
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

< Back to Alex Krizhevsky's home page

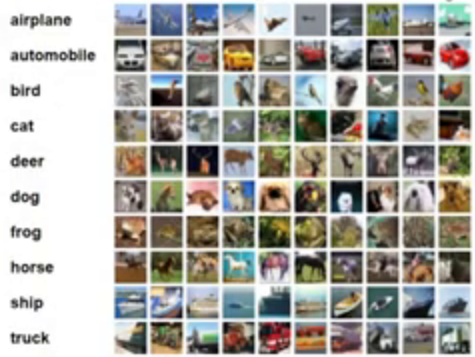
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

## The 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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

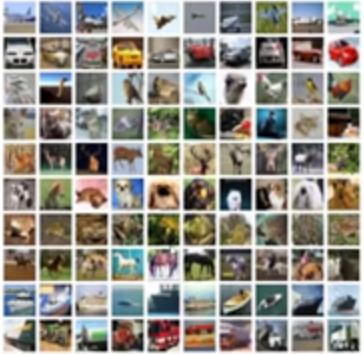



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes

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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:




The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

## Download

If you're going to use this dataset, please cite the tech report at the bottom of this page.

| Version | Size | md5sum |
| --- | --- | --- |

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 7 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
datasets.cifar10.load_data()
```

Type here to search

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 7 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
datasets.cifar10.load_data()
```

Type here to search

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
(X_train, y_train)datasets.cifar10.load_data()
```

Type here to search

CIFAR-10 and CIFAR-100 datasets | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla...

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
```

Type here to search

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]: (50000, 32, 32, 3)

Type here to search

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]:
```
(50000, 32, 32, 3)
```

In [ ]:

jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

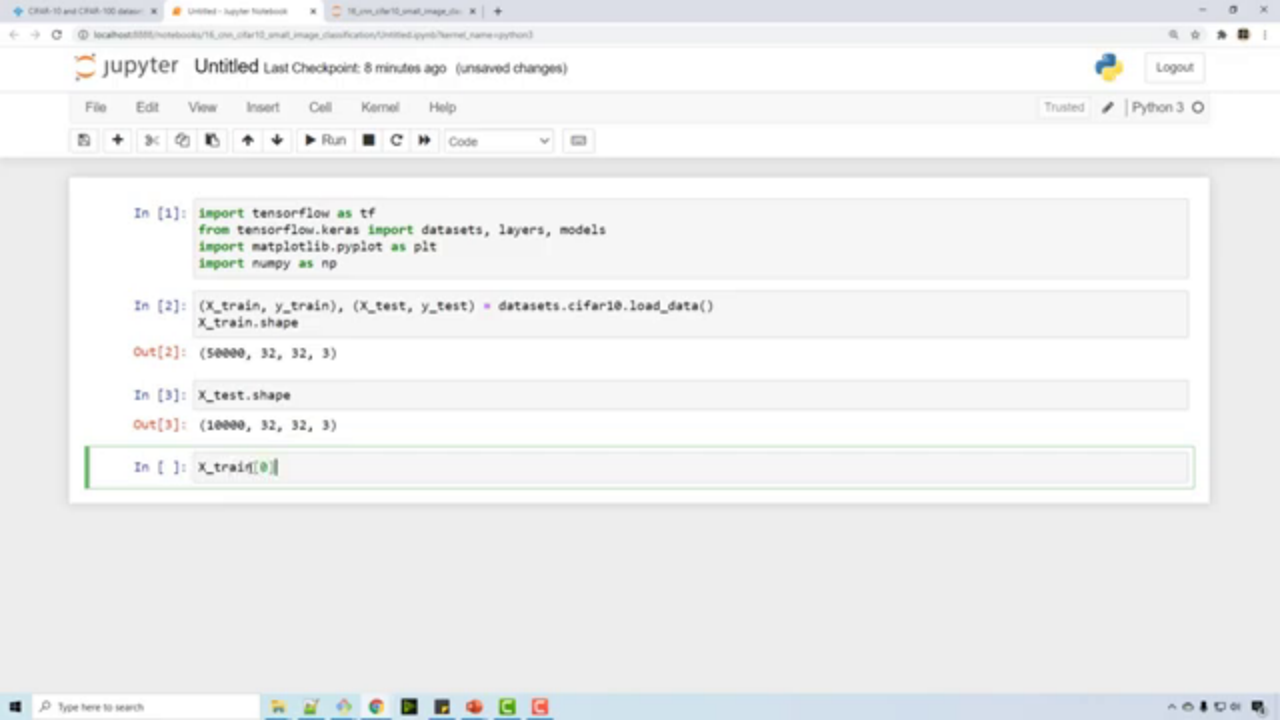
In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

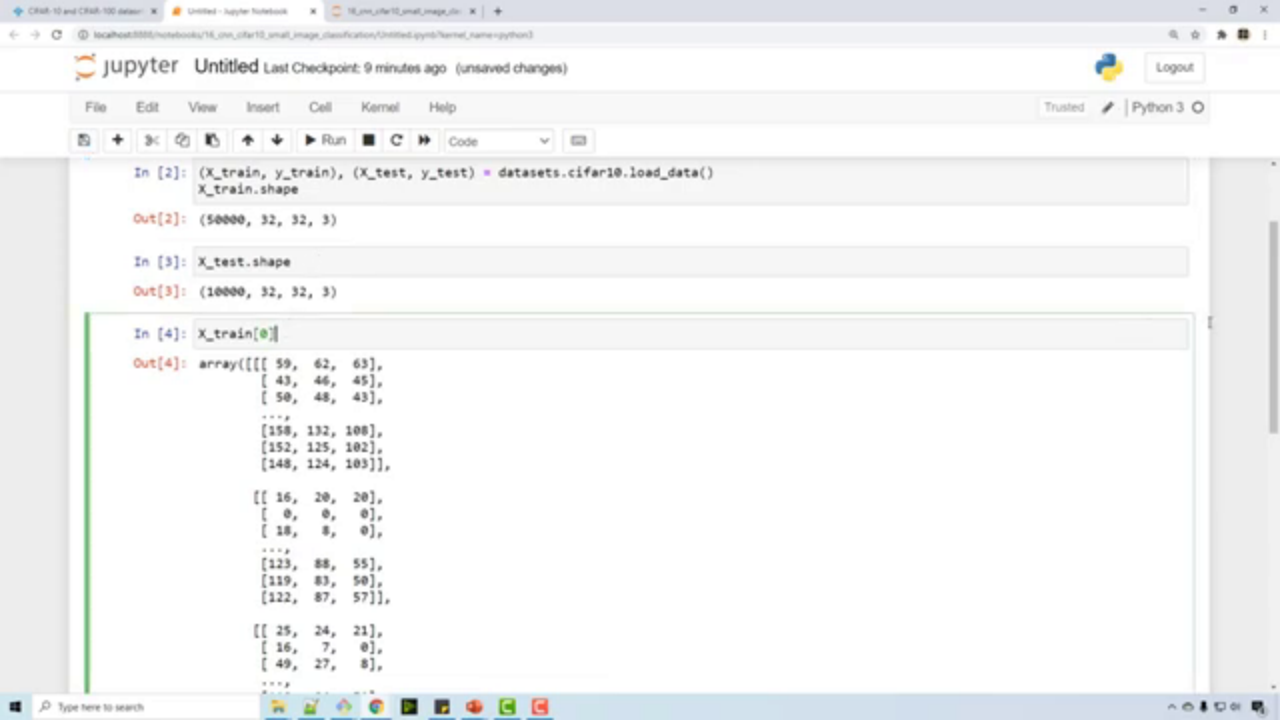
Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [ ]:


jupyter Untitled Last Checkpoint: 8 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
In [1]:
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
In [2]:
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
Out[2]:
(50000, 32, 32, 3)
In [3]:
X_test.shape
Out[3]:
(10000, 32, 32, 3)
In [ ]:
X_train[0]


jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
In [2]: (X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
Out[2]: (50000, 32, 32, 3)
In [3]: X_test.shape
Out[3]: (10000, 32, 32, 3)
In [4]: X_train[0]
Out[4]: array([[[ 59, 62, 63],
[ 43, 46, 45],
[ 50, 48, 43],
...,
[158, 132, 108],
[152, 125, 102],
[148, 124, 103]],
[[ 16, 20, 20],
[ 0, 0, 0],
[ 18, 8, 0],
...,
[123, 88, 55],
[119, 83, 50],
[122, 87, 57]],
[[ 25, 24, 21],
[ 16, 7, 0],
[ 49, 27, 8],

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [4]:
```
plt.X_train[0]
```

Out[4]:
```
array([[[ 59,  62,  63],
        [ 43,  46,  45],
        [ 50,  48,  43],
        ...,
        [158, 132, 108],
        [152, 125, 102],
        [148, 124, 103]],

       [[ 16,  20,  20],
        [  0,   0,   0],
        [ 18,   8,   0],
        ...,
        [123,  88,  55],
        [119,  83,  50],
        [122,  87,  57]],

       [[ 25,  24,  21],
        [ 16,   7,   0],
        [ 49,  27,   8],
        ...,
```

Type here to search

jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

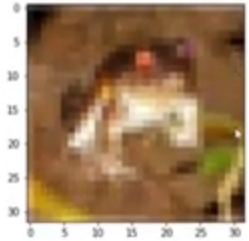
Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [5]:
```
plt.imshow(X_train[0])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x24c0d02c4c0>
```



jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

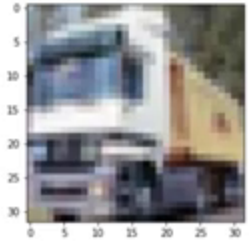
Run Code

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [6]:
```
plt.imshow(X_train[1])
```

Out[6]: <matplotlib.image.AxesImage at 0x24c0d109730>



jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [7]:
```
plt.figure(figsize = (15,2))
plt.imshow(X_train[1])
```

Out[7]: <matplotlib.image.AxesImage at 0x24c0d1702b0>

jupyter Untitled Last Checkpoint: 9 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

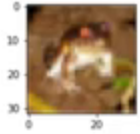
Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X_train[0])
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>



jupyter Untitled Last Checkpoint: 10 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

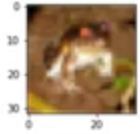
Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
```

Out[8]: <matplotlib.figure.AxesImage at 0x24c0d1b9af0>



< Back to Alex Krizhevsky's home page

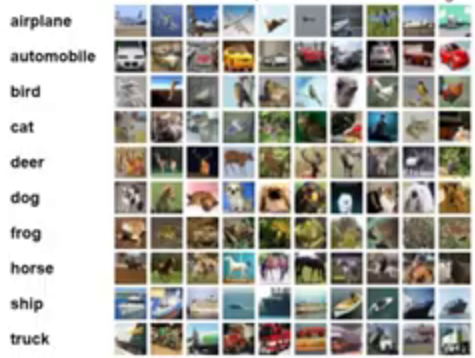
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

## The 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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:




The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes

jupyter Untitled Last Checkpoint: 10 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [2]:
```
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train.shape
```

Out[2]:
```
(50000, 32, 32, 3)
```

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel()
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```

jupyter Untitled Last Checkpoint: 10 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [ ]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel()
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>

jupyter Untitled Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

Out[2]: (50000, 32, 32, 3)

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>

jupyter Untitled Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [ ]:
```

```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>

jupyter Untitled Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [10]:
```
y_train.shape
```

Out[10]: (50000, 1)

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>

< Back to Alex Krizhevsky's home page

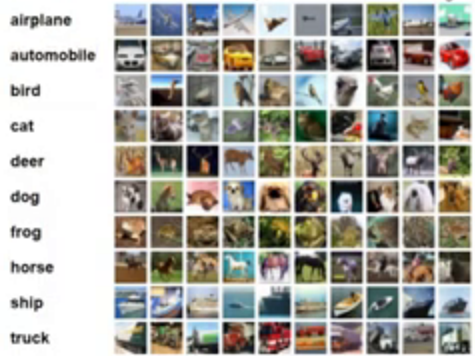
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

## The 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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:




The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes

jupyter Untitled Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]: (10000, 32, 32, 3)

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]: <matplotlib.image.AxesImage at 0x24c0d1b9af0>

jupyter Untitled Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

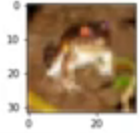
In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```



jupyter Untitled Last Checkpoint: 12 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Notebook saved Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

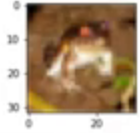
In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```



jupyter Untitled Last Checkpoint: 12 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [ ]:
```
y_train = y_train.s
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```

jupyter Untitled Last Checkpoint: 12 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [ ]:
```
y_train = y_train.shape()
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.figure.Figure at 0x24c0d1b9af0>
```

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter **Untitled** Last Checkpoint: 12 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [ ]:
```
y_train = y_train.shape(-1,)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```


Type here to search

jupyter Untitled Last Checkpoint: 12 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [ ]:
```
y_train = y_train.shape(-1,)
y_train
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```

jupyter Untitled Last Checkpoint: 12 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [3]:
```
X_test.shape
```

Out[3]:
```
(10000, 32, 32, 3)
```

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```

jupyter Untitled Last Checkpoint: 13 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.figure.Figure at 0x24c0d1b9af0>
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

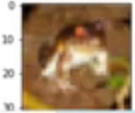
In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [14]:
```
classes[9]
```

Out[14]:
```
'truck'
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```



```
[1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

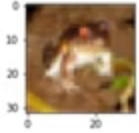
In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [14]:
```
classes[9]
```

Out[14]:
```
'truck'
```

In [8]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x24c0d1b9af0>
```



CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter Untitled Last Checkpoint: 13 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [15]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [ ]:

Type here to search

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_clas

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter **Untitled** Last Checkpoint: 13 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run | Code

In [11]:
```
y_train[:5]
```

Out[11]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [13]:
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [15]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [ ]:
```
plot_sample(X_train, y_train, )
```

Type here to search

jupyter Untitled Last Checkpoint: 13 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

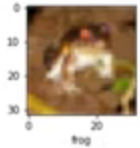
```
y_train[:5]
```

Out[13]:
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [15]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

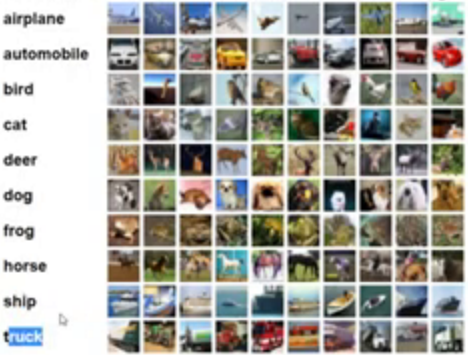
In [16]:
```
plot_sample(X_train, y_train, 0)
```



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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:




The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

## Download

If you're going to use this dataset, please cite the tech report at the bottom of this page.

| Version | Size | md5sum |
| --- | --- | --- |

jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

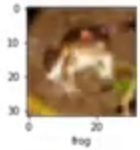
Trusted Python 3

Run Code

In [9]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [15]:
```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

In [16]:
```
plot_sample(X_train, y_train, 0)
```




In [*]:
```
plot_sample(X_train, y_train, 3)
```

jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

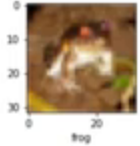
Logout

File Edit View Insert Cell Kernel Help

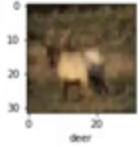
Trusted Python 3

Run Code

In [16]:
```
plot_sample(X_train, y_train, 0)
```




In [19]:
```
plot_sample(X_train, y_train, 3)
```



jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

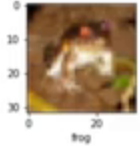
Logout

File Edit View Insert Cell Kernel Help

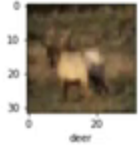
Trusted Python 3

Run Code

In [16]:
```
plot_sample(X_train, y_train, 0)
```




In [19]:
```
plot_sample(X_train, y_train, 3)
```



jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

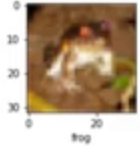
Logout

File Edit View Insert Cell Kernel Help

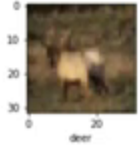
Trusted Python 3

Run Code

In [16]:
```
plot_sample(X_train, y_train, 0)
```




In [19]:
```
plot_sample(X_train, y_train, 3)
```



jupyter Untitled Last Checkpoint: 14 minutes ago (unsaved changes)

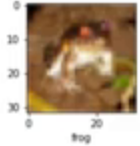
Logout

File Edit View Insert Cell Kernel Help

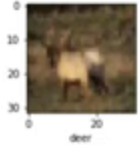
Trusted Python 3

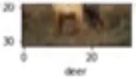
Run Code

In [16]:
```
plot_sample(X_train, y_train, 0)
```




In [19]:
```
plot_sample(X_train, y_train, 3)
```






In [20]:
```
X_train[0]
```

Out[20]:
```
array([[[ 59,  62,  63],
        [ 43,  46,  45],
        [ 50,  48,  43],
        ...,
        [158, 132, 108],
        [152, 125, 102],
        [148, 124, 103]],

       [[ 16,  20,  20],
        [  0,   0,   0],
        [ 18,   8,   0],
        ...,
        [123,  88,  55],
        [119,  83,  50],
        [122,  87,  57]],

       [[ 25,  24,  21],
        [ 16,   7,   0],
        [ 49,  27,   8],
        ...,
        [118,  84,  50],
        [120,  84,  50],
        [109,  73,  42]],
```


In [21]:
```
X_train[0]/255
```

Out[21]:
```
array([[[0.23137255, 0.24313725, 0.24705882],
        [0.16862745, 0.18039216, 0.17647059],
        [0.19607843, 0.18823529, 0.16862745],
        ...,
        [0.61960784, 0.51764706, 0.42352941],
        [0.59607843, 0.49019608, 0.4       ],
        [0.58039216, 0.48627451, 0.40392157]],

       [[0.0627451 , 0.07843137, 0.07843137],
        [0.        , 0.        , 0.        ],
        [0.07058824, 0.03137255, 0.        ],
        ...,
        [0.48235294, 0.34509804, 0.21568627],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117647, 0.22352941]],

       [[0.09803922, 0.09411765, 0.08235294],
        [0.0627451 , 0.02745098, 0.        ],
        [0.19215686, 0.10588235, 0.03137255],
        ...,
        [0.4627451 , 0.32941176, 0.19607843],
        [0.47058824, 0.32941176, 0.19607843],
        [0.42745098, 0.28627451, 0.16470588]],
```

jupyter Untitled Last Checkpoint: 15 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [19]:
```
plot_sample(X_train, y_train, 3)
```


In [21]:
```
X_train[0]/255
```

Out[21]:
```
array([[[0.23137255, 0.24313725, 0.24705882],
        [0.16862745, 0.18039216, 0.17647059],
        [0.19607843, 0.18823529, 0.16862745],
        ...,
        [0.61960784, 0.51764706, 0.42352941],
        [0.59607843, 0.49019608, 0.4       ],
        [0.58039216, 0.48627451, 0.40392157]],

       [[0.0627451 , 0.07843137, 0.07843137],
        [0.        , 0.        , 0.        ],
        [0.07058824, 0.03137255, 0.        ],
        ...,
        [0.48235294, 0.34509804, 0.21568627],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117647, 0.22352941]],
```

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter **Untitled** Last Checkpoint: 15 minutes ago (unsaved changes)

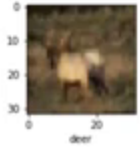
Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run | Code

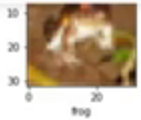
In [19]:
```
plot_sample(X_train, y_train, 3)
```



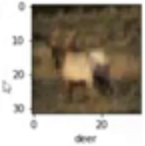

In [21]:
```
X_train = X_train / 255
X_test =
```

Out[21]:
```
array([[[0.23137255, 0.24313725, 0.24705882],
        [0.16862745, 0.18039216, 0.17647059],
        [0.19607843, 0.18823529, 0.16862745],
        ...,
        [0.61960784, 0.51764706, 0.42352941],
        [0.59607843, 0.49019608, 0.4       ],
        [0.58039216, 0.48627451, 0.40392157]],

       [[0.0627451 , 0.07843137, 0.07843137],
        [0.        , 0.        , 0.        ],
        [0.07058824, 0.03137255, 0.        ],
        ...,
        [0.48235294, 0.34509804, 0.21568627],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117647, 0.22352941]],
```

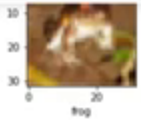
Type here to search




In [19]:
```
plot_sample(X_train, y_train, 3)
```

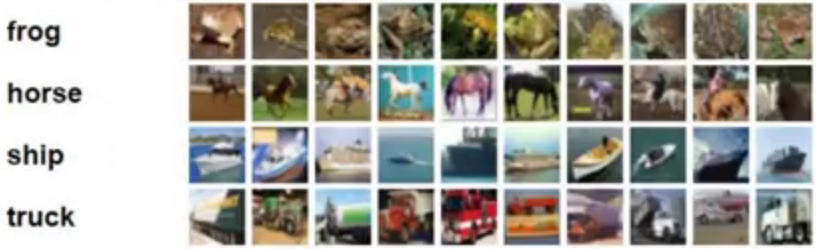



In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```




In [19]:
```
plot_sample(X_train, y_train, 3)
```


In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```




In [8]:
```
(X_train, y_train), (X_test,y_test) = tf.keras.datasets.cifar10.load_data()
```

In [9]:
```
X_train.shape
```

Out[9]:
```
(50000, 32, 32, 3)
```

In [10]:
```
y_train.shape
```

Out[10]:
```
(50000, 1)
```

**Data Visualization**

In [11]:
```
def plot_sample(index):
    plt.figure(figsize = (10,1))
    plt.imshow(X_train[index])
```

In [12]: plot_sample(0)

In [18]:
```
y_train_categorical[0:5]
```

Out[18]:
```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

**Model building and training**

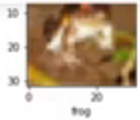
In [19]:
```
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32,32,3)),
        keras.layers.Dense(3000, activation='relu'),
        keras.layers.Dense(1000, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])

model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scaled, y_train_categorical, epochs=1)
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8642 - accuracy: 0.3328
```

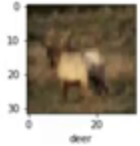
Out[19]:
```
<tensorflow.python.keras.callbacks.History at 0x24b348d8be0>
```

**Let's make some predictions**

In [ ]:
```
np.argmax(model.predict(X_test_scaled)[0])
```

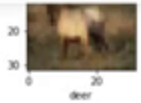



In [19]:
```
plot_sample(X_train, y_train, 3)
```



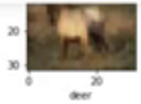

In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [ ]:
```

```



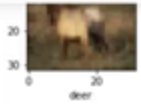

In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [ ]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```



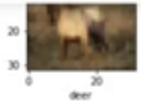

In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [ ]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```




In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [*]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```




In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [*]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1245/1563 [======================>.......] - ETA: 0s - loss: 1.5738 - accuracy: 0.4452
```

In [18]:
```
y_train_categorical[0:5]
```

Out[18]:
```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

**Model building and training**

In [19]:
```
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32,32,3)),
        keras.layers.Dense(3000, activation='relu'),
        keras.layers.Dense(1000, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])

model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_scaled, y_train_categorical, epochs=1)
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8642 - accuracy: 0.3328
```

Out[19]:
```
<tensorflow.python.keras.callbacks.History at 0x24b348d8be0>
```

**Let's make some predictions**

In [ ]:
```
np.argmax(model.predict(X_test_scaled)[0])
```

jupyter Untitled Last Checkpoint: 18 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

plane

tomobile

d

er

g

g

rse

ip

ck

y

0
0
0
0
0
0
0
0
1
0

y = 8

loss='sparse_categorical_crossentropy'

loss='categorical_crossentropy'

91 - accuracy: 0.3366

84 - accuracy: 0.4129

94 - accuracy: 0.4474

33 - accuracy: 0.4677

76 - accuracy: 0.4857

Type here to search

airplane 

automobile 

bird 

cat 

deer 

dog 

frog 

horse 

ship 

truck 

y

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

loss='categorical_crossentropy'

y = 8

loss='sparse_categorical_crossentropy'

airplane 

automobile 

bird 

cat 

deer 

dog 

frog 

horse 

ship 

truck 

$$y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$y = 8$

```
loss='sparse_categorical_crossentropy'
```

```
loss='categorical_crossentropy'
```

airplane 

automobile 

bird 

cat 

deer 

dog 

frog 

horse 

ship 

truck 

$$y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$y = 8$

```
loss='sparse_categorical_crossentropy'
```

```
loss='categorical_crossentropy'
```

```
    y_test, num_classes=10, dtype='float32'
)
```

In [17]:
```
y_train[0:5]
```

Out[17]:
```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

In [18]:
```
y_train_categorical[0:5]
```

Out[18]:
```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

**Model building and training**

In [19]:
```
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(32,32,3)),
        keras.layers.Dense(3000, activation='relu'),
        keras.layers.Dense(1000, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])

model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```


In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [23]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5704 - accuracy: 0.4474
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5083 - accuracy: 0.4677
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4576 - accuracy: 0.4857
```

In [22]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [23]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5704 - accuracy: 0.4474
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5083 - accuracy: 0.4677
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4576 - accuracy: 0.4857
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x24c0976edf0>
```

In [24]:
```
ann.evaluate(X_test, y_test)
```

```
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5704 - accuracy: 0.4474
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5083 - accuracy: 0.4677
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4576 - accuracy: 0.4857
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x24c0976edf0>
```

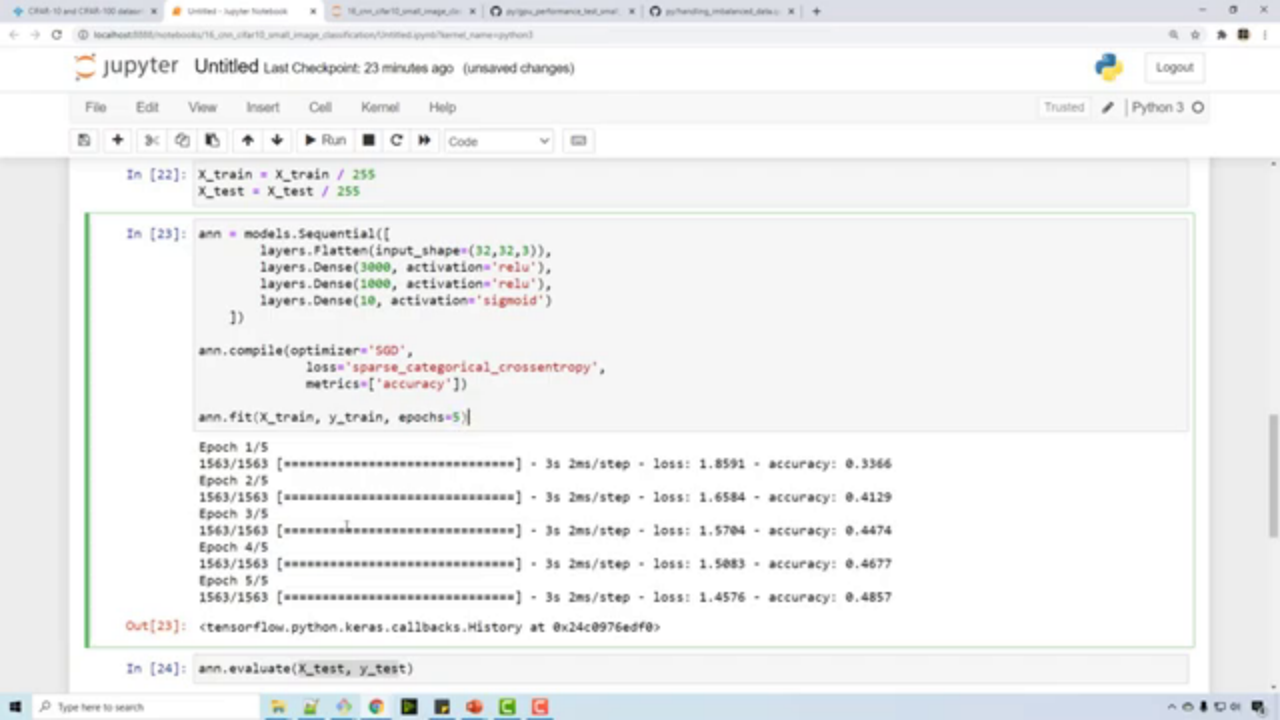
In [24]:
```
ann.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.4823 - accuracy: 0.4759
```

Out[24]:
```
[1.4822779893875122, 0.47589999437332153]
```

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
```


jupyter Untitled Last Checkpoint: 23 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
In [22]:
X_train = X_train / 255
X_test = X_test / 255
In [23]:
ann = models.Sequential([
layers.Flatten(input_shape=(32,32,3)),
layers.Dense(3000, activation='relu'),
layers.Dense(1000, activation='relu'),
layers.Dense(10, activation='sigmoid')
])
ann.compile(optimizer='SGD',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
ann.fit(X_train, y_train, epochs=5)
Epoch 1/5
1563/1563 [==========] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==========] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1563/1563 [==========] - 3s 2ms/step - loss: 1.5704 - accuracy: 0.4474
Epoch 4/5
1563/1563 [==========] - 3s 2ms/step - loss: 1.5083 - accuracy: 0.4677
Epoch 5/5
1563/1563 [==========] - 3s 2ms/step - loss: 1.4576 - accuracy: 0.4857
Out[23]:
<tensorflow.python.keras.callbacks.History at 0x24c0976edf0>
In [24]:
ann.evaluate(X_test, y_test)

jupyter Untitled Last Checkpoint: 23 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

jupyter Untitled Last Checkpoint: 23 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla | py/gpu_performance_test_small | py/handling_imbalanced_data.ip

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter **Untitled** Last Checkpoint: 23 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

Type here to search

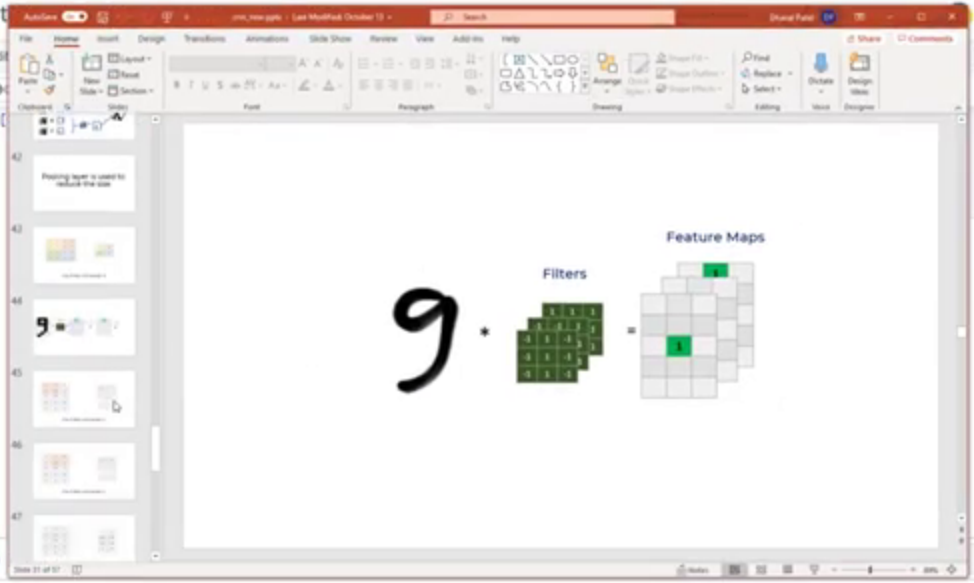
In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

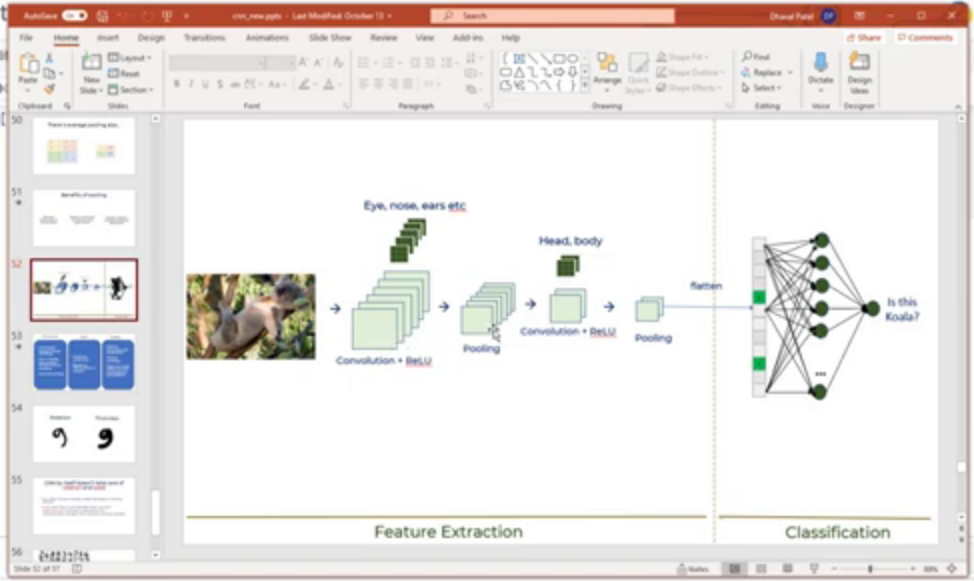
```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

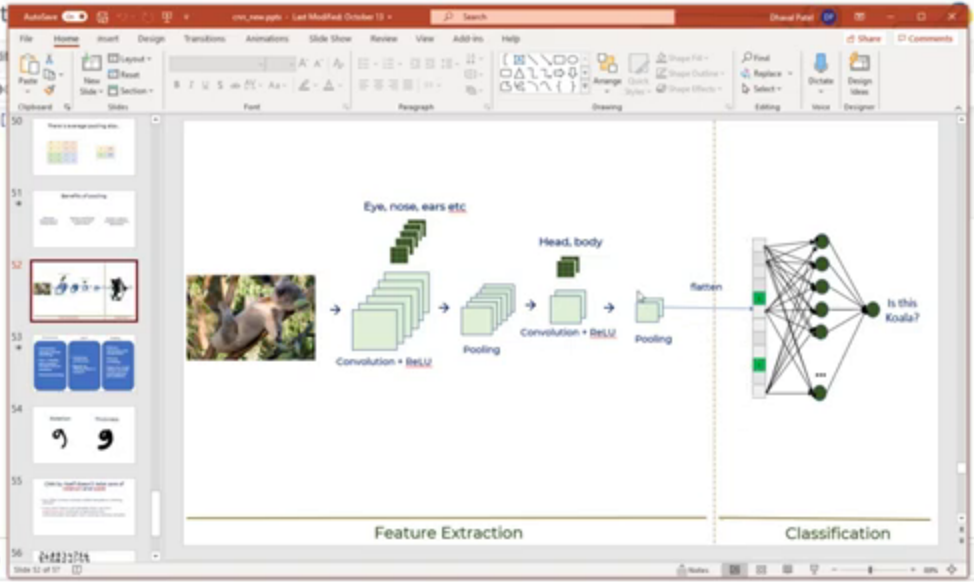
In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

In [ ]:


Feature Maps
Filters


Eye, nose, ears etc
Head, body
Convolution + ReLU
Pooling
Convolution + ReLU
Pooling
flatten
Is this Koala?
Feature Extraction
Classification


Eye, nose, ears etc
Head, body
Convolution + ReLU
Pooling
Convolution + ReLU
Pooling
flatten
Is this Koala?
Feature Extraction
Classification

jupyter Untitled Last Checkpoint: 24 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

In [ ]:

Type here to search

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])
```

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

    # cnn
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(3000, activation='relu'),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='sigmoid')
])
```

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])
```

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])
```

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

    # cnn
    # dense
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='sigmoid')
])
```

| | | | | |
|---|---|---|---|---|
| 0 | 0.71 | 0.35 | 0.47 | 1000 |
| 1 | 0.55 | 0.71 | 0.62 | 1000 |
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

    # cnn
    # dense
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

| | | | | |
|---|---|---|---|---|
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [ ]:

| | | | | |
|---|---|---|---|---|
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [ ]:
```
1: 0.45
2: 0.67
```

| | | | | |
|---|---|---|---|---|
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [ ]:
```
1: 0.45
2: 0.67

1: (0,45)
```

| | | | | |
|---|---|---|---|---|
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [ ]:
```
1: 0.45
2: 0.67

1: (0.45) / (0.45+0.67)
```

jupyter Untitled Last Checkpoint: 26 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

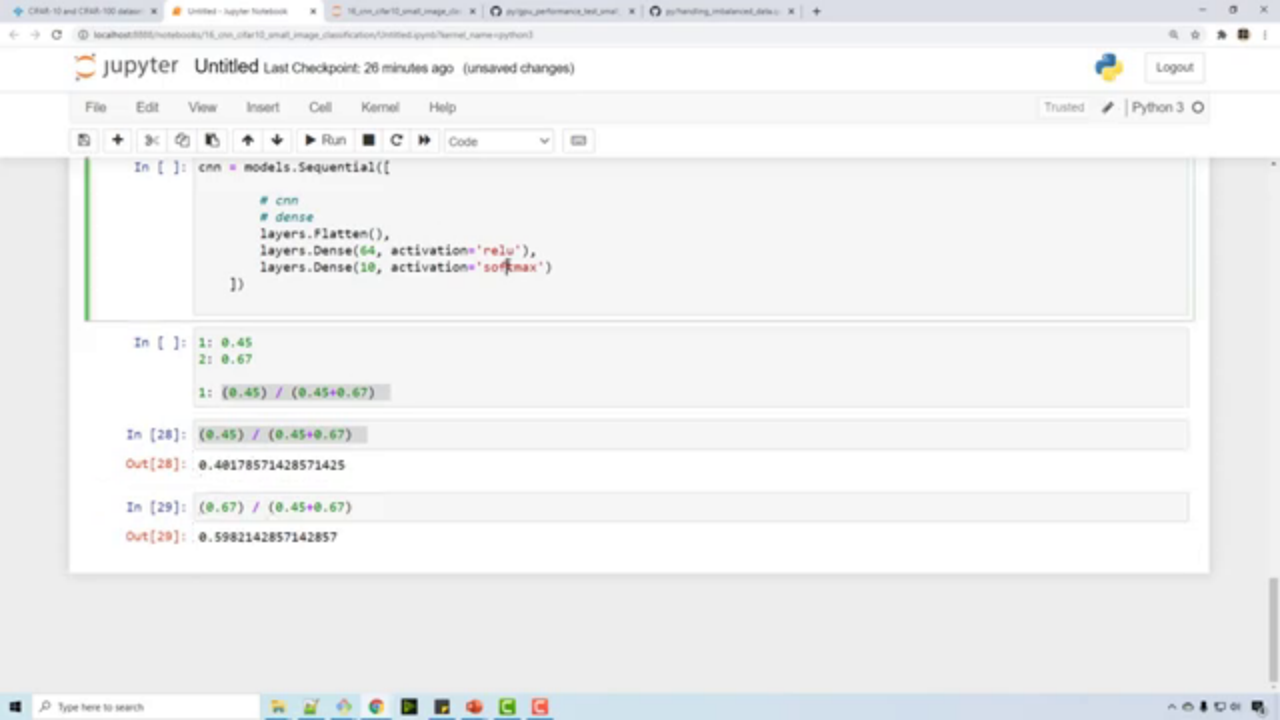
In [ ]:
```
cnn = models.Sequential([

        # cnn
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [ ]:
```
1: 0.45
2: 0.67

1: (0.45) / (0.45+0.67)
```

In [28]:
```
(0.45) / (0.45+0.67)
```

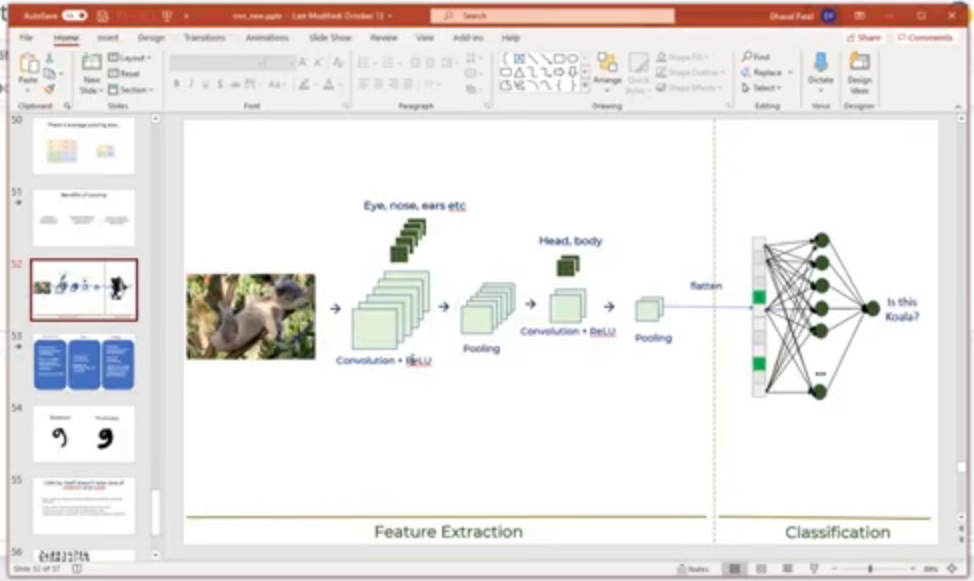
Out[28]:
```
0.40178571428571425
```

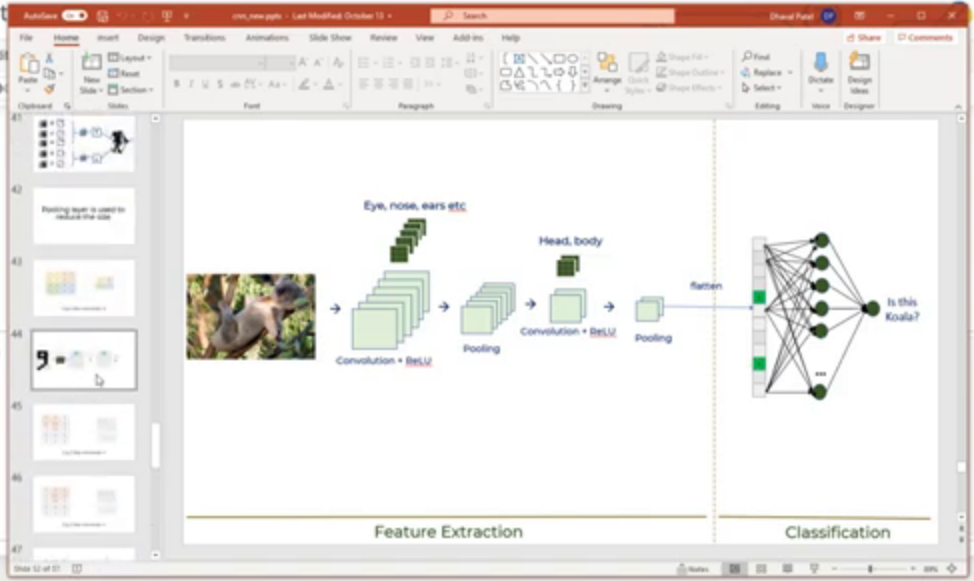
In [ ]:
```
(0.45) / (0.45+0.67)
```

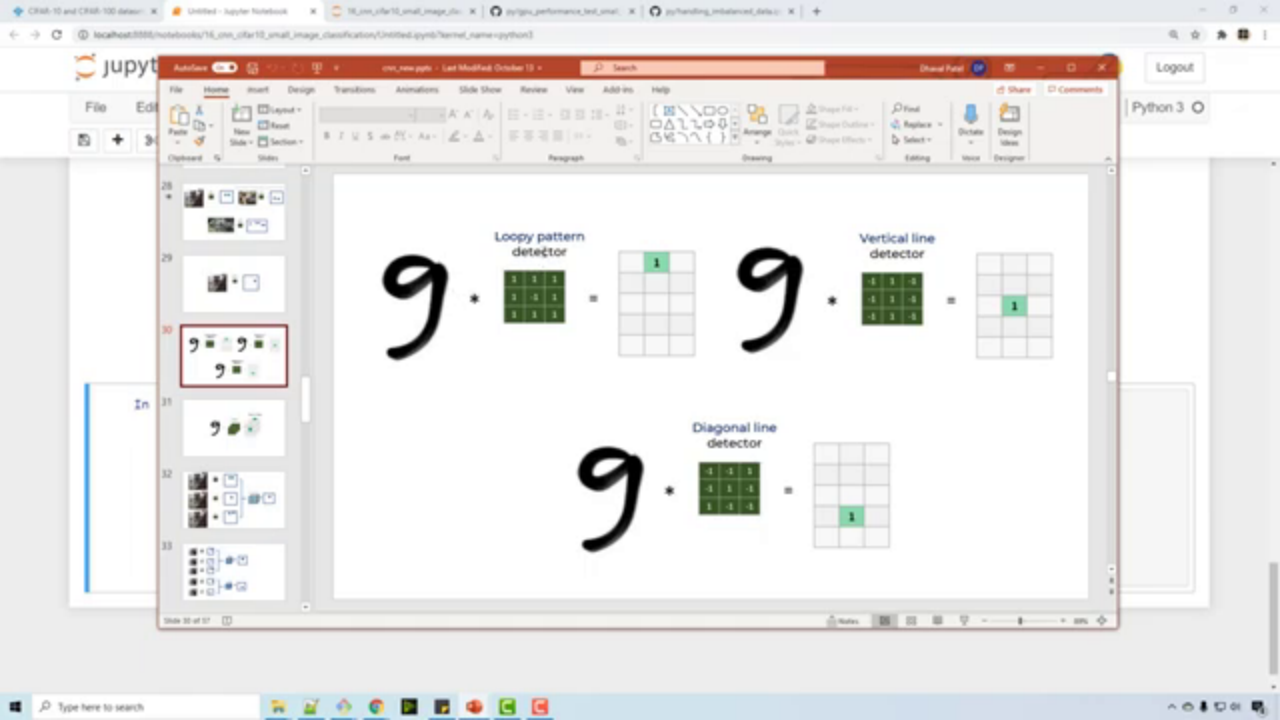

jupyter Untitled Last Checkpoint: 26 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
In [ ]:
cnn = models.Sequential([
# cnn
# dense
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])
In [ ]:
1: 0.45
2: 0.67
1: (0.45) / (0.45+0.67)
In [28]:
(0.45) / (0.45+0.67)
Out[28]:
0.40178571428571425
In [29]:
(0.67) / (0.45+0.67)
Out[29]:
0.5982142857142857

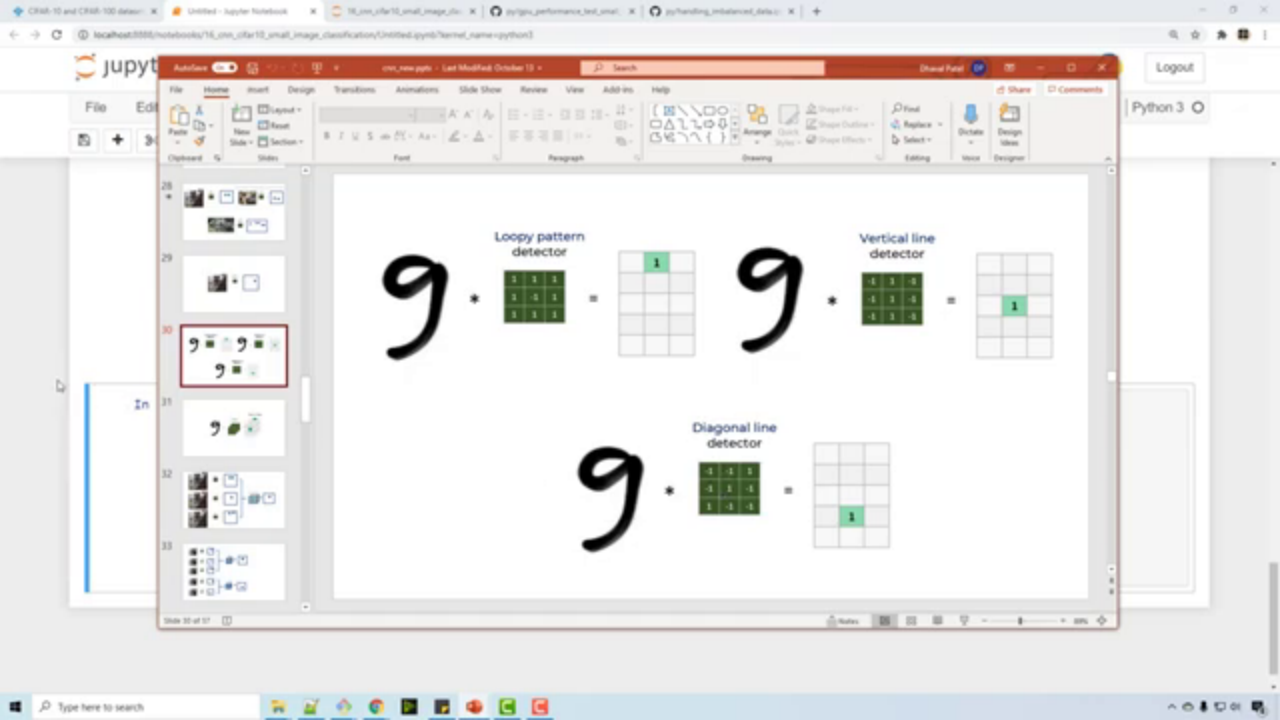
| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```


Eye, nose, ears etc
Head, body
flatten
Is this Koala?
Convolution + ReLU
Pooling
Convolution + ReLU
Pooling
Feature Extraction
Classification

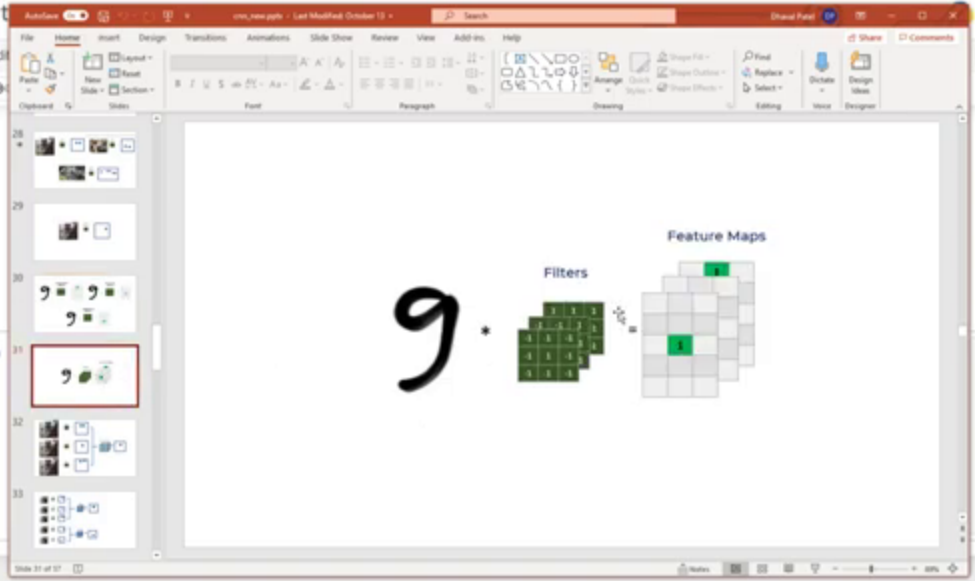

Eye, nose, ears etc
Head, body
Convolution + ReLU
Pooling
Convolution + ReLU
Pooling
flatten
Is this Koala?
Feature Extraction
Classification


Loopy pattern detector
Vertical line detector
Diagonal line detector


Loopy pattern detector
Vertical line detector
Diagonal line detector

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

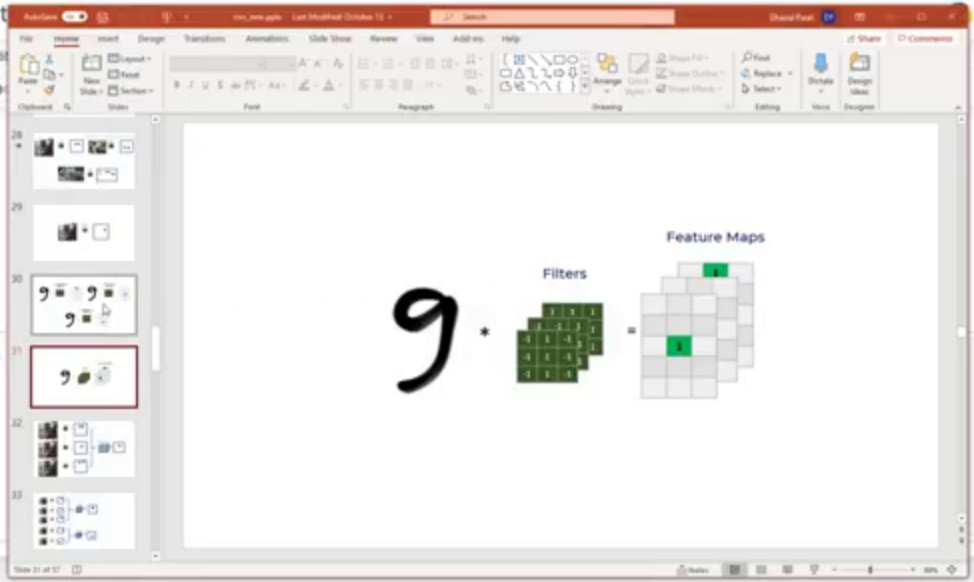

Feature Maps
Filters

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, ),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```


Feature Maps
Filters

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

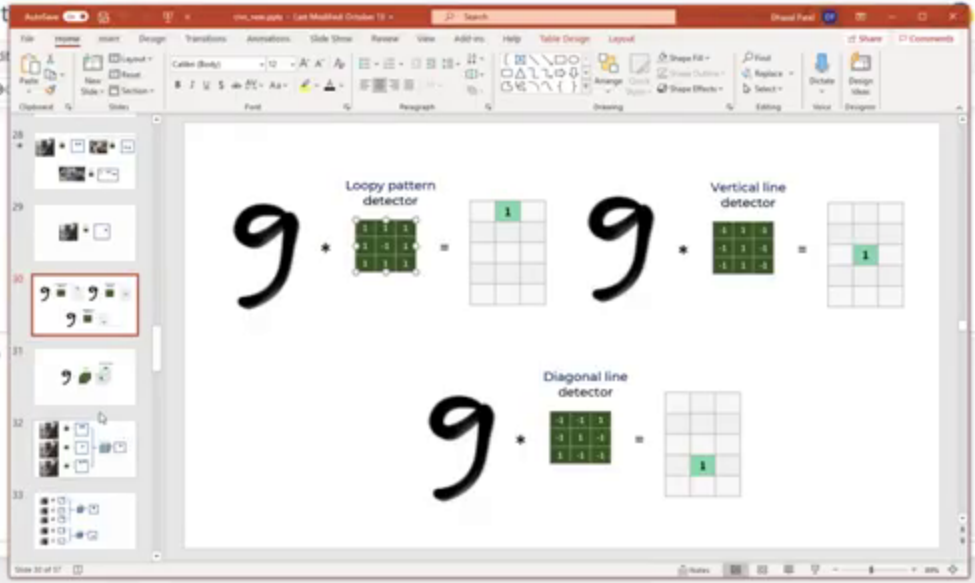
In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, activation='relu', kernel_size=(3,3)),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, activation='relu', kernel_size=(3,3)),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```


Loopy pattern detector
Vertical line detector
Diagonal line detector

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D(),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 2 | 0.33 | 0.41 | 0.36 | 1000 |
| 3 | 0.44 | 0.15 | 0.23 | 1000 |
| 4 | 0.34 | 0.51 | 0.40 | 1000 |
| 5 | 0.43 | 0.33 | 0.37 | 1000 |
| 6 | 0.43 | 0.68 | 0.52 | 1000 |
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| 7 | 0.63 | 0.45 | 0.53 | 1000 |
| 8 | 0.57 | 0.69 | 0.62 | 1000 |
| 9 | 0.59 | 0.48 | 0.53 | 1000 |
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [ ]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [30]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

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

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [30]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

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

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [30]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [31]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-bea1f0aede08> in <module>
----> 1 model.compile(optimizer='adam',
      2               loss='sparse_categorical_crossentropy',
      3               metrics=['accuracy'])

NameError: name 'model' is not defined
```

| | | | | |
|---|---|---|---|---|
| accuracy | | | 0.48 | 10000 |
| macro avg | 0.50 | 0.48 | 0.47 | 10000 |
| weighted avg | 0.50 | 0.48 | 0.47 | 10000 |

In [30]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [32]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

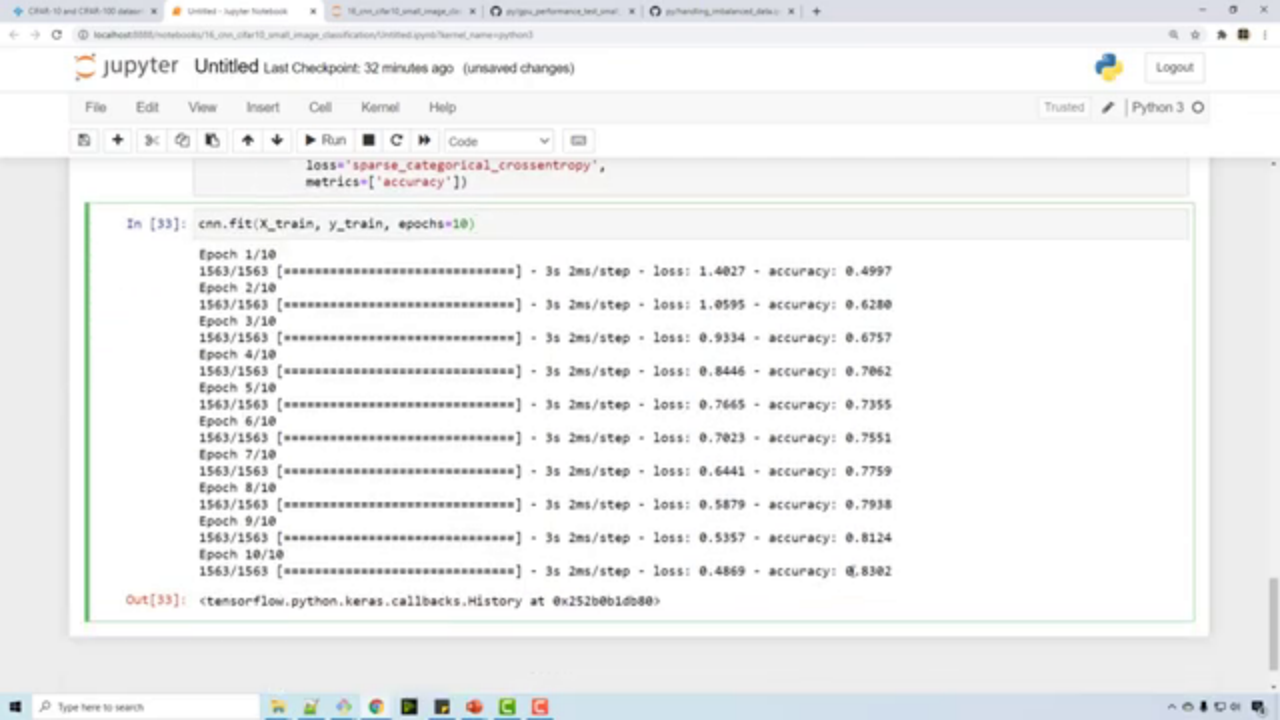
In [ ]:
```

```

```
        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [32]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

In [33]:
```
cnn.fit(X_train,y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4027 - accuracy: 0.4997
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
```


jupyter Untitled Last Checkpoint: 32 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted Python 3
Run Code
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
In [33]: cnn.fit(X_train, y_train, epochs=10)
Epoch 1/10
1563/1563 [==========] - 3s 2ms/step - loss: 1.4027 - accuracy: 0.4997
Epoch 2/10
1563/1563 [==========] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==========] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
Out[33]: <tensorflow.python.keras.callbacks.History at 0x252b0b1db80>
Type here to search

jupyter Untitled Last Checkpoint: 32 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [23]:
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='sigmoid')
    ])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8591 - accuracy: 0.3366
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.6584 - accuracy: 0.4129
Epoch 3/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5704 - accuracy: 0.4474
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.5083 - accuracy: 0.4677
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4576 - accuracy: 0.4857
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x24c0976edf0>
```

In [24]:
```
ann.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.4823 - accuracy: 0.4759
```

Out[24]:
```
[1.4823279809951782, 0.4758999943733215]
```

jupyter Untitled Last Checkpoint: 33 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [33]:
```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4027 - accuracy: 0.4997
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
```

In [ ]:

```
In [33]: cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4027 - accuracy: 0.4997
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
```

```
In [34]: cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

```
Out[34]: [1.0087039470672607, 0.6955999732017517]
```

```
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # dense
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

In [32]:
```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

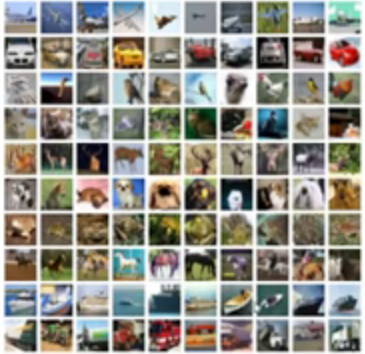
In [33]:
```
cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.4027 - accuracy: 0.4997
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
```

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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:




The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

## Download

If you're going to use this dataset, please cite the tech report at the bottom of this page.

| Version | Size | md5sum |
| --- | --- | --- |

```
Epoch 2/10
1563/1563 [==============================] - 3s 2ms/step - loss: 1.0595 - accuracy: 0.6280
Epoch 3/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
```

In [34]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
```

In [34]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [ ]:

```
1563/1563 [==============================] - 3s 2ms/step - loss: 0.9334 - accuracy: 0.6757
Epoch 4/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.8446 - accuracy: 0.7062
Epoch 5/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7665 - accuracy: 0.7355
Epoch 6/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.7023 - accuracy: 0.7551
Epoch 7/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.6441 - accuracy: 0.7759
Epoch 8/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5879 - accuracy: 0.7938
Epoch 9/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.8124
Epoch 10/10
1563/1563 [==============================] - 3s 2ms/step - loss: 0.4869 - accuracy: 0.8302
```

In [34]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [ ]:
```
plot_sample(X_test, y_test, 1)
```

jupyter Untitled Last Checkpoint: 34 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [34]:
```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [35]:
```
plot_sample(X_test, y_test, 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-fd538c14b1d6> in <module>
----> 1 plot_sample(X_test, y_test, 1)

<ipython-input-15-73f4cf12957f> in plot_sample(X, y, index)
      2     plt.figure(figsize = (15,2))
      3     plt.imshow(X[index])
----> 4     plt.xlabel(classes[y[index]])

TypeError: only integer scalar arrays can be converted to a scalar index
```

jupyter Untitled Last Checkpoint: 34 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

Out[34]: [1.0087039470672607, 0.6955999732017517]

In [ ]: y

In [35]:
```
y_test = y_test.reshape(-1,)
plot_sample(X_test, y_test, 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-fd538c14b1d6> in <module>
----> 1 plot_sample(X_test, y_test, 1)

<ipython-input-15-73f4cf12957f> in plot_sample(X, y, index)
      2     plt.figure(figsize = (15,2))
      3     plt.imshow(X[index])
----> 4     plt.xlabel(classes[y[index]])

TypeError: only integer scalar arrays can be converted to a scalar index
```

Out[34]: [1.0087039470672607, 0.6955999732017517]

In [37]:
```
y_test = y_test.reshape(-1,)
y_test[:5]
```

Out[37]: array([3, 8, 8, 0, 6], dtype=uint8)

In [35]:
```
plot_sample(X_test, y_test, 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-fd538c14b1d6> in <module>
----> 1 plot_sample(X_test, y_test, 1)

<ipython-input-15-73f4cf12957f> in plot_sample(X, y, index)
      2     plt.figure(figsize = (15,2))
      3     plt.imshow(X[index])
----> 4     plt.xlabel(classes[y[index]])

TypeError: only integer scalar arrays can be converted to a scalar index
```

jupyter Untitled Last Checkpoint: 35 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [34]:
```
cnn.evaluate(X_test,y_test)
```

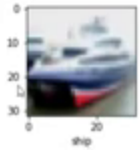
```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [37]:
```
y_test = y_test.reshape(-1,)
y_test[:5]
```

Out[37]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [38]:
```
plot_sample(X_test, y_test, 1)
```




In [ ]:

jupyter Untitled Last Checkpoint: 35 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

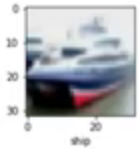
```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [37]:
```
y_test = y_test.reshape(-1,)
y_test[:5]
```

Out[37]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [38]:
```
plot_sample(X_test, y_test, 1)
```




In [ ]:

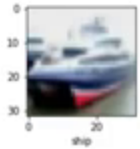
```
313/313 [==============================] - 0s 1ms/step - loss: 1.0087 - accuracy: 0.6956
```

Out[34]:
```
[1.0087039470672607, 0.6955999732017517]
```

In [37]:
```
y_test = y_test.reshape(-1,)
y_test[:5]
```

Out[37]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [38]:
```
plot_sample(X_test, y_test, 1)
```




In [39]:
```
y_pred = model.predict(X_test)
y_pred[:5]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-39-0c4c3462da8a> in <module>
----> 1 y_pred = model.predict(X_test)
      2 y_pred[:5]

NameError: name 'model' is not defined
```

jupyter Untitled Last Checkpoint: 35 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code


In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723928e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23988665e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```


In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
         1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
         5.23908669e-03, 3.95585485e-05],
        [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
         4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
         9.95130420e-01, 1.16837355e-05],
        [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
         4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
         9.75176334e-01, 1.32042286e-03],
        [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
         4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
         1.18420972e-03, 2.00188282e-04],
        [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
         5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
         2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

jupyter Untitled Last Checkpoint: 36 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code


In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
         1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
         5.23908669e-03, 3.95585485e-05],
        [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
         4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
         9.95130420e-01, 1.16837355e-05],
        [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
         4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
         9.75176334e-01, 1.32042286e-03],
        [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
         4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
         1.18420972e-03, 2.00188282e-04],
        [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
         5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
         2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [ ]:
```
np.
```


In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [41]:
```
np.argmax([5,12,1,2])
```

Out[41]:
```
1
```

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [41]:
```
np.argmax([5,12,1,2])
```

Out[41]:
```
1
```

ship

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [42]:
```
np.argmax([5,12,167,2])
```

Out[42]:
```
2
```

ship

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [43]:
```
np.argmax(y_pred[0])
```

Out[43]:
```
6
```

ship

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [43]:
```
y_classes = np.argmax(y_pred[0])
```

Out[43]:
```
5
```

ship

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [43]:
```
y_classes = [np.argmax(element) for eleme
```

Out[43]:
```
5
```

ship

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [43]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_
```

Out[43]:
```
5
```

CIFAR-10 and CIFAR-100 dataset | Untitled - Jupyter Notebook | 16_cnn_cifar10_small_image_cla | py/gpu_performance_test_small | py/handling_imbalanced_data.ip

localhost:8888/notebooks/16_cnn_cifar10_small_image_classification/Untitled.ipynb?kernel_name=python3

jupyter **Untitled** Last Checkpoint: 37 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run | Code

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [ ]:
```
y_test[:]
```

Type here to search

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [40]:
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

Out[40]:
```
array([[3.91144975e-04, 4.33854322e-04, 1.43723926e-04, 3.99961412e-01,
        1.47574965e-05, 5.93668640e-01, 6.97241412e-05, 3.79756239e-05,
        5.23908669e-03, 3.95585485e-05],
       [1.05845742e-04, 4.75216005e-03, 1.78524562e-09, 4.15040696e-10,
        4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
        9.95130420e-01, 1.16837355e-05],
       [1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
        4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
        9.75176334e-01, 1.32042286e-03],
       [9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
        4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
        1.18420972e-03, 2.00188282e-04],
       [1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
        5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
        2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
4.30966957e-10, 5.29897237e-10, 9.62223190e-10, 1.35456984e-14,
 9.95130420e-01, 1.16837355e-05],
[1.01393769e-02, 1.31144105e-02, 1.32235728e-04, 3.10876094e-05,
 4.81918505e-06, 4.86466633e-05, 3.23765053e-05, 4.00682353e-07,
 9.75176334e-01, 1.32042286e-03],
[9.41294730e-01, 2.95371632e-04, 4.70674075e-02, 4.77683637e-03,
 4.87070996e-03, 3.40701081e-05, 2.73908488e-04, 2.67446762e-06,
 1.18420972e-03, 2.00188282e-04],
[1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
 5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
 2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [ ]:
```
plot_sample(X_test, y_test, 1)
```

jupyter Untitled Last Checkpoint: 38 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

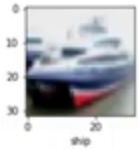
In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [46]:
```
plot_sample(X_test, y_test, 1)
```




In [ ]:

jupyter Untitled Last Checkpoint: 38 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

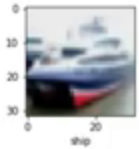
In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [46]:
```
plot_sample(X_test, y_test, 1)
```




In [ ]:
```
classes[y_classes[1]]
```

```
       5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
       2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

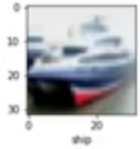
In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [46]:
```
plot_sample(X_test, y_test, 1)
```




In [48]:
```
classes[y_classes[1]]
```

Out[48]:
```
'ship'
```

```
[1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
 5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
 2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

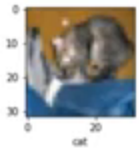
In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [49]:
```
plot_sample(X_test, y_test, 0)
```




In [50]:
```
classes[y_classes[0]]
```

Out[50]:
```
'dog'
```

jupyter Untitled Last Checkpoint: 38 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

```
[1.18030968e-07, 1.14894130e-04, 7.65499286e-03, 8.58013406e-02,
 5.15409887e-01, 4.64179739e-03, 3.86348605e-01, 6.14111229e-07,
 2.77474282e-05, 2.12781082e-08]], dtype=float32)
```

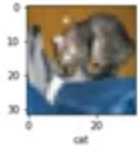
In [44]:
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

Out[44]:
```
[5, 8, 8, 0, 4]
```

In [45]:
```
y_test[:5]
```

Out[45]:
```
array([3, 8, 8, 0, 6], dtype=uint8)
```

In [49]:
```
plot_sample(X_test, y_test, 0)
```




In [50]:
```
classes[y_classes[0]]
```

Out[50]:
```
'dog'
```

jupyter Untitled Last Checkpoint: 38 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

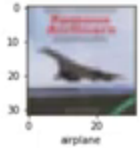
Run Code

Out[44]: [5, 8, 8, 0, 4]

In [45]: y_test[:5]

Out[45]: array([3, 8, 8, 0, 6], dtype=uint8)

In [52]: plot_sample(X_test, y_test,3)




In [50]: classes[y_classes[0]]

Out[50]: 'dog'


In [53]:
```
classes[y_classes[3]]
```

Out[53]: 'airplane'

In [54]:
```
print("Classification Report: \n", classification_report(y_test, y_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.72      0.73      1000
           1       0.78      0.85      0.81      1000
           2       0.60      0.59      0.60      1000
           3       0.62      0.40      0.49      1000
           4       0.65      0.65      0.65      1000
           5       0.52      0.67      0.59      1000
           6       0.75      0.79      0.77      1000
           7       0.78      0.71      0.74      1000
           8       0.75      0.84      0.79      1000
           9       0.80      0.73      0.76      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.70      0.70      0.69     10000
```


In [53]:
```
classes[y_classes[3]]
```

Out[53]: 'airplane'

In [54]:
```
print("Classification Report: \n", classification_report(y_test, y_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.72      0.73      1000
           1       0.78      0.85      0.81      1000
           2       0.60      0.59      0.60      1000
           3       0.62      0.40      0.49      1000
           4       0.65      0.65      0.65      1000
           5       0.52      0.67      0.59      1000
           6       0.75      0.79      0.77      1000
           7       0.78      0.71      0.74      1000
           8       0.75      0.84      0.79      1000
           9       0.80      0.73      0.76      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.70      0.70      0.69     10000
```

jupyter Untitled Last Checkpoint: 39 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Trusted Python 3

Run Code

In [27]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.35      0.47      1000
           1       0.55      0.71      0.62      1000
           2       0.33      0.41      0.36      1000
           3       0.44      0.15      0.23      1000
           4       0.34      0.51      0.40      1000
           5       0.43      0.33      0.37      1000
           6       0.43      0.68      0.52      1000
           7       0.63      0.45      0.53      1000
           8       0.57      0.69      0.62      1000
           9       0.59      0.48      0.53      1000

    accuracy                           0.48     10000
   macro avg       0.50      0.48      0.47     10000
weighted avg       0.50      0.48      0.47     10000
```

In [30]:
```
cnn = models.Sequential([
        # cnn
        layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)),
        layers.MaxPooling2D((2,2)),

        layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu')
```

Type here to search

Out[53]: 'airplane'

In [54]:
```
print("Classification Report: \n", classification_report(y_test, y_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.72      0.73      1000
           1       0.78      0.85      0.81      1000
           2       0.60      0.59      0.60      1000
           3       0.62      0.40      0.49      1000
           4       0.65      0.65      0.65      1000
           5       0.52      0.67      0.59      1000
           6       0.75      0.79      0.77      1000
           7       0.78      0.71      0.74      1000
           8       0.75      0.84      0.79      1000
           9       0.80      0.73      0.76      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.70      0.70      0.69     10000
```

433 lines (433 sloc) 14.8 KB Raw Blame

# Handwritten digits classification using CNN

In this notebook we will classify handwritten digits using a simple neural network (ANN) first and than repeat same thing with convolutional neural network. We will see how accuracy improves clickly when you use convolutional neural network.

In [27]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [28]:
```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

In [29]:
```
X_train.shape
```

Out[29]:
```
(60000, 28, 28)
```

In [30]:
```
X_test.shape
```

Out[30]:
```
(10000, 28, 28)
```

In [31]:
```
X_train[0].shape
```

# Handwritten digits classification using CNN

In this notebook we will classify handwritten digits using a simple neural network (ANN) first and than repeat same thing with convolutional neural network. We will see how accuracy improves clickly when you use convolutional neural network.

In [27]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [28]:
```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

In [29]:
```
X_train.shape
```

Out[29]:
```
(60000, 28, 28)
```

In [30]:
```
X_test.shape
```

Out[30]:
```
(10000, 28, 28)
```

In [31]:
```
X_train[0].shape
```

master

py / DeepLearningML / 16_cnn_cifar10_small_image_classification / cnn_cifar10_dataset.ipynb

Go to file

codebasics cifar10 image classification

Latest commit 594643a 42 minutes ago History

0 contributors

609 lines (609 sloc) 45.7 KB

Raw Blame

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
       0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.09019608, 0.25882353,
       0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
       0.03529412, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.21568627, 0.6745098 , 0.88627451, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.95686275, 0.52156863, 0.04313725,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.53333333, 0.99215686, 0.99215686, 0.99215686, 0.83137255,
       0.52941176, 0.51764706, 0.0627451 , 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.
```

In [59]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2959 - accuracy: 0.9185
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1368 - accuracy: 0.9603
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0995 - accuracy: 0.9703
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0771 - accuracy: 0.9772
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0628 - accuracy: 0.9806
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0519 - accuracy: 0.9841
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0442 - accuracy: 0.9865
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0369 - accuracy: 0.9886
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0300 - accuracy: 0.9910
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0264 - accuracy: 0.9917
```

Out[59]: <tensorflow.python.keras.callbacks.History at 0x1fe24629e80>

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution

```
In [173]: y_classes = [np.argmax(element) for element in y_pred]
          y_classes[:5]

Out[173]: [3, 8, 8, 0, 6]

In [175]: classes[y_classes[1]]

Out[175]: 'ship'

In [176]: classes[y_classes[3]]

Out[176]: 'airplane'
```

## Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/py/blob/master/DeepLearningML/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution