```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Load CIFAR-10
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# # Normalize
# x_train = x_train / 255.0
# x_test = x_test / 255.0
```

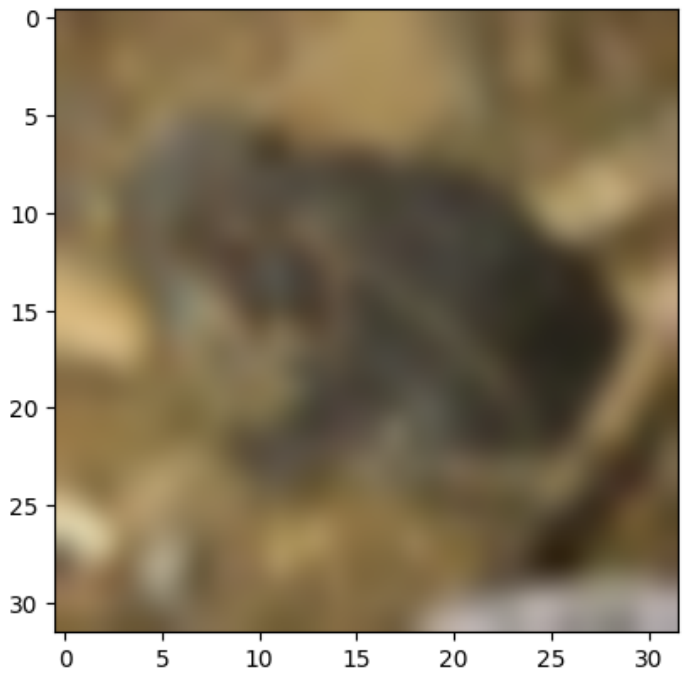
```
x_train.shape
```

```
(50000, 32, 32, 3)
```

```
x_train=x_train.reshape(-1,32,32,3)
```

```
plt.imshow(x_train[23],interpolation='bicubic')
```

```
<matplotlib.image.AxesImage at 0x7ade7a883e10>
```



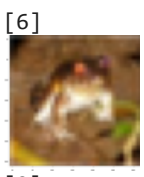

```
img=x_train[3]
```

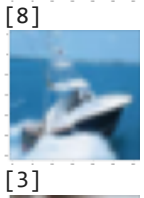
```
import matplotlib.pyplot as plt
import numpy as np

# Suppose img is your image array with shape (height, width, channels)
height, width = img.shape[:2]

plt.figure(figsize=(10,10), dpi=10)  # Set figure size proportional to image size
plt.imshow(img,interpolation='bilinear')
plt.axis('off')
plt.show()
```

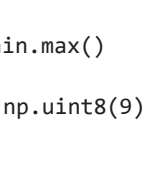
```
y_test
```

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

```
for i in range(10):
  plt.figure(figsize=(10,10),dpi=10)
  plt.imshow(x_train[i])
  print(y_train[i])
  plt.show()
```








```
y_train.max()
```

```
np.uint8(9)
```

```
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.activations import leaky_relu

class cnn():
    def __init__(self, x_train, x_test, y_train, y_test):
        self.x_train = x_train
        self.x_test = x_test
        self.y_train = y_train
        self.y_test = y_test

    def packs(self):
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import tensorflow as tf
        from tensorflow.keras.utils import to_categorical
        from tensorflow.keras.models import Sequential
        from tensorflow.keras import layers
        from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
        from tensorflow.keras.optimizers import Adam

    def eda(self):
        store = {}
        self.l = [self.x_train, self.x_test, self.y_train, self.y_test]
        self.names = ['x_train', 'x_test', 'y_train', 'y_test']
        for i, j in zip(self.l, self.names):
            store[j] = i.shape
        for i in range(4):
            if np.array_equal(self.l[i], self.x_train) or np.array_equal(self.l[i], self.x_test):
                self.x_train = self.x_train / 255
                self.x_test = self.x_test / 255
                self.x_train = self.x_train.reshape(-1, 32, 32, 3)
                self.x_test = self.x_test.reshape(-1, 32, 32, 3)
```

```
def labeling(self):
    self.y_train = to_categorical(self.y_train)
    self.y_test = to_categorical(self.y_test)

def dataaugmetation(self):
    self.aug = models.Sequential([
        layers.RandomCrop(32, 32),
        layers.RandomBrightness(0.2),
        layers.RandomZoom(0.3),
        layers.RandomRotation(0.3)
    ])

def model(self):
    self.model1 = models.Sequential([
        self.aug,
        layers.Conv2D(32, (3, 3), input_shape=(32, 32, 3)),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Flatten(),
        layers.Dense(128, activation='relu', kernel_regularizer=l1(0.01)),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu', kernel_regularizer=l1(0.01)),
        layers.Dense(10, activation='softmax')
    ])

def modelmoves(self):
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

    self.adam = Adam(learning_rate=0.001)
    self.early_stopping = EarlyStopping(patience=5, restore_best_weights=True)
    self.redulr = ReduceLROnPlateau(patience=4, factor=0.1)
    self.model1.compile(optimizer=self.adam, metrics=['accuracy'], loss='categorical_crossentropy')
    self.hist = self.model1.fit(self.x_train, self.y_train, epochs=34, batch_size=128,
                                validation_split=0.3, callbacks=[self.early_stopping, self.redulr])

def lossplots(self):
    import matplotlib.pyplot as plt
    plt.plot(self.hist.history['loss'])
    plt.plot(self.hist.history['val_loss'])

def acc(self):
    import matplotlib.pyplot as plt
    plt.plot(self.hist.history['accuracy'])
    plt.plot(self.hist.history['val_accuracy'])

def savemodel(self):
    self.model1.save("cirf_model.keras")
    self.m=self.model1

def model_predictions(self,path,m):
    from PIL import Image, ImageOps
    import numpy as np
    self.img = Image.open(path).convert("RGB")
    self.img = ImageOps.invert(self.img)
    self.img = self.img.resize((32, 32))
    self.img_array = np.array(self.img)
    self.img_array = self.img_array.astype('float') / 255
    self.img_array = self.img_array.reshape(1, 32, 32, 3)
    self.preds = m.predict(self.img_array)
    self.predic = np.argmax(self.preds)
    self.cofidence = np.max(self.preds)
    print(self.preds, self.cofidence, self.predic)


from keras.regularizers import l1


from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau


from tensorflow.keras.optimizers import Adam


c=cnn(x_train,x_test,y_train,y_test)
# c.packs()
# c.eda()
# c.labeling()
# c.dataaugmetation()
# c.x_train



c.packs()


c.eda()


c.x_train.shape
```

```
(50000, 32, 32, 3)
```

```
c.labeling()
```

```
c.dataaugmetation()
```

```
c.model()
```

```
c.modelmoves()
```

Show hidden output

```
c.lossplots()
```

Show hidden output

```
c.acc()
```

Show hidden output

```
from PIL import Image,ImageOps
```

```
path1='/content/download.jpeg'
```

```
c.savemodel()
```

```
c.m
```

```
<Sequential name=sequential_9, built=True>
```

```
c.model_predictions(path1,c.m)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
[[0.09889804 0.10007735 0.10094159 0.09946489 0.10065495 0.09951927
  0.10058565 0.10036055 0.09990454 0.09959321]] 0.10094159 2
```

```
# c.model1.save("cirf_model.keras")
```

```
# model1=c.model1
```

```
# model_predictions(path1,model1)
```

```
plt.imshow
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
# # c.savemodel()
# def model_predictions(path,mod):
#     from PIL import Image, ImageOps
#     import numpy as np

#     img = Image.open(path).convert("RGB")
#     img = ImageOps.invert(img)
#     img = img.resize((32, 32))
#     img_array = np.array(img)
#     img_array = img_array.astype('float') / 255
#     img_array = img_array.reshape(1, 32, 32, 3)  # Add batch dimension

#     preds = mod.predict(img_array)  # assuming model1 is accessible here
#     predic = np.argmax(preds)
#     cofidence = np.max(preds)
#     print(preds, cofidence, predic)
```

```
store={}
```

```
l=[1,2,3,4,5]
```

```
b=['a','k','h','i','l']
```

```
for i in l:
  store[i]=b


store


l=[2,3,4,5]
for i in range(4):
  if l[i]==2 or l[i]==3:
    l[i]=l[i]/4


l


Start coding or generate with AI.
```