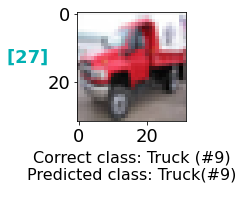


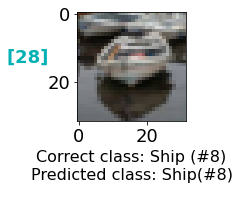


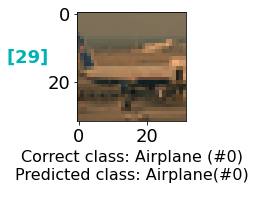


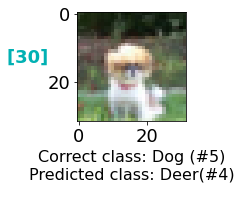












The classification probabilites for each class can also be printed for every image:[¶](https://n4m9dioyxt.clg07azjl.paperspacegradient.com/nbconvert/html/cnn_cifar10.ipynb?download=false#The-classification-probabilites-for-each-class-can-also-be-printed-for-every-image:)

In [27]:

print('\nOutput probabilities:\n ')  
  
**for** i **in** range(sample\_img\_count):  
 print(f'[{i**+**1}]:\n')  
  
 np**.**set\_printoptions(precision**=**4, suppress**=True**)  
 print(test\_image\_predictions[i])  
 np**.**set\_printoptions()  
 print('\n\n')

Output probabilities:  
   
[1]:  
  
[0. 0. 0. 0.0739 0.0003 0.9193 0. 0.0064 0. 0. ]  
  
  
  
[2]:  
  
[0. 0. 0.0002 0. 0. 0. 0. 0. 0.9998 0. ]  
  
  
  
[3]:  
  
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
  
  
  
[4]:  
  
[0. 0.0001 0.0002 0.0027 0. 0.0001 0. 0.0001 0.0002 0.9966]  
  
  
  
[5]:  
  
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]  
  
  
  
[6]:  
  
[0. 0. 0.0008 0.0022 0. 0.0002 0.9967 0. 0. 0. ]  
  
  
  
[7]:  
  
[0.1682 0.7352 0. 0. 0.0001 0. 0. 0.0007 0.0007 0.0952]  
  
  
  
[8]:  
  
[0. 0. 0. 0. 0.0057 0. 0. 0.9943 0. 0. ]  
  
  
  
[9]:  
  
[0. 0. 0.0009 0. 0.999 0. 0. 0. 0. 0. ]  
  
  
  
[10]:  
  
[0.0002 0. 0.0289 0.7106 0.0116 0.1898 0.0587 0.0002 0. 0. ]  
  
  
  
[11]:  
  
[0.0747 0.0027 0.0005 0.0001 0.0003 0. 0. 0.0001 0.0005 0.9211]  
  
  
  
[12]:  
  
[0. 0. 0.0162 0.0363 0.0056 0.7971 0.1379 0.006 0.0003 0.0005]  
  
  
  
[13]:  
  
[0.0004 0. 0.9752 0.0001 0.0003 0. 0.0239 0. 0. 0. ]  
  
  
  
[14]:  
  
[0.9629 0. 0.0371 0. 0. 0. 0. 0. 0. 0. ]  
  
  
  
[15]:  
  
[0.1256 0.0002 0. 0.0001 0.0097 0. 0. 0.0001 0.0004 0.8639]  
  
  
  
[16]:  
  
[0.0005 0. 0.0715 0.0288 0.8635 0.0273 0.0041 0.0044 0. 0. ]  
  
  
  
[17]:  
  
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
  
  
  
[18]:  
  
[0.0006 0.9675 0. 0.0001 0. 0. 0. 0. 0.0289 0.0029]  
  
  
  
[19]:  
  
[0.0003 0.4012 0. 0. 0. 0. 0. 0. 0.58 0.0185]  
  
  
  
[20]:  
  
[0.0001 0. 0.0111 0.0973 0.8187 0.0259 0.0072 0.0399 0. 0. ]  
  
  
  
[21]:  
  
[0.0874 0. 0.1484 0.2026 0.3155 0.0661 0.0082 0.1716 0.0001 0. ]  
  
  
  
[22]:  
  
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
  
  
  
[23]:  
  
[0.0022 0.0073 0.5345 0.0291 0.0415 0.1738 0.121 0.0868 0.0038 0.0001]  
  
  
  
[24]:  
  
[0. 0. 0.6867 0.0009 0.0002 0.3121 0.0001 0. 0. 0. ]  
  
  
  
[25]:  
  
[0.9858 0. 0.0004 0.0005 0. 0.0024 0.0082 0. 0.0026 0.0001]  
  
  
  
[26]:  
  
[0.0064 0. 0.9913 0.0005 0.0001 0. 0.0017 0. 0. 0. ]  
  
  
  
[27]:  
  
[0. 0.0048 0. 0. 0. 0. 0. 0. 0. 0.9952]  
  
  
  
[28]:  
  
[0. 0.0047 0.0033 0. 0. 0. 0. 0. 0.9919 0. ]  
  
  
  
[29]:  
  
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
  
  
  
[30]:  
  
[0. 0. 0.0187 0.0018 0.8222 0.1569 0.0004 0. 0. 0. ]